FPGA ARCHITECTURE, CHALLENGES AND APPLICATION: SURVEY
MALLIKARJUN SARSAMBA
Assistant Professor, Department of E. & C.E. BKIT, Bhalki, India

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) have become one of the key digital circuit implementation media over the last decade. A necessary 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.

KEYWORDS: FPGAs, PAL, SRAM, etc

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 convincing advantages over fixed function Application Specific Integrated Circuit (ASIC) technologies such as standard cells [N.Cohen,1999]: 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 [I. Kuon and J. Rose,2007]. These disadvantages arise largely from an FPGA’s programmable routing fabric which trades area, speed, and power in return for “instant” fabrication. 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. 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 programmable 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 ability to program function into the chip after silicon fabrication is complete. These customizations 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. 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.
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 [R. Minnick, 1967] typically consisted of a two-dimensional array of simple logic cells with fixed, point-to-point communication. These first rays, such as the Maitracascad [K. Maitra, 1962], contained logic cells which could be programmed via metallization 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 “cut point” cellular arrays. 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 [R. Minnick, 1967]. As a result, field-customization allowed for simplified array manufacturing and wider applicability. 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). 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 macro cells.

With these devices, logic functions must be implemented using one or more levels of two-level logic structures. Device inputs and intermediate combinational sums are fed into the array via a

![Figure 1.1: Basic FPGA structure](image)

![Figure 2.1: PLA and PAL architectures](image)
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 Ahlstrom in 1967 [S. E. Wahlstrom, 1967]. 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, when the cost per transistor was sufficiently lowered.

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 PROM EEPROM flash static memory [W. Carter, 1986], andante-fuses [J. Birkner, 1992]. 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.

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, Lattice and Altera. In these devices, static memory cells, such as the 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. The first static memory-based FPGA (commonly called an SRAM based FPGA) was proposed by Ahlstrom in 1967 [Actel Corporation, 2007]. 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, when the cost per transistor was sufficiently lowered.
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.

**Anti-fuse Programming Technology**

An alternative to SRAM and floating gate-based technologies is antifuse programming technology. This technology is based on structures which exhibit very high-resistance under normal circumstances but can be programmable “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, indirectly 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 [E. Hamdy, 1988]. The applications 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 [P. L. McEuen, 2002]. 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 or silicon oxide between two metal layers. Again, a high voltage breaks down the anti-fuse and causes the fuse to conduct.

**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 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.

**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 [W. Carter, 1986] or island-style.

**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 routings 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. 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 [M. Lin and A, 2007] and routing structures. 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, 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 significant amount of the FPGA die area [E. Hamdy, 1988], 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.

**CONCLUSIONS**

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**


