ZUMA: An Open FPGA Overlay Architecture

Alexander Brant
Dept. of ECE
UBC
Vancouver, BC
Email: alexb@ece.ubc.ca

Guy G.F. Lemieux
Dept. of ECE
UBC
Vancouver, BC
Email: lemieux@ece.ubc.ca

***Abstract*—This paper presents the ZUMA open FPGA overlay architecture. It is an open-source, cross-compatible embedded FPGA architecture that is intended to overlay on top of an existing FPGA, in essence an "FPGA-on-an-FPGA." This approach has a number of benefits, including bitstream compatibility between different vendors and parts, compatibility with open FPGA tool flows, and the ability to embed some programmable logic into systems on FPGAs without the need for releasing or recompiling the master netlist. These options can enhance design possibilities and improve designer productivity. Previous attempts to map an FPGA architecture into a commercial FPGA have had an area penalty of 100x at best [4]. Through careful architectural and implementation choices to exploit low-level elements of the host architecture, ZUMA reduces this penalty to as low as 40x. Using the VTR (VPR6) academic tool flow, we have been able to compile the entire MCNC benchmark suite to ZUMA. We invite authors of other tool flows to target ZUMA.**

***Keywords*-Field programmable gate arrays; Productivity; Reconfigurable architectures;**

I. Introduction

FPGAs are used widely in research and industry, but FPGA devices themselves are predominantly designed by a few large companies, and access to low-level details is often limited or simply unavailable. There have been calls for a completely open and portable tool flow that allows true portability of designs across all FPGA devices offered by major vendors. Due to factors including the complex and proprietary nature of FPGA designs, vendors have been slow to respond. This paper presents the ZUMA embedded FPGA architecture. ZUMA is a free, open, and cross-compatible embedded FPGA (eFPGA) architecture that compiles onto Xilinx and Altera host FPGAs. It is designed as an open architecture for open CAD tools and user designs which must be independent of the underlying device architecture. Through this approach, we hope to enable exploration of new programmable logic implementations in both commercial and research applications.

Initially a generic architecture was created in pure Verilog, which was used as a baseline for the development of the ZUMA architecture tailored for implementation on modern FPGAs. A CAD flow is in place for this architecture utilizing the VTR (VPR 6) project [3]. The final resource usage of the ZUMA architecture is less than one third of the generic design, and less than half of previous research attempts [4], at as little as 40 host LUTs per ZUMA embedded LUT. Throughout this paper, the term embedded LUT (eLUT) is used to differentiate between the LUTs of the host FPGA and the LUTs in the ZUMA architecture, and resource usage per eLUT is used to compare densities.

The key contributions of this paper are:

- adoption of configurable LUTRAMs as the basis for implementing both programmable LUTs and routing MUXs
- design of a Clos-style Input Interconnect Block (IIB) network for improved area efficiency of the internal crossbar of the cluster
- design of a resource-efficient configuration controller
- architectural modeling to determine the most efficient parameters for mapping to a given host architecture.

A. Motivation

The development and improvement of an FPGA-like overlay architecture is motivated by a variety of applications and research needs, as well as the opportunity for improved designer productivity. ZUMA can help address these goals by providing open access to all of the underlying details of an FPGA architecture, as an instrument for study and for implementation.

ZUMA can act as a compatibility layer, allowing interoperability of designs and bitstreams between different vendors and parts, in an analogous manner to a virtual machine in a computing environment. ZUMA can also be used to embed small amounts of programmable logic into existing FPGA based systems without relying upon the vendor's underlying partial reconfiguration infrastructure. The embedded logic will be customizable for specific tasks. More importantly, it can be made portable across different FPGA vendors that have different mechanisms for partial reconfiguration. This also allows for sections of the design, such as glue logic, to be reconfigured without going through an entire CAD iteration with vendor-specific tools.

ZUMA is also a tool for FPGA research and education. ZUMA's design is intended as a prototyping vehicle for

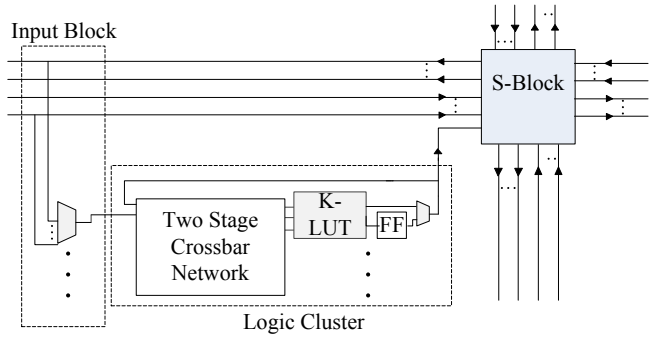

Figure 1. ZUMA tile layout and logic cluster design (note: inputs are distributed on all sides in real design)

future programmable logic architectures. Although incompatible with some architectural features such as bidirectional routing, ZUMA is useful for testing features that go beyond simple density or speed improvements and offer new functionality. Finally, the open nature of the design allows for an open tool flow from HDL-to-bitstream, amenable to current FPGA CAD research. Though a compilation flow is in place with VTR, a new approach to ZUMA tools, eg. based on JHDL, Lava, Torc, OpenRC, or RapidSmith can also be created. For example, one key application would be ultra-lightweight place-and-route tools that can run on a soft processor implemented in the same FPGA as ZUMA (not implemented in ZUMA, although that would be possible as well).

II. GENERIC BASELINE ARCHITECTURE

In designing the ZUMA FPGA architecture, we first created a purely generic 'vanilla' LUT-based FPGA architecture supported by VPR. The initial generic architecture design was created as a parameterized Verilog description. The design is similar to the standard architectures used in classical VPR experiments [5]. Since bidirectional routing is not suitable to implement or emulate in modern FPGAs, a unidirectional routing architecture is used instead. As unidirectional routing has only one driver per wire, the routing S block and output C block must be combined [2].

The generic logic cluster is comprised of a first stage depopulated input block for cluster inputs, followed by a fully populated internal crossbar, which is connected to the N K-LUTs. A total of I inputs are fed into the cluster by the input block, while all I inputs and N feedback signals are available to any basic logic element (BLE) input pin. The BLEs are single K-LUTs, followed by a single flip-flop which can be bypassed using a MUX. This generic version contains configuration bits stored in shift registers. Resource usage is detailed in table I.

III. ZUMA ARCHITECTURE

The implementation of the ZUMA FPGA takes the baseline architecture outlined in the previous section, and utilizes

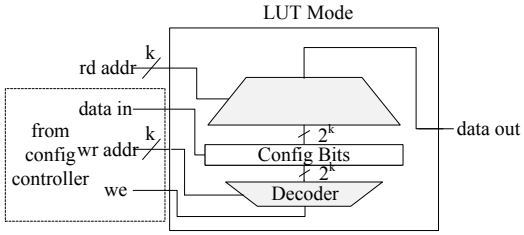

Figure 2. LUTRAM used as LUT

the unique resources available on a modern FPGA to minimize the area overhead. The design has been implemented on both Xilinx and Altera FPGAs.

A. LUTRAM Usage

The ZUMA architecture takes advantage of the reprogrammability of LUTs in the host architectures to create space efficient configurable logic and routing. In new generations of Altera and Xilinx FPGAs, logic LUTs can be configured as distributed RAM blocks, called LUTRAMs. Both vendors allow fully combinational read paths for these RAMs, permitting them to be used as normal k-input LUTs (figure 2). These are useful for directly implementing the programmable eLUTs of the ZUMA architecture, but they can also be used to improve the implementation efficiency of the routing network. By limiting the LUTRAM configurations to a simple pass-through of one input, a k-input LUTRAM becomes equivalent to a k:1 routing multiplexer. These MUXs will be the backbone of the global and local interconnection networks of the ZUMA FPGA. These LUTRAM MUXs consume fewer resources than MUXs implemented with generic Verilog constructs, as they require fewer host LUTs and configuration flip-flops due to the lack of configuration bits. As well, to save power by preventing unneeded switching when a routing MUX is inactive, each MUX in the generic version needs to be able to be configured to output ground, which can also increase resource usage, while the LUTRAMs can simply be configured by setting all configuration bits to zero. The difference in resource usage is illustrated in figure 3.

B. Clos Network Local Interconnect

The design of the internal connection block of the FPGA cluster is driven by a need for area efficiency, flexibility,

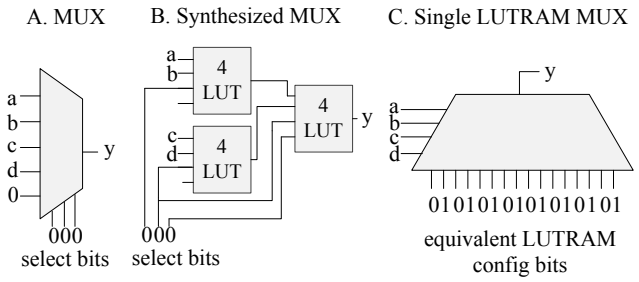

Figure 3. A: 4 to 1 MUX, B: 4-to-1 MUX synthesized from 4-LUTS, C: 4-to-1 MUX created one 4 input LUTRAM

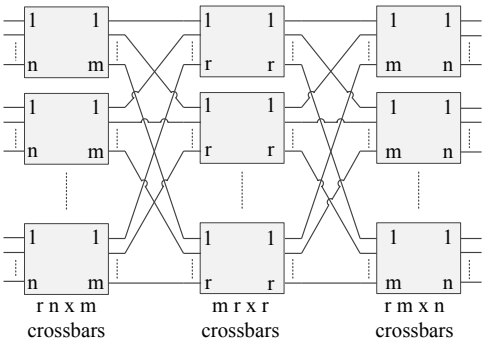

Figure 4. Clos network

and CAD compatibility. An IIB based on a Clos network was designed. This modified shuffle network is paired with a depopulated first level connection block of MUXs, in order to allow sufficient routing flexibility, including compatibility with modern CAD tools.

A Clos network (figure 4), is composed of three stages of connected crossbars. The number of inputs and outputs are identical. It is defined by three parameters: n, m and r, which define the number and dimensions of the crossbar stages. As long as m is greater than or equal to n, the network can route any input to any output [1]. Given that we have 6:1 MUXs as our basic switching element for a 6-LUT host FPGA, a first approach is to set the parameters as r=n=m=6. This allows us to build a 36 x 36 Clos network out of 108 6:1 MUXs. If we take the outputs of the network to be the inputs of the 6-LUTs, we can see that the last stage can be eliminated, as the order of the input to each LUT is unimportant, and the inputs to each LUT will always be different. We then have a two-stage network with 72 MUXs, feeding into a possible 6 ZUMA eLUTs, giving a total cost of 12 MUXs per eLUT for the IIB, with no reduction in routability(figure 5). In contrast, a full crossbar implemented with the same elements would require seven 6-LUTs per input, or 42 total MUXs per eLUT.

If the inputs are connected directly to the global routing tracks, only 36 - N tracks will have access to the LUT, since N of the inputs will be reserved for LUT feedback connections. Instead, additional MUXs are added before the first stage to give adequate flexibility, giving an overall design that can be routed with a VPR without extensive modification.

When implemented, we found that building a single 6-to-N memory from LUTRAMs is more efficient than N 6-to-1 memories on both Altera and Xilinx platforms, as each LUTRAM has overhead for write ports. All LUTRAMs used in a crossbar have the same input signals, and each output, when configured properly will only depend on the value of one of the inputs. As a result, each crossbar can be constructed out of a single multi-bit wide memory to increase density without changing the functionality.

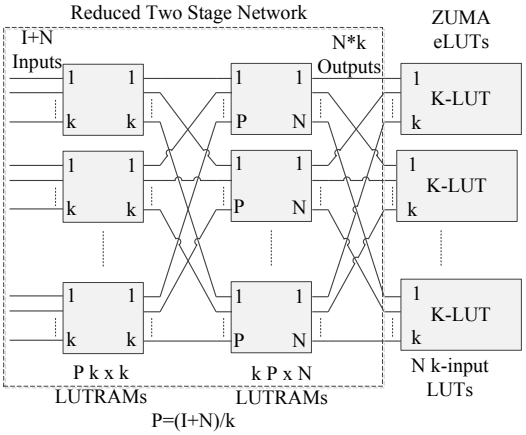

Figure 5. Modified two stage LUTRAM network

C. Configuration Controller

The design of the eFPGA also requires a configuration controller to rewrite the LUTRAMs and flip-flops that control the functionality of ZUMA's logic and routing. A configuration controller was designed to program all of the base elements of the eFPGA. The implementation is parameterized, for tradeoff between resource usage and configuration speed. For a ZUMA overlay comprised of k-input LUTRAMs, each LUTRAM has k address signals, one write-enable, and one write-data signal that must be driven by the configuration controller. The k-bit write address can be shared by all LUTRAMs in the design. However, to initialize each LUTRAM independently, a set of LUTRAMs can share a write-enable or write-data signal with each other, but not both. Each unique write-data or write-enable will require at least one additional LUT or flip-flop on the FPGA to drive the signal. ZUMA's LUTRAMs are divided into groups by location, made up of one or more ZUMA tile, and given separate write-data signals, while sharing a write-enable. The groups are written to serially, therefore a shift chain is used to propagate the write-enable between each group. This will utilizes only one additional flip-flop per write-enable signal. A 6-bit counter is then used to set the LUTRAM write addresses. For a large design, each group will add only one additional flip-flop to the design. By increasing the number of LUTRAMs programmed simultaneously, faster configuration can be achieved at a higher area cost.

D. Architectural Model for Area and Routing Efficiency

The correct choice of architectural parameters is very important for the efficiency of the eFPGA, due to the coarseness of the resources of the underlying fabric. Modeling studies were performed to explore optimal settings for the ZUMA FPGA parameters as well as the impact of the host FPGA architecture on the overall mapping efficiency. Given the base resource used in our design, for example a 6 input LUTRAM, parameters that create structures that fit exactly into these LUTs will be most efficient. The internal connection network is most efficient for a 6-input

	Xilinx Virtex 5 Implementation				Altera Stratix IV Implementation			
	Generic Architecture		ZUMA Architecture		Generic Architecture		ZUMA Architecture	
	Host LUTs	Host FFs	Host LUTs	Host FFs	Host LUTs	Host FFs	Host LUTs	Host FFs
Switch Block	121	156	104	0	121	156	121	156
Input Block	56	288	56	0	56	84	56	84
Crossbar	288	248	144	0	288	248	152	137
BLEs	528	520	16	8	528	520	16	40
Total	993	1212	320	8	993	1008	345	417
Per eLUT	124.1	151.5	40	1	124.1	126	43.1	52.1

Table I
RESOURCE BREAKDOWN PER CLUSTER

host LUT when the number of inputs is 36 or lower. The formula for an adequate number of inputs to a cluster is given in [5] as (N+1)*k/2. Given that N inputs that must be feedback connections for each LUT, we have the formula (N+1)*k/2+N = 36. Fixing K at 6 and solving for N, we get a cluster size of eight 6-LUTs, which requires 27 inputs, and a 35 input crossbar. The resource requirements for the ZUMA architecture were independently modeled based on profiling each building block, and minimum area overhead was obtained at N=8 eLUTs per cluster. The routing width is fixed at 112, which was determined by experiments in VPR to find the minimum width allowing the routing of all MCNC benchmarks for this architecture.

IV. RESULTS

A. Xilinx Results

Given eight 6-LUTs per cluster, a routing width of 112 and routing wire length of 4, an absolute cluster input flexibility of 6, a fractional cluster output flexibility of 3/8 and a switch block flexibility of 3, the resources consumed per ZUMA tile are presented in table I, as compiled on a Xilinx Virtex 5 FPGA. The results are a two thirds reduction in LUT usage, and an almost complete reduction in flip-flop usage, in comparison to the generic implementation.

B. Altera Results

The Altera version is currently less efficient than Xilinx, as a single LUTRAM mapped to the device consumes an entire LAB, plus additional registers, regardless of the width of the LUTRAM. This greatly restricts the size of the array that can be implemented on a single FPGA, though we are hopeful that a more efficient LUTRAM mapping may be available in future software releases. Given this drawback, the switch block and input block are implemented in the ZUMA version with generic Verilog multiplexers. Given the same architectural details as in the Xilinx results, the resources consumed per tile are presented in table I, as compiled to a Altera Stratix IV FPGA. LUT usage is reduced similarly to Xilinx when compared to the generic version, while flip-flop usage is halved.

V. CONCLUSIONS

This paper presented the ZUMA overlay architecture. ZUMA is an open, cross-compatible embedded FPGA architecture that compiles onto Xilinx and Altera FPGAs. It is designed as an open architecture for open CAD tools. The ZUMA overlay is intended to enable both applications for FPGAs and further research into FPGA CAD and architecture. Starting from an island style FPGA architecture, modifications to increase density and resource usage when compiled to a host FPGA were introduced, while preserving the functionality and CAD tool compatibility with VPR. Through utilization of LUTRAMs as reprogrammable MUXs and LUTs, we are able to reduce the resource overhead of implementing the overlay system by two thirds. Our modified Clos network internal crossbar makes use of these elements, as well as taking advantage of the routing requirements of the cluster architecture, to allow the use of CAD tools such as VPR while reducing resource usage. Through study and modeling, we found guidelines for architectural parameter sets that map efficiently to various hardware host architectures. With careful selection, an overhead of 40x can be created while still maintaining routability and mapping efficiency for user designs.

ACKNOWLEDGMENT

Our thanks to NSERC for funding this research.

REFERENCES

[1] C. Clos: A Study of Non-Blocking Switching Networks, *Bell System Technical Journal*, pp. 406-424. March 1953.

[2] G. Lemieux, E. Lee, M. Tom, and A. Yu, Directional and Single-Driver Wires in FPGA Interconnect, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, Brisbane, Australia, pp. 41-48, Dec. 2004.

[3] J. Rose, J. Luu, C-W Yu, O. Densmore, J. Goeders, A. Somerville, K.B. Kent, P. Jamieson and J. Anderson. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing, *in Proceedings of the 20th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 77-86, February 2012.

[4] R. Lysecky, K. Miller, F. Vahid, and K. Vissers. Firm-core Virtual FPGA for Just-in-Time FPGA Compilation. *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, 2005.

[5] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Boston: Kluwer Academic Publishers, 1999.