OPTIMIZING PROGRAMMABLE GATE ARRAY DESIGNS

Steven K. Knapp
Field Applications Engineer
XILINX, Inc.
2069 Hamilton Avenue
San Jose,CA 95125

INTRODUCTION

As designers increasingly rely upon high-density programmable logic devices for higher system integration and performance, integrated circuit manufacturers must turn toward novel architectures to deliver the required capabilities, instead of relying solely on process technology. Leading this trend is the XILINX family of programmable gate arrays. The XILINX™ XC3000-Series Programmable Gate Array (PGA) family can implement a wide range of digital logic applications with system clocks up to 40 MHz. Its fast and flexible gate-array-style architecture affords the designer gate-array-style design freedom while maintaining the manufacturing and development freedom associated with programmable logic devices (PLDs). This family of high-density, CMOS devices contains many advanced architectural features including internal three-state bussing and ranges in capacity from 2,000 to 9,000 usable gates as shown in Table I.

Table I. XC3000-Series Programmable Gate Array Family

XILINX Array	Gate Capacity	Logic Blocks	User I/O	Internal Bus	Internal Flipflops
XC3020	2,000	64	64	16-bits	128
XC3030	3,000	100	80	20-bits	200
XC3042	4,200	144	96	24-bits	288
XC3064	6,400	224	120	32-bits	448
XC3090	9000	320	144	40-bits	640

However, with the advent of any new high-density logic device comes new requirements for development software. A unique architecture may make a device faster and more flexible but it also injects new uncertainties for the design engineer. How will the designer use these new architectural elements without having to learn a completely new design entry method? Ideally, a new device family should use a familiar entry method such as TTL-equivalent elements from a schematic capture library or Boolean equation entry for those designers familiar with PAL® devices. The development system software should take the various input formats and optimize them to better fit the new architecture. This becomes especially important for new, highly-integrated programmable logic devices where multiple entry methods and optimization techniques may be required since various types of functions may be reside all on the same integrated circuit.

This paper describes the XILINX 3000-Series Programmable Gate Array architecture and its implications on design entry and software optimization methods. A simple microprocessor peripheral demonstrates the flexibility of both the PGA architecture and the development system. Multiple design entry methods are used including schematic capture and Boolean equation entry. Software optimization methods are used both at the design entry level and at the physical implementation level.

THE XC3000-SERIES ARCHITECTURE

Before delving further into design entry methods, it is important to understand the XC3000 architecture. The Logic Cell™ Array (LCA) architecture (or Programmable Gate Array (PGA) as it is called generically) consists of three primary elements as shown in Figure 1. A ring of programmable *110* Blocks (IOBs) surround a core array of Configurable Logic Blocks (CLBs). Programmable interconnect joins the various block structures to implement the designer's logic.

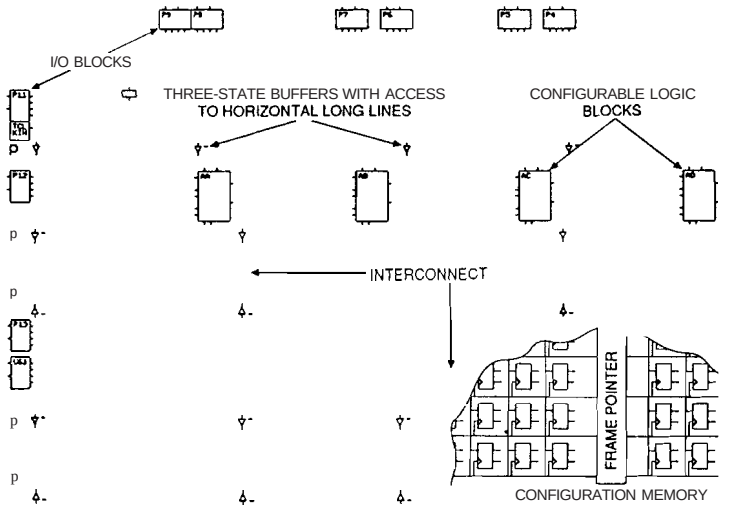

Figure1. An overview of the Programmable Gate Array (PGA) architecture showing the Configurable Logic Blocks (CLBs), the I/O Blocks (IOBs), and the programmable interconnect. An underlying distributed configuration memory array controls these elements.

Input/Output Blocks (IOBs)

Each of the programmable I/O Blocks (IOBs), shown in Figure 2, provides an interface between the external package pin of the device and the internal logic. From the logic description extracted from a schematic drawing or equation entry, a designer can program a block as an input, an output, or a bidirectional I/O pin.

As an input, an IOB supports both direct and registered inputs, simultaneously. The input register may be

programmed as either a positive-edge-triggered D-flipflop or an active-low, level-sensitive latch. The clock/latch-enable signal is common to all IOBs along an edge of the device though the source for this signal may originated from anywhere on or off the chip. In addition, the clock signal can be inverted along each edge.

Outputs may be either true or complement of the output signal and may be either combinational or registered, depending on programming. The output register is a positive-edge-triggered D-flipflop, also with an invertible output clock signal common to all the IOBs along an edge. The output buffer may be turned on, turned off, or enabled by an independent three-state control signal (also invertible).

An independent slew rate control allows each output buffer to drive at maximum frequency or, through programming, the slew rate may be limited to a more gradual switching edge. A gradual switching edge, used primarily for non-speed-eritical signals, reduces the system noise induced by switching transients. Fast switching edges are a common nuisance in high-speed CMOS designs. Since I/O pins on a CMOS device should never be left floating (because they can self-oscillate), the development system automatically defines unused IOBs as inputs and pulls them HIGH with 50kΩ to 100kΩ pull-up resistors.

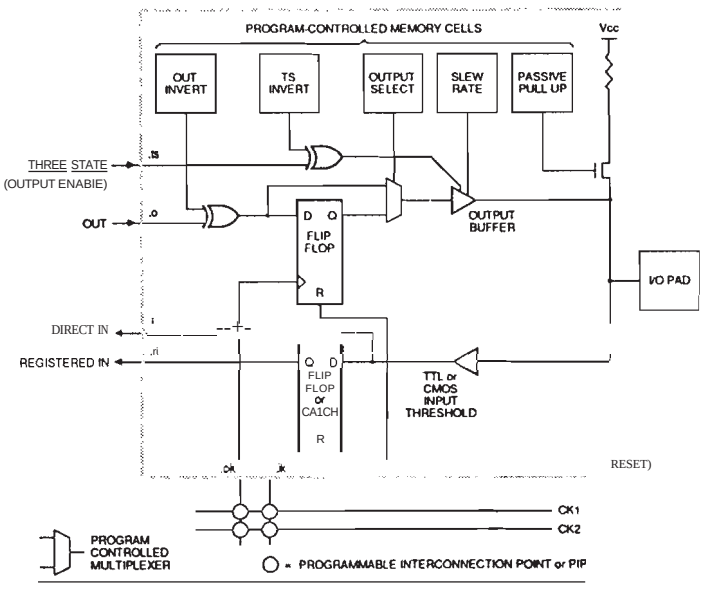

Figure 2. The Input/Output Blocks (IOB) includes input and output registers and various I/O options selected through programming.

Conflgurable Logic Blocks (CLBs)

An array of Configurable Logic Blocks (CLBs) comprise the elements used to build logic functions. Arranged in a matrix surrounded by the user I/O, each CLB contains a combinational logic section and two flipflops as shown in Figure 3. The actual configuration for a CLB is determined after a design is partitioned from schematic or equation entry.

Five inputs feed the combinational logic section which is built from a 32-by-1 look-up table. Each input acts as an address line for the look-up table. Using this approach to implement logic, each block can implement any possible function of five inputs, any two functions of up to four inputs, and some limited functions of six or seven inputs (six or seven input functions use feedback from the CLB internal flipflops).

The two D-type flipflops share a common clock input, a common asynchronous reset input, and a clock enable. The clocking structures on the device allow a wide variety of applications ranging from fully synchronous systems with a single clock to those clocking each CLB individually. Also the clock inputs to a CLB are invertible which eliminates the need to route both phases of a clock signal throughout the device. All inputs may be driven from the interconnect resources surrounding each block. "Floating" inputs are automatically tied to an appropriate value by the development system (i.e., if the clock enable pin is left floating or unused in the design, the development system sets it to a logic HIGH, or always enabled).

Data inputs to either flipflop are supplied from the combinational logic functions labeled F or G in Figure 3 or can come from the block's direct data input pin.

Each CLB has two outputs which may drive interconnect networks since a single CLB may hold two, independent functions.

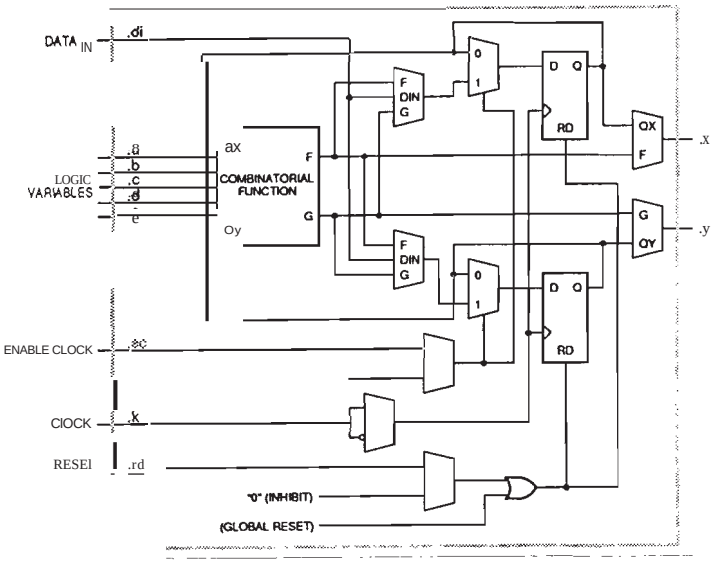

Figure 3. Each Configurable Logic Block (CLB) includes a combinational logic section, two flipflops and various program-controlled multiplexers that define logic flow through the block.

Programmable Interconnect

Programmable interconnect resources provide signal connections between the inputs and outputs of the I/O and logic blocks. These interconnections are composed from a two-layer grid of metal segments. There are three types of interconnect resources as shown in Figure 4: Direct Interconnect, General-Purpose Interconnect, and Longline Interconnect. After placing the designer's logic on the device, the Automated Placement and Routing (APR) software will use these various interconnect resources to join the blocks.

Direct interconnect consists of short, dedicated pieces of metal connecting only adjacent blocks. For example, a single

CLB has direct connections to its four nearest neighbors–the blocks above, below, left, and right. Direct interconnect is ideal for register transfer functions like shift registers or counters. Since direct interconnect is also the fastest type, shift registers built with a PGA can operated up to 60 MHz worst-case.

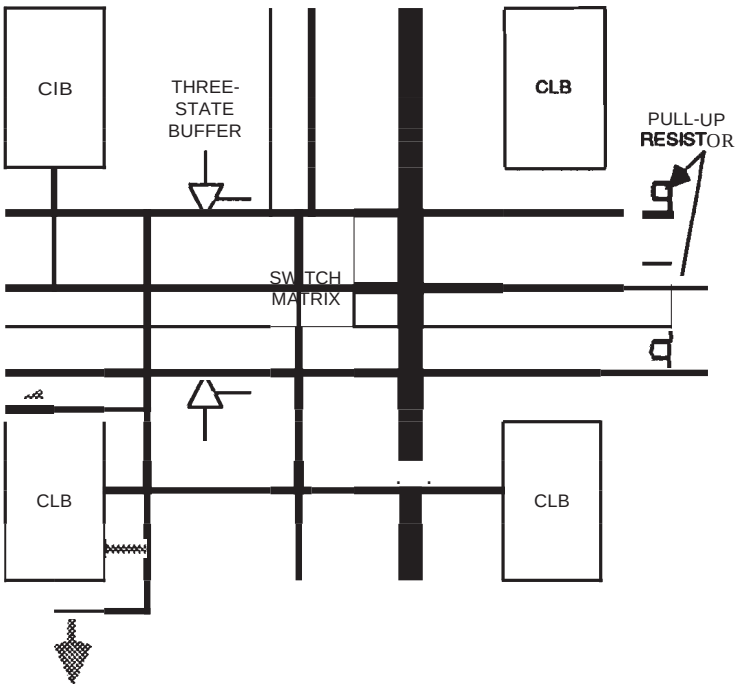

Figure 4. Programmable interconnect consists of three different resources. Direct interconnect (shown as shaded lines) joins adjacent blocks. General interconnect (shown as narrow solid lines) uses switching matrices to build flexible, localized connections. Longlines (shown as heavy solid lines) are ideal for clock signals since they have almost no skew.

General-purpose interconnect consists of a grid of five horizontal and five vertical metal segments located between the rows and columns of logic and I/O blocks. Switching matrices connect the various metal segments into the signal networks required to build the designer's logic. General-purpose interconnect is ideally used for medium fanout, localized connections since each additional switching matrix path represents an additional small delay.

Longline interconnect bypasses the switching matrices and is intended for high fanout signals or those requiring minimum skew (like clock signals), or those that must travel a long distance across the device. Longlines are single metal lines that traverse the entire height or width of the device. Since each longline is a continuous piece of metal, it represents a minimum skew path for signals tapping off from it.

Each interconnect column has four such longlines. The outer two vertical lines in each column are dedicated to clocking functions while the middle two are intended for general use. Each interconnect row has two horizontal longlines. An additional two longlines are located adjacent to the outer edges of the device for driving lOBs or for connecting other longlines together.

The horizontal longlines have additional capabilities. Each longline has a number of three-state buffers that may drive signals onto the common longline. With this capability, the horizontal longlines may be used to build bidirectional, internal busses on the device. In addition, the three-state buffers can be used to build wired-AND functions for fast decoding since each longline has two optional pull-up resistors at each end (their process-dependent values range from 2kΩ to 8kΩ). The smallest of the XC3000-Series, the XC3020, can include a 16-bit bidirectional bus while the largest, the XC3090, contains a 40-bit bidirectional bus capability.

SOFTWARE APPROACHESTO DESIGN ENTRY AND OPTIMIZATION

OVERVIEW

The Programmable Gate Array design process is shown in Figure 5. The various design modules may be entered using schematic capture or using a PALASM® or JEDEC input file. After entry, the logic within the each module may be optimized for the PGA architecture. The various design modules are merged together into a single, coherent logic description possibly from a hierarchical netlist.

After entry, the design is physically partitioned into the logic blocks on the device. Some optimization of the logic is performed to remove any unused or disabled logic functions. Further optimization of the physical design occurs when the logic blocks are placed and routed on the device. Human intervention on the physical design is optional using the XACT™ Design Editor which allows manual optimization of placement and routing.

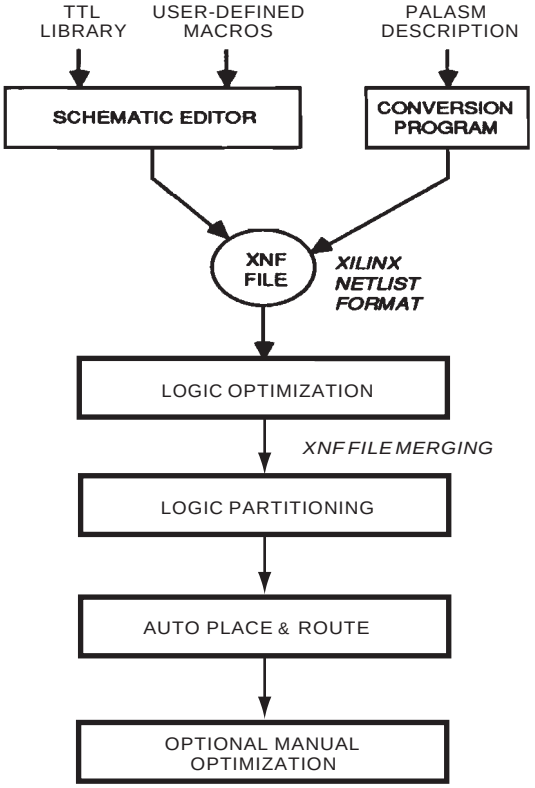

Figure 5. The Programmable Gate Array design process showing various stages of design and optimization steps.

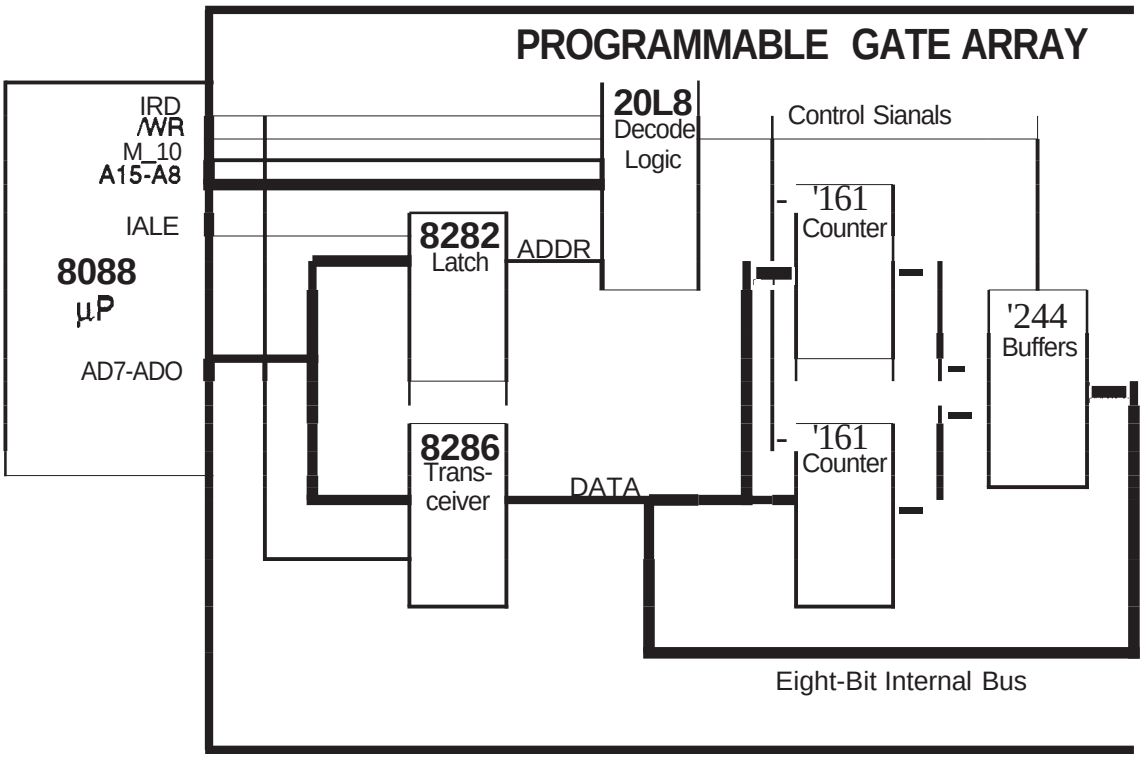

Figure6. An 8088 microprocessor peripheral design using a Programmable Gate Array. The circuit shown above requires a small portion of the 2000-gate XC3020 (13 of 64 CLBs).

DESIGN ENTRY

A Microprocessor Peripheral Example

The best way to illustrate a concept is through example. For the purpose of this discussion, assume that a design engineer was confronted with the microprocessor peripheral design shown in Figure 6 (which is just a portion of the overall design). He is asked by management to integrate the design, reduce the power consumption, and all the typical design challenges that confront a design engineer.

The design consists of an 8088 processor which will be implemented as a discrete device (though something of Z80 complexity would theoretically fit within an XC3090). Also, an 8282 octal bus latch demultiplexes the processor's eight-bit address/data bus while an 8286 bus transceiver directs data to and from the data bus. A pair of '161 preloadable binary counters implement an eight-bit counter which is preloaded and read by the processor. The output of the counter attaches to the processor's data bus via '244 three-state buffers. A PAL device decodes the address bus for controlling the counter and three-state buffers. The equations for this PAL device are shown in Figure 7. Much more logic will fit into the 2,OOO-gate XC3020 PGA, but for this example, a small design easily illustrates the various methods of design entry.

The goal of this simple design example is to integrate the octal latch, the bus transceiver, the '161s, the '244, and the PAL device into a highly-integrated solution. A standard PLD approach does not work because of the address/data bus

```
; This PAL decodes the address bus for
; loading, reading, and resetting counter

; WRITE ADDR = FFEO   load counter
; READ  ADDR = FFE1   read counter
; WRITE ADDR = FFE2   clear counter

CHIP  ELECTRO  PAL20LS

A15  A14  A13  A12  All  A10  A9  A8  A7  A6  A5  GND  A4  A3  A2
A1  A0  /RD  /WR  M_IO  /LOAD  /READ  /CLEAR  VCC

EQUATIONS

/LOAD   = A15 * A14 * A13 * A12 * All * A10 * A9 *
          A8 * A7 * A6 * A5 * A4 * /A3 * /A2 *
          /A1 * /A0 * M_IO * /WR

/READ   = A15 * A14 * A13 * A12 * All * A10 * A9 *
          A8 * A7 * A6 * A5 * A4 * /A3 * /A2 *
          /A1 * A0 * M_IO * /RD

/CLEAR  = A15 * A14 * A13 * A12 * All * A10 * A9 *
          A8 * A7 * A6 * A5 * A4 * /A3 * /A2 *
          A1 * /A0 * M_IO * /WR
```

Figure 7. The 20L8 PAL equations describe the address decoding for loading, reading, and clearing the counter.

demultiplexing and because of the bidirectional bussing between functions. Other than the XC3000 family, no single-chip programmable solution exists.

The engineer, of course, wants to implement this design using a familiar design methodology while still utilizing the advanced features of the architecture. Schematic capture offers an easy and familiar method of connecting various devices together. The TTL library allows the designer to implement his logic with familiar elements. Boolean equation entry allows for easy conversion of PAL devices. And finally, the hierarchical structure of the design entry process allows the designer to build his own macro functions for elements not already in the library.

Once each element is entered using the easiest method, the various pieces are merged together to form a cohesive, single design description. This logical description, in the Xilinx Netlist Format (XNF), is then automatically partitioned and reduced to optimally fit into the configurable logic blocks and I/O blocks.

Implementing the counters

The two '161 counters are implemented using the XILINX schematic library. The designer would simply bring up the '161 symbols from the library and connect the appropriate signal wires between the counters and the '244-style buffers. For aid in debugging and simulation, signal names should be attached to the various signal wires, though the schematic system will assign default names if none are given by the designer.

Implementing the octal latch and bus transceiver

The I/O block structure accommodates a wide range of interface applications. For this design example, both the 8282 octal latch and the 8286 bus transceiver fit into just eight I/O pins (with eight internal three-state buffers (TBUFs) used as part of the transceiver). Each IOB input path has both a direct and a registered input available simultaneously for both the latch and transceiver. Also, each IOB has a three-state-controlled output buffer for use by the transceiver.

The latch demultiplexes the lower eight bits of address (A7 to A0) from the address/data (AD7 to AD0) bus using the /ALE signal from the processor. The XILINX schematic library does not have an 8282 element but the designer may build his own from the lower-level primitives available in the library. From the schematic shown in Figure 8, the input register is set as a latched input (a '373-type latch) with the /ALE used as the active-low latch-enable signal. Therefore, addresses are latched during the address cycle while the data can flow directly into the device during the data cycle on the direct inputs.

To build the bus transceiver, merely add logic to the latch drawing. Each IOB contains a three-state output buffer with an invertible three-state control. The processors read strobe (/RD) controls the direction of data. When the processor is writing data, the output buffers are turned off and the internal three-state buffers are enabled. When the processor reads data, the output buffers are enabled while the internal buffers are high-impedance. The invertible three-state control on the

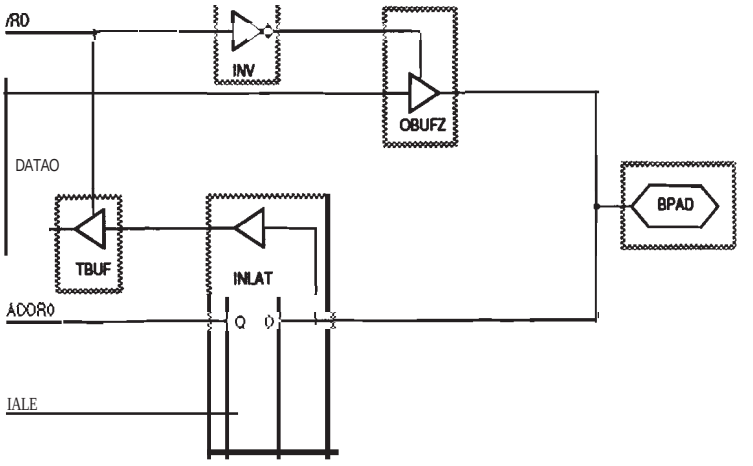

Figure 8. Both the 8282 bus latch and the 8286 bus transceiver are integrated into eight I/O Blocks. The diagram shows how a macro would be built for one bit of this function using the primitives in the XILINX library.

IOB allows a single line to control both the IOB buffer and the internal TBUF.

Converting the PAL Device into the Xilinx Netlist Format

Incorporating PAL devices within the Programmable Gate Array used to be a more difficult problem for designers. PAL devices have a sum-of-products (AND-OR) architecture much different from the PGA's array architecture. In a sum-of-products device, all the logic is broken down into a two-level logic structure. The inputs and feedback terms drive very wide AND gates which, in turn, feed smaller OR gates. Therefore, a design which may be optimized for the wide combinational logic of a PAL device may not fit well in a PGA if merely converted directly from the PAL device implementation.

To help designers with this conversion process, however, XILINX has created a translator and optimizer for PAL designs written in PALASM format (or translated directly from the JEDEC programming file).

The translator takes the PALASM or JEDEC file and directly converts it to a XILINX Netlist Format (XNF) file for use by XILINX development system. However, a direct translation will usually yield a sub-optimal design which will affect density and performance. Therefore, an optimizer program takes the XNF file (which is merely a logic description of the design) and optimizes it to better fit the PGA architecture through a technology called logic synthesis.

Logic Synthesis and Design Optimization

Logic synthesis technology allows a designer to enter his logic without regard to the specifics of the device architecture. In this example, a design specifically implemented in a sum-of-products architecture was remapped to better fit the architecture of the Programmable Gate Array.

Designers often use basic two-level minimization and logic synthesis when designing with PALs. Espresso, developed

jointly by International Business Machines and the University of California Berkeley, remains one of the best programs for logic minimization, when evaluated over a large set of logic functions.

However, two-level minimizers are designed for devices limited by logic complexity such as a PAL device where the first level is a programmable AND field followed by a fixed OR field. As a simple example, a five-input XOR requires two passes through standard PAL logic (not using a more expensive XOR PAL). The constraint in this example is the number of product terms allowed in the fixed OR field. A simple five-input XOR, when mapped into the sum-of-products architecture of a PAL device, requires 16 product terms (or a 16-input OR gate) while most PAL devices have only eight product terms. So a two-level minimizer takes this logic complexity constraint into account when synthesizing logic.

However, programs that perform two-level minimization do not generally apply to the PGA architecture. In a PGA device, logic is not limited by complexity but rather by fanin. Each logic block can implement any possible function of five, but only five, inputs (though limited functions of six and seven are possible). The five-input XOR function in the example above easily fits into a single CLB.

Logic synthesis is helpful for PGA designs requiring many inputs. For our design example, the address decoding for reading and writing the counter is just such a case. The decoding function fits into a single pass of the PAL device's wide combination logic and uses three of the PAL device's macrocells. To map the same function into the PGA, the designer must be cognizant that each CLB has only five inputs so the design requires multiple levels of logic. A multilevel approach would also be required if the PAL function were implemented in a gate array.

To make this process easier and automatic, XILINX developed a PAL design translator and optimizer. The answer derived by logic synthesis for this design example, which requires a 20L8, reduces the PAL design down to five CLBs. While this is a simple example, more dramatic results occur--especially in complex designs which tax human comprehension.

The designer may tell the optimizer to give him a quick answer or allow the software to search for a solution which optimally fits the PGA architecture. During the course of the search, the software will apply different optimization techniques and will keep track of two solutions. One solution tracks the minimum number of CLBs required and another searches for the minimum levels of logic required. The first tightly packs the CLBs for maximum density while possibly using more logic levels. The second solution may use more CLBs, all working in parallel, for best performance. In most cases, the result for best density matches that for best performance, but in some cases, the results can be dramatically different.

The optimizer is a descendant of the MIS and Espresso logic reduction algorithms, both of which originated at U.C. Berkeley. However, it employs proprietary technology for optimizing fanin-limited logic.

Other Uses of Logic Synthesis

For this design example, the optimization program performed the architectural remapping required to implement the PAL-based decoder in the PGA. Technology remapping is not limited to PAL devices only but can be applied to just about any other type of logic technology like TTL.

There are other uses for the optimizer, however. For example, a designer already comfortable with Boolean equation entry as a design method, can use PALASM to build a PGA directly from equations. Though PALASM is not sophisticated enough to describe some of the elements in the PGA architecture (like the internal three-state buffers), it is useful for describing combinational logic.

Merging the various entry formats

Each design entry method for XILINX PGAs creates an XNF file. Various sections of a design may have been entered using different methods. In order to merge these files into a cohesive design, a separate program combines the various XNF files into a single XNF file before partitioning the logic into CLBs. This merging program also flattens the design hierarchy into a single-level netlist. Hierarchy allows designers to use macros, to apply different types of optimization on separate portions of his design, and to build the design in modules.

Modularity provides the same benefits for hardware design as it does for software. Software designers use modularity to speed development and to isolate bugs to a specific module or subroutine. Similarly, in hardware design, each logic module can be designed and debugged separately. After the modules are completed, they are joined hierarchically to form the overall design. If bugs should appear, they can be quickly isolated to a specific module.

PHYSICAL IMPLEMENTATION

Partitioning the Design into Logic Blocks

The software effort to this point has been in entering or optimizing the design. Another program partitions the logic from a flattened and merged XNF file to build the logic specifically for CLBs. Up until partitioning, the design exists only as a logic description. The partitioner implements the design at the physical level.

This program not only partitions the logic but also removes any unused logic. For example, the terminal count (TC) generated by the second '161 counter in the design example would not lead to an output. If not removed, this would be wasted logic. The partitioner, therefore, removes any unused inputs or outputs, or single-input gates, or disabled functions (such as assigning a clock enable signal HIGH or always enabled).

Placement and Routing

After partitioning the logic into blocks, the blocks must be assigned to a physical location on the die and their

connections routed. One can think of the CLBs as generic components on silicon printed-circuit board. At this point, the software knows which "components" are to be used but not where to place those components on the circuit board.

The placement is performed using an algorithm called "simulated annealing" which models the physical process of annealing metals. During an annealing process, a metal which contains either deformations or imperfections, is heated to just above the melting point. At elevated temperatures, the atoms within the metal are allowed to move about freely in order to remove the defects. As the metal is slowly cooled, allowing for equilibrium at every stage, crystallization or an ordering process begins. Finally, at the freezing point, the metal is crystallized in a more orderly state. The simulated annealing algorithm, in general, is used to find good solutions to complex optimization problems. Automatic placement is just one application of the algorithm.

The algorithm "melts" the design in order to allow the CLBs and lOBs to move about freely. At high temperatures, blocks are allowed to move long distances across the device accomplishing global placement. Therefore, at high temperatures, blocks trapped on the wrong side of the device can quickly move to a more optimal location. As the temperature decreases, the blocks are restricted to a more limited move set. At low temperatures, only local placement occurs.

The goal is to "cool" the design slowly enough so that no new imperfections or irregularities are introduced. If the cooling phase proceeds too quickly, "quenching" occurs meaning that any imperfections (in this case, blocks placed in a suboptimal location) are frozen into the design. To avoid quenching, the temperature decreases in small decrements, especially when near the "freezing point." Also, the design is allowed to stabilize, or reach equilibrium, at each temperature.

At each successively lower temperature, the placement is evaluated. Blocks are typically only allowed to move to locations which will provide a better placement. However, depending how high the "temperature" is, a block has a certain probability that it will be allowed to move to a worse location. The advantage of this process is that it prevents the design from becoming frozen in a suboptimal state. A placement algorithm which only strives for a better placement will typically find only a localized solution and never find the global, optimal solution.

After placement, the program assigns which signals will use longlines and then routes the signal networks. After routing, the program calculates the worst-case interconnect delays for each network from the network source to each load. The delay report and all other information about the placement and routing is summarized in a report file automatically created by the program.

The placer and router treats all elements as though they have the same importance. In most designs, however, certain portions are more important or more time critical than others. Therefore, the designer can define "constraints" for the placer and router. For example, from the schematic, the design may define the placement of the I/O blocks or whether a net requires critical timing, or where to use longline interconnect. In addition, the automated placement and routing program accepts ASCII text instructions describing any design constraints.

Human Intervention

In cases where a design has particularly stringent performance requirements, the designer may elect to pre-route or pre-place portions of the design himself using the XACT Design Editor. The design editor allows the designer to specify elements at the device level in a mouse-based, graphical environment. By pre-placing and pre-routing critical sections of the application, the designer can directly control the performance of his application. As the critical signals are routed with the design editor, the system calculates the worst-case interconnect delays and displays them for evaluation. By adjusting the placement and the routing, the design performance can be "tweaked."

By pre-placing and pre-routing portions of the design, and then locking them in place through the ASCII text constraints file, the designer guarantees that his application will work at the required performance level after the first pass through automatic placement and routing. Also, by pre-placing portions of the design, the computational effort required to generate an optimal placement is reduced along with the execution time.

In cases where the automatic placement and routing software did not fully route all of the signal networks, the design editor is useful to complete any unrouted connections. As with printed-eircuit board placement and routing, a human designer is "smarter" than the average computer program but not as inclined to perform the tedious task of routing every net manually.

Completing the Design

The design is complete after placement and routing. The programming bitstream can be created for downloading into the part. Optional steps include simulating the design with the post-placement routing delays or debugging the design in-eircuit with the XILINX XACTOR In-Circuit Verifyer.

SUMMARY

In Programmable Gate Array (PGA) designs, optimization occurs at two levels. At the first level, the logic is optimized to best fit the PGA architecture. Logic design optimization provides the capability of mapping from an alien architecture, like PAL devices or TTL, into a form which optimally fits the PGA.

The second level of optimization occurs during the physical implementation process. The logic is further optimized as it is physically mapped into the Configurable Logic Blocks (CLBs). The placement and interconnection of the CLBs and lOBs is optimized using an advanced simulated annealing algorithm.

Design optimization, at both the logical and physical levels, is a focus of continuing research and development at XILINX. By optimizing designs remapped from other technologies, designer can apply new devices, with new architectures, to better solve old problems.