A Time-Multiplexed FPGA

Steve Trimberger, Dean Carberry, Anders Johnson,
Jennifer Wong

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
408-559-7778

steve.trimberger@xilinx.com

Abstract

This paper describes the architecture of a time-multiplexed FPGA. Eight configurations of the FPGA are stored in on-chip memory. This inactive on-chip memory is distributed around the chip, and accessible so that the entire configuration of the FPGA can be changed in a single cycle of the memory. The entire configuration of the FPGA can be loaded from this on-chip memory in 30ns. Inactive memory is accessible as block RAM for applications. The FPGA is based on the Xilinx XC4000E FPGA, and includes extensions for dealing with state saving and forwarding and for increased routing demand due to time-multiplexing the hardware.

Background

This paper reports on architecture development project that began in 1991. The architecture builds on the work of Ong [1995], who proposed rapidly reconfiguring an FPGA to increase logic capacity. Bhat [1993], DeHon [1995] and Tau [1995] also described rapidly-reconfigured FPGAs. These devices were deficient in at least one of the following critical areas:

- Device Capacity. Fundamentally, logic is shared to increase capacity. A base FPGA of low capacity is impractical, since a larger non-time-multiplexed FPGA is faster and easier to use.
- State Storage. Although combinational logic can be shared, state values cannot. They must be stored or forwarded until they are used. Simplistic solutions to this problem result in FPGA logic resources consumed with saving state. Very little remains for implementing logic.
- Memory. Time-multiplexed systems consume memory for configuration data and for staged design data. Facilities must be provided for this data.
- Static Logic. All systems include some logic that must always be active and cannot be multiplexed. A usable time-multiplexed device must be able to also supply this non-time-multiplexed logic.

This paper is divided into three sections. First is an introduction to the method of time multiplexing used in this architecture. The second section discusses modes of operation of a time-multiplexed device. The third section describes the device itself and highlights the device features and how they support the modes of operation.

The Basic Concept

The time-multiplexed FPGA is an extension of the Xilinx XC4000E product family. We gain logic capacity by dynamically re-using hardware. We add SRAM bits rather than CLBs.

The FPGA holds one active configuration and eight inactive configurations. The configuration memory is distributed throughout the die, with each configuration memory cell backed by eight bits of inactive storage in the configuration SRAM. This distributed inactive memory can be viewed as eight *configuration memory planes* (figure 1). Each plane is a very large word of memory (100,000 bits in a 20x20 device). When the device is *flash reconfigured*, all bits in the logic and interconnect array are updated simultaneously from one memory plane. This process takes about 5ns. After flash reconfiguration, about 25ns is required for signals in the design to settle.

Configuration memory planes can be loaded from off-chip while the FPGA is operating. They can also be read and written by on-chip logic, giving applications access to a single large block of RAM.

0-8186-8159-4/97 $10.00 © 1997 IEEE

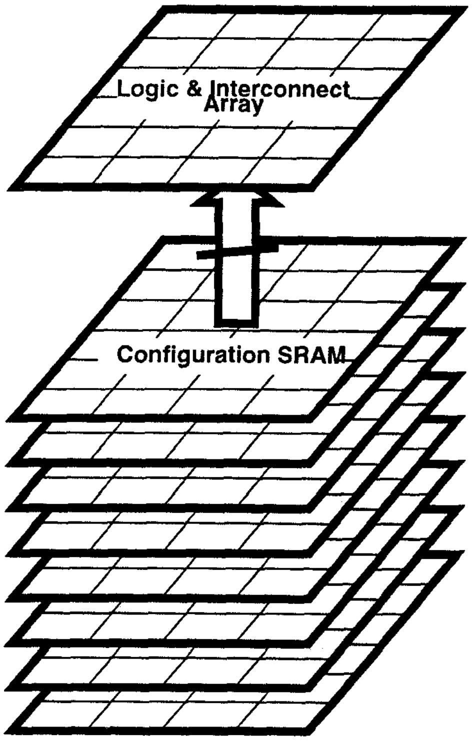

Figure 1. Time-Multiplexed FPGA Configuration Model

Modes of Operation

A rapidly-reconfigurable FPGA is a mere curiosity without a model of use that can be used as a design target and automated. We envision three modes of operation of the device: logic engine mode, time-share mode and static mode.

Logic Engine Mode

In logic engine (LE) mode, the time-multiplexing capability of the FPGA is used to emulate a single large design. Designs are modelled as Mealy state machines (figure 2). Combinational logic receives inputs from the device inputs and from flip flop outputs; and device outputs come from combinational logic and from flip flops. The combinational logic can be split into pieces and LUTs in the FPGA can be time-multiplexed during the calculation of those results.

When operating in logic engine mode, the FPGA sequences through multiple configurations called *microcycles*. The sequencing of microcycles is synchronized with the user's clock. One pass through all microcycles is called a *user cycle*. All combinational logic is evaluated and all flip-flop values are updated in one user cycle.

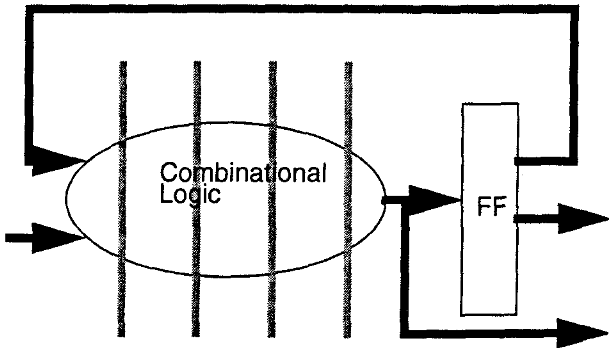

Figure 2. Logic Model

In this mode, the FPGA is reconfigured several times per user clock cycle -- the reconfiguration clock is faster than the user clock. Partial results from one configuration of the device must be saved and passed to subsequent configurations. Storage must be provided for flip flops, which cannot be time shared, since all values are required at future times.

Time Share Mode

In time share (TS) mode, the FPGA emulates several independent, communicating FPGAs in a virtual hardware environment. The FPGA remains in a single configuration for multiple user clock cycles before switching to another configuration -- the reconfiguration clock is slower than the user clock rate. The FPGA may reconfigure at irregular intervals or upon interrupt. To support virtual logic in time share mode, values computed in one configuration must be stored and shared with logic in other configurations. When a configuration is re-loaded into the FPGA, its state must be restored so it can resume operation as if it had never been removed.

Static Mode

In static mode, the FPGA, or part of it, does not appear to be reconfiguring at all. Static mode is used to build logic that must always be resident and active -- for example the logic that controls the time share or logic engine sequencing, or asynchronous logic. We implement static logic by programming memory cells of multiple configuration bits to be identical. When the new configuration is loaded, it operates identically. We ensure that reconfiguration does not glitch control points if they are unchanged, so the emulated logic operates without interruption.

Partial reconfiguration is not sufficient to support static logic, since dynamic logic interconnect may pass through

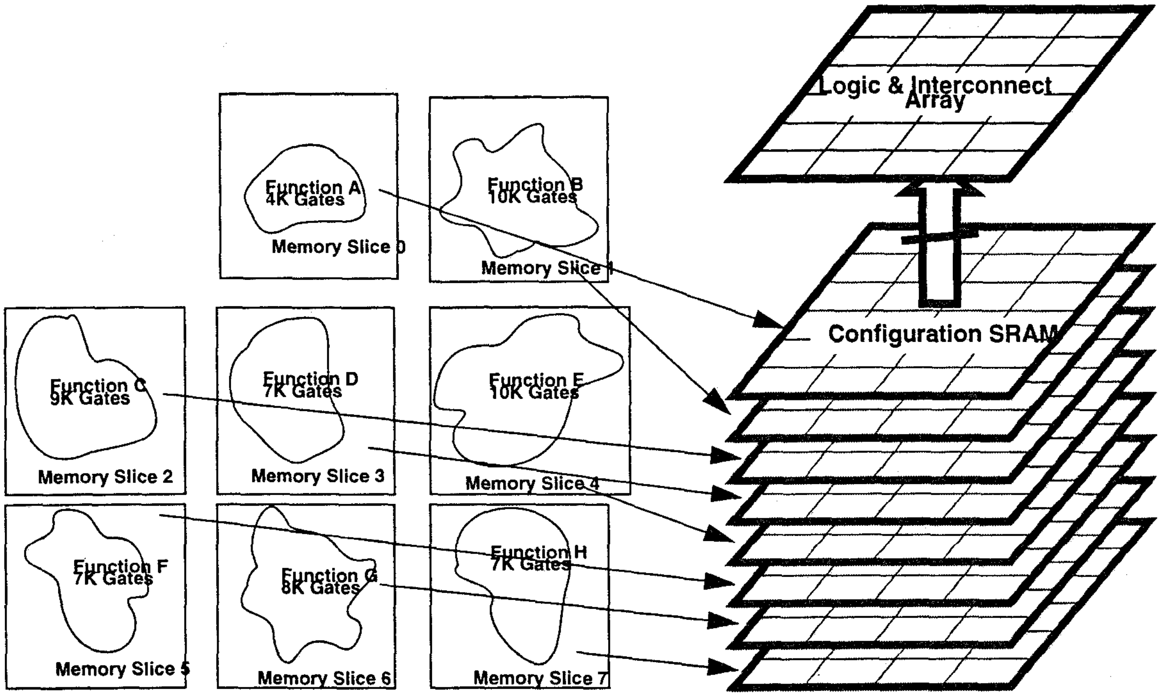

Figure 3. Time Share Model

static logic regions on the chip. A design where a region of the device is not reconfigured would allow static logic to exist, but such a design is too restrictive because it prevents device resources in the static region from being accessed by dynamic logic.

Memory Access

In addition to these operation modes, a configuration memory plane can be used as a block RAM of approximately 100,000 bits. The RAM mode allows user designs to read and write the memory directly, leading to the ability to create self-modifying hardware. Although this mode of operation is intriguing, we can see only one application: building a clever loader (perhaps decrypting data as it configures the plane).

Mixed Modes

We expect that these operation modes will be mixed on the chip, as shown in figure 4. A single application may have a few memory planes used as memory, and the logic part of the array split between static logic and time-shared logic or logic engine logic. A common occurrence of mixed modes is on the chip outputs in Logic Engine mode: the output side of the IOs must be static to properly emulate the outputs of the design. However, inputs may be dynamic, cycling in logic engine mode.

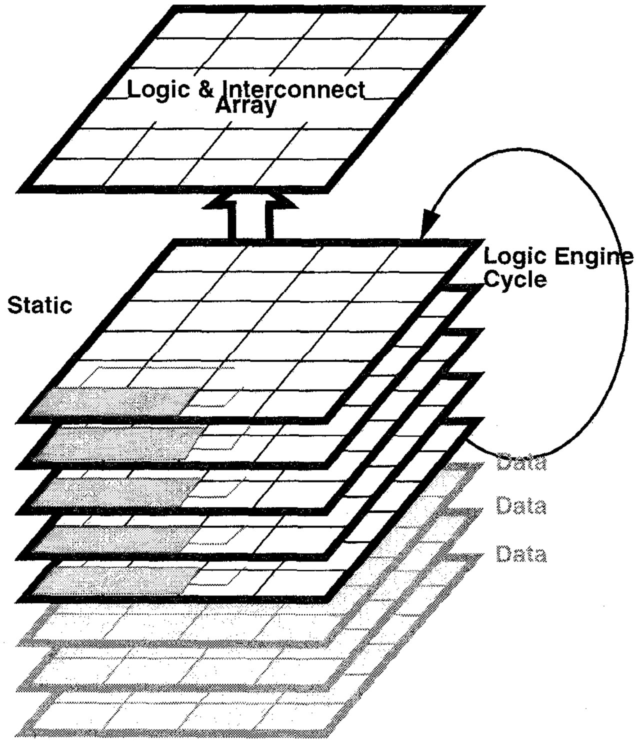

Figure 4. Mixed Modes

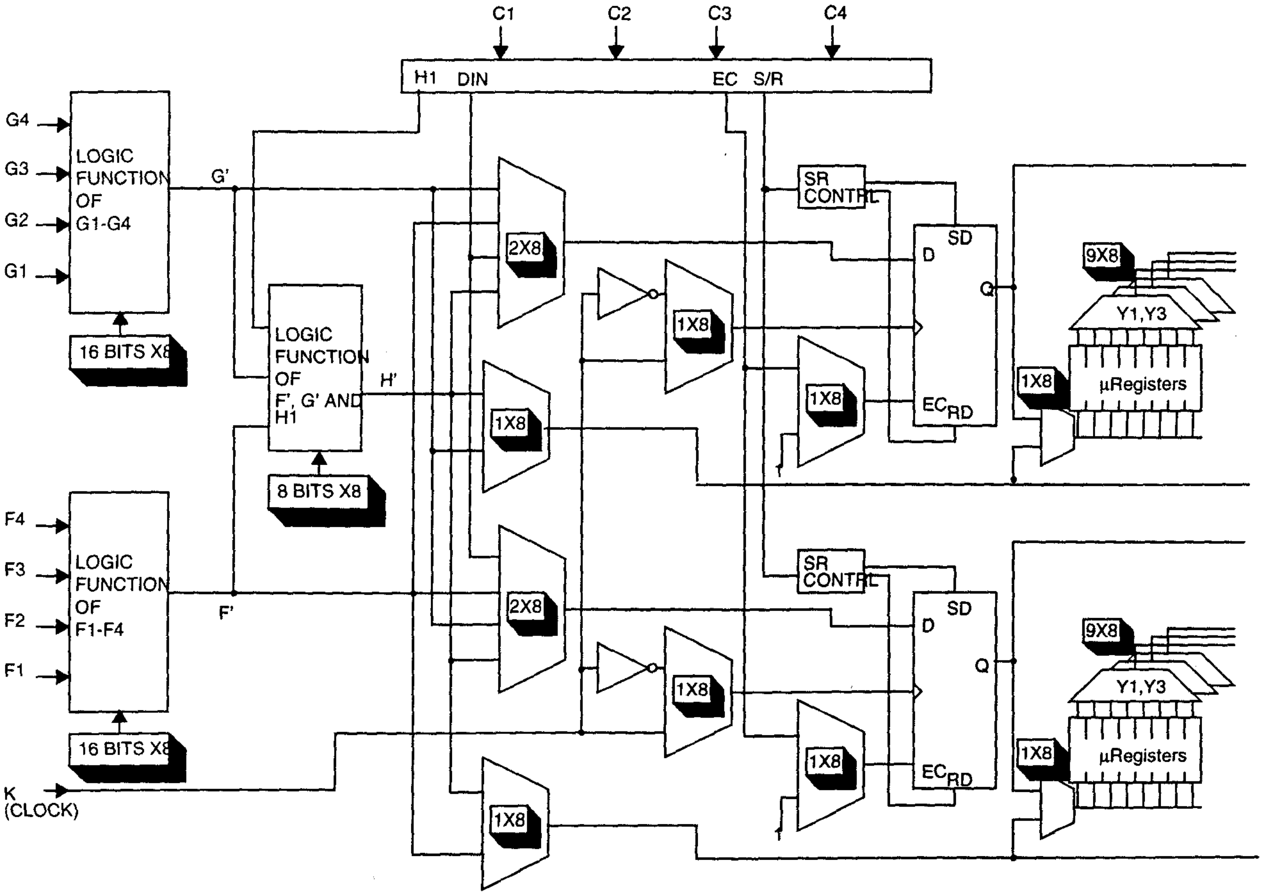

Figure 5. Time Multiplexed FPGA Architecture

Configurable Logic Block

As stated previously, the reconfigurable FPGA is based on the XC4000E device. Figure 5 shows a block diagram of the CLB. All configuration points in the device are backed by eight memory cells, as shown by the shadowed boxes. All interconnect points are similarly backed by memory cells. A significant difference between this CLB and the XC4000E is the addition of the micro registers near the CLB outputs on the right side.

Micro Registers

A micro register stores the CLB output (either the combinational or sequential output) when the FPGA changes configuration. This state saving is automatic, though each micro register can use the CLB's clock signal as a clock enable, which can be used to control state saving. The state-save-enable feature is used to support multiple clocks in logic engine mode.

Logic can access all values stored in micro registers from any configuration. The micro registers are multiplexed with the CLB's combinational and sequential results onto the CLB outputs. Like all other configurable features of the chip, those multiplexers are controlled by multiple configuration memory cells. Micro register signals are routed through normal programmable interconnect to their destinations.

In logic engine mode, micro registers store intermediate values and flip flop values. This is a critically important capability. Although combinational logic can be multiplexed among several functions, state storage cannot. As a result, although eight LUTs in a design may share a single physical LUT, eight flip flops must all be provided for subsequent logic to access the results. The storage for these flip flops is in the micro registers.

In time share mode, the micro registers serve to pass data from one configuration to another, and to save the state of a swapped-out configuration for restoration when it is re-activated.

In figure 5, access outside the CLB is limited to three of the eight micro registers during any one micro cycle. This restriction was derived empirically. We built an optimizing scheduler to partition many designs onto a time-multiplexed FPGA model. The results indicated that three outputs were sufficient to allow required access to micro registers without over-constraining placement.

Interconnect

Like CLB configuration cells, all configuration cells that control interconnect are backed by eight inactive memory cells.

Fundamentally, the signals routed from the micro registers to their destinations represent additional nets that must be routed, increasing wiring demand. Therefore, additional interconnect is required on the device. The interconnect capacity in the time-multiplexed FPGA is shown in table 1. This capacity chosen to provide a 97% probability of place and route success for a full 20x20 array of CLBs with additional interconnect demand due to logic engine wiring. A high success rate is required for a single configuration because a single design may be composed of up to eight configurations, all of which must route successfully for the design to operate.

	Vertical	Horizontal
Singles	8	8
Quads	8	8
Octals	8	8
Long Lines	0	6

Table 1. Interconnect Summary.

User Memory

A single memory access port located in the center of the chip uses the configuration address and data lines to access the configuration memory without interfering with FPGA logic. The address is decoded into a memory plane and a reference within the plane. The memory access port allows 8-bit, 16-bit or 32-bit access to the configuration memory.

Configuration Controller

Several options on reconfiguration are controlled by the *reconfiguration controller.* The controller supports time share and logic engine modes.

Time Share Mode

To support timeshare mode, flash reconfiguration can be initiated by an external or internal signal. The address of the new configuration plane can come from an internal or external source. Therefore, there is no restriction on which configuration is the next to be run. The reconfiguration operation proceeds as follows:

1. Save all CLB flip flop values in micro registers
2. Load the new configuration
3. Restore CLB flip flop values from micro registers

Flip flops are restored to allow a swapped-out configuration to resume operation where it ended, allowing us to swap "virtual logic" in and out of the device.

Logic Engine Mode

In logic engine mode, the sequence of reconfiguration is known in advance. The speed of reconfiguration is critical, since multiple configurations are required to complete a single cycle in the emulated design. The controller includes a next-address calculator. The controller reads the memory plane at that address and holds it, awaiting the completion of the current micro cycle.

The reconfiguration operation (microcycle) proceeds as follows:

1. Save all CLB outputs in micro registers.
2. Activate the new configuration.
3. Perform a user memory access (if any).
4. Pre-fetch the next configuration.

User memory access (#3) and pre-fetching the next configuration (#4) can be pipelined with the operation of the logic in the configuration (#2). Steps #3 and #4 are both memory operations, and both used the configuration memory bus. Therefore they must be serialized. However, since each memory operation takes about 5ns, the delay for these operations can be completely hidden.

Notice the CLB flip flop values are not restored. The CLB flip flop is not used in LE mode -- flip flop values are stored in micro registers.

Conceptually, all micro cycles are of the same duration, but in practice, the length of a micro cycle is the settling time -- the amount of time required for values to propagate from micro registers through combinational logic and set up the next-stage micro registers. This delay includes routing delay, which is dependent on the placement and routing of the logic plane. Therefore, each microcycle includes a duration counter, which determines how long the FPGA remains in the current micro cycle before proceeding to the next one. The duration is set by software after placement and routing is complete. The software timing verifier determines the length of the longest path after routing and sets the microcycle duration accordingly.

Static Mode

Since static mode is supported by programming some bits identically, the controller has no special sequencing to support it. The control over restoring flip flops is actually stored as bits in each CLB, so state restoration is permitted on a per-CLB basis. This flexibility is required to allow static flip flops to be distributed among time-shared logic. The CLBs containing static flip flops are not overwritten during reconfiguration, while those in time-shared logic are overwritten.

Configuration Memory Design

Figure 6 shows the circuitry for the configuration memory. Eight SRAM cells are connected to a single bit line for the cell. The current control value is held in the latch. To write a memory cell, a word line (Wn) is brought high allowing the data value on the bit line to overwrite the corresponding memory cell (MCn). Typically, only one word line on the chip is high during a memory write.

For flash reconfiguration, a single word line (Wn) is brought high, enabling the corresponding memory cell onto the bit line. When the bit line is stable, the latch is clocked to store the new configuration value. Typically, the bit lines for the same memory plane are read simultaneously for every control point.

The timing of μClock ensures that the bit line is stable before the latch becomes transparent -- this allows glitch-free transitions from one configuration to the next when the two values are the same -- a requirement for static logic.

The storage latch lets memory operations on the inactive memory proceed without affecting the active configuration. These operations include loading a memory plane with configuration data or design data from an outside source, and accessing the memory through the user memory port on the chip.

The latch also allows us to pipeline configuration fetch with operation of the device. We can pre-fetch the next configuration of the device, then instantly switch to the next configuration by clocking the latch. This is most useful in logic engine mode, where the configuration proceeds through a pre-defined sequence of steps, and the next configuration address is always known.

Power Consumption

Power consumption during reconfiguration can be very high. Although each bit line has a very small capacitance, there are 100,000 bit lines in a 20x20 array. Further, the signals on time-multiplexed interconnect on the FPGA is not expected to auto-correlate from cycle to cycle, as is the case from cycle to cycle in a traditional FPGA. A logic engine design operating at 40MHz can consume tens of watts. We

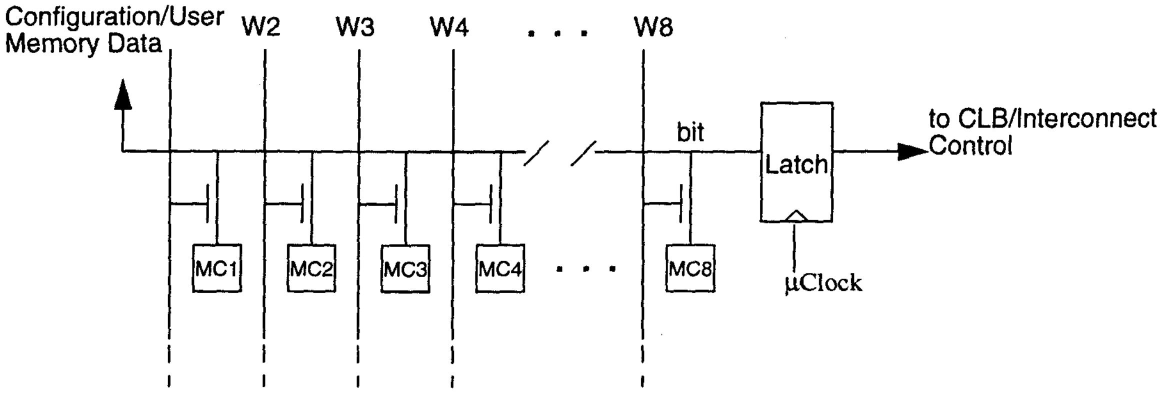

Figure 6. Memory Circuit

addressed power consumption two ways. First, we lowered the voltage swing on the bit lines. Secondly, we reduced the number of memory cells required to configure the device. However, power consumption remains a concern, particularly with larger devices.

Layout

As shown in figure 7, the design consists of columns of memory blocks, interleaved with logic. Each memory block contains 16 bits of memory, eight of which control field programmable logic on the left, eight on the right. The area between the memory cell columns is used to build the field-programmable logic: lookup tables, micro registers and programmable interconnect.

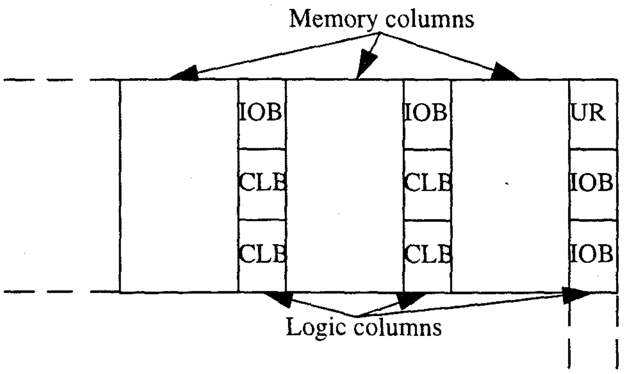

Figure 7. Chip Layout Floorplan. Upper Right Corner

The FPGA area is dominated by the memory and memory overhead circuitry. To reduce area, we reduced the memory cell count wherever possible. For example, the control bits for all multiplexers are fully encoded, reducing the memory cell count, which reduced the area and power consumption.

The layout was done in 0.5μm CMOS. The chip has not yet been fabricated.

Summary

This paper describes an architecture and modes of operation of a time-multiplexed FPGA. Modes of operation include:

- Logic engine mode, where the device emulates a single large FPGA.
- Time share mode, where the device emulates several communicating FPGAs.
- Static mode, where the logic remains active and unchanged during configuration.

Architectural and circuit innovations include:

- Micro registers for state storage and forwarding partial results. Micro registers also hold state for restoration of logic in a virtual hardware environment.
- A configuration controller that sequences configurations intelligently for logic engine and timeshare mode.
- Storage for the active configuration to allow pipelining of configuration memory fetch with FPGA operation and for allowing the configuration storage to be used as a block memory efficiently.

References

N. Bhat, K. Chaudhary, E.S. Kuh, "Performance-Oriented Fully Routable Dynamic Architecture for a Field Programmable Logic Device", M93/42, U.C. Berkeley, 1993.

A. DeHon, "DPGA-Coupled Microprocessors: Commodity ICs for the 21st Century", *IEEE Workshop on FPGAs for Custom Computing Machines*, 1995.

D. Gajski, N. Dutt, A. Wu, S. Lin, *High Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, 1994.

R. Ong, "Programmable Logic Device Which Stores More Than One Configuration and Means for Switching Configurations", *U.S. Patent* 5,426,378, 1995.

E. Tau, D. Chen, I. Eslick, J. Brown, A. DeHon, "A First Generation DPGA Implementation', *FPD '94 - Third Canadian Workshop on Field-Programmable Devices*, 1995.

S. Trimberger, "Mapping Large Designs into a Time-Multiplexed FPGA", private communication, 1997.

Xilinx, *The Programmable Logic Data Book*, 1996.