Foundations and Trends® in
Electronic Design Automation
Vol. 2, No. 2 (2007) 135–253

DOI: 10.1561/1000000005

FPGA Architecture: Survey and Challenges

Ian Kuon[1], Russell Tessier[2] and Jonathan Rose[1]

[1] *The Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, {ikuon, jayar}@eecg.utoronto.ca*

[2] *Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA, tessier@ecs.umass.edu*

Abstract

Field-Programmable Gate Arrays (FPGAs) have become one of the key digital circuit implementation media over the last decade. A crucial part of their creation lies in their architecture, which governs the nature of their programmable logic functionality and their programmable interconnect. FPGA architecture has a dramatic effect on the quality of the final device's speed performance, area efficiency, and power consumption. This survey reviews the historical development of programmable logic devices, the fundamental programming technologies that the programmability is built on, and then describes the basic understandings gleaned from research on architectures. We include a survey of the key elements of modern commercial FPGA architecture, and look toward future trends in the field.

1

Introduction

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system. They provide a number of compelling advantages over fixed-function Application Specific Integrated Circuit (ASIC) technologies such as standard cells [62]: ASICs typically take months to fabricate and cost hundreds of thousands to millions of dollars to obtain the first device; FPGAs are configured in less than a second (and can often be reconfigured if a mistake is made) and cost anywhere from a few dollars to a few thousand dollars.

The flexible nature of an FPGA comes at a significant cost in area, delay, and power consumption: an FPGA requires approximately 20 to 35 times more area than a standard cell ASIC, has a speed performance roughly 3 to 4 times slower than an ASIC and consumes roughly 10 times as much dynamic power [120]. These disadvantages arise largely from an FPGA's programmable routing fabric which trades area, speed, and power in return for "instant" fabrication.

Despite these disadvantages, FPGAs present a compelling alternative for digital system implementation based on their fast-turnaround and low volume cost. For small enterprises or small entities within large corporations, FPGAs provide the only economical access to the scalability and performance provided by Moore's law. As Moore's law progresses, the ensuing difficulties brought about by state-of-the-art deep submicron processes make ASIC design more difficult and expensive. The investment required to produce a useful ASIC consists of several very large items in terms of time and money:

1. State-of-the-art ASIC CAD tools for synthesis, placement, routing, extraction, simulation, timing analysis, and power analysis are extremely costly.
2. The mask costs of a fully-fabricated device can be millions of dollars. This cost can be reduced if prototyping costs are shared among different, smaller ASICs, or if a "structured ASIC" approach, which requires fewer masks, is used.
3. The loaded cost of an engineering team required to develop a large ASIC over multiple years is huge. (This cost would be related, but smaller for an FPGA design team.)

These high costs, and the need for a proportionally higher return on investment, drive most digital design starts toward FPGA implementation.

The two essential technologies which distinguish FPGAs are architecture and the computer-aided design (CAD) tools that a user must employ to create FPGA designs. The goal of this survey is to examine the existing state of the art in FPGA architecture and to project future trends; a companion paper on CAD for FPGAs appeared in a previous edition of this journal [54].

The survey is organized as follows: we first give a brief overview of programmable logic to provide a context for the subsequent sections which review the history of programmable logic, and the underlying programming technologies. The following sections define the terminology of FPGA architecture, and then describe the foundations and trends in logic block architecture and routing architecture including a discussion of power management techniques and related circuit design issues. A brief overview of the input/output structures and architectural questions is then presented followed by an explicit comparison between FPGAs and competing ASIC standard cell technology. Finally, the survey concludes with a review of some of the design challenges facing FPGAs and a look at emerging architectures for FPGAs.

1.1 Overview

FPGAs, as illustrated in Figure 1.1, consist of an array of programmable logic blocks of potentially different types, including general logic, memory and multiplier blocks, surrounded by a programmable routing fabric that allows blocks to be programmably interconnected. The array is surrounded by programmable input/output blocks, labeled I/O in the figure, that connect the chip to the outside world.

The "programmable" term in FPGA indicates an ability to program a function into the chip after silicon fabrication is complete. This customization is made possible by the programming technology, which is a method that can cause a change in the behavior of the pre-fabricated chip after fabrication, in the "field," where system users create designs.

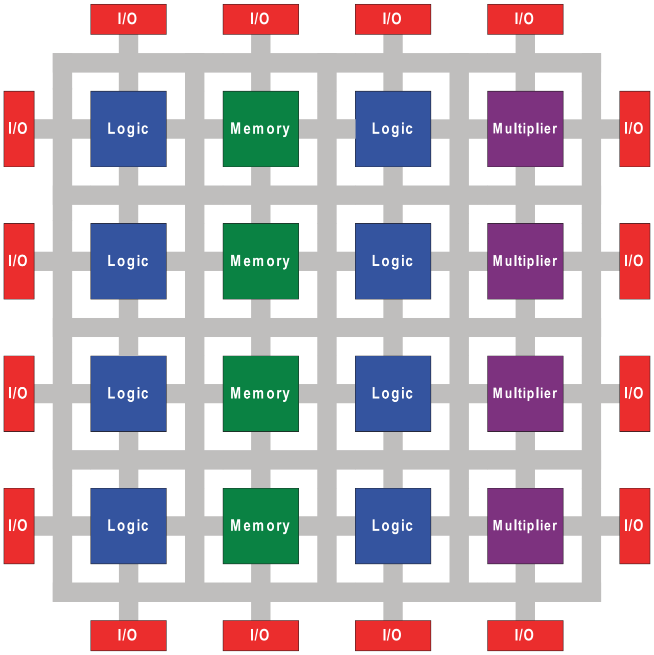


Fig. 1.1 Basic FPGA structure.

The first programmable logic devices used very small fuses as the programming technology. These devices are described briefly in the following section on the history of programmable logic. Section 3 goes into more detail on the three principal programming technologies in use today in modern FPGAs.

2

Early History of Programmable Logic

The origins of the contemporary Field-Programmable Gate Array are tied to the development of the integrated circuit in the early 1960s. Early programmable devices employed architectural regularity and functional flexibility. Cellular arrays [150] typically consisted of a two-dimensional array of simple logic cells with fixed, point-to-point communication. These first arrays, such as the Maitra cascade [143], contained logic cells which could be programmed via metalization during manufacturing to implement a range of two-input logic functions. By the mid-1960s, field-programmability, the ability to change the logic function of a chip after the fabrication process, was achieved via the introduction of “cutpoint” cellular arrays [150]. Although the connections between the elements of the array were fixed, the functionality of each logic cell in the array could be determined by setting programmable fuses. These fuses could be programmed in the field through the use of programming currents or photo-conductive exposure [150]. As a result, field-customization allowed for simplified array manufacturing and wider applicability.

In the 1970s, a series of read-only memory (ROM)-based programmable devices were introduced and provided a new way to implement logic functions. Although mask-programmable ROMs and fuse-programmable ROMs (PROMs) with N address inputs can implement any N-input logic function, area efficiency quickly becomes an issue for all but small values of N due to the exponential dependence of area on N. The first programmable logic arrays (PLAs) improved on this with two-level AND–OR logic planes (each plane in a wired-AND or wired-OR structure along with inverters can build any AND or OR logic term) that closely match the structure of common logic functions and are significantly more area-efficient. An example PLA is shown in Figure 2.1(a).

These architectures evolved further with the realization that sufficient flexibility was provided by a programmable AND plane followed by a fixed OR plane, in the programmable array logic (PAL) devices that were introduced in 1977 by Monolithic Memories Incorporated (MMI) [39]. As shown in Figure 2.1(b), it is notable that these devices contained programmable combinational logic which fed fixed sequential logic in the form of D-type flip–flop macrocells.

With these devices, logic functions must be implemented using one or more levels of two-level logic structures. Device inputs and

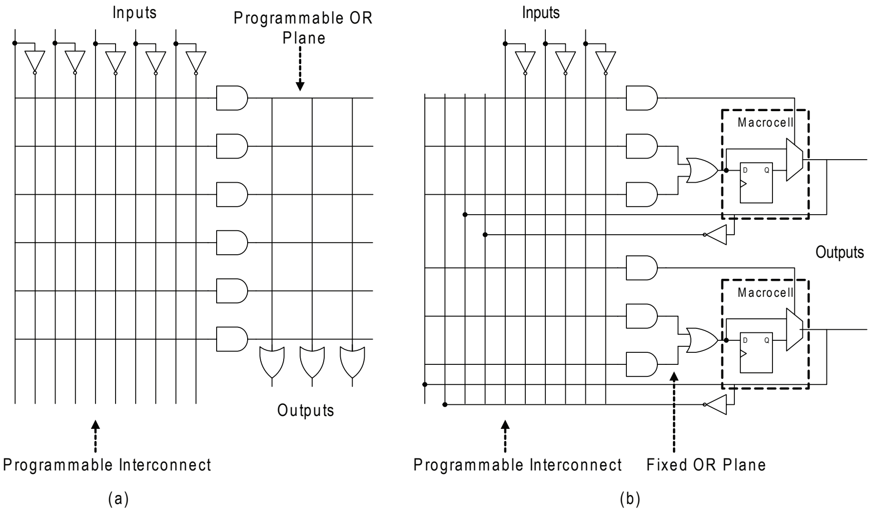


Fig. 2.1 PLA and PAL architectures.

intermediate combinational sums are fed into the array via a programmable interconnect that is typically a full cross-bar, leading to significant interconnect costs for this programmable architecture. For datapath and multi-level circuits, the area costs of two-level implementation quickly become prohibitive.

The first static memory-based FPGA (commonly called an SRAM-based FPGA) was proposed by Wahlstrom in 1967 [203]. This architecture allowed for both logic and interconnection configuration using a stream of configuration bits. Unlike its contemporary cellular array counterparts, both wide-input logic functions and storage elements could be implemented in each logic cell. Additionally, the programmable inter-cell connections could be easily changed (through memory-configurability) to enable the implementation of a variety of circuit topologies. Although static memory offers the most flexible approach to device programmability, it requires a significant increase in area per programmable switch compared to ROM implementations. It is likely this issue delayed the introduction of commercial static memory-based programmable devices until the mid-1980's, when the cost per transistor was sufficiently lowered.

The first modern-era FPGA was introduced by Xilinx in 1984 [49]. It contained the now classic array of Configurable Logic Blocks. From that first FPGA which contained 64 logic blocks and 58 inputs and outputs [49], FPGAs have grown enormously in complexity. Modern FPGAs now can contain approximately 330,000 equivalent logic blocks and around 1100 inputs and outputs [23, 231] in addition to a large number of more specialized blocks that have greatly expanded the capabilities of FPGAs. These massive increases in capabilities have been accompanied by significant architectural changes that will be described in the remainder of this survey.

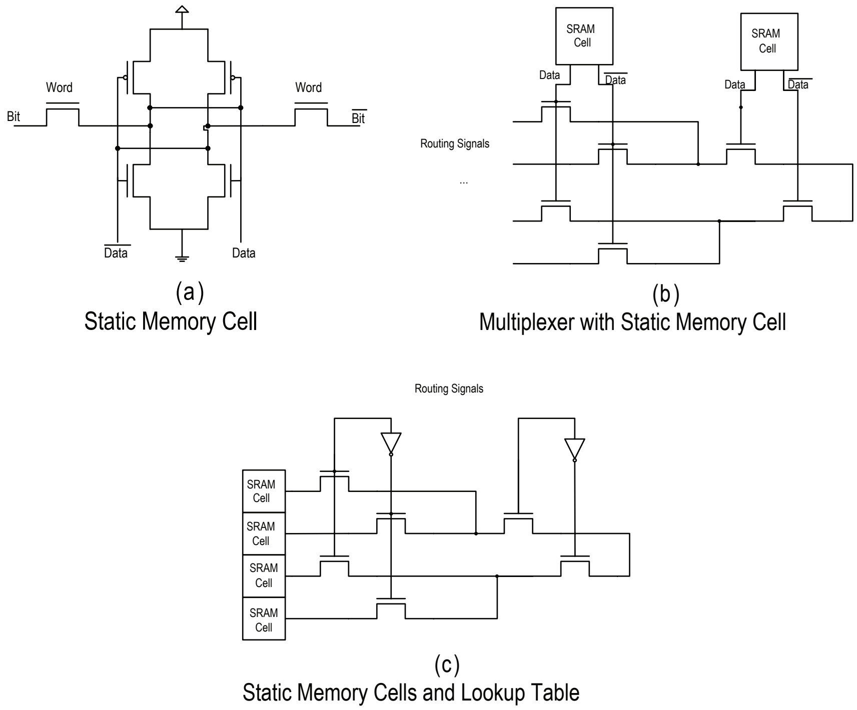
3

Programming Technologies

Every FPGA relies on an underlying programming technology that is used to control the programmable switches that give FPGAs their programmability. There are a number of programming technologies and their differences have a significant effect on programmable logic architecture. The approaches that have been used historically include EPROM [81], EEPROM [68, 174], flash [92], static memory [49], and anti-fuses [38, 93]. Of these approaches, only the flash, static memory and anti-fuse approaches are widely used in modern FPGAs. This survey focuses primarily on static memory-based FPGAs but, in this section, all these modern programming technologies will be reviewed to provide a more complete understanding of the advantages and disadvantages of static memory-based programming.

3.1 Static Memory Programming Technology

Static memory cells are the basis for SRAM programming technology which is widely used and can be found in devices from Xilinx [221, 224, 225, 227, 228, 229, 231], Lattice [124, 127], and Altera [18, 21, 22, 23, 24, 25]. In these devices, static memory cells, such as the



(a)
Static Memory Cell

(b)
Multiplexer with Static Memory Cell

(c)
Static Memory Cells and Lookup Table

Fig. 3.1 Use of static memory cells.

one shown in Figure 3.1(a), are distributed throughout the FPGA to provide configurability. There are two primary uses for the SRAM cells. Most are used to set the select lines to multiplexers that steer interconnect signals. The majority of the remaining SRAM cells are used to store the data in the lookup-tables (LUTs) that are typically used in SRAM-based FPGAs to implement logic functions. Figures 3.1(b) and 3.1(c) illustrate these two different approaches. Historically, SRAM cells were used to control the tri-state buffers and simple pass transistors that were also used for programmable interconnect but, as will be discussed in Section 5, such interconnect structures are no longer commonly used.

SRAM programming technology has become the dominant approach for FPGAs because of its two primary advantages: re-programmability and the use of standard CMOS process technology. From a practical standpoint, an SRAM cell can be programmed an indefinite number of times. Dedicated circuitry on the FPGA initializes all the SRAM bits on power up and configures the bits with a user-supplied configuration. Unlike other programming technologies, the use of SRAM cells requires no special integrated circuit processing steps beyond standard CMOS. As a result, SRAM-based FPGAs can use the latest CMOS technology available and, therefore, benefit from the increased integration, the higher speeds and the lower dynamic power consumption of new processes with smaller minimum geometries.

There are however a number of drawbacks to SRAM-based programming technologies:

(1) *Size.* The SRAM cell requires either 5 or 6 transistors and the programmable element used to interconnect signals requires at least a single transistor.

(2) *Volatility.* The volatility of the SRAM cell necessitates the use of external devices to permanently store the configuration data when the device is powered down. These external flash or EEPROM devices add to the cost of an SRAM-based FPGA. (We note that there have recently been a few devices that use on-chip SRAM as the main programmability mechanism, but that also include on-chip banks of flash memory to load this SRAM upon power-up [17, 234].)

(3) *Security.* Since the configuration information must be loaded into the device at power up, there is the possibility that the configuration information could be intercepted and stolen for use in a competing system. (We note that several modern FPGA families provide encryption techniques for configuration information that effectively eliminates this risk [23, 231].)

(4) *Electrical properties of pass transistors.* SRAM-based FPGAs typically rely on the use of pass transistors to implement multiplexers. However, they are far from ideal switches as they have significant on-resistances and present an appreciable capacitive load. As FPGAs migrate to smaller device geometries these issues may be exacerbated.

3.2 Flash/EEPROM Programming Technology

One alternative that addresses some of the shortcomings of SRAM-based technology is the use of floating gate programming technologies that inject charge onto a gate that "floats" above the transistor. This approach is used in flash or EEPROM memory cells. These cells are non-volatile; they do not lose information when the device is powered down.

Historically, EEPROM memory cells were not used to directly switch FPGA signals and, instead, the cells were commonly used to implement wired-AND functions in PLD-style devices [45]. Except for very low-capacity devices [11, 232], such approaches are no longer commonly used, in part because of the static power dissipation inherent in such schemes. With modern IC fabrication processes, it has become possible to use the floating gate cells directly as switches. Flash memory cells, in particular, are now used because of their improved area efficiency. The widespread use of flash memory cells for non-volatile memory chips ensures that flash manufacturing processes will benefit from steady decreases in process geometries. Figure 3.2 illustrates the flash-based approach used in Actel's ProASIC devices [3, 186]. The smaller programming transistor is used for programming the floating gate (injecting charge that remains even while the power is off) while

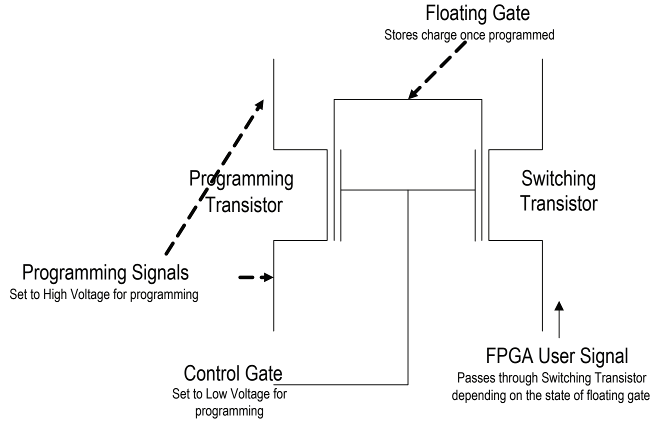

Floating Gate
Stores charge once programmed
Programming Transistor
Switching Transistor
Programming Signals
Set to High Voltage for programming
Control Gate
Set to Low Voltage for programming
FPGA User Signal
Passes through Switching Transistor depending on the state of floating gate

Fig. 3.2 Floating gate transistor.

the larger switching transistor serves as the programmable switch. It is also necessary to use the switching transistor when erasing the device [3].

This flash-based programming technology offers several unique advantages, most importantly non-volatility. This feature eliminates the need for the external resources required to store and load configuration data when SRAM-based programming technology is used. Additionally, a flash-based device can function immediately upon power-up instead of having to wait for the loading of configuration data. The flash approach is also more area efficient than SRAM-based technology which requires up to six transistors to implement the programmable storage. The programming circuitry, such as the high and low voltage buffers needed to program the cell, contributes an area overhead not present in SRAM-based devices. However, this cost is relatively modest as it is amortized across numerous programmable elements. In comparison to anti-fuses, an alternative non-volatile programming technology, flash-based FPGAs are reconfigurable and can be programmed without being removed from a printed circuit board. The use of a floating-gate to control the switching transistor adds design complexity because care must be taken to ensure the source–drain voltage remains sufficiently low to prevent charge injection into the floating gate [142]. Since newer processes require lower voltage levels, this issue may become less of a concern in the future.

One disadvantage of flash-based devices is that they cannot be reprogrammed an infinite number of times. Charge buildup in the oxide eventually prevents a flash-based device from being properly erased and programmed [156]. Current devices such as the Actel ProASIC3 are only rated for 500 programming cycles [3]. For most uses of FPGAs, this programming count is more than sufficient. In many cases FPGAs are programmed for only one use. Another significant disadvantage of flash devices is the need for a non-standard CMOS process. Also, like the static memory-based technology, this programming technology suffers from relatively high resistance and capacitance due to the use of transistor-based switches.

One trend that has recently emerged is the use of flash storage in combination with SRAM programming technology [17, 125, 126, 134, 234]. In these devices from Altera, Xilinx and Lattice, on-chip flash memory is used to provide non-volatile storage while SRAM cells are still used to control the programmable elements in the design. This addresses the problems associated with the volatility of pure-SRAM approaches, such as the cost of additional storage devices or the possibility of configuration data interception, while maintaining the infinite reconfigurability of SRAM-based devices. It is important to recognize that, since the programming technology is still based on SRAM cells, the devices are no different than pure-SRAM based devices from an FPGA architecture standpoint. However, the incorporation of flash memory generally means that the processing technology will not be as advanced as pure-SRAM devices. Additionally, the devices incur more area overhead than pure-SRAM devices since both flash and SRAM bits are required for every programmable element.

3.3 Anti-fuse Programming Technology

An alternative to SRAM and floating gate-based technologies is anti-fuse programming technology. This technology is based on structures which exhibit very high-resistance under normal circumstances but can be programmably "blown" (in reality, connected) to create a low resistance link. Unlike SRAM or floating gate programming technologies, this link is permanent. The programmable element, an anti-fuse, is directly used for transmitting FPGA signals. Two approaches have been used to implement anti-fuses. Dielectric anti-fuses are composed of an oxide–nitride-oxide dielectric positioned between N+ diffusion and polysilicon [93]. The application of a high voltage causes the dielectric to break down and form a conductive link with a resistance of typically between 100 and 600 ohms [55, 90]. This dielectric approach has been largely replaced by metal-to-metal-based anti-fuses. These anti-fuses are formed by sandwiching an insulating material such as amorphous silicon [88] or silicon oxide [61, 242] between two metal layers. Again, a high voltage breaks down the anti-fuse and causes the fuse to conduct. The advantage of this metal-to-metal anti-fuse is that the on resistance can be between 20 and 100 ohms [181] and the fuse itself requires no silicon area. This metal-to-metal approach is used in recent FPGAs from Actel [2, 5] and QuickLogic [161, 162].

The primary advantage of anti-fuse programming technology is its low area. With metal-to-metal anti-fuses, no silicon area is required to make connections, decreasing the area overhead of programmability. However, this decrease is slightly offset by the need for large programming transistors that supply the large currents needed to program the anti-fuse. While this area can be amortized across a number of fuses with clever programming architecture, it contributes significantly to the total area. Anti-fuses have an additional advantage; they have lower on resistances and parasitic capacitances than other programming technologies. The low area, resistance, and capacitance of the fuses means it is possible to include more switches per device than is practical in other technologies. Non-volatility also means that the device works instantly once programmed. This lowers system costs since additional memory for storing the programming information is not required and it also allows the FPGA to be used in situations that require operation immediately upon power up. Finally, since programming, and hence transmitting the bitstream to the FPGA, need only be done once, this can be done in a secure environment which improves the security of the design on the FPGA. To further enable this security, current devices offer security modes which disable accesses through the programming interface once the device is programmed [2].

There are also significant disadvantages to this programming technology. In particular, since anti-fuse-based FPGAs require a non-standard CMOS process, they are typically well behind in the manufacturing processes that they can adopt compared to SRAM-based FPGAs. Furthermore, the fundamental mechanism of programming, which involves significant changes to the properties of the materials in the fuse, leads to scaling challenges when new IC fabrication processes are considered. Indeed, there is some evidence that anti-fuses are no longer scaling as the most advanced devices use $0.15\,\mu$m technology [2] which is many generations behind the technology used for new standard CMOS devices.

The inability to reprogram anti-fuses also makes them unsuitable for applications where configuration changes are required. Unlike alternative technologies, in-system programming is not possible with these devices. Instead, special programmers must be used to program a device before it is mounted on a final product. Finally, the one-time programmability of anti-fuses makes it impossible for manufacturing tests to detect all possible faults. Some faults will only be uncovered after programming and, therefore, the yield after programming will be less than the 100% yield of SRAM or floating-gate devices. Some current devices are only expected to be programmed successfully with 90% confidence [2].

3.4 Summary

While the three programming technologies reviewed in this section are used in modern devices, SRAM-based programming technology has become the most widely used. Table 3.1 summarizes the differences between the different programming technologies. An ideal technology would be non-volatile and reprogrammable using a standard CMOS process and offer low on resistances and low parasitic capacitances. Clearly, none of the technologies satisfies all these requirements. One of the primary reasons SRAM technology has dominated is its use of standard CMOS manufacturing processes. For this same reason, the dominance of SRAM-based programming technology can be expected to continue for the foreseeable future of CMOS technology.

Table 3.1 Programming technology properties summary.

	SRAM	Flash	Anti-fuse
Volatile?	Yes	No	No
Reprogrammable?	Yes	Yes	No
Area (storage element size)	High (6 transistors)	Moderate (1 transistor)	Low (0 transistors)
Manufacturing process?	Standard CMOS	Flash Process	Anti-fuse needs special development
In-system programmable?	Yes	Yes	No
Switch resistance	∼500–1000 Ω	∼500–1000 Ω	20–100 Ω
Switch capacitance	∼1–2 fF	∼1–2 fF	<1 fF
Programming yield	100%	100%	> 90%

4

Logic Block Architecture

FPGAs consist of programmable logic blocks which implement logic functions, programmable routing to interconnect these functions and I/O blocks to make off-chip connections. This section will review trends in logic block architecture while programmable routing and I/O architecture will be considered in Sections 5 and 6.

Although many of the fundamental challenges and issues in FPGAs involve programmable routing circuit design and architecture, the logic block architecture of an FPGA is also extremely important because it has a dramatic effect on how much programmable routing is required. In this section, we discuss the basic trade-offs in logic block architecture design and the nomenclature needed to describe it. Our analysis includes a discussion of heterogeneous mixtures of different logic blocks. To conclude the section, we survey a number of commercial architectures and describe how these architectures fit into previously defined models.

4.1 FPGA Logic Block Fundamentals and Trade-Offs

The purpose of a logic block in an FPGA is to provide the basic computation and storage elements used in digital logic systems. As used in the original gate arrays, the most simple and un-specific way of providing this capability is to use a transistor as the basic logic element, and build gates and storage elements from it. This approach was indeed attempted in a commercial FPGA from the now-defunct company Crosspoint [144]. This kind of very fine-grained logic block, however, requires the use of large amounts of programmable interconnect to create any typical logic function. It will result in an FPGA that is bound to suffer from area-inefficiency (because programmable routing is expensive in terms of area), low performance (each routing "hop" is slow), and high power consumption (because of the higher capacitance of programmable interconnect that must be charged and discharged).

At the other extreme, a logic block could be an entire processor. This approach exists in the commercial space, although processors are mixed with some more fine grained logic blocks in a device [13, 188, 189, 195, 196, 228, 231, 235]. Such a logic block on its own would not have the performance gains that come from customizable hardware. In addition, if such a block was used to implement a 2-input AND gate, it would be incredibly inefficient, which illustrates the danger of using logic blocks that are too coarse-grained.

In between these extremes is a spectrum of logic block choices ranging from fine to coarse-grain logic blocks. FPGA architects over the last two decades have selected basic logic blocks made of transistors (noted above) [144], NAND gates [160], an interconnection of multiplexers [79], lookup tables [49], and PAL-style wide-input gates [217]. These choices were originally driven by intuitive insights on the part of architects, typically with very little data or analysis, with a few exceptions [79]. In this survey, we discuss how the research foundations for choosing a logic block were established, and then focus on the effect of logic block functionality on the three key metrics: area, speed, and power. In this discussion, we use silicon area as the proxy for cost, as is common. For a more detailed survey on the specifics of the logic blocks mentioned above, see [167] or [43].

In addition to a basic logic block, many modern FPGAs contain a heterogeneous mixture of different blocks, some of which can only be used for very specific functions, such as dedicated memory blocks or multipliers. These structures are very efficient at implementing specific functions, yet go to waste if unused. A central issue in FPGA architecture design is the selection of specific, hard circuits for inclusion in an FPGA. Section 4.8 will discuss these hard, heterogeneous circuit blocks in more detail.

4.2 Methodology

In general, we are interested in knowing the effect of an FPGA's architecture on the area-efficiency, speed, and power of a set of application circuits implemented in the FPGA. The set of applications represent the "target market" of the FPGA. Ideally, this set would include all digital hardware applications if just a single FPGA architecture could serve all possible markets, maximizing its advantage as a single standard device. Modern commercial practise requires the use of several architectural families to serve different market segments.

It is a common practise in FPGA architecture research to employ an empirical approach to study and explore different architectures. Here, application circuits are synthesized into different architectures through a CAD flow that is able to vary the architectural elements under study, as illustrated in Figure 4.1. This approach is similar to the one employed in computer architecture research [98], in which a number of software applications are compiled into different processor

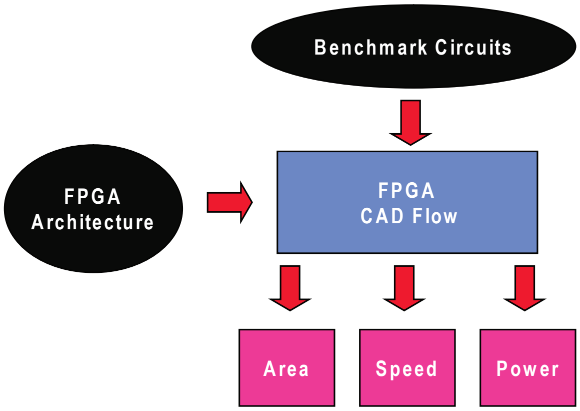


Fig. 4.1 Architecture exploration empirical CAD flow.

architectures to determine the effect of processor architecture on performance. The only alternative to this kind of empirical approach is the use of a theoretical approach in which applications are modeled in a statistical/graph-theoretical way, and the mapping of an application onto an architecture is modeled through a probabilistic calculation or through other theoretical machinery. While there have been a number of attempts to do this [44, 52, 78, 79, 91, 159, 244] these studies have necessarily focused on a fairly narrow spectrum of architectures and tools. They have typically been used to study routing architecture, rather than logic block architecture. As an aside, the use of theory in architecture research remains an open and possibly important question, if some fundamental and usable theory can be created.

There are four aspects of an experimental flow used to study architecture that need to be described when presenting experimental results:

(1) *The depth of the CAD flow.* This depth represents how far through synthesis, packing, placement, and routing the circuits are processed. (A survey of all these standard steps in the FPGA CAD flow has been covered in a previous issue [54].) The deeper the CAD flow, the more precise and believable the results, but deeper flows require a great deal more development effort and computation time.

(2) *The quality of the CAD tools used.* Low-quality tools can give misleading architectural results. It is thus important to both use the best tools available in CAD flows and to realize that a breakthrough in CAD tool development could dramatically impact conclusions from experimental studies and possibly the body of FPGA architectural knowledge. Yan et al. studied this issue [237] and described several situations where specific tools could create misleading architectural conclusions.

(3) *The set of benchmark circuits used.* The quality of results depends on how representative the benchmark circuits are with respect to typical circuits created by target users of a device.

(4) *The quality of the models and tools used to determine area, speed, and power.* For example, full timing analysis with

proper RC delay models can be used to evaluate circuit speed, or maximum logic block depth can be used as a proxy. The latter, of course, is much less accurate than the former, and does not account for significant routing effects.

We now explore some specific trade-offs in FPGA logic block architecture.

4.3 Logic Block Trade-Offs with Area

FPGA area-efficiency is a key metric because the size of the FPGA die dictates a significant portion of its cost, particularly for devices with a large logic capacity. (For smaller devices, I/O and packaging also become significant in the cost of the devices; we discuss I/O in Section 6.)

To understand area-efficiency trade-offs we must have a clear definition of it. Area-efficiency can only be defined given a set of target benchmark circuits, B_i. If we are interested in comparing a set of FPGA architectures, F_j, then each benchmark circuit B_i must be synthesized into the architecture F_j and the resulting area required is measured and labeled as A_j^i. Here it is important to note that in most such architectural exploration, the size of the FPGA array of logic blocks (and typically the amount of routing supplied in that array) is allowed to vary to meet the need of a particular circuit in a particular architecture — that is, the size of the array and routing is varied until the minimum needed is determined. This notion has been, at times, controversial because most engineers think of an FPGA as a device which is fixed in size and that does not vary in response to the needs of the application circuit except in a very coarse way across the different sizes in a commercial device family. FPGA architects have permitted this more fine variation as a necessary way to determine the quality of an architecture with more precision than a few fixed devices would allow.

Once the area A_j^i of every circuit implemented in a specific architecture is known, the areas are typically normalized to a base case architecture (i.e., $j = 1$) and averaged across all B_i benchmarks, often using the geometric average. Thus, for each architecture, a single average area number represents its efficiency with respect to the base architecture.

For a homogeneous FPGA array (one that employs just one type of logic block) the fundamental area trade-offs of an architecture are as follows:

- As the functionality of the logic block increases, fewer logic blocks are needed to implement a given design. Up to a point, fewer logic blocks reduce the total area required by a design.
- As the functionality of the logic block increases, its size (and the amount of routing it needs per block) increases.

If we label the number of logic blocks required to implement circuit B_i on architecture F_j as N^i_j and the sum of the logic block area and surrounding routing area required to implement each of those blocks as LR^i_j, then the area of a given benchmark circuit on an architecture is

$$A^i_j = N^i_j \times LR^i_j. \tag{4.1}$$

This relationship implies that an FPGA architect should increase logic block functionality up to the point that the gain in area efficiency due to a reduced number of blocks is offset by the growth in the size of the logic and routing area of an individual block.

These fundamentals were first articulated in [168, 169] using a fairly limited depth of CAD flow (only global routing) and simple area models. More recently, [7, 8, 9] employed the deep CAD flow and high-quality area models of [37] to re-explore area trade-offs with much higher quality synthesis, placement, and routing tools.

The more recent area models are worthy of note. To determine the area of the logic and routing, all structures are designed at the transistor level (both the logic and the programmable routing fabric) and each transistor is sized with a proper circuit design approach, typically with the goal of minimizing the area-delay product of basic circuit structures. The total area is measured in terms of minimum-width transistor areas (MWTAs) wherein the relative silicon area of each transistor is normalized to the area of a minimum sized transistor in the process. This normalization allows for process-independent comparisons. With this area model, each distinct transistor incurs a fixed area cost and increasing the width of the transistor amortizes the fixed cost such that a transistor with twice the minimum width requires less area than two independent minimum width transistors. Further details of this area model can be found in [37].

The work of [7, 8, 9] first explored the effect of lookup table (LUT) size on area and speed performance. Figure 4.2 illustrates the basic trade-off discussed above, for area. The X-axis of Figure 4.2 is the size of the lookup table (or K, the number of inputs to the lookup table). For this architecture, a “cluster” size of 1 was used, which means that each logic block contained exactly one LUT and flip–flop. The left-hand Y-axis (and dashed line) of Figure 4.2 gives the area of the logic block and its surrounding routing while the right-hand Y-axis (and solid line) of Figure 4.2 gives the geometric average of the number of K-input LUT/flip–flop blocks needed to implement the 28 circuits used in the experiment.

This experiment illustrates the fundamental trade-offs described above — as the LUT size (K) increases, the number of LUTs required to implement the circuits significantly decreases. However, the area cost

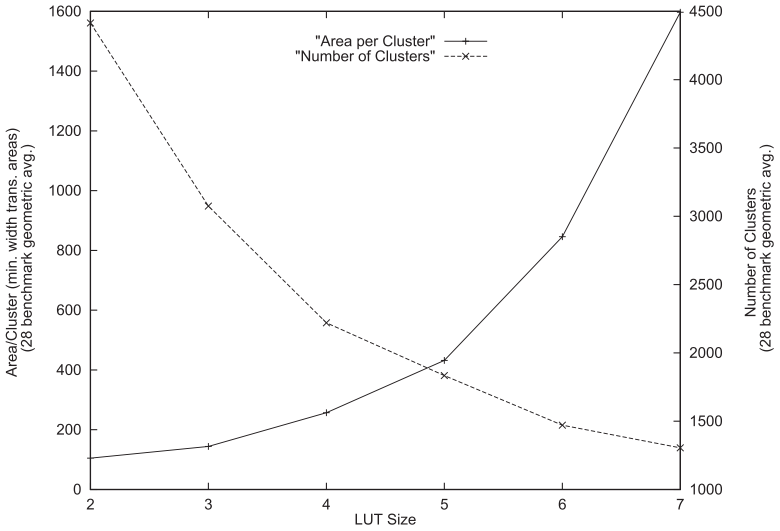


Fig. 4.2 Number of logic blocks and area/block vs. logic block functionality [7].

of implementing the logic and routing for each block increases significantly with K due to the following reasons:

(1) The number of programming bits in a K-input lookup table is 2^K, indicating an exponential area increase with K, and
(2) The number of routing tracks surrounding the logic required for successful routing increases as the number of pins connecting into the logic block increases (as determined by K). It is fundamentally true that increasing the number of pins connected to a logic block requires an increase in the number of tracks per channel, as first modeled by El Gamal [78].

When the two curves of Figure 4.2 are multiplied to obtain the total area, the curve of Figure 4.3 is obtained. This curve shows that, at first, a reduction in block count reduces total area, but then an increase in block size leads to an area increase as the LUT size increases. This curve is typical of any area versus granularity experiment in FPGA architecture.

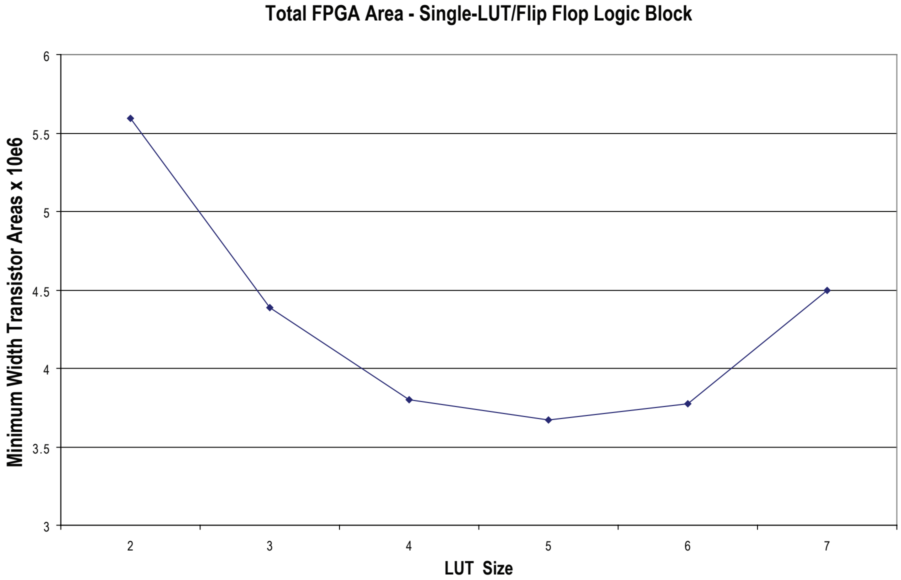


Fig. 4.3 Total area of FPGA vs. LUT size [7].

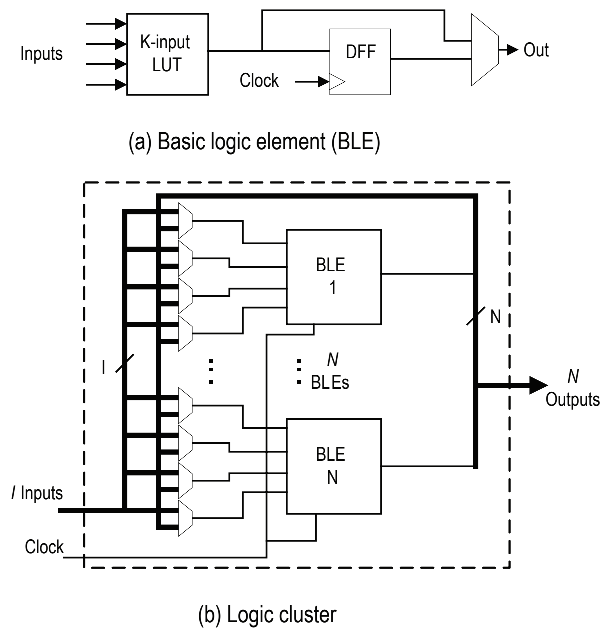


Fig. 4.4 FPGA basic logic element and cluster [37].

An alternate way to change the granularity of an FPGA logic block is to use clusters of LUTs and flip–flops. This hierarchical approach is now commonly used in most industrial FPGAs. Here, several basic logic elements are grouped together and programmably connected by a local interconnect structure, as illustrated in Figure 4.4 from [37], which uses a LUT/flip–flop basic logic element grouped into a cluster of size N. By increasing logic block granularity in this manner (as opposed to growing it by making the LUT size larger) the size of the logic and internal routing to supply the complete crossbar connectivity within the cluster only grows quadratically (vs. exponential growth for LUT size increases).

In general, there are fewer inputs to the cluster from the external inter-cluster routing than the total number of inputs to the basic logic elements inside the cluster. This reduction is possible because cluster input signals are often used as inputs to multiple logic elements in clusters with a sufficient number of logic elements. This observation was first captured in an equation for clusters of N 4-input lookup tables by Betz and Rose [34]: I, the number of pins needed to fully occupy a cluster of N 4-input lookup tables is $2N + 2$, as opposed to the total number of input pins on all basic logic elements, $4N$. Ahmed et al. [7, 8, 9] later generalized this relationship for N-sized clusters of K-input lookup tables:

$$I = \frac{K}{2}(N + 1), \tag{4.2}$$

which is significantly less than the maximum of KN.

Typically, intra-cluster routing in contemporary LUT-based FPGAs does not exhibit full crossbar connectivity. One study [129] determined that at least half of the connections between cluster inputs and logic element inputs can be removed and between 50% and 75% of the feedback connections from logic element outputs to logic element inputs can be removed with no impact on delay or the number of logic clusters required. This switch depopulation results in about a 10% area reduction for FPGAs with cluster sizes similar to commercial offerings. The reduction in intra-cluster routing flexibility requires an FPGA router to extend the search for wiring paths into logic clusters. This extended search can increase routing time by up to a factor of four [129].

4.4 Speed Trade-Offs

For a homogeneous FPGA array that employs just one type of logic block, fundamental architectural effects on speed include:

- As the functionality of the logic block increases, fewer logic blocks are used on the critical path of a given circuit, resulting in the need for fewer logic levels and higher overall speed performance. A reduction in logic levels reduces the required amount of inter-logic block routing, which contributes a substantial portion of the overall delay.
- As the functionality of the logic block increases, its internal delay increases, possibly to the point where the delay increase offsets the gain due to the reduced logic levels and reduced inter-logic block routing.

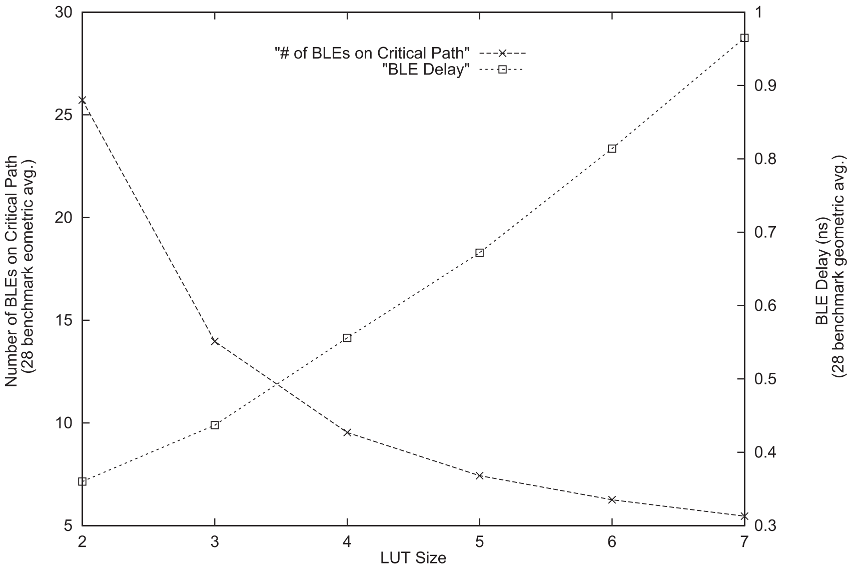


Fig. 4.5 Number of LUTs on critical path & LUT delay vs. LUT size [7].

Ahmed [7] explored this trade-off in a series of experiments, as illustrated in Figure 4.5. The figure gives the average number of levels of LUTs (called BLE for "basic logic element" in the figure) on the critical path across a suite of 28 circuits, for different LUT sizes. The figure also shows the intrinsic delay of circuits mapped to different LUT sizes. For these experiments the full CAD flow including synthesis, technology mapping, packing, placement, and routing was performed. Area and delay information for the placement and routing tools was determined through transistor-level sizing of the FPGAs resources [7].

Total FPGA delay as a function of LUT size includes the routing delay for each level of logic. This total critical path delay is plotted versus LUT size in Figure 4.6 for cluster sizes ranging from 1 to 10. The figure shows that as both LUT and cluster size increase, the critical path delay monotonically decreases with diminishing returns. There are significant returns to increasing LUT size up to six and cluster size up to three or four. Recent trends in commercial architectures have indeed moved toward larger LUT sizes to capture these gains [135, 231].

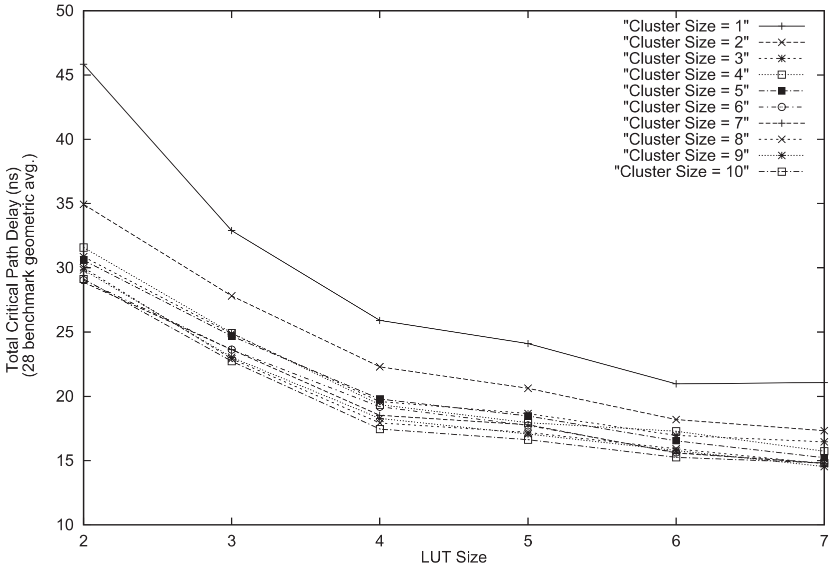


Fig. 4.6 Total critical path delay as a function of LUT and cluster size [7].

4.5 Logic Block Power Trade-Offs

Power consumption in FPGAs, as with all integrated circuits, is generally divided into two categories: dynamic power and static power. Dynamic power is the power consumed by the transitioning of signals on the device. Even in the absence of signal transitions, power continues to be consumed and that power consumption is known as static or leakage power. For FPGAs, dynamic power consumption trade-offs exist that are similar to those described above for area and speed. Results from [138] suggest that the best logic block architectures for area are also the best logic block architectures for power consumption. This assessment is likely a result of the basic nature of power consumption, capacitance tracks with area, so a reduction in area results in a related reduction in power consumption. Li et al. [138] concludes that the best LUT and cluster sizes in terms of area-efficiency from [7] are also the best sizes for minimized dynamic power consumption.

More recently, Cheng et al. [56] showed how to optimize a logic block architecture in concert with dynamic and static power reduction techniques. For a fixed, standard 4-LUT architecture, it was shown that sleep transistors and threshold voltage settings can be used to achieve significant power consumption reductions.

4.6 PLA/PAL-Style Types of Logic Blocks

Although lookup-table-based logic blocks dominate commercial FPGA architectures, AND/OR-based programmable logic blocks have also received consideration. A variable AND array, fixed OR-plane PAL style of logic is the most common AND/OR configuration. Cong et al. [65] showed that a fairly small PAL-like structure, with 7–10 inputs and 10–13 product terms obtains performance gains of up to 33% for a 27% increase in area. With a different routing architecture, it was possible to improve performance by up to 27% while also reducing area by 17%. Earlier, Kouloheris and El Gamal [116, 117, 118] showed that K-input multiple-output PAL-style logic blocks were more area-efficient than 4-input LUTs but PAL-based implementations typically consumed excessive static power.

4.7 Heterogeneous Mixtures of Soft Logic Blocks

Our previous discussion on basic area, speed, and power trade-offs versus logic block and cluster size assumed a homogeneous array of logic blocks. The trade-offs, with larger LUT and cluster sizes being advantageous for speed but smaller sizes being best for area, however, suggest that a mixture of two kinds of logic blocks may be better than a homogeneous approach. A choice of logic resources allows a CAD tool to appropriately select which of the resources to use.

This idea was first explored in [97]. A mixture of two different sizes of lookup tables was investigated to determine if the resulting FPGA could be more area efficient than a homogeneous offering. The experimental flow used to evaluate the architecture only went as deep as logic synthesis, as the authors developed a new heterogeneous technology mapping tool that could synthesize logic circuits into fixed ratios of two different sizes of lookup tables. To determine area, they counted the total number of bits in the lookup tables, and the number of pins connected via routes. They concluded that, compared to a homogeneous 4-LUT logic architecture, a small (10%) reduction in pin count could be obtained with a mixture of 5-input and 2-input lookup tables, or 4-input and 2-input lookup tables. In addition, they showed a reduction in pins for a 6-input/4-input LUT combination, but a fairly significant increase in the total number of LUT bits. They suggested, without showing experimental evidence, that this combination would have good speed–area benefits due to the presence of the faster, 6-input LUT.

Similarly, in [66], it was found that a heterogeneous mixture of 3-LUTs, 4-LUTs, 5-LUTs, and 6-LUTs offered a performance improvement of at least 25% over the homogeneous FPGA consisting of only one type of those LUTs. Compared to the heterogeneous FPGA, the homogeneous 3-LUT FPGA was 34% smaller while the homogeneous 6-LUT FPGA was 137% larger. Again, this study only considered the CAD flow up to technology mapping.

Kaviani and Brown [112, 113] proposed employing a mixture of PAL-like logic blocks and LUT-based logic blocks in a single architecture. The PAL blocks provide improved circuit speed and the LUT blocks provide area efficiency. Their experiments, also performed at the depth of technology mapping, suggested that there were significant area gains possible with these heterogeneous mixtures.

4.8 Heterogeneity

Up to this point, we have discussed the architectural trade-offs of a specific logic block choice or mixtures of different general purpose logic blocks. By contrast, any function that is implemented directly in *specific-purpose logic* on the FPGA will typically have superior area, speed, and power consumption over its implementation in general-purpose logic blocks. However, if the function is not used in a target application, the silicon area devoted to the function will be wasted, causing a net reduction in area efficiency. These observations imply a rich field of architectural trade-offs for FPGAs with a *heterogeneous* mixture of general-purpose logic blocks and specific-purpose logic blocks. The key questions in this domain are:

(1) Which kinds of specific logic functions should be implemented?
(2) What should be the ratio of specific functions to general purpose functions?
(3) What can be done about specific-purpose logic blocks that are not used in a specific application?

To address these questions we provide some nomenclature. The *soft logic fabric* of an FPGA is the array of combinational logic elements, each consisting of a logic function implemented as a gate or LUT, that is connected through a programmable routing fabric. Any other circuitry employed in the device is a *hard circuit structure* which we define as a structure that allows the implementation of a logic function that could *also be implemented in the soft logic fabric*. By this definition, a dedicated flip–flop inside a logic block is considered a hard circuit structure, since it is possible to build flip–flops from an interconnection of programmable LUTs or gates. Dedicated flip–flops are now universally used in commercial FPGAs, although some early FPGAs were built without them [1, 144]. In addition, modern commercial FPGAs contain dedicated logic within each general purpose block to support arithmetic carry and sum functions [18, 21, 22, 23, 24, 25, 221, 224, 225, 227, 228, 229, 231] and some memory functions [23, 231].

It is appropriate to distinguish between two kinds of heterogeneity: one kind is exemplified by the flip–flop and dedicated carry logic which appears alongside the combinational logic in every logic block that makes up the soft logic fabric. This type of heterogeneity represents *soft fabric heterogeneity*. An FPGA with only soft fabric heterogeneity can be constructed from an array of identical tiles, each containing the basic soft logic block and soft fabric heterogeneous elements.

A second type of FPGA heterogeneity involves independent blocks of logic that are not paired with soft combinational logic elements. In this case, distinct tiles containing dedicated hard circuit structures are added to the array of tiles. For example, multi-bit block RAMs that appear in modern FPGAs are common hard circuit structures. These RAMs range in size from hundreds to tens or hundreds of thousands of bits and are commonly found in contemporary devices such as the Altera Flex 10K, Flex 20K, Stratix, Stratix II, and Stratix III families [14, 15, 18, 22, 23], and the Xilinx Virtex, Virtex II, II Pro, Virtex 4 and 5 and Spartan II, and III families [221, 224, 227, 228, 229, 231, 235]. Block RAMs typically are aligned in vertical columns within the basic tile array, as shown in Figure 4.7. Other common hard circuit structures found in contemporary commercial FPGAs include the multiply-accumulate (MAC) blocks that appear in the Stratix I, Stratix II, and Stratix III FPGAs and the multiplier blocks that appear in Xilinx Virtex II, Virtex 4, and Virtex 5 FPGAs. Heterogeneity which includes hard circuit structures can be referred to as *tile-based heterogeneity* to reflect the inclusion of diverse tiles on the same FPGA substrate. Figure 4.7 illustrates an FPGA with a mixture of different blocks with tile-based heterogeneity.

We now discuss various types of hard structures that appear in commercial FPGAs or that have been proposed by FPGA researchers.

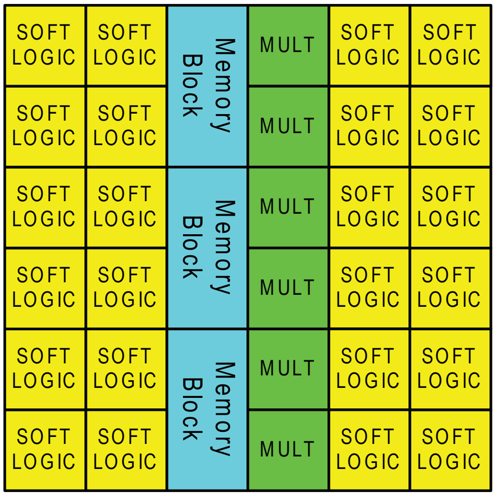


Fig. 4.7 Illustration of tile based heterogeneity.

4.8.1 Soft Fabric Heterogeneity

As mentioned earlier in this section, since their introduction nearly all commercial FPGAs have included flip–flops in their basic logic elements. Early research [168, 169] investigated the area efficiency of FPGAs with and without dedicated flip–flops, and clearly established the significant benefits of including flip–flop circuits within logic elements. Modern FPGA flip–flops are typically edge-triggered and include a variety of set, reset, load, enable and clocking capabilities. As noted earlier, modern FPGAs typically group basic logic elements into clusters. Individual FPGA families may be distinguished by the intra-cluster connectivity of reset, enable, load, and clock signals applied to flip–flops. Some architectures force these signals to be the same cluster wide, while others allow the signals to be programmably selected [23, 231].

Many modern FPGAs include explicit circuitry for addition/subtraction/carry logic to make adders and subtraction units smaller and faster [23, 25, 227, 231]. Several modern FPGAs also include more advanced carry lookahead and carry-skip logic [23, 231]. Woo [218] and Hauck [96] explored several versions of carry logic formation and carry lookahead to achieve higher performance.

The Xilinx XC4000 series FPGAs [103], and all subsequent Xilinx FPGAs provide the ability to turn LUTs in the soft fabric into small memories. These memories can be connected together to form larger memories. This ability to convert LUTs to memory has also been added in Altera's recent FPGAs [23]. As well, in all Xilinx FPGAs since the original Virtex [224, 227, 228, 229, 231], the LUT can also be configured to act as a shift register.

4.8.2 Memory

The first type of heterogeneous tile used in FPGAs was block memory [14, 152] which first appeared commercially in the Altera Flex 10K series FPGA [14]. This memory block consisted of 2K bits of static RAM, which could be configured as either a 2048 × 1, 1024 × 2, 512 × 4 or 256 × 8 bit memory. This flexibility [152, 209] is a crucial aspect of memory blocks in FPGAs and tile-based heterogeneity in general. Since different applications will need memory configured in many different aspect ratios, basic memory blocks must be flexible and configurable. Furthermore, the software system should make it easy to combine memory blocks, with the addition of a small amount of soft logic, into large blocks of memory with an even wider range of aspect ratios. All contemporary FPGAs include memory blocks. and they have grown to cover a significant fraction of the FPGA die area. This trend is likely to continue as memory becomes more important in larger systems.

Most contemporary FPGAs employ memory blocks that have dual-port functionality. Some dual port memories allow simultaneous read and write operations, while others allow mixtures of read and write operations [23, 231]. Recent FPGAs have special features such as the ability to support FIFO configurations [231]. The cost of this added flexibility requires additional memory ports and internal memory complexity. Complex memory operation can be supported with control circuits implemented in soft logic. Often, it can be costly to combine memory blocks to perform large, complicated memory functions due to logic and routing overheads.

The demand for memory across an application set can vary quite widely. Some applications requires almost no memory while others require large amounts. A number of research projects have examined how to use memory blocks that are not required for memory storage. Several projects have explored converting unused memory blocks into large lookup tables which implement combinational functions [67, 210, 211, 212, 213].

4.8.3 Computation-Oriented Tiles

An early example of a computation-oriented tile is the multiplier integrated into the Xilinx Virtex II FPGA [229]. This tile consisted of an 18 × 18 2's complement multiplier that sat alongside a block memory tile. Since the introduction of the Virtex II, Xilinx and other manufacturers have introduced more sophisticated hard computational units that include multiplier-accumulators, and some multiplexer functions [228, 231]. The Altera Stratix series is notable for its ability to fracture larger multipliers into groups of smaller multipliers. For example, the Stratix I [22] contains a single 36 × 36 multiplier accumulator block that can be broken into eight 9 × 9 multipliers and an adder to sum results.

If multipliers are not needed by an application, the multiplier tiles located inside a target FPGA provide little benefit. One way to deal with this issue is to create multiple sub-families within a device family that use different ratios of soft logic to hard-logic. A device family typically consists of a set of FPGAs with the same basic architecture that contain differing amounts of resources. For example, devices in the Lattice EPC2 family are available in six sizes including devices with between 6,000 LUTs and 68,000 LUTs [124]. A designer can select the device with the most appropriate ratio, minimizing "wasted" computational tiles. The cost of this flexibility is incurred by the FPGA vendor, who must support a larger number of devices. This concept works against the fundamental economies of a single family serving many applications. The ratio-based approach to resource allocation was first introduced commercially by Xilinx for the Virtex 4 [228] family. This family has three sub-families, one with a focus on soft logic and memory, one with a focus on arithmetic computational units, and one with a processor and high-speed serial interface focus. Ratio-based subfamilies are also available for the Xilinx Virtex 5 family [231] and the Altera Stratix III family [23]. A similar approach was initially suggested by Betz and Rose [33].

4.8.4 Microprocessors

Microprocessors are vital components in many digital systems. Since they are often used in conjunction with FPGA logic, it makes sense to consider their integration into an FPGA logic fabric. An initial offering by Triscend Corporation interfaced an embedded processor to an FPGA-like fabric via a bus [195, 196] (Triscend was later acquired by Xilinx [226]). Later, Altera introduced Excalibur, an FPGA which included a hard ARM core connected to an Altera Apex 20 K series FPGA [13]. Nearly simultaneously, Xilinx introduced Virtex II Pro FPGAs which included one, two, or four IBM Power PC cores integrated with a Virtex II logic fabric [235]. Several Xilinx Virtex 4 and Virtex 5 subfamilies also support Power PC cores [228, 231]. A challenging aspect of including a hard processor on an FPGA is the development of the interfaces between the processor, memory system, and the soft fabric. The alternative to a hard processor is a soft processor, built out of the soft fabric and other hard logic. The latter is generally slower in performance and larger in terms of area. However, the soft processor can often be customized to exactly suit the needs of the application to gain back some of the lost performance and area-efficiency.

4.9 Commercial Logic Blocks

In general, published research on logic block architecture tends to model and explore relatively simple basic logic elements, such as the pure K-input lookup table or PLA style blocks. In contrast, commercial logic blocks have undergone an evolution that typically has led to the development of more complex blocks in an attempt to gain more functionality.

For example, one of the earliest FPGAs, the Xilinx XC3000 FPGA [222] employed a complex logic block, as illustrated in Figure 4.8. The basic block is a 5-input lookup table (using an additional multiplexer is not shown in the figure), but it is augmented to allow the creation of two 4-input functions (labeled F and G in the figure) that share most of the inputs. Unfortunately, the complexity of the logic block made it difficult for synthesis tools to find efficient logic mappings for designs. In addition, it was difficult to manually design functions that could map to this architecture.

As result, numerous subsequent FPGAs (the Xilinx XC4000 family [103], Virtex series up to and including the Virtex 4 family, and Altera Flex, Apex, Cyclone, and Stratix I families) used basic clusters of 4-input lookup tables to implement logic. The input signal fanout leveraged in the Xilinx XC3000 architecture is achieved by distributing cluster inputs to multiple LUT-based basic logic elements.

More recently, the basic logic elements in commercial FPGAs have returned to being more complex. The Altera Stratix II [106, 135]

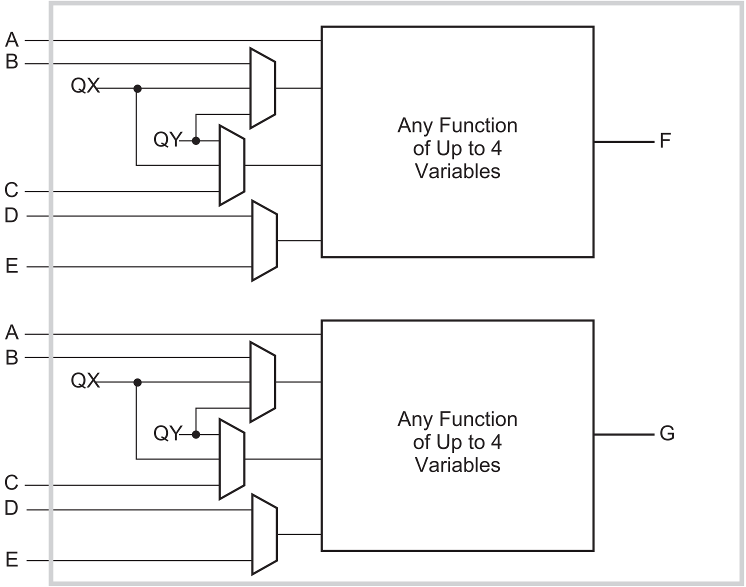

A
B
QX
QY
C
D
E
Any Function of Up to 4 Variables
F
A
B
QX
QY
C
D
E
Any Function of Up to 4 Variables
G

Fig. 4.8 One possible configuration of a Xilinx XC3000 logic block.

architecture employs a fracturable 6-input lookup table, called the Adaptive Logic Module (ALM), as illustrated in Figure 4.9.

The ALM is an 8-input structure that can implement many combinations of logic functions, including:

- One 6-input logic function
- Two 4-input logic functions
- One 5-input and one 3-input function
- Two 6-input functions that share the same logic function and 4 inputs

The Virtex 5 FPGA [231], employs a 6-input lookup table that can also implement two 5-input functions that share five inputs, or two 4-input functions that share fewer inputs. This level of logic block complexity is now supported by enhanced synthesis algorithms which can evaluate a range of possible combinational implementations. To ensure mapping efficiency, the ALM was developed in concert with

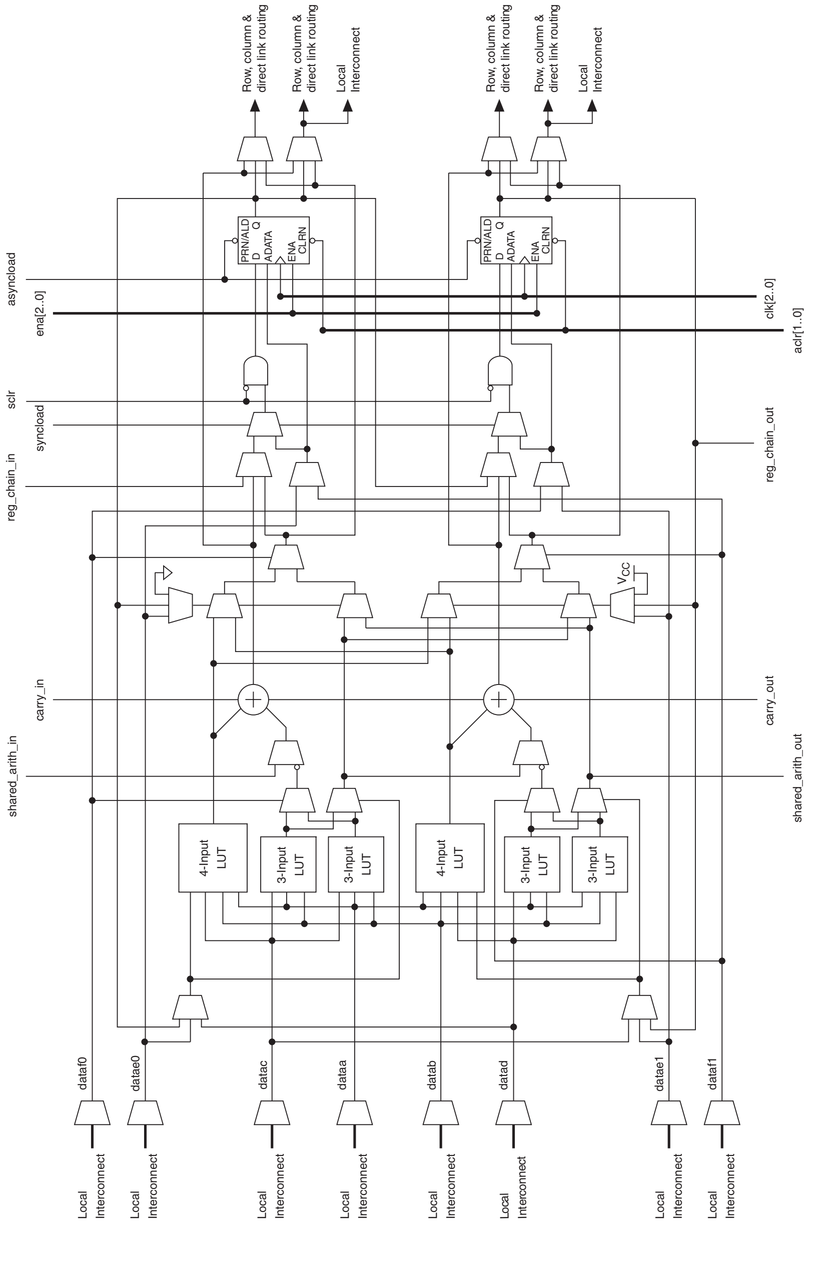


Fig. 4.9 The Altera Stratix II adaptive logic module [18].

a supporting synthesis tool [135]. Currently, designers almost never deal with the architecture of a logic element directly, but rather use vendor-supplied synthesis tools. In rare cases, when very high logic density is desired for a highly replicated function, manual mapping may be explored.

A notable anti-fuse-based logic block appeared in a series of devices from Actel. These blocks used a multiplexer as a basic logic element [79]. A configuration of three 2-to-1 multiplexers whose inputs were connected to constants or input signals, was used to provide a wide variety of logical functions. The most recent anti-fuse-based logic block from Actel employs a form of heterogeneity by using a mixture of combinational logic-only modules (C-modules, shown in Figure 4.10) and sequential modules that contain flip–flops.

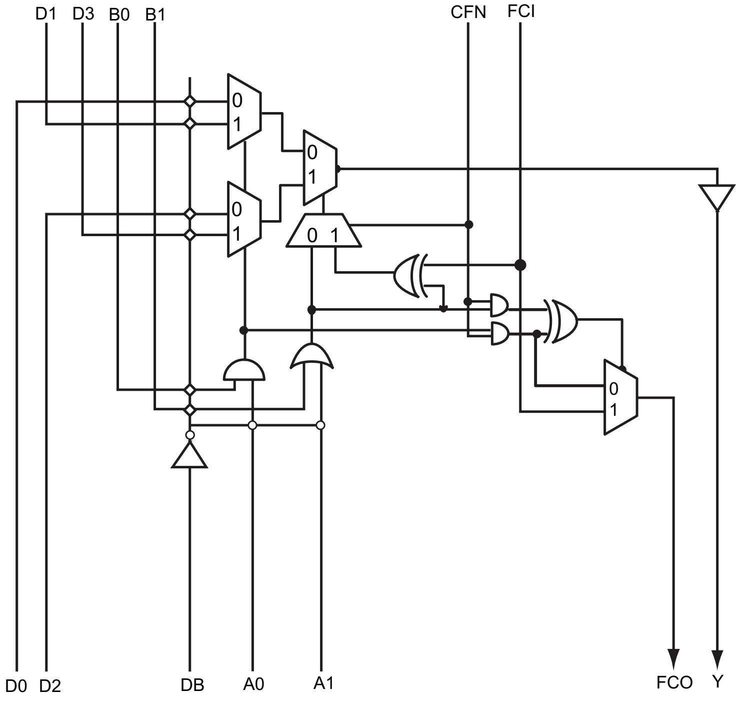


Fig. 4.10 Actel axcelerator combinational logic block [2].

4.10 Challenges in Basic Logic Block Architecture

A crucial goal in the evolution of FPGAs is the reduction of the area, performance, and power penalty of using the devices versus ASICs. One possible approach to reaching this goal is to integrate more hard blocks into FPGAs. Desirable blocks would be applicable to multiple application domains and offer significant benefits in circuit area, speed, and power consumption versus soft logic. Next generation FPGAs may well include more sophisticated hard blocks that perform computation. For example, hard floating point units [32] are a possibility. For practical purposes, these types of blocks must be flexible. If, for example, fixed integer arithmetic blocks could be integrated into a fixed floating point block, the result would be a block of broad utility that meets the basic general-applicability mandate of FPGAs.

A more radical way to provide flexibility would be to marry every hard block to its own soft logic that could be used in the event the hard logic is not needed. One potential approach is shown in Figure 4.11 from [109]. Instead of only containing hard logic, a soft logic block, labeled as a shadow cluster, is also available and an additional multiplexer allows one to select from either the hard or soft logic. This approach ensures that the routing into and out of the block can always be used even when the hard logic is not required. These kinds of approaches, that

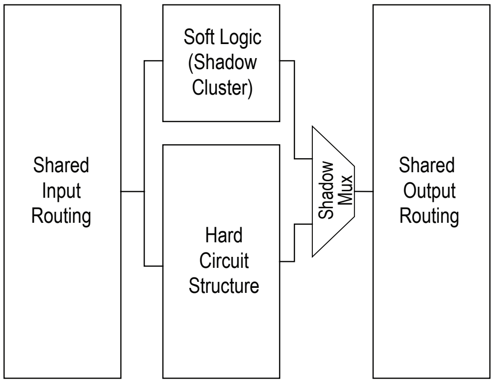


Fig. 4.11 Combined hard and soft logic block [109].

create flexibility in hard logic, are among the most important ways to reduce the large area gap between FPGAs and ASICs.

Finally, as fabrication technology progresses, it is clear that power consumption, both dynamic and static, has become a serious issue. It is important for architects to continue to think of high-level architectural methods to reduce power consumption.

5

Routing Architecture

The programmable routing in an FPGA provides connections among logic blocks and I/O blocks to complete a user-designed circuit. It consists of wires and programmable switches (configured by one of the programming technologies described in Section 3) that form the desired connections.

To accommodate a wide variety of circuits, the interconnect structure must be flexible enough to support widely varying local and distant routing demands together with the design goals of speed performance and power consumption.

Although the routing demand of logic circuits varies from design to design, certain common characteristics of these designs exert a strong influence on the architecture of FPGA routing. For example, most circuits exhibit locality, necessitating an abundance of short, fast, routing wires, while simultaneously requiring at least some intermediate and longer wires to support more distant connections.

Additionally, circuits also contain a number of signals such as clocks and resets that must be widely distributed across the FPGA. Modern FPGAs all contain dedicated interconnect networks that handle the distribution of these signals. Typically, these networks are carefully designed to be low skew for use in distributing clock signals. They generally can be directly connected to flip–flops and the networks can only be driven by a limited number of resources on the FPGA. There are many interesting architectural questions that must be considered when designing these networks and some of these details are discussed in [122]. The remainder of this discussion will focus exclusively on the FPGA's general purpose routing.

5.1 FPGA Routing Architecture Overview

A basic issue in FPGA design is the organization of the global routing architecture, which is the macroscopic allocation of wires with no focus on the more microscopic switching between wires. The global routing architecture defines the relative position of routing channels in relation to the positioning of logic blocks, how each channel connects to other channels, and the number of wires in each channel. The detailed routing architecture specifies the lengths of the wires, and the specific switching quantity and patterns between and among wires and logic block pins. In recent years, the issue of single-driver versus multiple-driver wires, which gives rise to wires that send signals in a specific direction, has also arisen as an important part of detailed routing architecture. We begin with an overview of the two main types of global routing architecture, and then move to a discussion of various aspects of detailed routing architecture.

FPGA global routing architectures can be characterized as either hierarchical [6] or island-style [37, 45].

5.1.1 Hierarchical Routing Architectures

Hierarchical routing architectures separate FPGA logic blocks into distinct groups [6, 197]. Connections between logic blocks within a group can be made using wire segments at the lowest level of the routing hierarchy. Connections between logic blocks in distant groups require the traversal of one or more levels (of the hierarchy) of routing segments. As shown in Figure 5.1, only one level of routing (Level 1) directly connects to the logic blocks. Programmable connections are represented with crosses and circles. Generally, the width of routing channels is widest at levels furthest from the logic blocks.

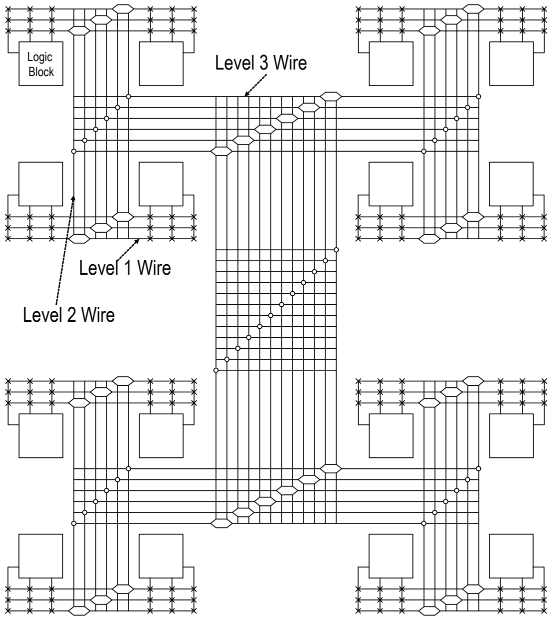


Fig. 5.1 Example of hierarchical FPGA [197].

This hierarchical global routing architecture has been used in a number of commercial FPGA families including Altera Flex10K [14], Apex [15], and Apex II [12] architectures. Although the hierarchical routing architecture offers somewhat more predictable inter-logic block delay following design placement and superior performance for some logic designs [6], design mapping can be an issue. If the distribution of design wire lengths does not match the hierarchical routing architecture distribution (or indeed the hierarchy of the design does not match the hierarchy of the FPGA's routing architecture), logic block use in each hierarchical group may need to be reduced. In addition, each level of the hierarchy presents a hard boundary that, once traversed, usually incurs a significant delay penalty. This penalty will be present even if two logic blocks are physically close together but apart with respect to the hierarchy, which is counter-productive. Also, although any given level of hierarchy typically purports to provide a constant delay between the same members of the hierarchy, the physical distance and resulting differences in capacitance and resistance in the interconnect, do actually result in a fairly wide variation in inter-block delay. This effect is more pronounced for modern IC fabrication processes.

It is possible to ignore this variation and model it as a constant by choosing the worst case value, but this gives up important opportunities for delay optimization. For these reasons, most recent commercial FPGA routing architectures do not use this type of global routing architecture and, instead, use only one level of hierarchy to create a flat, island-style global routing architecture.

5.1.2 Island-Style Routing Architecture

As shown in Figure 5.2, island-style FPGAs logic blocks are arranged in a two dimensional mesh with routing resources evenly distributed

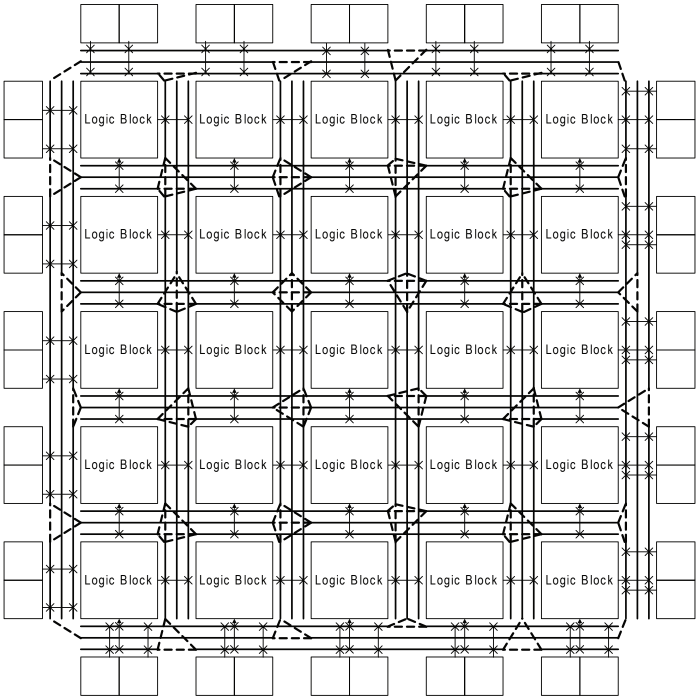


Fig. 5.2 Island-style FPGA.

throughout the mesh. An island-style global routing architecture typically has routing channels on all four sides of the logic blocks. The number of wires contained in a channel, W, is pre-set during fabrication, and is one of the key choices made by the architect. Island-style routing architectures generally employ wire segments of different lengths in each channel in an attempt to provide the most appropriate length for each given connection. They also typically stagger the starting point of the wire segments so that each logic block has a chance of connecting at the beginning of a wire of the most appropriate length.

Currently, most commercial SRAM-based FPGA architectures [18, 23, 125, 228, 231] use island-style architectures. This routing structure offers a number of desirable properties. Since routing wires of different lengths are in close physical proximity to logic blocks, efficient connections for a variety of design net lengths can be formed. By staggering the start and end points of channel segments of the same length, the physical layout for each logic block and surrounding routing channels can be optimized to form a single tile. This combined logic and routing tile can be replicated in two dimensions to form the FPGA array. As a result of this regularity, the minimum feasible routing delay between logic blocks can quickly be estimated. For the remainder of this survey we will exclusively examine island-style architectures, unless otherwise noted.

5.1.3 Detailed Island-Style Routing Architecture

The detailed routing architecture of an island-style FPGA defines the logical structure of interconnection between wire segments in routing channels and between logic block I/O and routing channel wire segments. The pins of a logic block that the routing architecture connects are divided into input pins (that drive data into the block) and the block's outputs. As shown in Figure 5.3, a logic block input pin connects to channel wire segments through switches in an input connection block [166]. The logic block output pins connect to channel wire segments via an output connection block. The fraction of wire segments in a channel which connect to an input logic block pin is the input connection block flexibility, $F_{c,\mathrm{in}}$. Similarly, the fraction of wire segments

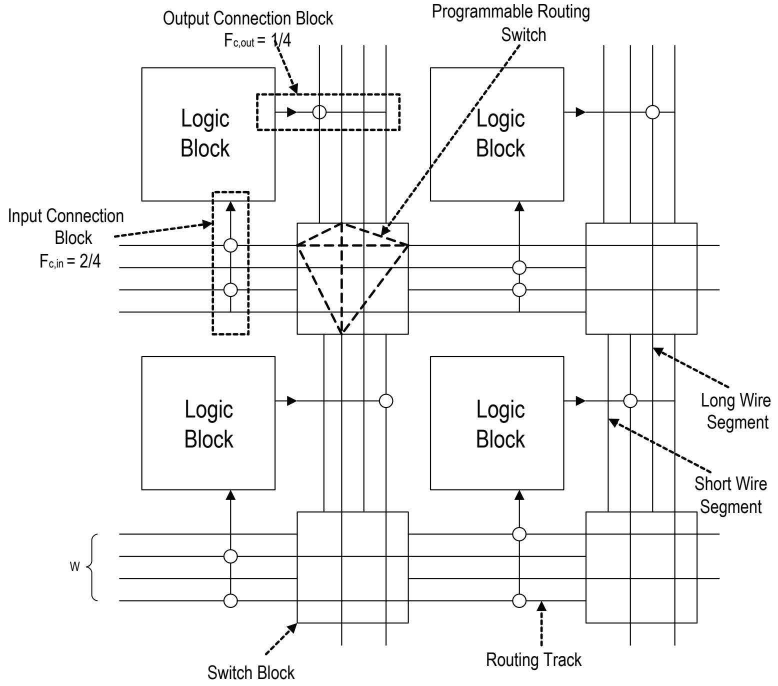


Fig. 5.3 Detailed routing architecture of an Island-style FPGA [37].

in a channel which connect to an output logic block pin is the output connection block flexibility, $F_{c,\text{out}}$.

A switch block [166] forms connections between wire segments at every intersection of a horizontal and vertical channel. Each switch block contains a set of switches which allow input wire segments to connect to wire segments in adjacent channels. The number of possible connections a wire segment can make to other wire segments is the switch block flexibility, F_s. In Figure 5.3, an example set of switch connections is represented as dashed lines and F_s is 3. As described in the next section, modern FPGA architectures often physically coalesce switch block and connection block structures into a single structure termed a routing driver block. Wire segments may start, end, or pass through a switch block unbroken. Each short wire segment spans one logic block while long wire segments span multiple logic blocks [37].

The organization of segment-to-segment connections inside a switch block has been a topic of significant study. Early switch block designers [208, 220] typically assumed a simple FPGA routing structure which only contained wire segments which span a single logic block throughout while still supporting $F_s = 3$. The routability of common switch block styles, such as the disjoint [220] and Wilton [208] switch blocks shown in Figure 5.4 were validated through extensive experimentation. The disjoint switchbox has been used in a number of commercial FPGAs including devices from the Xilinx XC4000 family. As seen in Figure 5.4(a), a wire entering a disjoint switch block can only connect to other wires with the same numerical designation via programmable switches. As a result, potential source–destination routes in the FPGA are isolated into distinct routing domains, limiting routing flexibility. The Wilton switch block uses the same number of routing switches as the disjoint switch block but overcomes the domain issue by allowing for a change in domain assignment on connections that turn. For example, in Figure 5.4(b), the connection from the track in domain 0 at the left can connect to the track in domain 3 on the bottom or domain 0 on the top. This ability to change domains in at least one direction facilitates routing as a greater diversity of routing paths from a net source to a destination is possible. In addition to the Wilton and disjoint switch blocks, a number of alternative designs, such as the Universal switch block, have also been suggested [52, 53, 80]. A full review of additional

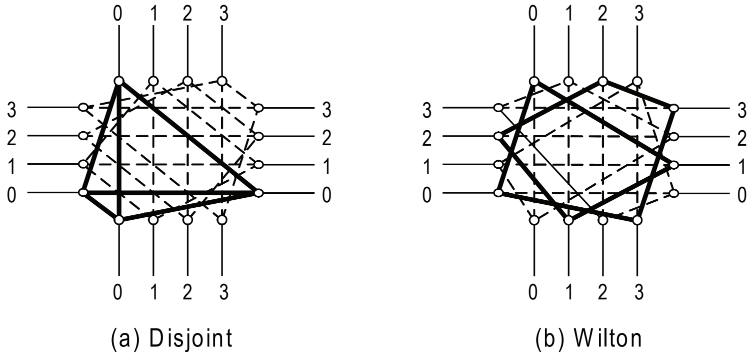


Fig. 5.4 Disjoint and wilton switch blocks.

island-style switch blocks optimized for length 1 wire segments can be found in [132].

Since contemporary FPGAs contain a variety of wire segment lengths, more recent switch block design has focused on optimizing connections in channels with segments that span both single and multiple logic blocks. Although switch blocks which rotate tracks are effective for length 1 segments, inefficiencies arise when they are used to make connections to multi-block wire segments. Due to the requirement for domain changes, a multi-block segment which passes through the Wilton switch block requires two switches to connect to two multi-block segments which are perpendicular to the segment. In contrast, for disjoint switch blocks, each multi-block wire segment only connects to one perpendicular segment. These differences are illustrated in Figure 5.5. For multi-block segments, the added routing flexibility offered by the extra connections does not overcome the area overhead of the extra switches [37, 132].

The majority of recent switch block designs only allow switch connections between wire endpoints or between wire midpoints, but not between endpoints and midpoints. Midpoint-to-midpoint connections are made using single disjoint connections. Example switch blocks in this style include the Imran [146] and shifty [130] switch blocks. The Imran switch block uses a Wilton switch block to connect endpoints of wires and single-transistor disjoint connections to connect

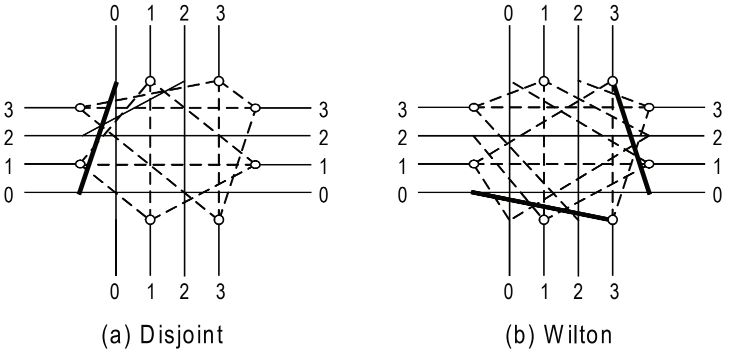


Fig. 5.5 Switch boxes for multiple block length wires.

midpoints. This switch block has been shown to be more area efficient than disjoint, universal, or Wilton switch blocks [146]. The shifty switch block similarly allows for domain changes on endpoint turns and disjoint connections at midpoints. Experimentation has shown that shifty and Imran switch blocks give similar area and delay results [130]. Both switch blocks are superior to disjoint switch blocks in area and delay performance due to their ability to allow for diverse routing paths.

A number of studies have examined routing architectures that include a variety of segment lengths in each routing channel. The fraction of segments of a specific length in each channel defines the segmentation distribution [37]. For example, in Figure 5.6, taken from [37], 40% of tracks are of length 1, 40% are of length 2, and 20% are of length 4. Several studies [91, 170] have examined segment length distribution for antifuse-programmed row-based FPGAs. Roy and Mahendale [170] determined row segmentation distributions using a tool which created a Poisson distribution of segment lengths based on design statistics. It was determined that a Poisson distribution of segment lengths was more area efficient than a uniform distribution of length 4 wires. More recent studies of segmentation for row-based architectures have used graph-theoretic formulations to evaluate segmentation distribution. Chang et al. [51] determined the wire distribution of a specific design using a graph-matching approach to match design wires to specific segment lengths. These segments were then packed into channels in a tree-like fashion. Later work [243] extended this effort by using a maximum spanning tree algorithm to perform the graph matching.

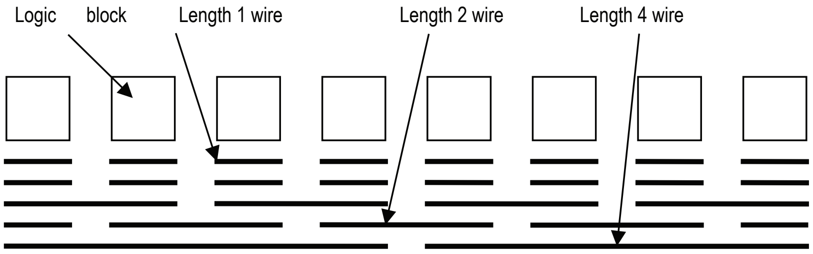


Fig. 5.6 Example channel segmentation distribution [37].

The need for a mix of segment lengths in island-style FPGA devices is motivated by the characteristics of benchmark designs targeted to the devices. The amount of interconnect required by a circuit has been found to be related to Rent's rule [123], a well-known relationship between the size of a group of logic and the number of its external connections. This relationship indicates that the amount of pins, P, needed for an amount of logic, G, grows as $P = K \times G^B$, where B is a parameter known as Rent's exponent and K is a scaling constant. The range of segment lengths required by a design is related to the Rent exponent B based on the equation $f_L = L^{2B-3}$, where f_L is the fraction of segments of length L [72]. The value of B varies between 0.5 and 0.75 for most circuits [30]. This wire length distribution has been used to determine FPGA segment lengths for a series of target designs [182]. Experiments determined that tuning the segment lengths to match predicted wire lengths and net fanouts can reduce FPGA area and delay on a per design basis. However, since designs targeting FPGAs often have widely varying wiring requirements it can be difficult to design a single segmentation distribution using analytical techniques that meets all the requirements. FPGA routing is generally architected to support all designs, including those with the largest B values. This choice allows almost all designs to fully use logic resources and complete routing successfully. As a result, in many cases, a sizable fraction of available routing resources is left unused. To confirm the efficiency of a new routing architecture, a device routability evaluation is usually performed via experimental methods (like those described in Section 4) with a suite of test circuits [136] rather than through the use of analytical techniques.

Several studies have attempted to determine FPGA segmentation by routing a series of designs and examining wire lengths. Brown et al. [42] used global routing followed by detailed routing to complete the FPGA design. Although this study questioned the need for segment lengths of greater than length 2 or 3, the two-step router increased the difficulty of wire sharing and limited the use of longer segments [37]. Betz et al. used a contemporary FPGA router which combines global and detailed routing into one step to evaluate segmentation [36]. This study verified the importance of including significant medium length segments which span between 4 and 6 logic blocks in an island-style routing architecture. This finding was validated during the development of the Stratix architecture [136], which contains significant length 4 and length 8 segments.

In addition to connection pattern and quantification parameters, FPGA detailed routing architecture performance is governed by the types of switches used to make connections, the size of transistors used to build programmable switches and the metal width and spacing of FPGA wires [36]. Routing switches are typically made from collections of basic transistor structures including pass transistors, buffers, and multiplexers. As VLSI technology used to implement FPGAs has evolved over the years, so has the design of switches used in the routing fabric.

Many FPGA architectures have been developed that use pass transistors and tri-state buffers as routing switches [36, 131, 179]. Figure 5.7 (taken from [36]) illustrates a routing architecture which contains both pass transistors and tri-state buffers. Both of these switch implementations support bidirectional wire segments since each segment can be driven by switches in multiple switch blocks. The relative usage of each type of switch dictates FPGA area and performance. Pass transistors minimize area consumption and are faster than buffered connections

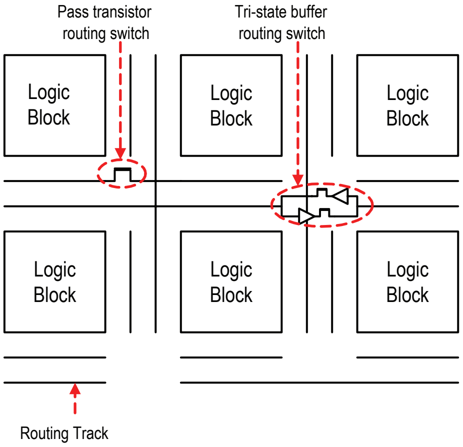


Fig. 5.7 Routing switches in an island-style FPGA routing architecture [36].

for short wiring paths that pass through a small number of switches. Generally, tri-state buffers provide faster interconnect for connections that pass through many switches. As a result, FPGA devices that intersperse pass transistors and tri-state buffers in the routing fabric provide better delay characteristics with the same area consumption as those that provide only one type of switch. In [36], it was shown that the fastest routing architecture that uses these switches contains 50% buffered routing tracks and 50% pass transistors. This experiment restricted signal paths to wire segments that exclusively use one type of switch. In [131], an attempt was made to modify this restriction by alternating buffer and pass transistor connections along each net path. Unfortunately, no delay improvements were achieved with this approach.

5.2 Unidirectional Single-Driver Routing Architectures

All of the pass transistor and buffered routing architectures described previously use bidirectional wire segments that can be driven by switch blocks on both ends of the segment. As shown in Figure 5.8(a), bidirectional wires are connected with bidirectional switches (e.g., two back-to-back tri-state drivers). The use of bidirectional wire segments can leave

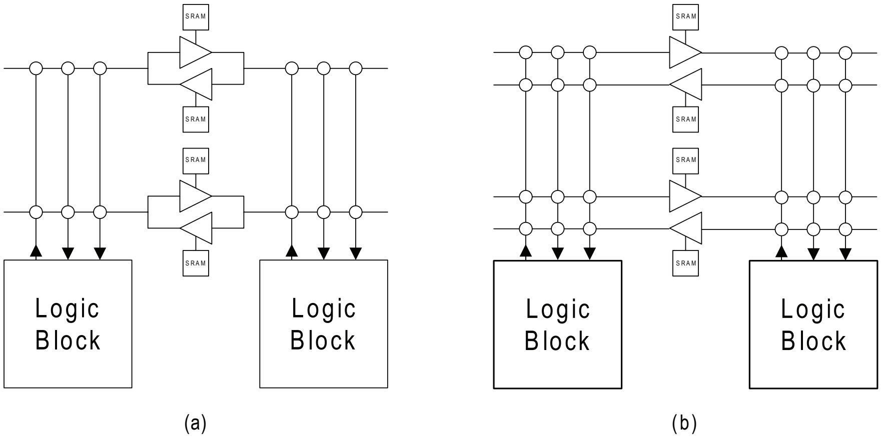


Fig. 5.8 Bidirectional and directional wires [128].

many routing switches unused [128]. Once programmed, each switch will be enabled to only drive one wire segment leaving the remaining switch resources unused. For example, in Figure 5.8(a), at least 50% of the tri-state buffers will be inactive. Additionally, the extra sinks per wire segment increase capacitance, impacting delay.

In contrast, a directional wire segment [239] is driven in a single direction. As seen in Figure 5.8(b), this halves the required tri-state buffers per switch [128].

Figure 5.9 shows two distinct choices that exist for the implementation of directional wire segment switches in a routing driver block. In the first implementation, shown in Figure 5.9(a), each wire segment is driven by adjacent wire segments from a single switch. These connections represent the logical switch block connections shown in Figure 5.3. Additionally, each wire segment may be driven by one or more logic block output pins via a pass transistor. In [128], this implementation is referred to as a directional tri-state (dir-tri) implementation. An alternate approach to implementing directional wires is shown in Figure 5.9(b). In this single-driver implementation, a switch multiplexer selects inputs from both wire segment and logic block sources for each output wire segment. As a result, each single-driver wire segment can be driven by a non-tri-state buffer, improving drive strength.

This single-driver approach physically combines the logical switch and output connection blocks to form the routing driver block shown in Figure 5.10. All logic block outputs which source the switch must come from physically adjacent blocks. This constraint can be limiting for wire segments that span multiple logic blocks. Although not as prevalent,

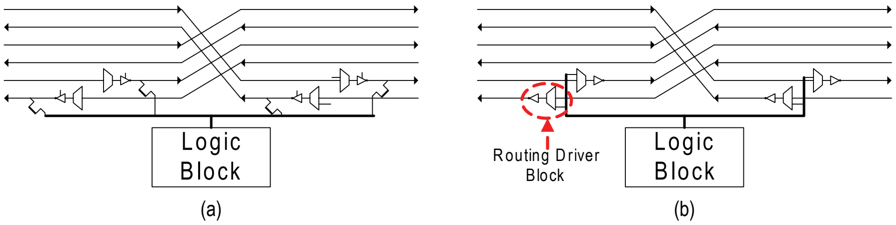


Fig. 5.9 Directional routing connection blocks: (a) dir-tri (b) single-driver.

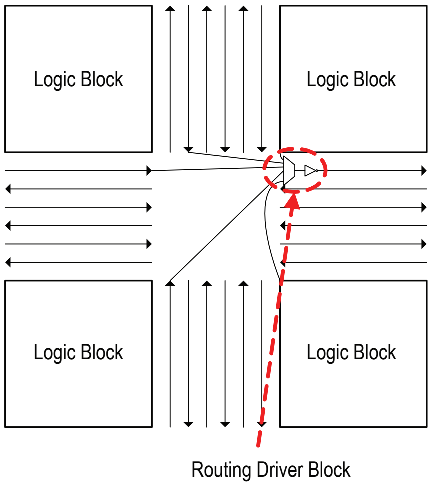


Fig. 5.10 Routing driver block implementation.

logic block outputs can also be integrated into routing switches for bidirectional wire segments [131, 153].

An initial concern when using directional wire segments in place of bidirectional segments is a possible increase in the number of required wire segments per channel. As seen in Figure 5.8, twice as many wires per channel are required for directional routing to include the same number of drivers as their bidirectional counterparts. However, experimentation indicates that roughly the same number of tracks per channel is needed to achieve the same routability for dir-tri [128] and single-driver switch implementations [128, 136]. As a result of this track count finding, the overall area required to implement a dir-tri routing architecture was found to be about 20% less than the corresponding bidirectional equivalent [128]. However, the directionality of wire segments was found to negatively impact the flexibility and drive capability of pass transistor connections from logic block outputs to wire segments, causing a delay increase of 3% for the dir-tri case versus a corresponding bi-directional architecture [128].

In contrast to the dir-tri results, experimental results generated using a single-driver routing architecture show both area and delay improvements over bidirectional routing architectures. In [136], an 18%

area improvement and 16% delay improvement was noted for a 100% single-driver architecture versus a directional architecture that contains a 50–50 mix of pass transistors and buffers. In calculating these results, a spectrum of architectures containing a mix of bidirectional and single-driver wire segments was considered, but the 100% single-driver case always gave the best results for area and delay. In a similar experiment, [128] showed that a single-driver routing architecture was superior to an architecture with bidirectional buffers by 25% for area and 9% for delay. Much of the area improvement was found to be due to the unidirectional nature of the routing (20% improvement). Delay reduction was primarily due to reduced driver capacitance on each wire and the reduced overall routing area.

Experiments with single-driver routing architectures [128] have confirmed that low connectivity ($F_s = 3$) is appropriate for these architectures. Since wire segments in these architectures can only be driven from endpoints, switch block design does not require the assessment of wire midpoint connections.

5.3 Additional Routing Structure Improvement Approaches

In addition to the relatively recent introduction of single-driver architectures, several other efforts have attempted to improve FPGA routing performance through the inclusion of less generally programmable or "hard-wired" connections. Numerous commercial FPGAs [23, 125, 231] allow for direct connections between logic blocks to avoid the need to drive the interconnect fabric. The work in [165] showed that these connections, which avoid delays in traversing connection blocks and switch blocks for very near neighbor connections, can improve speed by 6.4% at a small (3.8%) area cost. In [107], an architecture which drives wires of length 5 directly between logic blocks is proposed although the specific benefits of the technique are not enumerated.

A technique to replace switch block connections with fixed metal connections between horizontal and vertical wire segments was described in [185]. A study was first performed to determine if wire segments are often programmably connected into specific patterns such as

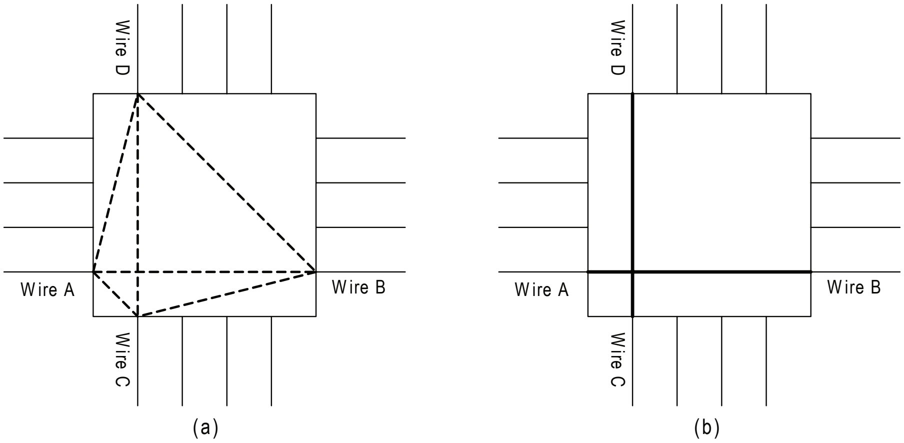


Fig. 5.11 Hard wired switch blocks [185].

T's or L's. This motivates the removal of some programmable switches to create fixed metal wiring that fit these patterns. For example, if the switches in Figure 5.11(a) (represented as dashed lines), are frequently programmed to create a + connection between Wires A, B, C, and D, it may be advantageous to remove the switches and form one piece of metal (as seen in Figure 5.11(b)). However, the removal of the switches now precludes the ability to form separate horizontal and vertical connections between wire segments if a + connection is unneeded. In [185], the effect of removing a subset of pass transistor and buffered switches to form L, +, and T shaped connections is evaluated with a full routing experiment. A 22% improvement in critical path delay and a 6% area reduction is seen for a set of 10 benchmark designs. A form of this approach has been included in the Xilinx Virtex 5 FPGA family, which includes wires that connect diagonally [231].

5.3.1 Circuit-Level Techniques to Improve Routing

Several researchers have attempted to improve the performance of interconnect wires through increased wire spacing, because in modern IC fabrication technologies, the proximity of two routing tracks gives rise to a capacitive effect known as crosstalk. By spacing wires farther apart this effect can be reduced, resulting in reduced capacitance on the wire and increased speed. The work in [35] determined that a 13% circuit speedup could be achieved by using 5 times minimum wire spacing on 20% of the routing tracks in each island-style channel. Although wire spacing improves performance for architectures which contain pass transistors and buffered switches, increasing physical wire width does not due to the need to increase driver size. Increased track spacing was implemented in a commercial architecture [107] which assigns 20% of routing wires to these fast routing resources.

Another circuit-level technique to improve performance involves the use of routing multiplexers which contain fast paths, as shown in Figure 5.12. The number of pass transistors required to traverse different paths in the multiplexer is imbalanced leading to fast paths for critical inputs and slower paths for regular inputs. Lewis et al. [135] report that this technique has been integrated into the routing architecture for Altera Stratix II devices. Like the spacing approach, critical paths are assigned to fast routing resources by the FPGA router. It was found that the availability of imbalanced multiplexers improved design performance by 3% without impacting device area.

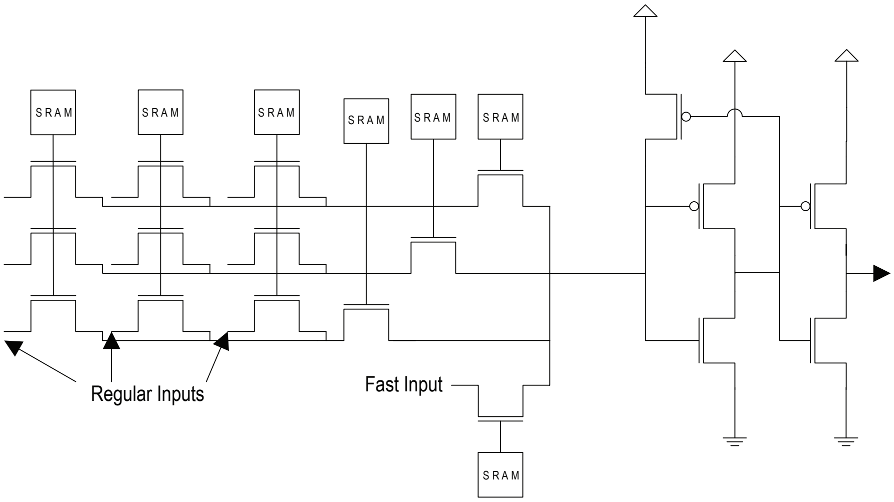


Fig. 5.12 Fast inputs to routing multiplexers [135].

Recent trends in single-driver wiring for FPGA routing have been motivated by the implementation of novel transistor-level circuit structures in building the routing switches. Three specific issues have been investigated to support routing delay reduction: buffer construction, transistor sizing, and routing driver multiplexer sizing. As shown in Figure 5.13, FPGA routing buffers are typically implemented as a sequence of CMOS inverters. In [131], it was determined that a series of three inverters, rather than two or four, is desirable since balanced rise/fall characteristics are achieved at the buffer output due to its inversion of the input signal. Two or four-stage buffers could be used instead if a non-inverting buffer is preferred.

In [128], a series of experiments related to sizing of single-driver architecture switches is described. In these experiments, $0.18\,\mu$m transistors in a three-inverter buffer are evaluated to determine appropriate transistor sizing for delay and area-delay optimization. The relative size of the PMOS and NMOS transistors in each inverter for a given technology plays an important role in buffer performance. For example, for the $0.18\,\mu$m technology used in [128], it was determined that the relative sizing of PMOS and NMOS transistor width (W_p/W_n) for the three stage buffer should be 1/3.5 for the sense stage, 1/1 for the intermediate stage, and 1.4/1 for the drive stage. After determining various inverter

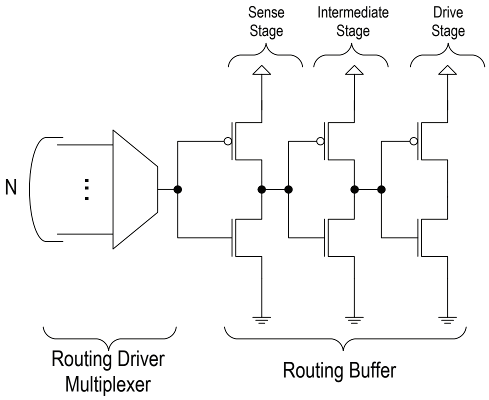


Fig. 5.13 Transistor-level implementation of routing driver multiplexer and routing buffer [131].

ratios, [128] examined a range of possible input counts per single-driver routing switch from delay and area-delay perspectives. For delay optimization, it was determined that the fastest single-driver switch contains a 4-to-1 routing driver multiplexer. A single-driver switch with an 8-to-1 multiplexer was determined to be optimal for area-delay product.

5.3.2 Bus-Based and Pipelined FPGA Routing Structures

As FPGAs have grown in size, the amount of datapath circuitry targeted to the devices has grown. This has led several research projects to consider both bus-based connections and pipelining the interconnect signals.

5.3.2.1 Bus-Based Routing

A number of researchers have noted that if the signals on an FPGA are grouped together as buses, then the control of the programmable switches could be amortized across the entire bus. This notion was first proposed in [58] and [59]. In addition, the regularity of the bus connections can be employed to reduce the total number of switches needed.

In [238], a series of routing architecture optimizations are considered that take advantage of the bus structure of many datapath circuits. The bit-sliced example shown in Figure 5.14 illustrates how switches in the interconnect fabric can be removed to take advantage of datapath regularity. Consider implementing the four-bit slice datapath shown in Figure 5.14(a). This circuit could be implemented using conventional logic block bit slices, as shown in Figure 5.14(b). A minimum of four FPGA logic block and interconnect tiles are needed to implement the needed logic and communicate the needed circuit inputs and outputs.

Using bus-based connections, the four logic blocks shown in Figure 5.14(b) are combined to form a multi-bit logic block in Figure 5.14(c) and individual wires in the original implementation are grouped into four-bit routing buses. Although both implementations require eight logic block input and output connections, the number of switch block connections for the bussed implementation is reduced

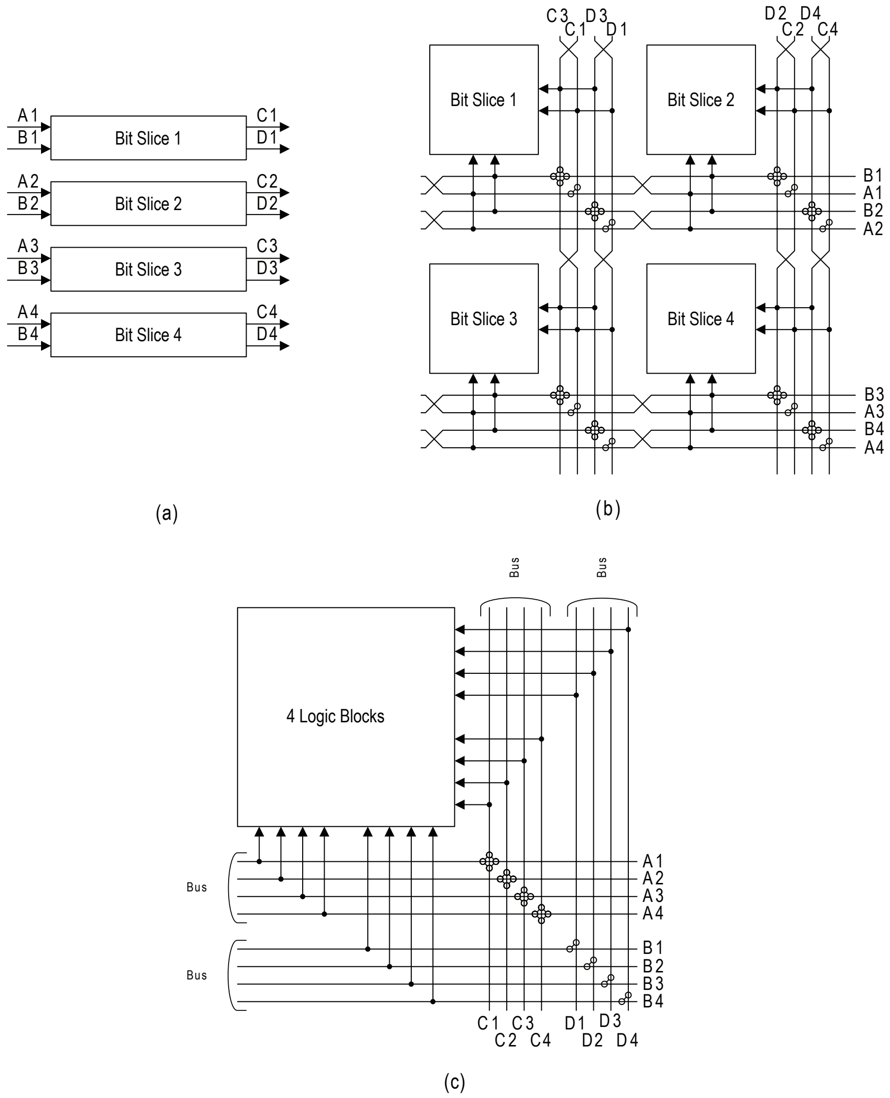


Fig. 5.14 Bus-based routing FPGAs [238].

by half. This bus-based approach reduces the need for wiring flexibility by taking advantage of per-bit bus regularity. Not only are the number of storage bits needed to configure routing connections significantly reduced due to smaller switch count, but configuration bits can be shared across adjacent routing switches since they have the same setting, leading to additional savings. Through experimentation [238], it was determined that 14% routing area can be saved using this approach for datapath oriented designs leading to an overall 10% FPGA area reduction. Prior research anticipated a larger area saving that was not realized in this detailed implementation, largely because a significant fraction of the wires have to be non-bussed to handle non-bus signals. Approximately 50% of routing wires were needed for dedicated four-bit buses while the switches for remaining wires were individually programmed.

5.3.2.2 Pipelined (Registered) Routing

Although recent FPGA system clock speeds approach 200–400 MHz, they still lag far behind their microprocessor counterparts. In addition, while a specific microprocessor operates at the same frequency for each application, FPGA operating frequencies vary from application to application. In general, the long and variable interconnect delays associated with FPGA routing are responsible for both of these issues. Several FPGA research projects [183, 197, 204] have examined adding pipeline registers to FPGA interconnect to address these concerns. The basic idea is to have a router insert registers as it routes, increasing the clock frequency in exchange for added cycle latency.

While registers allow for enhanced raw clock rates, they complicate the FPGA routing problem since the number of flip–flops on paths which converge on a logic block must be matched to allow for causal behavior. In [197], flip–flops are added to all interconnect switches and logic block inputs and outputs for a routing network organized in the hierarchical topology shown in Figure 5.1. This approach of pipelining segment-to-segment connections and logic block I/O allows all designs mapped to the FPGA to run at the same system clock frequency. To account for routing paths which traverse different counts of interconnect flip–flops, an adjustable value of up to seven flip–flops is allocated per logic block input. The inclusion of the routing flip–flops leads to a 50% increase in overall routing area.

Two research projects have examined including flip–flops directly into the segmented routing fabric of an island-style routing architecture. In [204], all horizontal segment to vertical segment connections are buffered by flip–flops. The authors account for the presence of the flip–flops in their FPGA routing algorithm by attempting to minimize the number of corner turns included in net routes and by re-timing nets to equally distribute flip–flops along paths. Although this approach shows promise, the area-normalized throughput of designs does not significantly improve for the proposed architecture and tool flow. Additionally, designs not suited to re-timing may suffer a latency penalty.

In [183], the interconnect fabric of a standard island-style architecture is enhanced with routing flip–flops. Unlike [197], only a small fraction of switch block connections are pipelined, and only one additional flip–flop is needed amongst the logic block inputs. As seen in Figure 5.15, in cases when the routing switch flip–flop is unneeded, it can be avoided through the use of a programmable multiplexer. Since both pipelined and unpipelined routing segments are available, retiming algorithms can selectively move the position of design flip–flops from logic blocks to the interconnect fabric to achieve the fastest possible design clock frequency. Through experimentation it was determined that about 12%–25% of routing switches should contain a flip–flop.

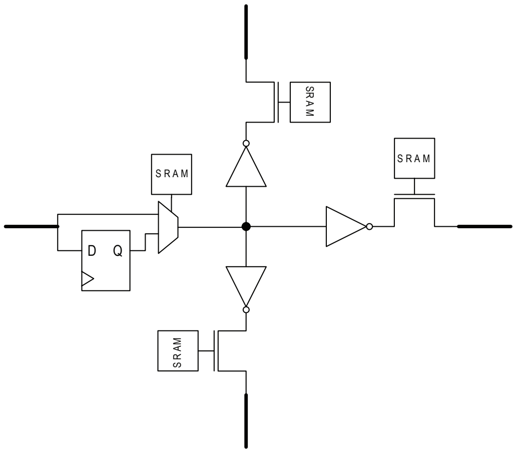


Fig. 5.15 Registered routing switch [183].

The inclusion of these flip–flops results in an average design speedup of about 25% with a 10% overall FPGA area increase.

5.4 Power Related Issues

Although FPGA routing has traditionally been constructed to balance area and delay, power consumption has recently become an important issue. Several studies [84, 138, 178, 198] have indicated that between 60%–70% of FPGA dynamic and static power consumption is located in the programmable interconnect. Leakage power is a particular concern since much of the interconnect resources within the FPGA are not actively used following device programming. Most transistor leakage in FPGAs is a result of source-to-drain subthreshold leakage and gate-to-source gate oxide leakage. Subthreshold leakage increases exponentially as transistor threshold voltage, V_t, is reduced to obtain high performance. Gate oxide leakage is a result of electron tunneling as the transistor gate oxide is thinned. The leakage current due to tunneling increases exponentially with oxide thinning. If a transistor is not being used, both components of leakage can be eliminated by removing the supply voltage, V_{DD}, from the transistor. In contrast, dynamic power reduction techniques target the activity of interconnect transistors that are used. Dynamic power can be saved by reducing the supply voltage associated with circuits since this value is proportional to $V_{DD}{}^2$.

A variety of techniques have been applied to reduce dynamic and static power consumption in FPGA routing. For single-driver wiring, these techniques have been applied to both routing multiplexers and buffers. In [137], V_{DD} for each routing buffer can be driven by two separate sources, a full-rail V_{DD} (V_{DDH}) and a reduced V_{DD} (V_{DDL}). As shown in Figure 5.16, two transistors provide three possible routing states, high-performance (transistor M1 active), reduced performance (transistor M2 active), and sleep mode (both transistors shut off). Through experimentation it was determined that a large fraction (88%) of interconnect buffers could be placed into sleep mode and 85% of active routing buffers could be driven with V_{DDL} without increasing circuit delay for a 100 nm process. This combination results in an 80% overall reduction in interconnect leakage and a 38% reduction in

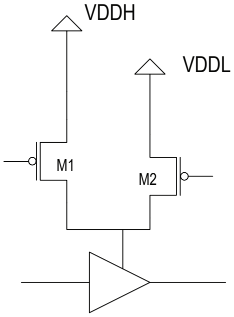


Fig. 5.16 V_{DD} programmability [137].

interconnect dynamic power for an overall interconnect power reduction of 56%. Later work by the same authors [141] explored techniques to reduce SRAM bit count by combining the SRAM bits used to control V_{DD} selection. Both of these projects assumed a buffered tri-state interconnect although the same power-reducing techniques can be applied to buffers in single-driver architectures.

A similar approach of using separate transistors and V_{DD} levels was described in [82]. A 22% reduction in interconnect dynamic power and an 81% reduction in interconnect leakage power was achieved. Power calculations were performed after timing-driven place and route.

Although the previous dual-V_{DD} approaches show good power reduction, they require the chip-wide distribution of multiple V_{DD} values. A V_{DD}-selection approach for routing buffers which does not have this limitation [29] appears in Figure 5.17. This work proposes a new switch architecture which adds two transistors, MNX and MPX, to the buffer in the routing switch (this differs from [137] which used two PMOS devices connected to two different power rails). The two extra transistors allow the switch to operate in a variety of modes, depending on their gate voltages. When both are turned on, the effective supply voltage, V_{VD}, seen by the buffer is V_{DD} (assuming minimal voltage loss due to parasitics in the MNX/MPX transistors), and the buffer operates in a traditional, high-performance mode. When MNX is enabled and MPX is disabled (e.g., the buffer is not in either sleep or low power mode), the switch's effective supply voltage V_{VD} is reduced

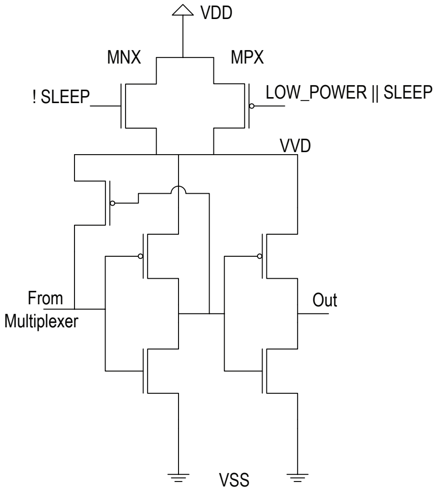


Fig. 5.17 Low power programmable routing buffer.

by V_t, since MNX is an NMOS transistor and can therefore pass only a weak-1 signal. This configuration allows the buffer to operate in a low-power mode.

With respect to the decreased performance, the authors note that in their benchmark circuits, 75% of routing resources could tolerate a slowdown of 50%, so most switches could withstand the lower performance for a 70 nm process. With respect to passing a weak-1, the authors found that most routing switches drive other routing switches. Since level-restoration is built into the switch, weak-1 inputs are permissible. When both MNX and MPX are disabled, the transistors are powered off, which greatly reduces leakage power by cutting off the supply voltage from the buffer, effectively turning it off. It was found that leakage power (including effects of the new transistors) was reduced by about 35% when operating in low-power mode (i.e., MPX off, MNX on) and up to 61% when operating in sleep mode (i.e., MPX and MNX off) for a single-driver routing architecture. The authors also found a 28% reduction in dynamic switching power when operating in low-power mode. Estimated area measurements predict a per-tile area increase of about 10%.

In addition to V_{DD} modulation to save routing power, several researchers have examined the use of FPGA interconnect transistors which have differing threshold voltages (V_t). The use of high V_t increases transistor delay but reduces static power consumption. Several papers [82, 137, 164] describe the use of high V_t transistors to implement configuration SRAM bits. Since these bits are not subsequently read, performance is not an issue. These techniques are likely used in commercial devices in addition to the use of thicker oxides to further reduce the leakage of these devices which are not performance critical [27, 233].

Other research projects have examined the use of routing resources with a mix of standard and high V_t routing transistors. In [164], an experiment is performed which replaces a fixed percentage of transistors in routing multiplexers with high V_t equivalents. The authors claim that since most connections are not timing critical, they can increase delay in 50% of connections by 50% and not affect overall system performance by using high-V_t multiplexer transistors. They also note that routing utilization on the higher performance nets will be increased, since they will be preferred by the router. Overall, the authors estimate that they can reduce leakage power in the switch blocks by 40% if 50% of the switch blocks use high V_t transistors.

One interesting approach to using high-V_t transistors to control power consumption involves using redundant SRAM bits to control unused paths in routing buffers [164]. Traditionally, when a multiplexer has many inputs, it is implemented in multiple stages to help decrease the parasitic capacitance on the output node of the multiplexer. In these configurations, SRAM bits are minimized by having one bank of SRAM cells feed all multiplexers in a given level. This is illustrated in Figure 5.18(a).

While this approach minimizes the number of SRAM cells needed, multiple pass transistors are activated on unused paths, leading to unnecessary leakage power dissipation. For example, in Figure 5.18(b), only one pass transistor in the first stage passes the output value and the remaining transistors can be shut off. This goal requires additional SRAM bits since each path in the first stage needs to be controllable. The more SRAM bits used, the finer the granularity, which means that more unused multiplexer transistors in a level can be disabled.

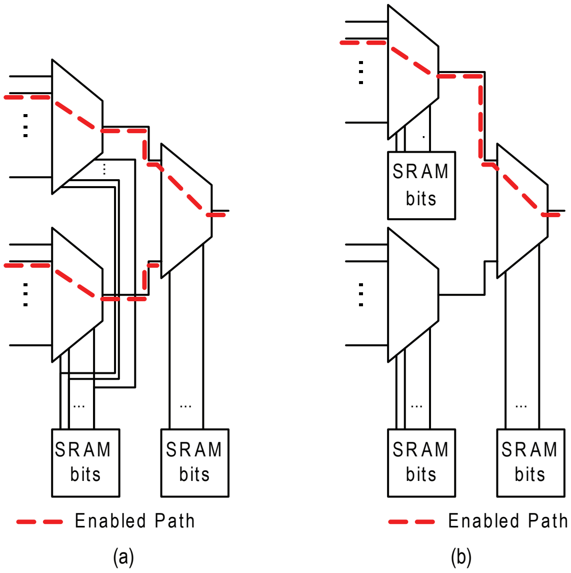


Fig. 5.18 Multiplexer with redundant memory. (a) Minimum memory usage. (b) Redundant memory.

It is important to note that the extra SRAM bits may also contribute leakage current, but since these bits are not timing critical, they can be implemented with high-threshold, low-power transistors. Another downside to this approach is that the extra SRAM bits require more area. The authors find that for a 30-to-1 multiplexer implemented in 2 levels, doubling the number of SRAM bits can decrease average leakage power by a factor of 2 but increases interconnect area by 30%–50%.

Two other techniques to control FPGA interconnect power involve gate and body biasing of unused interconnect transistors. By applying a bias voltage to the body terminal of a transistor, it is possible to alter its effective threshold voltage, and therefore reduce sub-threshold leakage [164]. The fabrication of a transistor which will support body biasing is complicated since a multi-well process is required which consumes area due to minimum well-spacing requirements and other design rules. The bias voltages can be controlled by circuitry that fixes threshold voltages to a target level. In [164], a 2-stage, 30-to-1 multiplexer is investigated for 2 well and 6 well processes. The experiments determined that for an area increase of 1.6X to 2X, leakage current can be reduced by 1.7X to 2.5X. A further extension of this body-biasing is proposed in [151] which adaptively adjusts the body-bias for all the blocks within the FPGA. This scheme reduced leakage power by 3X with the added benefit that within-die and die-to-die variation was also reduced.

5.5 Challenges in Basic Routing Architecture

Although much progress has been made in defining the routing architecture of commercial FPGAs, the changing role of programmable devices in embedded and desktop computing systems will necessitate changes in the near future. Before FPGAs can be effectively used in portable electronics, issues related to the static power consumption of routing must be addressed. One possibility may involve selectively shutting down power to regions of the routing fabric based on resource demand. This approach may potentially require FPGAs to have a more well-defined routing hierarchy so that the interface between active and inactive routing is clearly defined. The isolation of FPGA routing into a distinct hierarchy may also help provide clearly defined boundaries between functional components built in FPGA logic, an important issue for embedded security [105]. At the system level, the increased use of multiple microprocessors built from logic within an FPGA may necessitate the inclusion of optimized inter-processor routing resources within the FPGA routing fabric.

Recent trends in FPGA device and packaging also present new challenges for FPGA routing. Renewed interest in 3D device packaging has direct applicability to FPGA logic [140] and routing structures [139]. As transistor channel widths shrink deep into the submicron realm, the manufacturability of reliable FPGA routing becomes an issue. Although this issue has been examined for routing in non-silicon based devices [71], effective run-time fault avoidance in contemporary commercial FPGAs remains an active research area.

Despite years of research, several issues remain for existing island-style commercial FPGA architectures. Since FPGA routing consumes a significant amount of the FPGA die area [9, 37] and typically only a fraction of routing switches and wires are actively used per design, additional research is needed to understand how to better minimize circuit routing needs for FPGA routing architectures. Additional research is also needed to define acceptable switch block patterns for single-driver switches and connection block patterns for hard blocks, such as memory blocks and multipliers.

6

Input/Output Architecture and Capabilities

The logic and routing structures described in the previous sections serve as a general purpose platform that can be used in many different applications. This platform must interface at many different speeds and voltages with the wide range of external components that may connect to an FPGA. This is done through dedicated input/output pads and cells on FPGAs and, in this section, we discuss architecture-level issues and features in the design of these cells.

We refer to the I/O pad and surrounding supporting logic and circuitry as an input/output cell. These cells are important components of an FPGA both because this interface sets the rate for external communication and these cells along with their supporting peripherals consume a significant portion of an FPGA's area. For example, in the Altera Stratix 1S20 and the Altera Cyclone 1C20, I/O's and peripheral circuitry occupy 43% and 30% of the total silicon area, respectively [133].

A crucial consideration in I/O cell design is the selection of which interface standards to support. In this section, these challenges are examined and current input/output architectures are reviewed.

6.1 Basic I/O Standards

The major challenge in input/output architecture design is the great diversity in input/output standards. For example, different standards may require different input voltage thresholds and output voltage levels. To support these differences, different I/O supply voltages are often needed for each standard. They may also require a reference voltage to compare against the input voltages. Other standards require clamping diodes which allow specific abnormally high or low voltages to be tolerated. Many standards also rely on differential signaling to improve noise immunity and enable increased data transmission speeds. Proper termination is also essential for maintaining signal integrity but different standards have different termination requirements.

Table 6.1 summarizes these requirements for common standards supported by current FPGAs [23, 124, 231]. The table lists the I/O standard, the required supply voltage, the reference voltage if necessary, and the termination required. Besides these major differences, there are a multitude of other options that may be desirable such as tristate drivers, programmable drive strengths and pull-up or pull-down behavior.

Table 6.1 Commonly supported I/O standards (N/R = Not Required).

I/O Standard	Output supply voltage	Reference voltage	Termination voltage
LVTTL	3.3V	N/R	N/R
LVCMOS	3.3V	N/R	N/R
3.3V PCI	3.3V	N/R	N/R
3.3V PCI-X	3.3V	N/R	N/R
SSTL-2 Class I	2.5V	1.25V	1.25V
SSTL-18 Class I	1.8V	0.90V	0.90V
1.8V HSTL Class I	1.8V	0.90V	0.90V
1.5V HSTL Class I	1.5V	0.75V	0.75V
1.2V HSTL Class I	1.2V	0.60V	0.60V
Differential SSTL-2 Class II	2.5V	N/R	1.25V
Differential SSTL-18 Class II	1.8V	N/R	0.90V
1.8V differential HSTL Class II	1.8V	N/R	0.90V
1.5V differential HSTL Class II	1.5V	N/R	0.75V
1.2V differential HSTL Class II	1.2V	N/R	0.60V
LVDS	2.5V	N/R	N/R
HyperTransport	2.5V	N/R	N/R

6.2 I/O Architecture Issues

One of the most significant decisions in input/output architecture design is the selection of the standards that will be supported. This involves carefully made trade-offs because, unlike general-purpose logic structures, such as the LUT, which can implement any digital function, an I/O cell can generally only implement the standards selected by the I/O cell designer. However, the decision regarding which standards to support is far from straightforward. Supporting a greater number of standards can increase the silicon area required for the I/O cells significantly. Additionally, the pin capacitance may increase with each additional supported standard, which can limit performance [200]. However, the usefulness of FPGAs depends on their flexibility, including the ability to support different signaling standards. Given these conflicting factors, the final selection of standards often depends heavily on business factors as opposed to technical factors, and therefore the choice of which standards to support is typically made by a marketing research arm of an FPGA vendor.

Once the I/O standards to be supported are known, it is necessary to determine which input/output pins will support each standard. At one extreme every pin can support every standard and feature. This approach is clearly the most expensive and may result in implementation difficulties as a large capacitance will be attached to each pin. However, this generality gives the printed circuit board designer who uses the chip the most flexibility. At the other extreme, different standards can be limited to different groups of I/O pins. This approach may result in easier electrical design and lower-cost (to the FPGA vendor) I/O cells, but it may limit flexibility for the printed circuit board designer.

It is desirable to make all the input/output pins in an FPGA equivalent. Until recently, this was generally the case for commercial FPGA offerings. However, the increase in the number of input/output pins on an FPGA and the number of standards for inter-chip communication has made full equivalency impractical. Many standards have conflicting requirements such as differing output voltages. Given such differences it would be impractical for every pin to be able to independently support every standard. Instead, most modern FPGAs have adopted an I/O banking scheme in which input/output cells are grouped into predefined banks [23, 60, 124, 231]. Each bank shares supply and reference voltage supplies. A single bank therefore cannot support all the standards simultaneously, but different banks can have different supplies to support otherwise incompatible standards.

The use of different I/O banks is now a standard practise. However, there is less of a consensus regarding the number of pins in a bank or the equivalency of the banks. In some FPGA families the number of I/Os per bank is relatively constant for all device sizes at 64 pins per bank [228] or 40 pins per bank [231]. At the other extreme, some FPGA families adopt a fixed number of banks across all the devices of the FPGA family [18]. This latter approach means that the number of pins per bank will be significantly larger for the largest members of a device family. This can be very restrictive when using these large devices. A hybrid approach of having a variable number of banks with a variable number of pins per bank has recently been used [23]. Devices with more I/O pins have more banks but the number of pins per bank is allowed to increase as well. This avoids limiting the flexibility as extensively as the fixed number of banks approach, while also reducing the costs required to provide a large number of I/O banks. Table 6.2 lists the number of I/O's, the number of I/O banks and the resulting number of I/Os per bank for a few commercial devices. With the wide range in the number of I/Os available in an FPGA family, it is clear that, unless the number of banks is increased, the number of I/Os per bank can grow significantly.

Besides bank sizing, it is necessary to determine whether independent banks will be functionally equivalent. Each bank could independently support every I/O standard supported by the device. This is the approach used for some FPGAs [231]. This does not imply all the pins

Table 6.2 Comparison of current I/O architectures.

	Stratix II [18]	Stratix III [23]	Virtex 5 [231]
# of I/Os	342–1170	296–1120	172–1200
# of I/O Banks	10–12	12–24	7–33
# of I/Os/Bank	6, 34–168	24–48	20, 40

are completely flexible as the use of one standard within a bank may preclude the use of other standards within that same bank because of incompatible voltage requirements. Some FPGAs instead opt to limit the standards supported by each I/O bank [23]. This reduced flexibility may save area. Flexibility and hence marketability suggests that all banks should be equivalent with as few pins per bank as possible, but this increases the area requirements for the I/O cells. These trade-offs have not been studied extensively.

6.3 High-Speed I/O Support

Before the advent of high-speed signaling, an FPGA I/O cell only had to detect/drive a logic zero or one from/to the outside of the device. This functionality could be directly controlled by the FPGA's logic fabric. However, for high-speed differential inter-chip signaling or high-bandwidth memory interfaces, additional circuitry is often necessary because of the high speeds involved. At a minimum, standards that make use of differential signaling require two I/O cells to be paired together with differential transmitters/receivers. Additionally, inter-chip signaling and memory interfacing each demand specialized circuitry to facilitate these high-speed links.

General high-speed inter-chip interfaces require special-purpose serialization and deserialization (SERDES) blocks [23, 231] to support multi-gigabit interfaces. This high speed signaling often necessitates the use of source synchronous clocking in which a clock is transmitted along with the data signals. For such approaches, proper signal sampling is challenging and dedicated circuitry for performing dynamic clock phase adjustment or per bit delays is often required [101]. Dedicated circuitry to perform this phase adjustment [23] or individual bit delay [231] is now standard in commercial FPGAs.

High-speed communication requires more than the high-speed signaling that these analog features enable. Therefore, contemporary FPGAs frequently include dedicated digital circuits to support higher-level protocols [19, 228]. These features, such as Ethernet Media Access Controllers (MAC) [228], represent another form of logic block heterogeneity, similar to the type described in Section 4.

High-speed memory interfaces also need special-purpose hardware to accurately capture data flowing between memory chips and the FPGA. Delay-locked loops (DLLs) and phase-locked loops (PLLs) are used to adjust the phase of a transfer clock to ensure that data from the external memory is sampled when the data is valid. To account for potential timing variations due to process, voltage or temperature differences, phase adjustment is performed dynamically [23, 231]. I/O cells also generally contain circuitry to enable fine grained per bit delays of each data bit [23, 231] so that system level timing requirements can be met. The amount of soft logic required for the memory interface is reduced by this additional circuitry while high-speed operation is ensured.

Additional features, such as adjustable signal terminations (called Digitally Controlled Impedance [231] or On-chip Termination [23]), further simplify board design and ensure a proper signaling environment.

6.4 Challenges in I/O Architecture and Design

We have briefly discussed some of the many issues that must be considered in the design of an FPGA's I/O architecture. While modern commercial FPGAs provide some solutions to these design questions, FPGA I/O architecture remains a relatively unexplored area. A better understanding of the appropriate granularity for I/O banks is needed, and the extent to which equivalent I/O banks are necessary should be explored. Both issues require an understanding of the needs of state-of-the-art printed circuit board design.

The electrical design of broadly programmable I/O cells is a difficult and open problem. Are there clever ways to create a single cell that will support many I/O standards efficiently while using minimal area and ensuring that the support of one standard does not interfere with another?

These questions are very important since additional high-speed communication approaches for memory interfaces and inter-chip communication will be needed in the near future. Emerging standards will potentially necessitate even more dedicated circuitry on the FPGA. Decisions regarding which standards to support and how to support them will become ever more challenging.

7

Improving FPGAs

The FPGA architectural developments described previously have improved the area efficiency, performance, and power consumption of FPGAs. However, the overhead incurred to make FPGAs both general purpose and field-programmable often prevents the use of FPGAs for some applications. When the cost, performance and power consumption specifications go beyond what is possible on an FPGA, other alternatives must be considered. These alternatives include full-custom design and fabrication, ASIC cell-based design or partially prefabricated “structured” ASICs. Before reviewing these alternatives, it is useful to first quantify the area, performance, and power gap between FPGAs and ASICs since this will inform the search for customized alternatives to FPGAs and improvements to FPGAs.

7.1 The Gap Between FPGA and ASICs

It is well understood that FPGAs suffer in terms of area, performance, and power consumption relative to ASICs. The area penalty of using an FPGA over an ASIC at high volume results in a significantly higher unit price, for example.

The extent of these differences is often not fully appreciated; yet, it is what systems designers must gauge when determining whether their system can be implemented using an FPGA or one of the alternatives. This gap between FPGAs and ASICs is also what FPGA architects fundamentally aim to narrow. There have been various comparisons between FPGAs or similar devices and ASICs in the past [45, 214, 245]. Recently, a more thorough comparison has been performed [119, 120]. In the study, a 90 nm FPGA, the Altera Stratix II, was compared to an ASIC created using an STMicroelectronics 90 nm process. The approach used was an empirical one (much like the architectural explorations described in Section 4.2) that compared the area, performance, and power consumption of multiple benchmark circuits in both technologies. As discussed in Section 4.8, modern FPGAs contain tile-based heterogeneous structures and the Altera Stratix II, in particular, has multiplier/accumulator blocks (DSP blocks), and a number of different memory blocks. Since not all benchmarks make use of these heterogeneous features, the benchmarks were categorized according to the resources they used.

The results of this comparison are summarized in Table 7.1. The table lists the geometric average of the ratio of the FPGA measurement to the ASIC measurement across all benchmark circuits for specific metrics. Each row indicates the particular metric being compared: area consumed, critical path delay, and dynamic power. The results in each row are broken down, in each column, into the results for each category of benchmarks: pure soft logic (labeled “Soft Logic Only”), soft logic with DSP arithmetic computations (Soft Logic & DSP), soft logic with memory blocks (Soft Logic & Memory), and soft logic mixed with DSP and memory blocks (Soft Logic, Memory & DSP). For example, the

Table 7.1 The FPGA:ASIC gap from [120].

Metric	Soft logic only	Soft logic & DSP	Soft logic & memory	Soft logic, DSP & memory
Area	35	25	33	18
Delay	3.4	3.5	3.5	3.0
Dynamic power	14	12	14	7.1

"Soft Logic Only" column summarizes the averages for circuits that only used LUTs and flip–flops.

7.1.1 Area Gap

As shown in Table 7.1, the average ratio of area consumed by a 90 nm FPGA using just soft logic and a 90 nm ASIC is 35. This is clearly a significant difference that severely limits the size of circuits that can be handled on a modern FPGA. This large area gap is also one of the primary contributors to the higher cost of FPGAs relative to ASICs and directly affects the volumes at which FPGAs are no longer price competitive. The high profit margins, typically higher than 60%, that FPGA vendors are able to command also has a strong impact on the relative price of FPGAs vs. ASICs.

Fortunately, the hard logic blocks that are now employed in FPGAs reduce this area gap as shown in the last three columns of Table 7.1. The benchmarks that make use of the DSP blocks and soft logic (column 3 of the table) are only 25 times larger on average when implemented in an FPGA. When both memory and the DSP blocks were used (column 5 of the table), the average of 18 times is even lower than when just DSP blocks are used.

It is important to recognize that these measurements are somewhat optimistic because only the hard blocks that are used are included in the FPGA area measurement, rather than some proportion of unused blocks. As described in Section 4, real design implementations are forced to tolerate a fixed ratio of soft logic to memory to DSP blocks. In addition, the results generated using the heterogeneous blocks heavily depend on the nature of the specific circuits used in the comparison and, in particular, the number and size of the hard blocks that are required by each circuit. As more hard blocks are used, the ratio of FPGA to ASIC area will be reduced because the implementation area of the hard blocks is similar to that of an ASIC implementation, assuming that the logic dominates the area and not the routing. With full utilization of all the hard blocks available on the Stratix II, the area gap could potentially shrink to as low as 4.7 times, a lower bound described in [120].

7.1.2 Speed Gap

The average delay gap of 3.4 times for the soft logic case is not as large as the area gap. It is noteworthy that the DSP blocks, which dramatically improved the FPGA vs. ASIC area gap have little benefit on the FPGA vs. ASIC delay gap. This observation goes against a commonly held belief that the primary benefit of hard blocks is speed improvement. The reasons cited for this effect are

(1) A hard block may only speed up a portion of the critical path, with the remainder still relatively slow, so the net gain is not as dramatic as expected.
(2) A hard block may speed up some number of critical paths in the FPGA, but there may be other near-critical paths that are not sped up, and so the gain is limited.

Similarly, the use of the memory blocks in the FPGA had a negligible impact on the delay gap between FPGAs and ASICs and their main benefit also appears to be area savings. We note that [99] suggests that larger gains from hard blocks can be obtained if a full re-timing pass is done on the circuit with the hard embedded blocks in place, particularly for highly pipelined circuits. The tool flow used to generate the data reported above did not use re-timing.

7.1.3 Power Gap

Finally, the dynamic power consumption of an FPGA implementation relative to an ASIC implementation was found to be 14 times greater on average. This is a significant issue as power, rather than area or performance, is often the limiting factor for digital designs. Hard blocks were shown to offer only a slight reduction in the FPGA's power consumption relative to an ASIC's consumption.

7.2 Alternatives to FPGAs

The large area, performance, and power gap between FPGAs and ASICs prevents the use of FPGAs for some applications. To address this limitation, a range of alternatives to FPGAs exist.

7.2.1 Standard Cell ASICs

The primary alternative to FPGAs are standard-cell based ASICs [62]. The benefits of standard cell ASIC implementation relative to an FPGA are well known and, as described in Section 7.1, they include a lower price at higher volume, greater performance, and reduced power consumption. Given the large area, performance, and power advantage of ASICs it is not always necessary to use the most current, smallest-geometry process for an ASIC and, instead, a more mature, larger-geometry process can be used. However, the challenges of standard cell ASIC versus FPGA implementation include significantly higher NRE costs, a longer manufacturing time and an increasingly more complicated design process. The process of converting FPGA designs to a standard cell implementation is also not necessarily straightforward. Techniques to improve the performance of FPGAs are often not appropriate for ASIC designs. An FPGA implementation may utilize some features of a device simply because they are already available while every additional feature or gate used in an ASIC incurs a cost. This can alter the approach used for a design. Such issues potentially increase the effort required to convert an FPGA design to a standard cell ASIC design.

7.2.2 Structured ASICs

While an ASIC implementation can offer significant area, performance, and power benefits, the many difficulties that must be overcome to create an ASIC have led to the development of devices that lie in between an FPGA and an ASIC. A structured ASIC [16, 76, 154, 155, 180, 219] is a partially pre-fabricated die that is customized through additional manufacturing.

The pre-fabricated portion of the die includes an array of gates (similar to the older-generation mask-programmed gate arrays) as well as pre-placed memory, I/O, hard processor and/or DSP blocks, pre-planned power distribution and pre-routed interconnects [180]. The customization of these devices is performed during manufacturing using custom masks. A variety of different approaches for creating structured ASICs have been developed [16, 76, 154, 155, 180, 219]. Most of the masks for these approaches are fixed for all designs. These masks define the basic logic fabric. Some of these methods use one or more layers of metal for customization, while others use as little as a one via layer for customization in conjunction with SRAM programmability [76].

One factor that affects the density of the final implementation is the number of masks that are design dependent. With more masks, greater density is possible but costs and manufacturing time are increased. All of these factors are very different than the issues that must be considered when using FPGAs and, as a result, the logic architecture of structured ASICs is often very different from those found in FPGAs. It is beyond the scope of this survey to review the different logic architectures used for structured ASICs. However, it is noteworthy that some approaches use a structure that is functionally equivalent to an FPGA [16]. Such approaches simplify the process of converting FPGA designs to structured ASIC implementations.

The primary advantage of structured ASICs over FPGAs is the elimination of dedicated switches and the memory elements that configure those switches. This switch removal reduces the area, increases the performance, and reduces the power consumption of structured ASIC implementations in comparison to FPGA implementations. Structured ASICs achieve these gains without the high design costs of regular ASICs. For fabrication in $0.13\,\mu$m CMOS, a typical structured ASIC may have design costs on the order of $500k, while a full standard cell-based ASIC can easily cost $5.5M [219]. For newer processes in 90 nm and 65 nm CMOS, this difference may become larger as mask costs and design complexity increase. Since only a few layers are customized in a structured ASIC, manufacturing time is reduced in comparison to a cell-based ASIC. Unlike standard cell ASICs, there are many approaches for structured ASICs that are specifically aimed at FPGA design conversion [16, 28]. However, structured ASICs do not offer the full performance, power, and area efficiency of ASICs and, therefore, the additional costs of an ASIC implementation are not always prohibitive. In addition, since fabrication is necessary to complete the device, a structured ASIC suffers from many of the risks and delays associated with ASICs.

7.2.3 Design-Specific Testing

An alternative to creating new custom device designs that address FPGA inefficiencies is to leverage FPGA devices that are less than fully functional. In general, only a fraction of an FPGA's resources are used by an end user's circuit. Normally, FPGAs must be fully functional because every design may use different FPGA resources. However, once an end user's design enters production, a part which is less than fully functional may be sufficient as long as faulty resources are not used by the design. The 20–35 times area gap between FPGAs and ASICs increases FPGA costs both because greater silicon area is required versus ASICs and because this greater area leads to reduced yield [163]. Once the circuit to be implemented on an FPGA is fixed, most defects that occur in unused portions of the FPGA will not affect the operation of the end-user's circuit, except for catastrophic defects such as power-ground shorts. By developing custom test vectors based on the end user's circuit, it is possible to test parts which failed the test for full functionality but which may be sufficiently functional to implement a specific end-user circuit. This is the approach used for Xilinx EasyPath FPGAs and a potential methodology for this approach is described in US Patent 6,891,395 [205].

In comparison to other approaches, this design-specific testing improves manufacturing yield, and reduces the costs of FPGA testing. Note that these are only two factors that contribute to the cost difference between FPGAs and their alternatives. The underlying implementation is not changed and, therefore, the area, performance, and power will also remain unchanged. However, with the improved yield, the price per FPGA can be reduced. In comparison to other FPGA alternatives, a benefit of this approach is significantly reduced NRE costs (reportedly on the order of $75k [223]), a risk-free conversion process and a faster time to production. The conversion of a design to a design-specific FPGA involves no changes to the circuit structure and, instead, only the process of selecting parts is changed. There are a few possible approaches for part selection. The most likely approach involves testing parts for full functionality and then using partially functional parts as candidates for design-specific testing [205]. However, it is also possible that only gross functionality and design-specific testing could be performed which would reduce testing costs. Given the high gross margins on FPGAs, another possible but unlikely approach would be to perform only the testing for full functionality on the manufactured parts. The fully functional parts could then be sold as design-specific devices if the manufacturer was willing to accept reduced gross margins. In all cases, the end-user benefits from reduced per device costs without making any changes to their design.

8

Emerging Challenges and Architectures

In this survey, we have reviewed the basic foundations of FPGA logic, routing, and I/O architecture and assessed basic programming technologies. FPGA architecture has many degrees of freedom which have only been partially explored by vendors and researchers over the past 20 years. The challenges going forward, as we have suggested in previous sections, are still very great, and there is significant room for innovation and improvement in all areas of FPGAs.

With the continued improvements to process technology, new challenges arise and FPGA architectures must adapt to address these challenges while also taking advantage of the increased integration that improved process technology provides. In this section, some of these challenges and the potential approaches to address them are reviewed. We then examine new architectures that are being considered to either take advantage of the increased levels of integration or to resolve the issues that will be faced when improvements from traditional CMOS process scaling become more limited.

8.1 Technology Issues

The continued scaling of CMOS processes has made a number of transistor-level issues increasingly problematic to the point that FPGA architects must consider these problems to reduce their impact on FPGA users. Three of these issues, soft errors, process variability, and manufacturing defects, will be reviewed in this section.

8.1.1 Soft Errors

With shrinking device sizes, one increasing problem is that of soft errors or single event upsets. A soft error occurs when ionizing radiation corrupts the data stored in a circuit [31]. The error persists until new data is written. Soft errors have long been recognized as a potential problem as radiation can come from a variety of sources. Originally, radioactive impurities in the device packages were a dominant cause but more recently cosmic radiation either through direct interaction with silicon nuclei or through interaction with insulator materials have become the primary causes of soft errors [31]. Technology scaling often worsens the soft error problem since voltage scaling, and reduced node capacitances lower the charge threshold necessary to corrupt the data. As well, the greater level of integration increases the likelihood that soft errors will affect a given device.

These soft errors are a significant concern for FPGAs because they contain flip–flops in logic blocks and SRAM in heterogeneous memory tiles that can both be affected by soft errors. However, SRAM-based FPGAs also rely on SRAM bits to store the configuration of the FPGA. This latter source of soft errors is unique to FPGAs. We will briefly review the issues and mitigation techniques for both types of errors.

8.1.1.1 Configuration Memory

The SRAM cells that form the configuration memory of an SRAM-based FPGA make such FPGAs particularly vulnerable to soft errors. Any change to the configuration memory may alter the functionality of a user's circuit and such errors would persist until the FPGA is reprogrammed. FPGAs built using the flash or antifuse programming technologies described in Section 3 do not suffer from these errors [4, 10].

There are generally three classes of approaches to mitigating the impact of soft errors:

- *Circuit and technology-level* approaches aim to reduce the possibility of soft-errors through the selection of appropriate processes combined with circuit-level changes that decrease the soft error rate.
- *System-level* design techniques aim to ensure that the system can detect and recover from soft errors.
- *User design* methodologies make the implementation of a user's design on the FPGA as insensitive to soft errors as possible.

At the circuit-level a number of changes are possible that can reduce the occurrence of soft errors. Careful selection of the memory supply voltage is essential [64] but the conflicting demand for reduced leakage and enhanced gate oxide reliability limit the voltage level that can be used. Some approaches specific to configuration memories have been suggested [199, 202]. These involve the addition of metal capacitors to nodes in the memory which increases the amount of charge necessary to cause a single event upset. With more charge required, the soft error rate is reduced. Besides FPGA-specific techniques, any number of circuit-level soft error reduction techniques may be used.

System-level techniques involve detecting and recovering from errors. In recent FPGAs this can be done by configuring them to regularly verify their configuration memory by comparing the current values with the desired configuration state using cyclic redundancy checks [26]. An output from the FPGA signals when an error is detected which alerts the system that the FPGA needs to be reprogrammed. More recent approaches leverage the fact that, since resources like routing are not fully used, many configuration bits do not affect the functionality of the FPGA and, therefore, if the configuration bit is a "don't care" the error is not flagged [23, 26]. Alternatively, the use of duplication and partial reprogramming has been suggested to offer more fine-grained control of errors while also reducing the amount of time required to recover from an error [40].

Instead of changing the device architecture, soft errors can also be addressed by altering the user's design. One of the simplest, but also most expensive approaches, is to use triple modular redundancy (TMR) [23, 230] which involves replicating a design three times and using voting circuits to determine the correct output. An alternative to the complete replication of a design is to reduce the sensitivity to soft errors in the user's design by careful selection of the resources used. For example, in [87] an FPGA router that considers the number of sensitive configuration bits (i.e., the number of bits that must be set correctly for the circuit to function) was created. The router considers these bit counts in addition to delay. A 14% reduction in the number of necessary bits was achieved. Since routing is the cause of nearly 80% of the soft errors in the FPGA configuration memory [89], reducing the number of sensitive bits is clearly beneficial.

8.1.1.2 User-Visible Memory

The flip–flops and the heterogeneous memory blocks within a user's design are another potential source of soft errors. In practise, the flip–flops which, unlike SRAM bits, are not made using minimum-sized devices, are typically not as vulnerable to soft errors [26, 236]. The bits in memory blocks, however, are vulnerable due to their size.

Fortunately, using error correction on the data stored within the memory can significantly reduce the soft error rate. There is an area cost associated with the use of error correction due to the need to store additional data and the circuitry required to encode and decode the data. This cost has been partially reduced in recent FPGAs through the introduction of dedicated hard circuits to perform the memory encoding [23, 231]; however, small memories in current FPGAs and all memories in previous generations of FPGAs continue to require a soft-logic implementation of error correction blocks [26, 236]. Given this frequent need for error correction circuitry, this is another hard block which may be appropriate to include as a heterogeneous block in future FPGAs.

However, the inclusion of error correction does not eliminate soft errors, as any code is limited in the number of errors it can correct.

Some codes allow some number of errors to be detected but not corrected and, therefore, it is up to the user's circuit to handle such events. It should also be recognized that, since the memory is directly accessed by the user circuits, any error handling scheme could also be implemented in the user's design. Such schemes can be tailored to the circuit's environment and application such that mission critical applications in high radiation environments may have elaborate correction schemes while an application that can tolerate errors operating in a normal environment may be able to use only limited error correction circuitry. While FPGA vendors will strive to offer memories with soft error rates as low as possible, FPGA users will have to develop circuits (or vendors will need to supply soft IP) that can handle the level of errors that will occur in their environment.

8.1.2 IC Process Variation

Like soft errors, variability after manufacturing in transistor properties (such as V_t, oxide thickness and doping concentrations) is an issue faced by all integrated circuit designers. This variation can have a significant effect on performance and power consumption. Historically, most of this variation has occurred between dies [41, 74] and FPGA manufacturers have been able to take advantage of this variation by testing the speed of each FPGA after manufacturing and then binning each device according to its speed. Parts from faster speed bins could then be sold for greater prices. Leakage variation could partially be managed by testing power consumption and discarding parts with unacceptable leakage. However, with the continued shrinking of CMOS transistors, within-die variation has increased in significance [41, 74] and its significance will continue to increase for future technologies. This within-die variation cannot be leveraged in the same manner as the die-to-die variation; instead, operating speeds must be reduced to maintain functionality. For FPGAs it has been estimated that within-die variation in the speed of a logic element causes a speed reduction of 5.7% in 90 nm technology and this may potentially grow to 22.4% in 22 nm technology [176].

Architectural changes to FPGAs to reduce the impact of process variation have been suggested. The most straightforward is to select the logic block architecture parameters to minimize this variation and LUT size is found to be particularly important for reducing variation in either timing or leakage [216]. An alternative approach is to adaptively compensate for any variation through body-biasing [151]. The inherent regularity and reconfigurability of FPGAs makes it possible to include a characterization unit that can test each logic block in the FPGA and store an appropriate body-bias setting. Slow blocks are set to a body bias voltage that will decrease its threshold voltage thereby increasing the block's speed. Fast blocks will have their threshold voltage increased to reduce leakage power. This scheme incurs an area penalty on the order of 1%–2% while decreasing delay variability by 30% and leakage variability by 78%. For a slightly higher area penalty, variability in delay was reduced by 3.3X and in leakage by 18X [151]. This proposal appears promising but architectural questions involving the size of the adjustable blocks and the granularity of the body bias voltage levels must be thoroughly examined.

Other approaches for handling process variability in FPGAs rely on CAD-level changes. Proposals include introducing statistical static timing analysis (SSTA) to FPGA CAD tools to improve delays by avoiding the margins that are necessary for traditional static timing analysis [177, 184], testing multiple logically equivalent configurations of the FPGA to find one that is functional at the desired speed [177], generating critical paths that will be more robust in the face of variation [147] or customizing the implementation on the FPGA for the variations of each specific device [57, 111]. With the increased impact of variability that is expected in future process generations, it is likely a combination of architectural and circuit-level changes will be needed in conjunction with a number of CAD tool innovations.

8.1.3 Manufacturing Defects

As described previously, process variation can lead to reduced performance and widely varying leakage power consumption both of which threaten to reduce the yield on devices after manufacturing. However, this is not the only source of yield loss as manufacturing defects, in which a device is not functional, are in fact the main source of losses. For this reason, devices are tested after fabrication and defective parts are generally discarded. Minimizing the yield loss from defects is an ongoing concern and it may be an even greater challenge in future technologies. For FPGAs it has been predicted that the yield in 22 nm CMOS may be 25% lower than in 90 nm CMOS for the same amount of logic [48].

However, with FPGAs there is an opportunity to work around these defects and improve yields by leveraging their regular structure. A number of approaches for improving yields have been proposed and some are currently used in FPGAs [20, 50, 148]. One set of possibilities are software-only approaches in which the CAD tools configure the FPGA to avoid any detected faults [104, 121]. Such approaches have modest increased area costs but make programming each device significantly more complex. Another possibility is design-specific testing [205, 223] which was described in Section 7. That approach allows parts that are partially defective to be used by designs that do not make use of the defective resources; however, this is only useful for high volume designs for which development of a custom test program is cost effective.

The alternative to these two approaches is to add redundancy to the FPGA architecture. With redundancy, a defective FPGA can be "repaired," and the yield improved, by storing defect information in the device and using that information to adjust the programming of the device to avoid any errors. This has commonly been done in memories [114] and was also proposed for PLAs [207]. These redundant approaches can be classified as either coarse-grained or fine-grained depending on the size of the redundant regions. We will now describe these two approaches. In both cases, the approaches only aim to correct defects that are localized and there are many faults that are not tolerable such as a short between power and ground.

Coarse-grained approaches add additional rows or columns of tiles to the FPGA, with each tile containing a logic block and any adjacent interconnect. This was first suggested in [94] and was more recently evaluated in [241]. This coarse grain scheme is also used commercially [50, 148]. In these schemes, spare rows and/or columns are added to the FPGA and, in devices with defects, defective rows/columns are bypassed and the spare rows/columns are used. The overhead of this approach includes both the area for the spare rows/columns that are unused in defect-free FPGAs and the additional logic that must be added to enable the shifting. The latter factor was estimated to cause a 2% area increase for an early FPGA architecture [94] while the cost of the spare rows/columns depends on the number of spare rows/columns which is determined by the desired level of defect tolerance. Performance and power consumption are also impacted by this scheme because routing must be made longer to allow connections to span a defective row or column. This has been estimated to only cause a 5% increase to interconnect delays [94] but this has not been reevaluated for recent architectures. One new challenge for this defect tolerance scheme is the heterogeneous blocks that are commonly included in modern FPGAs as they add complexity to these coarse-grained redundancy approaches and, with few exceptions [172], such issues have yet to be examined.

An alternative to the coarse-grained approach of adding complete rows or columns of logic is a more fine-grained approach of adding additional switches throughout the FPGA interconnect [73, 240]. With another set of switches, connections can be shifted to avoid defective routing segments. The area and delay overheads of this approach on a modern architecture have been estimated to be 25%–40% and 15%–25%, respectively [240]. This approach is particularly interesting because in defect-free devices the additional switches can be used to provide additional flexibility and potentially improved performance which lowers the effective overheads of such schemes. However, the fine-grained nature of this scheme also necessitates the storage of a detailed defect map and the costs associated with storing that information have not been completely evaluated.

Both the fine-grained and the coarse-grained defect tolerance approaches have their advantages. Coarse-grained designs can handle defects in logic and routing while, to date, none of the fine-grained approaches can handle errors in the logic block. However, the fine-grained approach offers increased interconnect defect tolerance. A detailed comparison of these schemes was performed in [241].

Given these differing capabilities, it is likely that, as suggested in [241], hybrid architectures using both fine-grained and coarse-grained techniques may be advantageous in the future. The approaches that are used will depend on a number of factors including the defect rate, whether defects occur in clusters, the architecture to which the redundant structures are added and FPGA vendors' willingness to sacrifice performance for yield.

8.2 Emerging Architectures

While the continued scaling of CMOS gives rise to problems described previously, the increased integration both allows and forces architects to consider new architectures for FPGAs. In this section, we examine alternative architectures for FPGAs that are enabled by or necessitated by the continued improvements in process technology, along with new ideas and directions that have recently been proposed to simply achieve better performance, computational density, and power consumption.

8.2.1 Coarse-Grained FPGAs

As FPGAs grow increasingly large and more functionality can be implemented on a single chip, a move away from single bit operations, as are common in current FPGAs, to architectures that operate on coarser multi-bit data has been frequently proposed as one way to further improve FPGA performance and efficiency. An early step in this direction was the bus-based routing and logic blocks [58, 59, 238] described in Section 5.3.2. While that architecture operated at least partially on multi-bit buses of data, the underlying LUT-based logic element was left unchanged. Since then, a number of alternative architectures have been developed that propose a shift away from such LUT-based logic, which is inherently fine-grained, to logic units (including both raw ALUs and more complex instruction-set processors) that operate on coarse-grained data.

The rationale for this switch to coarse-grained blocks is that many computing tasks operate on multi-bit data such as integers and floating-point numbers. Using circuit structures dedicated to processing such coarse data may make the implementations faster and more area efficient. However, with the use of such alternative logic, a number of architectural questions must be considered including the selection of the logic block and the routing topology to complement that structure. A full review of these architectural issues is beyond the scope of this survey and, instead, we will only briefly review the core architectural questions.

As described in Section 4, there is a wide spectrum of possible fine-grained logic blocks. For coarse-grained blocks, the same architectural questions must be considered including whether to use a homogeneous or heterogeneous mixture of blocks and the specific functionality to include in the logic block.

A number of possibilities have been suggested ranging from a heterogeneous mixture of logic blocks based on multi-bit functional units including multipliers, memories, and arithmetic logic units (ALUs) [77] to homogeneous mixtures of full processors [192, 206]. The original developments in this area favored logic blocks with relatively simple functional unit blocks. This included, PipeRench, an architecture based on 8-bit 3-LUT logic blocks with additional circuitry to improve arithmetic operations [86, 175], CHESS, which was based on a 4-bit ALU [145], RaPiD, which contained a mixture of 16-bit ALUs, multipliers and memories [77] and DAPDNA-2, which combined 32-bit ALUs, delay elements, memories, and external memory access units [173].

In recent years, there have been a number of proposals for logic blocks that are complete processors. Examples include the RAW architecture which consists of logic blocks which are a 32-bit RISC processor, a floating point unit and instruction and data caches [192], the TILE processor which uses a very long instruction word (VLIW) processor as its logic block [206] and the Ambric architecture based on 32-bit RISC processors [47]. It is interesting to note that this is similar to the direction of general purpose processors [201], but these new architectures retain the flavor of FPGA-style interconnect. Currently, only hundreds of processor blocks can be integrated on a single device unlike the tens of thousands of logic blocks found on current FPGAs but with increased integration this will expand quickly. These coarse-grained architectures also appear to be moving away from heterogeneity as many of the recent designs are largely homogeneous [47, 175, 192, 206].

The diversity in the range of logic blocks has led to a wide range of approaches for interconnecting these blocks. The initial ALU based architectures [77, 145] used segmented routing interconnect that was very similar to the standard FPGA interconnect described in Section 5 except a granularity that matches the logic block is used. Processor-based devices have adopted interconnect networks based on dedicated processors controlling the routing crossbars [192, 206], circuit switched connections between flow-controlled channels [46] (which is somewhat like FPGA bus-based routing) or dynamic (i.e., packet-based) switches [102, 192, 206].

Clearly, coarse-grained architectures continue to develop and there have been a number of proposals besides those described here. This area is the focus of many commercial developments but it is not yet clear which, if any, of these coarse-grained logic block and interconnect architectures will achieve wide adoption because there have been only limited performance analyses of these architectures. This lack of performance comparisons may largely be due to the immaturity (and the resulting limited availability) of CAD tools targeting these platforms. Until high quality CAD tools (and instruction set compilers) that can effectively map benchmarks to these different architectures are developed, direct comparisons may be limited.

It is also interesting to consider that these coarser grained structures key advantage may simply be in the programming model that they present to the user, and that it is possible to map that same model on to a more traditional FPGA, and thereby regain the economic advantages of a “single” universal architecture. It could also be true that if a particular model works well, then its more closely allied architecture could well be the most efficient implementation.

8.2.2 Asynchronous FPGAs

Another alternative approach that has been proposed to enable further performance improvements in FPGAs is the use of asynchronous design elements. Conventionally, it has been a standard practise for digital circuits to be designed for synchronous operation with global clocks overseeing the circuit's operation. Accordingly, FPGA architecture, including the work described in this survey, has focused primarily on implementing synchronous circuits. Asynchronous design has been proposed to improve the energy efficiency of FPGAs since asynchronous designs offer potentially lower energy as energy is only consumed when necessary. They also have the potential to increase throughput since circuits operate based on their actual performance and not a theoretical worst case (which is particularly important given the increased impact of process variations described previously). Finally, asynchronous architectures may simplify the design process as complex clock distribution networks become unnecessary. Given these potential benefits, a number of proposals to use FPGAs to implement asynchronous designs have been suggested.

Architectural changes are generally necessary because modern synchronous-focused FPGAs lack many features needed by asynchronous circuits. In particular, FPGAs are not designed to avoid glitches but any glitch can lead to incorrect operation of asynchronous designs. As well, asynchronous circuits often rely on a specific ordering of signals to ensure that a data ready signal only occurs after the data signals have been set. However, the significant and variable routing delays of the signals within an FPGA can make it difficult to satisfy such constraints. Despite these issues, there have been attempts to implement asynchronous designs on synchronous FPGAs [100]; however, such approaches have been found to incur significant area and performance penalties [193]. Therefore, to enable more efficient asynchronous implementations, new FPGA architectures that incorporate asynchronous elements directly have been proposed [95, 110, 158, 193, 194, 215]. We will briefly review these architectures and then summarize the architectural issues facing asynchronous FPGA designs.

8.2.2.1 Asynchronous Architectures

The first asynchronous FPGA was developed by Hauck et al. [95]. It consisted of a modified version of a previously developed synchronous

FPGA architecture. The logic block resembled a conventional logic block with two significant changes: first, that the logic block provides for a fast feedback path to facilitate the implementation of asynchronous state elements. Second, the latch within the logic block can be used to initialize an asynchronous circuit. As in a regular (synchronous) FPGA, the latch can also be connected to a clock signal. Besides the regular logic block, [95] also introduces an arbiter block to resolve asynchronous contention that is added as a heterogeneous element to the array of tiles.

Teifel et al. [193, 194] proposed alternative asynchronous architectures designed for dataflow applications. The basic logic block consists of LUTs, similar to the synchronous FPGAs described previously, along with units useful for asynchronous computation such as a split unit which enables conditional forwarding of data and a merge unit that allows for the conditional selection of data from different sources. Unlike the other asynchronous architectures, all the data in the routing is transferred using asynchronous techniques. Starting from that base architecture, a higher performing architecture is developed by using many of the same approaches used in synchronous designs including increasing the LUT size, improving the carry logic and clustering groups of functional units. This approach is in commercial development [63].

An alternative approach for asynchronous FPGAs was suggested by Payne [157, 158]. This architecture is designed for a "bundled-data" approach that splits the control path from the datapath [158]. Two distinct types of cells, that are always paired, are used: a timing cell that implements the control-path using asynchronous signaling and a data cell that uses same structures as a synchronous FPGA. A similar approach was also suggested in [115]; however, that design opted for a more elaborate control path unit.

An alternative to fully asynchronous design is a globally asynchronous, locally synchronous (GALS) approach. Such an approach has been suggested for use in FPGAs by both Royal et al. [171] and Jia et al. [110]. Both designs involve introducing a level of hierarchy into the FPGA fabric. Standard hard or soft synchronous logic blocks are grouped together to form large synchronous blocks. Communication between these blocks is done asynchronously. As described in Section 5.1.1, the hard boundaries introduced by such hierarchy can be problematic given the wide range of designs FPGAs must accommodate. Jia and Vemuri [110] attempts to address this issue by allowing for some flexibility in the use of these asynchronous boundaries.

8.2.2.2 Asynchronous Architecture Issues

It is clear that, despite each architecture offering its own purported benefits, a number of architectural questions remain unresolved for asynchronous FPGAs. Many architectures rely on logic blocks similar to those used for synchronous designs [95, 110, 158, 171] and, therefore, the same architectural issues such as LUT size, cluster size, and routing topology must be investigated. In addition to those questions, asynchronous FPGAs also add the challenge of determining the appropriate synchronization methodology such as a bundled data approach [158] or an approach with only delay insensitive elements [193].

These questions remain unanswered in part because CAD software remains comparatively immature. The current trend appears to be toward developing software that hides the asynchronous design details [63, 215]. As was noted in the discussion of coarse-grained architectures, well-developed CAD tools will be necessary to enable the rigorous experimental methodologies, as described in Section 4.2, that are used to evaluate current FPGA architectures.

8.2.3 Nanotechnology Architectures

While the increased integration of CMOS devices, as described by Moore's Law, has greatly expanded the capabilities of FPGAs, significant challenges must be addressed for this trend to continue [108] and eventually fundamental physical limits will prevent further scaling of standard CMOS devices. New alternative or complementary approaches to CMOS may enable further increases in integration and performance improvements for FPGAs. Such technologies, which feature devices with sizes on the order of 10 nm and lower, are generally referred to as nanotechnology.

Unlike the lithographic-based CMOS process currently used to create integrated circuits, nanotechnology devices will likely be fabricated with a completely different bottom-up approach that can only create highly regular structures which will likely have high defect rates [85]. (Interestingly, this combination of a regular fabric that most be programmed to avoid defects may force more devices to adopt FPGA-like approaches.) A review of the fabrication techniques that may be used is beyond the scope of this survey. Interested readers are referred to [149, 187]. Instead we will review the architectures that will be used to leverage the strengths of these nanotechnologies.

A number of FPGA architectures based on nanoscale technologies have been proposed. As the underlying nanotechnologies have some limitations compared to lithographically designed CMOS, all the architectures are similar in that they assume a combination of CMOS and nanodevices will be necessary. The architectures differ in the division of capabilities between CMOS and the nanodevices.

One approach is to continue to use CMOS for implementing the LUT-based logic blocks while using nanodevices for the interconnect [75, 83]. This can enable increased integration because the routing consumes a significant portion of the silicon area. With all the routing implemented using nanowires and the switches implemented using molecular switches, it is found that a 30% reduction in area and a 32% improvement in average delay is possible compared to 22 nm CMOS technology [83]. With a different style of nanotechnology to implement the interconnect, smaller performance gains of 17.5% were obtained [75]. These relatively modest improvements suggest more aggressive use of the nanotechnology is necessary to achieve greater density improvements.

An alternative architecture, called the CMOL FPGA, shifts more of the functionality to the nanodevices [190, 191]. Implementing LUT-based logic with these nanodevices is not efficient [190] and, therefore, a PLA-based approach is used. (PLAs and some of their architectural issues were described in Sections 2 and 4.6.) The architecture consists of an array of CMOS tiles containing either inverters connected through pass transistors to a grid of metal interconnects or a flip–flop for storage. A grid of nanodevices lies above the CMOS fabric and is connected at specific points. Each nanodevice can be programmed to be either on (low resistance) or off (high resistance). When these programmable switches are connected together with a pull-down implemented using CMOS, a wired-OR is created. Connecting the wired-OR to the CMOS inverter restores the voltage levels and also makes the design functionally complete. This approach yielded improvements in density of on average 110 times compared to 45 nm CMOS [191]. However, the speed estimates indicate the design is considerably (5X) slower than the CMOS implementation [191].

Finally, an architecture known as the nanoPLA that moves almost all of the functionality to the nanodevices has been proposed [69, 70, 71]. Again logic is implemented using a NOR-based PLA; however, nanoscale FET devices are used to restore signal voltages. This obviates the need for CMOS logic after every logic stage. As well, it is possible to implement registers without using lithographic CMOS gates. A large device is created by replicating a single nanoPLA tile that contains the PLA planes, internal signal feedback paths, and buffering with selective inversion. This closely resembles the island-style FPGAs described earlier and therefore, the interconnection of these blocks faces many of the same routing architecture questions described earlier. However, with this nanoscale interconnect, those questions will need to be revisited. CMOS logic is used for inputs and outputs to the nanoPLA fabric as well as for programming the fabric. Compared to 22 nm CMOS, an improvement in density of one to two orders of magnitude appears possible [70]; however, improving the delay performance appears to be an unresolved challenge.

The diversity in the nanoscale architectures described is a result of the relative immaturity of the underlying nanotechnologies. As these nanoscale devices continue to evolve and it becomes more certain which approaches are manufacturable, the architectures will change to take advantage of the capabilities of these nanodevices. One significant challenge that requires further study is the handling of the defects that will likely be present in these future nanodevices. While these architectures [70, 83, 190] considered some of these issues, full solutions that incorporate both CAD and architecture will need to be developed.

8.3 Conclusion

This survey has explored many issues in the complex and rapidly evolving world of pre-fabricated FPGA architectures. While these devices have changed dramatically in last two decades, it is clear that many fundamental questions remain, driven by rapid changes in technology and applications.

References

[1] Actel Corporation, "ACT 1 series FPGAs," http://www.actel.com/documents/ACT1_DS.pdf, April 1996.

[2] Actel Corporation, "Axcelerator family FPGAs," http://www.actel.com/documents/AX_DS.pdf, May 2005.

[3] Actel Corporation, "ProASIC3 flash family FPGAs," http://www.actel.com/documents/PA3_DS.pdf, October 2005.

[4] Actel Corporation, "Single-event effects in FPGAs," http://www.actel.com/documents/FirmErrorPIB.pdf, 2007.

[5] Actel Corporation, "SX-A family FPGAs v5.3," http://www.actel.com/documents/SXA_DS.pdf, February 2007.

[6] A. Aggarwal and D. Lewis, "Routing architectures for hierarchical field-programmable gate arrays," in *IEEE International Conference on Computer Design*, pp. 475–478, October 1994.

[7] E. Ahmed, *The Effect of Logic Block Granularity on Deep-Submicron FPGA Performance and Density.* Master's thesis, University of Toronto, Department of Electrical and Computer Engineering, 2001.

[8] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, pp. 3–12, ACM Press, 2000.

[9] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 3, pp. 288–298, March 2004.

[10] G. R. Allen and G. M. Swift, “Single event effects test results for advanced field programmable gate arrays,” in *2006 IEEE Radiation Effects Data Workshop*, pp. 115–120, 2006.

[11] Altera Corporation, “Classic EPLD family data sheet. A-DS-CLASSIC-05,” http://www.altera.com/literature/ds/classic.pdf, May 1999.

[12] Altera Corporation, “APEX II programmable logic device family, DS-APEXII-3.0,” http://www.altera.com/literature/ds/ds_ap2.pdf, August 2002.

[13] Altera Corporation, “Excalibur device overview DS-EXCARM-2.0,” May 2002.

[14] Altera Corporation, “FLEX 10K embedded programmable logic device family, DS-F10K-4.2,” http://www.altera.com/literature/ds/dsf10k.pdf, January 2003.

[15] Altera Corporation, “APEX 20K programmable logic device family data sheet, DS-APEX20K-5.1,” http://www.altera.com/literature/ds/apex.pdf, March 2004.

[16] Altera Corporation, “HardCopy structured ASICs,” http://www.altera.com/products/devices/hardcopy/hrd-index.html, 2005.

[17] Altera Corporation, “MAX II device handbook,” http://www.altera.com/literature/hb/max2/max2_mii5v1.pdf, June 2005.

[18] Altera Corporation, “Stratix II device handbook SII5V1-3.1,” http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf, July 2005.

[19] Altera Corporation, “Stratix II GX device handbook SIIGX5V1-1.1,” http://www.altera.com/literature/hb/stx2gx/stxiigx_handbook.pdf, October 2005.

[20] Altera Corporation, “Altera’s strategy for delivering the benefits of the 65-nm semiconductor process WP-01002-1.1,” http://www.altera.com/literature/wp/wp-01002.pdf, September 2006.

[21] Altera Corporation, “Cyclone II device handbook, ver. CII5V1-3.0,” http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf, June 2006.

[22] Altera Corporation, “Stratix device family data sheet, Volume 1, S5V1-3.4,” http://www.altera.com/literature/hb/stx/stratix_vol_1.pdf, January 2006.

[23] Altera Corporation, “Stratix III device handbook, ver 1.0,” http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf, November 2006.

[24] Altera Corporation, “Cyclone device handbook. C5V1-2.1, ver. C5V1-2.1,” http://www.altera.com/literature/hb/cyc/cyc_c5v1.pdf, January 2007.

[25] Altera Corporation, “Cyclone III device handbook, ver. CIII5V1-1.2,” http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf, September 2007.

[26] Altera Corporation, “Robust SEU mitigation with Stratix III FPGAs, WP-01012-1.0,” http://www.altera.com/literature/wp/wp-01012.pdf, October 2007.

[27] Altera Corporation, “Stratix III FPGAs vs. Xilinx Virtex-5 devices: Architecture and performance comparison, Altera White Paper WP-01007-2.1,” http://www.altera.com/literature/wp/wp-01007.pdf, October 2007.

[28] AMI Semiconductor, “Structured digital products,” http://www.amis.com/pdf/structured_digital_brochure.pdf, September 2003.

[29] J. Anderson and F. Najm, “A novel low-power FPGA routing switch,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 719–722, October 2004.

[30] H. B. Bakaglu, *Circuits, Interconnection, and Packaging for VLSI*. Reading, MA: Addison Wesley, 1990.

[31] R. Baumann, “Soft errors in advanced computer systems,” *IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[32] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, “Embedded floating-point units in FPGAs,” in *FPGA'06: Proceedings of the Internation Symposium on Field Programmable Gate Arrays*, pp. 12–20, USA, New York, NY: ACM Press, 2006.

[33] V. Betz and J. Rose, “Improving FPGA performance via the use of architecture families,” in *3rd ACM Intl. Symposium on Field-Programmable Gate Arrays*, pp. 10–16, 1995.

[34] V. Betz and J. Rose, “How much logic should go in an FPGA logic block?,” *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 10–15, January–March 1998.

[35] V. Betz and J. Rose, “Circuit design, transistor sizing and wire layout of FPGA interconnect,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 171–174, 1999.

[36] V. Betz and J. Rose, “FPGA routing architecture: Segmentation and buffering to optimize speed and density,” in *Proceeding: ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 140–149, February 1999.

[37] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[38] 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 antifuse programmable elements,” *Microelectronics Journal*, vol. 23, no. 7, pp. 561–568, November 1992.

[39] J. M. Birkner and H. T. Chua, “Programmable array logic circuit,” U.S. Patent number 4124899, Filed May 23, 1977, November 1978.

[40] C. Bolchini, D. Quarta, and M. D. Santambrogio, “SEU mitigation for SRAM-based FPGAs through dynamic partial reconfiguration,” in *GLSVLSI '07: Proceedings of the 17th Great Lakes Symposium on VLSI*, pp. 55–60, USA, New York, NY: ACM Press, 2007.

[41] K. A. Bowman, S. G. Duvall, and J. D. Meindl, “Impact of die-to-die and within-die parameter fluctuations on themaximum clock frequency distribution for gigascale integration,” *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, 2002.

[42] S. Brown, M. Khellah, and Z. Vranesic, “Minimizing FPGA interconnect delays,” *IEEE Design and Test of Computers*, vol. 13, no. 4, pp. 16–23, Winter 1996.

[43] S. Brown and J. Rose, “FPGA and CPLD architectures: A tutorial,” *IEEE Design and Test of Computers*, vol. 12, no. 2, pp. 42–57, Summer 1996.

[44] S. Brown, J. Rose, and Z. Vranesic, “A stochastic model to predict the routability of field-programmable gate arrays,” *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, vol. 12, no. 12, pp. 1827–1838, December 1993.

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

[46] M. Butts, “Synchronization through communication in a massively parallel processor array,” *IEEE Micro*, vol. 27, no. 5, pp. 32–40, September–October 2007.

[47] M. Butts, A. M. Jones, and P. Wasson, “A structural object programming model, architecture, chip and tools for reconfigurable computing,” in *Field-Programmable Custom Computing Machines, 2007. 15th Annual IEEE Symposium on. FCCM 2007*, pp. 55–64, April 2007.

[48] N. Campregher, P. Y. K. Cheung, G. A. Constantinides, and M. Vasilko, “Analysis of yield loss due to random photolithographic defects in the interconnect structure of FPGAs,” in *FPGA ’05: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, pp. 138–148, USA, New York, NY: ACM Press, 2005.

[49] 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 reconfiguration gate array,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 233–235, May 1986.

[50] M. Chan, P. Leventis, D. Lewis, K. Zaveri, H. M. Yi, and C. Lane, “Redundancy structures and methods in a programmable logic device, US Patent 7,180,324,” February 2007.

[51] Y.-W. Chang, J.-M. Lin, and M. D. F. Wong, “Matching-based algorithm for FPGA channel segmentation design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 6, pp. 784–791, June 2001.

[52] Y.-W. Chang, D. F. Wong, and C. K. Wong, “Universal switch-module design for symmetric-array-based FPGAs,” in *Proceedings of the 1996 ACM Fourth International Symposium on Field-Programmable Gate Arrays*, pp. 80–86, February 1996.

[53] Y.-W. Chang, D. F. Wong, and C. K. Wong, “Universal switch modules for FPGA design,” *ACM Transactions Design Automation Electronic Systems*, vol. 1, no. 1, pp. 80–101, 1996.

[54] D. Chen, J. Cong, and P. Pan, “FPGA design automation: A survey,” *Foundations and Trends in Electronic Design Automation*, vol. 1, no. 3, September 2006.

[55] J. Chen, S. Eltoukhy, S. Yen, R. Wang, F. Issaq, G. Bakker, J. L. Yeh, E. Poon, D. Liu, and E. Hamdy, “A modular 0.8 um technology for high performance dielectric antifuse field programmable gate arrays,” in *Proceedings of 1993 International Symposium on VLSI Technology, Systems and Applications*, pp. 160–164, 1993.

[56] L. Cheng, F. Li, Y. Lin, P. Wong, and L. He, “Device and architecture cooptimization for FPGA power reduction,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 7, pp. 1211–1221, July 2007.

[57] L. Cheng, J. Xiong, L. He, and M. Hutton, "FPGA performance optimization via chipwise placement considering process variations," in *International Conference on Field-Programmable Logic and Applications,* vol. 6, pp. 44–49, 2006.

[58] D. Cherepacha and D. Lewis, "A datapath oriented architecture for FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 1994.

[59] D. Cherepacha and D. Lewis, "DP-FPGA: An FPGA architecture optimized for datapaths," in *VLSI Design*, pp. 329–343, 1996.

[60] S. Cheung, K. K. Chua, B. J. Ang, T. P. Chong, W. L. Goay, W. Y. Koay, S. W. Kuan, C. P. Lim, J. S. Oon, T. T. See, C. Sung, K. P. Tan, Y. F. Tan, and C. K. Wong, "A million gate PLD with 622 MHz I/O interface, multiple PLLs and high performance embedded CAM," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 143–146, May 2000.

[61] S. Chiang, R. Forouhi, W. Chen, F. Hawley, D. McCollum, E. Hamdy, and C. Hu, "Antifuse structure comparison for field programmable gate arrays," in *International Electron Devices Meeting 1992 Technical Digest*, pp. 611–614, December 1992.

[62] D. Chinnery and K. Keutzer, *Closing the Gap Between ASIC and Custom Tools and Techniques for High-Performance ASIC Design.* Kluwer Academic Publishers, 2002.

[63] P. Clarke, "CEO Interview: John Lofton Hold of Achronix," EE Times, http://www.eetimes.com/news/semi/showArticle.jhtml?articleID=187002064, May 2006.

[64] N. Cohen, T. S. Sriram, N. Leland, D. Moyer, S. Butler, and R. Flatley, "Soft error considerations for deep-submicron CMOS circuit applications," in *Electron Devices Meeting, 1999. IEDM Technical Digest. International*, pp. 315–318, 1999.

[65] J. Cong, H. Huang, and X. Yuan, "Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs," *TODAES*, vol. 10, pp. 3–23, January 2005.

[66] J. Cong and S. Xu, "Delay-optimal technology mapping for FPGAs with heterogeneous LUTs," in *Design Automation Conference, 1998. Proceedings*, pp. 704–707, 1998.

[67] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proc. ACM International Symposium on FPGA*, pp. 179–188, Monterey, California, February 1998.

[68] R. Cuppens, C. D. Hartgring, J. F. Verwey, H. L. Peek, F. A. H. Vollebraft, E. G. M. Devens, and I. A. Sens, "An EEPROM for microprocessors and custom logic," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 603–608, April 1985.

[69] A. DeHon, "Design of programmable interconnect for sublithographic programmable logic arrays," in *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, pp. 127–137, USA, New York, NY: ACM Press, 2005.

[70] A. Dehon, "Nanowire-based programmable architectures," *Journal on Emerging Technologies in Computing Systems*, vol. 1, no. 2, pp. 109–162, 2005.

[71] A. DeHon and M. J. Wilson, "Nanowire-based sublithographic programmable logic arrays," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, pp. 123–132, USA, New York, NY: ACM Press, 2004.

[72] W. E. Donath, "Wire length distribution for placements of computer logic," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 152–155, May 1981.

[73] A. Doumar and H. Ito, "Design of switching blocks tolerating defects/faults in FPGA interconnection resources," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2000*, pp. 134–142, 2000.

[74] S. Duvall, "Statistical circuit modeling and optimization," in *2000 5th International Workshop on Statistical Metrology*, pp. 56–63, 2000.

[75] S. Eachempati, A. Nieuwoudt, A. Gayasen, N. Vijaykrishnan, and Y. Massoud, "Assessing carbon nanotube bundle interconnect for future FPGA architectures," in *DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 307–312, USA, San Jose, CA: EDA Consortium, 2007.

[76] eASIC Corporation, "nextreme Structured ASIC," http://www.easic.com/pdf/asic/nextreme_asic_structured_asic.pdf.

[77] C. Ebeling, D. C. Cronquist, and P. Franklin, "RaPiD–reconfigurable pipelined datapath," in *6th International Workshop on Field-Programmable Logic and Applications*, pp. 126–135, Springer, 1996.

[78] A. El Gamal, "Two-dimensional stochastic model for interconnections in master slice integrated circuits," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 2, pp. 127–138, February 1981.

[79] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. A. El-Ayat, and A. Mohsen, "An architecture for electrically configurable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 394–398, April 1989.

[80] H. Fan, J. Liu, Y.-L. Wu, and C.-C. Cheung, "On optimal hyperuniversal and rearrangeable switch box designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 12, pp. 1637–1649, 2003.

[81] D. Frohman-Dentchkowsky, "A fully-decoded 2048-bit electrically programmable MOS ROM," in *IEEE International Solid State Circuits Conference Digest of Technical Papers*, pp. 80–81, February 1971.

[82] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "A dual-VDD low power FPGA architecture," in *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pp. 145–157, August 2004.

[83] A. Gayasen, N. Vijaykrishnan, and M. J. Irwin, "Exploring technology alternatives for nano-scale FPGA interconnects," in *DAC '05: Proceedings of the 42nd Annual Conference on Design Automation*, pp. 921–926, USA, New York, NY: ACM Press, 2005.

[84] V. George, H. Zhang, and J. Rabaey, "The design of a low energy FPGA," in *Proceedings: International Symposium on Low Power Electronics and Design*, pp. 188–193, August 1999.

[85] S. C. Goldstein, A. DeHon, and M. Butts, "Molecular electronics: Devices, systems and tools for gigagate, gigabit chips," in *Computer Aided Design,*

2002. IEEE/ACM International Conference on ICCAD 2002, vol. 00, pp. 433–440, USA, Los Alamitos, CA: IEEE Computer Society, 2002.

[86] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[87] S. Golshan and E. Bozorgzadeh, "Single-event-upset (SEU) awareness in FPGA routing," in *DAC '07: Proceedings of the 44th Annual Conference on Design Automation*, pp. 330–333, USA, New York, NY: ACM Press, 2007.

[88] K. E. Gordon and R. J. Wong, "Conducting filament of the programmed metal electrode amorphous silicon antifuse," in *International Electron Devices Meeting, 1993. Technical Digest., International*, pp. 27–30, 1993.

[89] P. Graham, M. Caffrey, J. Zimmermann, E. Johnson, P. Sundararajan, and C. Patterson, "Consequences and categories of SRAM FPGA configuration SEUs," in *Military and Aerospace Programmable Logic Devices International Conference,* Vol. 11, pp. 1–9, 2003. http://www.xilinx.com/esp/mil_aero/collateral/RadiationEffects/consequences_categories.pdf.

[90] J. Greene, E. Hamdy, and S. Beal, "Antifuse field programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1042–1056, July 1993.

[91] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, "Segmented channel routing," in *Proceedings: ACM/IEEE Design Automation Conference*, pp. 567–572, June 1990.

[92] D. C. Guterman, L. H. Rimawi, T.-L. Chiu, R. D. Halvorson, and D. J. McElroy, "An electrically alterable nonvolatile memory cell using a floating-gate structure," *IEEE Transactions on Electron Devices*, vol. 26, no. 4, pp. 576–586, April 1979.

[93] E. Hamdy, J. McCollum, S.-O. Chen, S. Chiang, S. Eltoukhy, J. Chang, T. Speers, and A. Mohsen, "Dielectric based antifuse for logic and memory IC," in *International Electron Devices Meeting Technical Digest*, pp. 786–789, December 1988.

[94] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki, "Introducing redundancy in field programmable gate arrays," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 7.1.1–7.1.4, 1993.

[95] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for implementing asynchronous circuits," *IEEE Design and Test of Computers*, vol. 11, no. 3, pp. 60–69, 1994.

[96] S. Hauck, M. M. Hosler, and T. W. Fry, "High-performance carry chains for FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 138–147, April 2000.

[97] J. He and J. Rose, "Advantages of heterogeneous logic block architectures for FPGAs," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 7.4.1–7.4.5, May 1993.

[98] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, third ed., 2003.

[99] C. H. Ho, P. H. W. Leong, W. Luk, S. J. E. Wilton, and S. Lopez-Buedo, "Virtual embedded blocks: A methodology for evaluating embedded elements in FPGAs," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 45–44, 2006.

[100] Q. T. Ho, J.-B. Rigaud, L. Fesquet, M. Renaudin, and R. Rolland, "Implementing asynchronous circuits on LUT based FPGAs," in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, pp. 36–46, London, UK: Springer-Verlag, 2002.

[101] M. Horowitz, C.-K. K. Yang, and S. Sidiropoulos, "High-speed electrical signalling: Overview and limitations," *IEEE Micro*, vol. 18, no. 1, pp. 12–24, January/February 1998.

[102] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[103] H.-C. Hsieh, W. S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 2/1–31.2/7, May 1990.

[104] W. J. Huang and E. J. McCluskey, "Column-based precompiled configuration techniques for FPGA fault tolerance," in *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2001. FCCM'01*, pp. 137–146, 2001.

[105] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *Proceedings: IEEE Symposium on Security and Privacy*, pp. 281–295, May 2007.

[106] M. Hutton, D. Lewis, B. Pedersen, J. Schleicher, R. Yuan, G. Baeckler, A. Lee, R. Saini, and H. Kim, "Fracturable FPGA logic elements," CP-01006-1.0 http://www.altera.com/literature/cp/cp-01006.pdf.

[107] M. Hutton, S. Shumarayev, V. Chan, P. Kazarian, V. Maruri, T. Ngai, J. Park, R. Patel, B. Pedersen, and J. Schleicher, "Interconnect enhancements for a high-speed PLD architecture," in *Proceedings: ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, pp. 3–10, February 2002.

[108] International Technology Roadmap for Semiconductors 2007 Edition http://www.itrs.net/reports.html, December 2007.

[109] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," in *IEEE International Conference on Field Programmable Technology (FPT06)*, pp. 1–8, Bangkok, Thailand, December 2006.

[110] X. Jia and R. Vemuri, "Studying a GALS FPGA architecture using a parameterized automatic design flow," in *ICCAD '06: Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, pp. 688–693, USA, New York, NY: ACM Press, 2006.

[111] K. Katsuki, M. Kotani, K. Kobayashi, and H. Onodera, "A yield and speed enhancement scheme under within-die variations on 90 nm LUT array," in

Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 596–599, 2005.

[112] A. Kaviani and S. Brown, “Hybrid FPGA architecture,” in *Proceedings: ACM/SIGDA International Symposium on Field-programmable Gate Arrays 1996, FPGA ’96*, pp. 1–7, Monterey, CA, February 1996.

[113] A. Kaviani and S. Brown, “The hybrid field-programmable architecture,” *IEEE Design and Test of Computers*, vol. 16, no. 2, pp. 74–83, April–June 1999.

[114] K. Kokkonen, P. Sharp, R. Albers, J. Dishaw, F. Louie, and R. Smith, “Redundancy techniques for fast static RAMs,” in *IEEE International, Solid-State Circuits Conference. Digest of Technical Papers 1981,* vol. 24, 1981.

[115] R. Konishi, H. Ito, H. Nakada, A. Nagoya, K. Oguri, N. Imlig, T. Shiozawa, M. Inamori, and K. Nagami, “PCA-1: A fully asynchronous, self-reconfigurable LSI,” in *Seventh International Symposium on Asynchronous Circuits and Systems, 2001. ASYNC 2001*, pp. 54–61, 2001.

[116] J. Kouloheris and A. El Gamal, “FPGA performance vs. cell granularity,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 61.2.1–6.2.4, May 1991.

[117] J. Kouloheris and A. El Gamal, “FPGA area versus cell granularity — lookup tables and PLA cells,” in *FPGA 92, ACM First International Workshop on Field-Programmable Gate Arrays*, pp. 9–14, February 1992.

[118] J. Kouloheris and A. El Gamal, “FPGA area versus cell granularity — PLA cells,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1992.

[119] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” in *2006 Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 21–30, USA, New York, NY: ACM Press, 2006.

[120] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[121] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, “Efficiently supporting fault-tolerance in FPGAs,” in *FPGA ’98: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, pp. 105–115, USA, New York, NY: ACM Press, 1998.

[122] J. Lamoureux and S. J. E. Wilton, “FPGA clock network architecture: Flexibility vs. area and power,” in *FPGA ’06: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, pp. 101–108, USA, New York, NY: ACM Press, 2006.

[123] B. Landman and R. Russo, “On a pin versus block relationship for partitions of a logic graph,” *IEEE Transactions on Computers*, vol. C-20, pp. 1469–1479, December 1971.

[124] Lattice Semiconductor Corporation, “LatticeECP/EC family data sheet, version 02.0,” http://www.latticesemi.com/lit/docs/datasheets/fpga/ecp_ec_datasheet.pdf, September 2005.

[125] Lattice Semiconductor Corporation, "LatticeXP family data sheet, version 03.1," http://www.latticesemi.com/lit/docs/datasheets/fpga/xp_data_sheet.pdf, September 2005.

[126] Lattice Semiconductor Corporation, "MachXO family data sheet, version 01.2," http://www.latticesemi.com/lit/docs/datasheets/cpld/machxo.pdf, November 2005.

[127] Lattice Semiconductor Corporation, "LatticeSC family data sheet DS1004 version 01.5," http://www.latticesemi.com/dynamic/view_document.cfm?document_id=19028, March 2007.

[128] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings: International Conference on Field-Programmable Technology*, pp. 41–48, December 2004.

[129] G. Lemieux and D. Lewis, "Using sparse crossbars within LUT clusters," in *Proceedings: ACM/SIGDA International Symposium on FPGAs*, pp. 59–68, February 2001.

[130] G. Lemieux and D. Lewis, "Analytical framework for switch block design," in *Proceedings: International Symposium on Field Programmable Logic and Applications*, pp. 122–131, September 2002.

[131] G. Lemieux and D. Lewis, "Circuit design of routing switches," in *Proceedings: ACM/SIGDA International Symposium on Field Programmable Gate Array*, pp. 19–28, February 2002.

[132] G. Lemieux and D. Lewis, *Design and Interconnection Networks for Programmable Logic*. Boston, MA: Kluwer Academic Publishers, 2004.

[133] P. Leventis, M. Chan, D. Lewis, B. Nouban, G. Powell, B. Vest, M. Wong, R. Xia, and J. Costello, "Cyclone: A low-cost, high-performance FPGA," in *Proceedings of the IEEE 2003 Custom Ingretated Circuits Conference*, pp. 49–52, 2003.

[134] P. Leventis, B. Vest, M. Hutton, and D. Lewis, "MAX II: A low-cost, high-performance LUT-based CPLD," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 443–446, October 2004.

[135] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, "The Stratix II logic and routing architecture," in *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, pp. 14–20, USA, New York, NY: ACM Press, 2005.

[136] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose, "The Stratix$^{\mathrm{TM}}$ routing and logic architecture," in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, pp. 12–20, ACM Press, 2003.

[137] F. Li, Y. Lin, and L. He, "Vdd programmability to reduce FPGA interconnect power," in *IEEE/ACM International Conference on Computer Aided Design*, 2004.

[138] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1712–1724, November 2005.

[139] M. Lin and A. El Gamal, "A routing fabric for monolithically stacked 3D-FPGA," in *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, pp. 3–12, USA, New York, NY: ACM Press, 2007.

[140] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 216–229, February 2007.

[141] Y. Lin, F. Li, and L. He, "Power modeling and architecture evaluation for FPGA with novel circuits for Vdd programmability," in *Proceedings: ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, pp. 199–207, February 2005.

[142] R. Lipp, R. Freeman, and T. Saxe, "A high density flash memory FPGA family," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 239–242, May 1996.

[143] K. Maitra, "Cascading switching networks of two-input flexible cells," *IEEE Transactions on Electronic Computing*, vol. EC-11, pp. 136–143, April 1962.

[144] D. Marple and L. Cooke, "An MPGA compatible FPGA architecture," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 4.2.1–4.2.4, 1992.

[145] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in *FPGA '99: Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, pp. 135–143, USA, New York, NY: ACM Press, 1999.

[146] M. I. Masud and S. Wilton, "A new switch block for segmented FPGAs," in *Proceedings: International Workshop on Field Programmable Logic and Applications*, pp. 274–281, August 1999.

[147] Y. Matsumoto, M. Hioki, T. Kawanami, T. Tsutsumi, T. Nakagawa, T. Sekigawa, and H. Koike, "Performance and yield enhancement of FPGAs with within-die variation using multiple configurations," in *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, pp. 169–177, USA, New York, NY: ACM Press, 2007.

[148] C. McClintock, A. L. Lee, and R. G. Cliff, "Redundancy circuitry for logic circuits," US Patent 6034536, March 2000.

[149] P. L. McEuen, M. S. Fuhrer, and H. Park, "Single-walled carbon nanotube electronics," *IEEE Transactions on Nanotechnology*, vol. 1, no. 1, pp. 78–85, March 2002.

[150] R. Minnick, "A survey of microcellular research," *Journal of the Association of Computing Machinery*, vol. 14, pp. 203–241, April 1967.

[151] G. Nabaa, N. Azizi, and F. N. Najm, "An adaptive FPGA architecture with process variation compensation and reduced leakage," in *Proceedings of the*

43rd Annual Conference on Design Automation, pp. 624–629, USA, New York, NY: ACM Press, 2006.

[152] T. Ngai, J. Rose, and S. Wilton, "An SRAM-programmable field-configurable memory," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 499–502, Santa Clara, CA, May 1995.

[153] E. Ochetta, P. Crotty, C. Erickson, C.-T. Huang, R. Jayaraman, R. Li, J. Linoff, L. Ngo, H. Nguyen, K. Pierce, D. Wieland, J. Zhuang, and S. Nance, "A novel predictable segmented FPGA routing architectures," in *Proceeding: ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 3–11, February 1998.

[154] T. Okamoto, T. Kimoto, and N. Maeda, "Design methodology and tools for NEC electronics' structured ASIC ISSP," in *ISPD '04: Proceedings of the 2004 International Symposium on Physical Design*, pp. 90–96, USA, New York, NY: ACM Press, 2004.

[155] C. Patel, A. Cozzie, H. Schmit, and L. Pileggi, "An architectural exploration of via patterned gate arrays," in *ISPD '03: Proceedings of the 2003 International Symposium on Physical Design*, pp. 184–189, USA, New York, NY: ACM Press, 2003.

[156] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proceedings of the IEEE*, vol. 85, no. 8, pp. 1248–1271, 1997.

[157] R. Payne, "Self-timed FPGA systems," in *Field-Programmable Logic and Applications*, pp. 21–35, Springer, Berlin/Heidelberg, 1995.

[158] R. Payne, "Asynchronous FPGA architectures," *IEEE Proceedings Computers and Digital Techniques*, vol. 143, no. 5, pp. 282–286, 1996.

[159] M. Pedram, B. Nobandegani, and B. T. Preas, "Architecture and routability analysis for row-based FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 230–235, November 1993.

[160] Plessey Semiconductor, "ERA60100 preliminary data sheet," 1989.

[161] QuickLogic Corporation, "Eclipse II family data sheet (Rev P)," http://www.quicklogic.com/images/EclipseII_Family_DS.pdf, February 2007.

[162] QuickLogic Corporation, "QuickLogic PolarPro Data Sheet (Rev H)," http://www.quicklogic.com/images/polarpro_DS.pdf, February 2007.

[163] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 1996.

[164] A. Rahman and V. Polavarapuv, "Evaluation of low-leakage design techniques for field programmable gate arrays," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 23–30, February 2004.

[165] A. Roopchansingh and J. Rose, "Nearest neighbour interconnect architecture in deep submicron FPGAs," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 59–62, San Diego, CA, May 2002.

[166] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, March 1991.

[167] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–1029, July 1993.

[168] J. Rose, R. J. Francis, P. Chow, and D. Lewis, "The effect of logic block complexity on area of programmable gate arrays," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 5.3.1–5.3.5, San Diego, May 1989.

[169] J. 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 Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, October 1990.

[170] K. Roy and M. Mehendale, "Optimization of channel segmentation for channeled architecture FPGAs," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 4.4.1–4.4.4, May 1992.

[171] A. Royal and P. Y. K. Cheung, "Globally asynchronous locally synchronous FPGA architectures," in *Field-programmable Logic and Applications: 13th International Conference, FPL 2003, Lisbon, Portugal, September 1–3, 2003: Proceedings*, Springer, 2003.

[172] R. Saini, A. Lee, and N. Ngo, "Programmable logic device having regions of non-repairable circuitry within an array of repairable circuitry and associated configuration hardware and method," US Patent 7,215,140, May 2007.

[173] T. Sato, H. Watanabe, and K. Shiba, "Implementation of dynamically reconfigurable processor DAPDNA-2," in *VLSI Design, Automation and Test, 2005.(VLSI-TSA-DAT). 2005 International Symposium on IEEE VLSI-TSA*, pp. 323–324, 2005.

[174] A. Scheibe and W. Krauss, "A two-transistor SIMOS EAROM cell," *IEEE Journal of Solid-State Circuits*, vol. 15, no. 3, pp. 353–357, June 1980.

[175] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 63–66, 2002.

[176] P. Sedcole and P. Y. K. Cheung, "Within-die delay variability in 90 nm FPGAs and beyond," in *Field Programmable Technology, 2006. IEEE International Conference on FPT 2006*, pp. 97–104, 2006.

[177] P. Sedcole and P. Y. K. Cheung, "Parametric yield in FPGAs due to within-die delay variations: A quantitative analysis," in *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, pp. 178–187, USA, New York, NY: ACM Press, 2007.

[178] L. Shang, A. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex-II FPGA family," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 157–164, February 2002.

[179] M. Sheng and J. Rose, "Mixing buffers and pass transistors in FPGA routing architectures," in *Proceedings: ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 75–84, February 2001.

[180] D. D. Sherlekar, "Design considerations for regular fabrics," in *ISPD '04: Proceedings of the 2004 International Symposium on Physical Design*, pp. 97–102, USA, New York, NY: ACM Press, 2004.

[181] C.-C. Shih, R. Lambertson, F. Hawley, F. Issaq, J. McCollum, E. H. H. Sakurai, H. Yuasa, H. Honda, T. Yamaoka, T. Wada, and C. Hu, "Characterization and modeling of a highly reliable metal- to-metal antifuse for high-performance and high-density field-programmable gate arrays," in *Proceedings of the 1997 IEEE International Reliability Physics Symposium*, pp. 25–33, 1997.

[182] A. Singh and M. Marek-Sadowska, "FPGA interconnect planning," in *Proceedings: International Workshop on System-Level Interconnect Planning*, pp. 23–30, April 2002.

[183] D. Singh and S. Brown, "The case for registered routing switches in field programmable gate arrays," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–169, February 2001.

[184] S. Sivaswamy and K. Bazargan, "Variation-aware routing for FPGAs," in *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, pp. 71–79, USA, New York, NY: ACM Press, 2007.

[185] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh, "HARP: hard-wired routing pattern FPGAs," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 21–29, February 2005.

[186] T. Speers, J. J. Wang, B. Cronquist, J. McCollum, H. Tseng, R. Katz, and I. Kleyner, "0.25 um flash memory based FPGA for space applications," in *MAPLD Conference*, Baltimore MD, 1999. www.actel.com/documents/FlashSpaceApps.pdf.

[187] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler, "Molecular electronics: From devices and interconnect to circuits and architecture," *Proceedings of the IEEE*, vol. 91, no. 11, pp. 1940–1957, 2003.

[188] Stretch Inc, "Stretch S5530 product brief, MK-5530C-0002-000," http://www.stretchinc.com/_files/Stretch_S5530_Software_Configurable_Processor.pdf, 2005.

[189] Stretch Inc, "S6100/S6105 product brief, MK-6000C-0001-000," http://www.stretchinc.com/_files/S6000.pdf, 2007.

[190] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, no. 6, pp. 888–900, 2005.

[191] D. B. Strukov and K. K. Likharev, "A reconfigurable architecture for hybrid CMOS/Nanodevice circuits," in *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, pp. 131–140, USA, New York, NY: ACM Press, 2006.

[192] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[193] J. Teifel and R. Manohar, "An asynchronous dataflow FPGA architecture," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1376–1392, 2004.

[194] J. Teifel and R. Manohar, "Highly pipelined asynchronous FPGAs," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, pp. 133–142, USA, New York, NY: ACM Press, 2004.

[195] Triscend Corporation, "Triscend A7V MKT012-0001-001," http://www.zylogic.com.cn/download/pdf/products01_2/A7VProductBrief.pdf.

[196] Triscend Corporation, "Triscend E5 configurable system-on-chip platform product description TCH300-0001-001," http://www.keil.com/dd/docs/datashts/triscend/te5xx.pdf, 2001.

[197] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, G. Varghese, J. Wawrzynek, and A. DeHon, "HSRA: High-speed, hierarchical synchronous reconfigurable array," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 125–134, February 1999.

[198] T. Tuan and B. Lai, "Leakage power analysis of a 90 nm FPGA," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 57–60, September 2003.

[199] J. E. Turner, "Programmable logic devices with stabilized configuration cells for reduced soft error rates," US Patent 6876572, April 2005.

[200] J. Tyhach, B. Wang, C. Sung, J. Huang, K. Nguyen, X. Wang, Y. Chong, P. Pan, H. Kim, G. Rangan, T.-C. Chang, and J. Tan, "A 90-nm FPGA I/O buffer design with 1.6-Gb/s data rate for source-synchronous system and 300-MHz clock rate for external memory interface," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 9, pp. 1829–1838, September 2005.

[201] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, 2008.

[202] M. L. Voogel and S. P. Young, "Memory cells utilizing metal-to-metal capacitors to reduce susceptibility to single event upsets," US Patent 7110281, September 2006.

[203] S. E. Wahlstrom, "Programmable logic arrays — cheaper by the millions," *Electronics*, vol. 40, pp. 90–95, December 1967.

[204] N. Weaver, J. Hauser, and J. Wawrzynek, "The SFRA: A corner-turn FPGA architecture," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 3–12, February 2004.

[205] R. W. Wells, Z.-M. Ling, R. D. Patrie, V. L. Tong, J. Cho, and S. Toutounchi, "Application-specific testing methods for programmable logic devices," US Patent 6,891,395, May 2005.

[206] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the Tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[207] C. Wey, M. Vai, and F. Lombardi, "On the design of a redundant programmable logic array RPLA," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 1, pp. 114–117, 1987.

[208] S. Wilton, *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories*. PhD thesis, University of Toronto, Department of Electrical and Computer Engineering, 1997.

[209] S. Wilton, J. Rose, and Z. Vranesic, "Architecture of centralized field-configurable memory," in *3rd ACM International Symposium on Field-Programmable Gate Arrays, FPGA*, pp. 97–103, 1995.

[210] S. J. E. Wilton, "Implementing logic in FPGA embedded memory arrays: Architectural implications," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1998.

[211] S. J. E. Wilton, "SMAP: Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 171–178, February 1998.

[212] S. J. E. Wilton, "Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 1, pp. 56–68, January 2000.

[213] S. J. E. Wilton, "Implementing logic in FPGA memory arrays: Heterogeneous memory architectures," in *IEEE International Conference on Field-Programmable Technology*, pp. 142–149, December 2002.

[214] S. J. E. Wilton, N. Kafafi, J. C. H. Wu, K. A. Bozman, V. Aken'Ova, and R. Saleh, "Design considerations for soft embedded programmable logic cores," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 2, pp. 485–497, February 2005.

[215] C. G. Wong, A. J. Martin, and P. Thomas, "An architecture for asynchronous FPGAs," in *Field-Programmable Technology (FPT), 2003. IEEE International Conference on Proceedings 2003*, pp. 170–177, 2003.

[216] H.-Y. Wong, L. Cheng, Y. Lin, and L. He, "FPGA device and architecture evaluation considering process variations," in *ICCAD '05: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, pp. 19–24, USA, Washington, DC: IEEE Computer Society, 2005.

[217] S. C. Wong, H. C. So, J. H. Ou, and J. Costello, "A 5000-gate CMOS EPLD with multiple logic and interconnect arrays," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 5.8.1–5.8.4, May 1989.

[218] N.-S. Woo, "Revisiting the cascade circuit in logic cells of lookup table based FPGAs," in *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 90–96, 1995.

[219] K.-C. Wu and Y.-W. Tsai, "Structured ASIC, evolution or revolution?," in *ISPD '04: Proceedings of the 2004 International Symposium on Physical Design*, pp. 103–106, USA, New York, NY: ACM Press, 2004.

[220] Y.-L. Wu and M. Marek-Sadowska, "Orthogonal greedy coupling — a new optimization approach for 2-D field-programmable gate arrays," in *Proceedings: ACM/IEEE Design Automation Conference*, pp. 568–573, June 1995.

[221] Xilinx, "Spartan-II 2.5V FPGA family complete data sheet," http://direct.xilinx.com/bvdocs/ publications/ds001.pdf.

[222] Xilinx, "Xilinx 3000 series data sheet," http://direct.xilinx.com/bvdocs/publications/3000.pdf.

[223] Xilinx, "Xilinx: EasyPath series overview," http://www.xilinx.com/products/silicon_solutions/fpgas/easypath/overview.htm.

[224] Xilinx, "Virtex 2.5V field programmable gate arrays, DS003-1(v2.5)," http://direct.xilinx.com/bvdocs/publications/ds003-1.pdf, April 2001.

[225] Xilinx, "Spartan and Spartan-XL families field programmable gate arrays. DS060 (v1.7)," http://direct.xilinx.com/bvdocs/publications/ds060.pdf, June 2002.

[226] Xilinx, "Xilinx announces acquisition of Triscend Corp.," Xilinx Press Release 0435, http://www.xilinx.com/prs_rls/xil_corp/0435_triscend_ acquisition.htm, March 2004.

[227] Xilinx, "Spartan-3 FPGA family: Complete data sheet," DS099, http://direct.xilinx.com/bvdocs/publications/ds099.pdf, August 2005.

[228] Xilinx, "Virtex-4 family overview," DS112(v1.4), http://direct.xilinx.com/bvdocs/publications/ds112.pdf, June 2005.

[229] Xilinx, "Virtex-II platform FPGAs: Complete data sheet. DS031(v3.4)," http://direct.xilinx.com/bvdocs/publications/ds031.pdf, March 2005.

[230] Xilinx, "Triple module redundancy design techniques for Virtex FPGAs," Xilinx Application Note 197, www.xilinx.com/support/documentation/application_notes/xapp197.pdf, July 2006.

[231] Xilinx, "Virtex-5 user guide," UG190 (v2.1), October 2006.

[232] Xilinx, "XC9500 in-system programmable CPLD family DS063(v5.4)," http://direct.xilinx.com/bvdocs/publications/DS063.pdf, April 2006.

[233] Xilinx, "Power consumption in 65 nm FPGAs, Xilinx White Paper WP246 (v1.2)," http://www.xilinx.com/support/documentation/white_papers/wp246.pdf, February 2007.

[234] Xilinx, "Spartan-3AN FPGA family data sheet, DS557," http://direct.xilinx.com/bvdocs/publications/ds557.pdf, February 2007.

[235] Xilinx, "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, DS083(v4.6)," http://direct.xilinx.com/bvdocs/publications/ds083.pdf, March 2007.

[236] Xilinx, "Xilinx FPGAs overcome the side effects of sub-90 nm technology, Xilinx White Paper WP256 v1.0.1," http://www.xilinx.com/support/documentation/white_papers/wp256.pdf, March 2007.

[237] A. Yan, R. Cheng, and S. J. E. Wilton, "On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques," in *FPGA '02: Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*, pp. 147–156, USA, New York, NY: ACM Press, 2002.

[238] A. Ye and J. Rose, "Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits," in *Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 3–13, February 2005.

[239] S. P. Young, T. J. Bauer, K. Chaudhary, and S. Krishnamurthy, "FPGA repeatable interconnect structure with bidirectional and unidirectional interconnect lines," US Patent 5,942,913, August 1999.

[240] A. J. Yu and G. G. Lemieux, “Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement,” in *International Conference on Field Programmable Logic and Applications, 2005*, pp. 255–262, 2005.

[241] A. J. Yu and G. G. Lemieux, “FPGA defect tolerance: Impact of granularity,” in *Field-Programmable Technology, 2005. IEEE International Conference on Proceedings 2005*, pp. 189–196, 2005.

[242] G. Zhang, E. Hu, Yu, S. Chiang, and E. Hamdy, “Metal-to-metal antifuses with very thin silicon dioxide films,” *IEEE Electron Device Letters*, vol. 15, no. 8, pp. 310–312, August 1994.

[243] C. Zhou and Y.-L. Wu, “Optimal MST-based graph algorithm on FPGA segmentation design,” in *Proceedings: International Conference on Communications, Circuits, and Systems*, pp. 1290–1294, June 2004.

[244] K. Zhu, D. F. Wong, and Y.-W. Chang, “Switch module design with application to two dimensional segmentation design,” in *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, pp. 480–485, November 1993.

[245] P. S. Zuchowski, C. B. Reynolds, R. J. Grupp, S. G. Davis, B. Cremen, and B. Troxel, “A hybrid ASIC and FPGA architecture,” in *ICCAD ’02*, pp. 187–194, November 2002.