Field-Programmable Gate Arrays

This year, more digital designs will be built on field-programmable gate arrays than on all traditional, mask-programmed gate arrays combined. What are FPGAs and what makes them so popular?

Like traditional gate arrays, FPGAs implement thousands of logic gates in multilevel structures. An FPGA manufacturer makes a single, standard device, like a programmable logic device (PLD), that users program to carry out desired functions. Field programmability comes at a cost in logic density and performance: FPGA capacity trails mask-programmed gate array capacity by about a factor of 10; FPGA performance trails mask-programmed gate arrays by about a factor of three.

On the other hand, a user can program an FPGA design in a few seconds or minutes, rather than the weeks or months required for the production of mask-programmed parts. FPGAs need no custom mask tooling, saving thousands of dollars over mask-programmed parts. The result is a low-risk design style, where the price of a logic error is small, both in money and project delay. The reduced risk makes FPGAs useful for rapid product development and prototyping. Moreover, FPGAs can be fully tested after manufacture, so users' designs do not require test program generation, automatic test pattern generation, and design for testability.

Types of FPGAs

Many kinds of programmable logic products are called FPGAs. Here, we use a broad definition of the term, including not only devices with internal structure similar to gate arrays, but also devices with internal structure similar to a collection of PLDs. The term FPGA is often reserved for the former type of part; the latter are also called complex PLDs (CPLDs) or programmable multilevel devices (PMDs).

Three programming technologies are commonly used for FPGAs. Each has associated area and performance costs, and the device architectures reflect those costs. Thus, we can categorize FPGAs according to their combination of programming technology and device architecture.

CPLDs. In a CPLD architecture (Figure 1 on p. 4), the user creates logic and interconnections by programming EPROM (or EEPROM) transistors to form wide fan-in gates. A CPLD consists of a few function blocks, each similar to a simple two-level PLD. Each function block contains a PLD AND-array that feeds its macrocells. The AND-array consists of a number of product terms. The user programs the AND-array by turning on...
Figure 1. CPLD FPGA structure.

Figure 2. SRAM FPGA structure.

EPROM transistors that allow selected inputs to be included in a product term.

A macrocell includes an OR gate to complete the two-level AND-OR logic and may also include registers and an I/O pad. The macrocell may contain additional EPROM cells to control multiplexers that select a registered or nonregistered output and decide whether or not the macrocell result is output on the I/O pad at that location. Macrocell outputs are connected as additional function block inputs or as inputs to a global universal interconnect mechanism (UIM) that reaches all function blocks on the chip. The function blocks, macrocells, and interconnect mechanisms vary from one product to another, giving a range of device capacities and speeds.

SRAM FPGAs. In an SRAM-programmed FPGA, static memory cells hold the programming. The SRAM FPGA implements logic as lookup tables made from the memory cells, with function inputs controlling the address lines. Each lookup table of $2^n$ memory cells implements any function of $n$ inputs. One or more lookup tables, combined with flip-flops, form a configurable logic block. The CLBs are arranged in a two-dimensional array with interconnect segments in channels (Figure 2), similar to an island style gate array architecture. Interconnect segments connect to CLB pins in the channels and to other segments in the switch boxes through pass transistors controlled by configuration memory cells. Because SRAM cells and pass transistors are comparatively expensive in area and delay, the switch boxes are not full crossbar switches.

An SRAM FPGA program consists of a single long program word. On-chip circuitry loads the program word, reading it serially out of an external memory every time power is applied to the chip. The program bits set the values of all configuration memory cells on the chip, thus setting the lookup-table values and selecting which segments connect to each other. SRAM FPGAs are inherently reprogrammable. They can be updated in the system, providing designers with new design options and capabilities, such as logic updates that do not require hardware modification and timeshared virtual logic.

Antifuse FPGAs. An antifuse is a two-terminal device that, when exposed to a high voltage, forms a permanent short circuit between the nodes on either side. Individual antifuses are small, so an antifuse-based architecture can have hundreds of thousands or millions of antifuses. To simplify the architecture and programming, antifuse FPGAs (Figure 3) usually consist of rows of configurable logic elements with interconnect channels between them, much like traditional gate arrays. The pins on the logic blocks extend into the channel. A logic block is usually a comparatively simple gate-level network, which one programs by connecting its input pins to fixed values or to interconnect nets. There are antifuses at every wire-to-pin intersection point in the channel and at all wire-to-wire intersection points where channels intersect.
Design process
The FPGA design process is similar to other gate array design. Input can come from a schematic netlist, a hardware description language, or a logic synthesis system. The first step in design implementation is to fit the logic in the input into the FPGA structures. This step is similar to "technology mapping" in logic synthesis. It is called "logic partitioning" by some FPGA manufacturers, and "logic fitting" in reference to CPLD-style FPGAs.

After partitioning, the design software assigns the logic, now described in terms of functional units on the FPGA, to particular physical locations on the device and chooses the routing paths. These last two steps are similar to traditional gate array placement and routing. They may be algorithmically simpler or more difficult, depending on the amount of routing resources available on the chip, the types of interconnect available, and the design constraints.

Articles
In this issue and the next, we present articles on all three types of FPGAs, on critical parts of the FPGA design process, and on software problems in FPGA design.

In this issue, "DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization," by Kuang-Chien Chen, Jason Cong, Yuzheng Ding, Andrew Kahng, and Peter Trajmar, describes an effective solution to the high-performance technology-mapping problem for look-up-table-based FPGAs.

Reprogrammable FPGAs provide programmable logic in the same way reprogrammable microprocessors provide programmable function. David E. Van den Bout, Joseph N. Morris, Douglas Thomas, Scott Labrozzi, Scott Wingo, and Dean Hallman describe an innovative use of reprogrammable FPGAs in "Any-Board: An FPGA-Based, Reconfigurable System." The system consists of a large amount of reprogrammable logic that can be reconfigured to perform various large-scale functions.

In our December issue, Zafar Hasan, Dave Harrison, and Maciej Ciesielski will describe a partitioning and fitting algorithm for CPLD devices in "Fast Partitioning Method for PLA-Based FPGAs." This crucial step determines not only whether a design will fit into the FPGA but also the performance of the design.

To achieve better performance and density, many FPGAs include dedicated logic for special functions—for example, carry logic for counters or adders. Designers and design systems often find this logic difficult to use. In another December article, "Shortening the Design Cycle for Programmable Logic Devices," Steven H. Kelem and Jorge P. Seidel describe a method of automating the use of dedicated logic on FPGAs.

David Marple describes the architecture and software for an antifuse-based FPGA in "An MPGA like FPGA," also in the December issue. Architecturally, the antifuse FPGA is more similar to mask-programmed gate arrays than any other type of FPGA, so the article should give readers familiar with conventional gate array design a good introduction to FPGA issues.

FPGA topics are becoming more common both at circuits conferences, such as the Custom Integrated Circuits Conference (CICC) and the International Conference on Computer Design (ICCD), and at computer-aided design conferences, such as the Design Automation Conference (DAC) and the International Conference on Computer-Aided Design (ICCAD). In addition, two FPGA workshops took place this year: FPGA in Berkeley, California, where some of the work described in this issue was originally reported; and FPL (Field-Programmable Logic) in Vienna, Austria. An FPGA workshop is planned for summer 1993 in Oxford, UK. Interested parties should contact Will R. Moore, Dept. of Engineering Science, Parks Road, Oxford, OX1 3PJ, UK; or moore@vax.oxford.ac.uk.

Stephen Trimberger is manager of advanced development at Xilinx, Inc., where he is responsible for new directions in FPGA architecture and software. He was a member of the Architecture Definition Group for the XC4000 FPGA and the technical leader for the XC4000 design automation software. Previously, he was a member of the Design Technology Group at VLSI Technology, working on computer-aided logical and physical ASIC design. Trimberger received his PhD in computer science from the California Institute of Technology. He is a member of the IEEE Computer Society.

Send questions or comments to Stephen Trimberger, Xilinx, Inc., 2100 Logic Dr., San Jose, CA 95124; steve@xilinx.com.