H. Grunbacher, R. W. Hartenstein, Eds., *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, Berlin: Springer-Verlag, pp. 44-51, 1993.

MONTAGE: An FPGA for Synchronous and Asynchronous Circuits

Scott Hauck, Gaetano Borriello, Steven Burns, Carl Ebeling
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Abstract

Field-programmable gate arrays are frequently used to implement system interfaces and glue logic. However, there has been little attention given to the special problems of these types of circuits in FPGA architectures. In this paper we describe Montage, a Triptych-based FPGA designed for implementing asynchronous logic and interfacing separately-clocked synchronous circuits. Asynchronous circuits have different requirements than synchronous circuits, which make standard FPGAs unusable for asynchronous applications. At the same time, many asynchronous design methodologies allow components with greatly different performance to be substituted for one another, making a design environment which migrates between FPGA, MPGA, and semi-custom implementations very attractive. Similar problems also exist for interfacing separately-clocked synchronous circuits. We discuss these problems, and demonstrate how the Montage FPGA satisfies the demands of these classes of circuits.

Introduction

Field-programmable gate arrays provide an ideal implementation medium for system interface and glue logic. They integrate large amounts of random logic and simple data paths, and can be easily reprogrammed to reflect changes in system components. Unfortunately, most of the effort in designing FPGA architectures has ignored the special problems of these types of circuitry. Interface and glue logic require support for interfacing asynchronous logic to synchronous logic, interconnecting separately-clocked synchronous components, and controlling certain circuit delays [Borriello 1988], all of which are largely ignored by current architectures.

Asynchronous circuits are also not well served by current FPGAs. Implementations of asynchronous logic must consider hazards in the logic, synchronization and arbitration of events, and strict adherence to the timing assumptions of the design methodologies [Martin 1990, Sutherland 1989]. Unfortunately, it is not possible to implement these circuits in a robust manner in current FPGAs. Some of the elements required (most importantly, arbiters that resolve conflicts between two concurrently arriving signals) are not implementable in the standard digital logic found in these devices. In addition, the logic and routing elements must be designed more carefully in order to avoid extra "glitches" on lines, since in asynchronous circuits every transition is important. Finally, routing resources must have predictable, optimizeable delays in order to meet timing assumptions in the design methodologies.

However, the problems are not restricted simply to the architectures themselves. The mapping software also must be altered to handle the demands of asynchronous logic. Primarily, there are much stricter timing demands that must be upheld. Bundled data and isochronic forks both require signals to be routed with special delay demands, demands that impact placement of logic cells as well. Also, the logic decomposition used to break logic blocks down to the size required by an FPGA (the covering problem) cannot simply use the algorithms for synchronous logic. For quasi-delay insensitive circuits, where the only timing assumptions made are those of isochronic forks, standard synchronous logic decomposition techniques can add extra levels of logic incorrectly, putting hazards into the circuit.

Related work [Brunvand 1991] has looked at mapping asynchronous circuits to Actel electrically-programmable gate arrays. Although the paper outlined an implementation strategy based on a fairly rich library of macro blocks, the underlying limitations of the Actel parts made it difficult to handle arbitration and synchronization, and the complex routing structure did not allow adequate control of routing delays. In addition, we feel that any antifuse-based FPGA is less desirable for asynchronous circuits because of the strict assumptions that must be made about circuit delays. These may require chip delay testing, which is impossible in an unprogrammed antifuse system as path delays cannot be measured until after programming.

The Architecture

The Montage FPGA is a version of the Triptych architecture designed to handle synchronous interface and asynchronous circuits. Since much of Montage is identical to Triptych, we direct readers unfamiliar with this architecture to [Hauck 1992]. Like Triptych, Montage is a RAM-based FPGA, which is preferable to antifuses because it allows the chip to be programmed for delay testing without permanently configuring it.

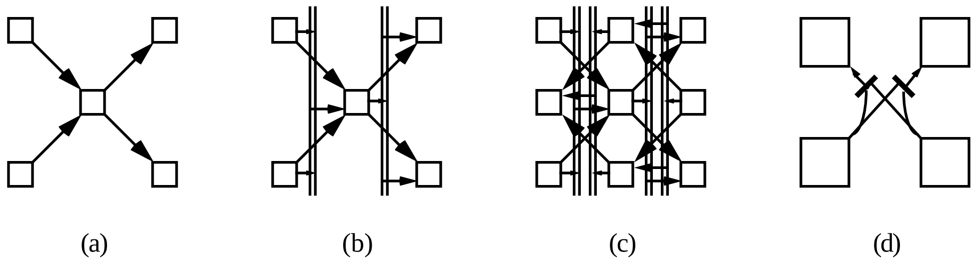

Figure 1. The overall structure of the Montage FPGA shown in a progression of steps. The basic fanin/fanout structure (a) is augmented with segmented routing channels (b) attached to a third RLB input and output. The structure (c) is obtained by merging two copies of (b), with data flowing in opposite directions in the two copies. Shown in (d) are the connections between the two copies, which permit communication between the two copies, in order to implement sequential logic.

The Montage global routing structure is identical to the Triptych routing structure, with diagonal connections between local cells, augmented with vertical segmented channels (see figure 1). This structure has proven to be effective for mapping general synchronous circuits. It is even better suited to asynchronous circuits, where one expects to find much more tightly connected subcircuits, and in general less random global routing. Also shared with Triptych is the general philosophy of allowing mappings to fix the tradeoff between logic and routing resources by having logic blocks capable of performing routing functions.

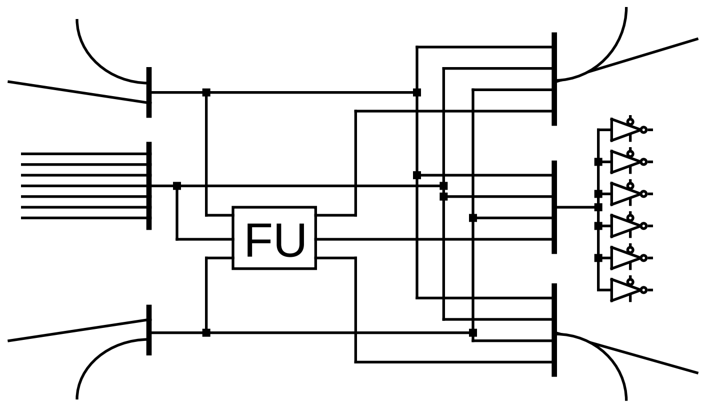

Figure 2. Montage routing and logic block (RLB) design. The RLB consists of: 3 multiplexers for the inputs, a functional unit, 3 multiplexers for the outputs, and tristate drivers for the segmented channels.

A Montage RLB (shown in figure 2) has three inputs and three outputs, and a functional unit (FU) which operates on the three inputs. There are two different types of functional units. The first is a logic block, which implements logic functions and stateholding elements. As shown in figure 3, the logic block has a function block capable of implementing any function of 3 inputs. The switch logic function block shown was chosen because it does not suffer from charge sharing. This is important because asynchronous circuits require very clean signals, with absolutely no extraneous transitions. The function output can be fed through a master-slave d-latch. This d-latch can be configured with one of two clocks in synchronous mode (allowing two independently clocked synchronous circuits to coexist on a chip), or with a choice of initialization state in asynchronous mode. In the asynchronous initialization mode, the latch is set to a value during programming and holds the function output to this value until enough time has passed for the circuit to reach a valid operating state, at which point the latch is made transparent. Each RLB can choose independently how to use the d-latch, so on a single chip there can be two separately clocked synchronous circuits, asynchronous elements initialized with the built-in circuitry, and unlatched logic blocks. Note that any one of the three logic block inputs can be replaced with a feedback line carrying the d-latch's output value. This feature is used in two separate ways. For synchronous circuit elements, this line carries the latch output. This allows a function to be dependent on its previous value, implementing simple state machines in a single cell. For asynchronous elements, this line allows state-holding (non-combinational) elements to be built. This is done by expressing the state-holding function of n inputs as a combinational function of (n+1) inputs, where the extra input is the function's previous value. Thus, a single logic block is capable of implementing any 3-input combinational function, or a 2-input stateholding function such as an asynchronous S-R flipflop or a Muller C-Element.

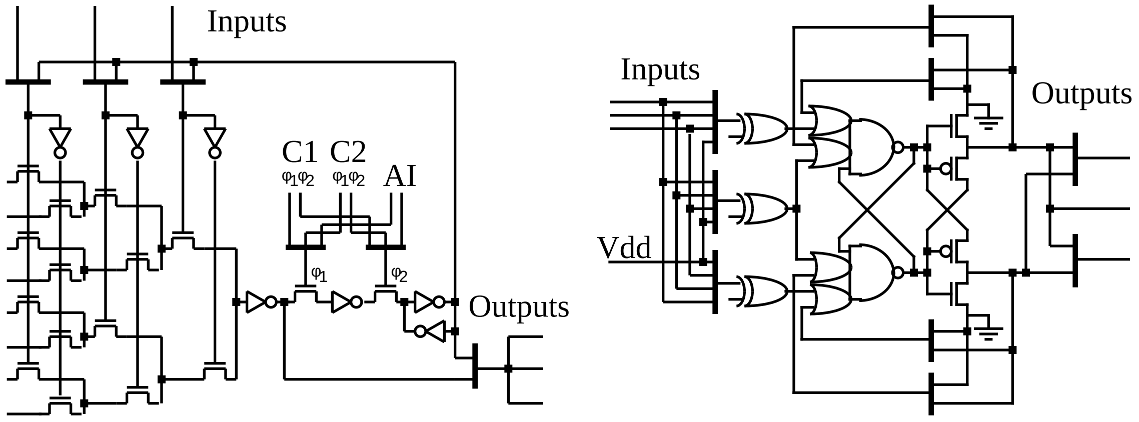

Figure 3. The two types of functional units: the logic block (left) and the arbiter unit (right).

The second type of functional unit is an arbiter block. This block is capable of implementing an arbiter, an enabled arbiter, or a synchronizer, with all inputs completely permutable and invertable. Although we expect these blocks to be used infrequently, the roles they serve in asynchronous circuits are essential, and are not implementable in standard digital logic. Thus, they must appear as special, built-in blocks in any FPGA which hopes to implement asynchronous circuits, but which does not allow mappings to program circuits at the transistor level (for an example of an EPGA which might allow sufficient transistor-level programming to implement an arbiter, see [Marple 1992]). For examples of how both types of blocks are used, please see figure 4.

Currently we plan to have a 15:1 ratio between the number of logic blocks and arbiter blocks. This number was chosen based on the relative infrequency of arbiters and synchronizers in typical asynchronous circuits. Since we found that typical Triptych mappings used 25% of their RLBs for routing only, jobs which the arbiter RLBs in Montage are capable of handling, we believe that most unused arbiters will be absorbed into this factor. However, we have taken care to ensure arbiter blocks occupy the same amount of area as logic blocks, allowing easy alteration of the arbiter mix in Montage implementations should it prove necessary.

A last important point to be made about the architecture is how the Montage routing structure handles bundled data and isochronic forks. For bundled data, the Montage routing structure's simplicity makes it much more feasible to design a router which ensures that bundled-data control signals take longer paths than all of their data bits. For isochronic forks, there are two different implementation styles dependent on the type of isochronic forks. Isochronic forks can be broken into two classes: one-way, where the only requirement is that a signal reach one end of the fork before it reaches the other; and two-way, where the signal is required to reach both ends of the fork at the same time. For one-way isochronic forks, the signal is routed to the RLB of the critical end of the fork, and is then routed back out of this block to the other side of the fork. Thus, the dual routing and logic nature of a Montage RLB ensures that the signal reaches one cell before the other. For two-way isochronic forks, the two ends of the fork are placed either off the same interconnect line, or off diagonals flowing from a shared source RLB. In this way, the isochronic fork depends on the correct speed of very localized elements, delays which can easily be checked during initial chip verification.

Future Work

The development of an FPGA for asynchronous circuits opens up several new avenues of exploration. The entire process of mapping for FPGAs must be re-evaluated for this domain. Most obviously, placement algorithms must take into account the constraints generated by bundled data and isochronic forks, and routers must ensure these constraints are met. A more subtle issue arises in the covering process. Quasi-delay insensitive circuits, which make no timing assumptions other than that of isochronic forks, cannot always be decomposed using standard logic decomposition techniques. The problem is that when a logic function is decomposed, the lines introduced to connect the subcomponents may now have transitions on them that did not exist in the original circuit. Since these transitions are obviously not properly sensed in the original circuit, where they did not even exist, it can cause the circuit to malfunction. Thus, new covering techniques must be explored, such as a complete resynthesis from the original circuit description.

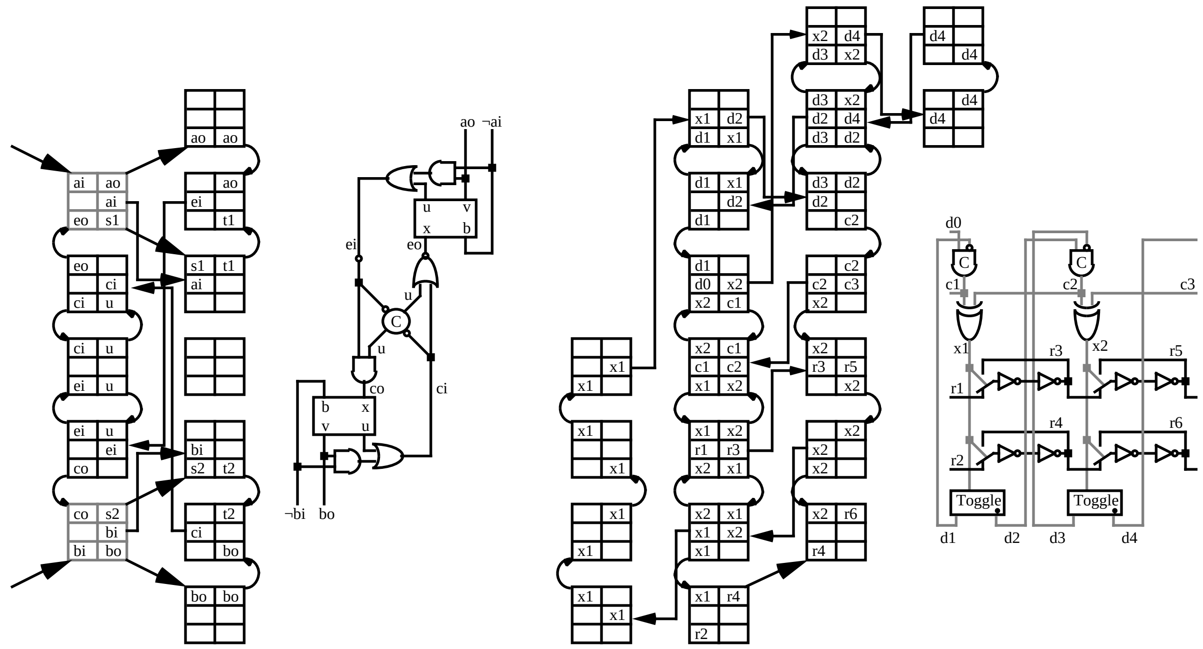

Figure 4. Two example circuits: (left) Martin's fair arbiter [Martin 1990], built with two synchronizers (arbiter blocks have grey outlines), and (right) Sutherland's micropipelined FIFO [Sutherland 1989]. Note that although only two levels of the FIFO are shown, the mapping fits together for longer FIFOs.

Acknowledgments

This research was funded in part by the Defense Advanced Research Projects Agency under Contract N00014-J-91-4041. Gaetano Borriello and Carl Ebeling were supported in part by NSF Presidential Young Investigator Awards, with matching funds provided by IBM Corporation and Sun Microsystems.

References

G. Borriello, "A New Interface Specification Methodology and its Application to Transducer Synthesis", Technical Report UCB/CSD 88/430, University Of California, Berkeley, May 1988.

E. Brunvand, "Implementing Self-Timed Systems with FPGAs", International Workshop on Field-Programmable Logic and Applications, Oxford, 1991.

S. Hauck, G. Borriello, C. Ebeling, "Triptych: An FPGA Architecture with Integrated Logic and Routing", Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems, March 1992.

D. Marple, L. Cooke, "An MPGA Compatible FPGA Architecture", First International ACM/SIGDA Workshop on Field-Programmable Gate Arrays, Berkeley, 1992.

A. Martin, "Programming in VLSI: From Communicating Processes to Delay-insensitive Circuits". In C. Hoare, "UT Year of Programming Institute on Concurrent Programming", Addison-Wesley, Reading, MA, 1990.

I. Sutherland, "Micropipelines", CACM, Volume 32, Number 6, June 1989.