XUMA: An Open FPGA Overlay Architecture

Alex D. Brant
Dept of ECE, UBC
Vancouver, Canada
alexb@ece.ubc.ca

Guy G.F. Lemieux
Dept of ECE, UBC
Vancouver, Canada
Lemieux@ece.ubc.ca

DRAFT – Please do not distribute

ABSTRACT

This paper presents the XUMA open FPGA overlay architecture. It is a free (as in both free speech and free beer), 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." XUMA is an island style architecture that can be compiled into a host commercial FPGA. This approach has a number of benefits, including compatibility of the bitstream 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 [17]. Through careful architectural and implementation choices to exploit low-level elements of the host architecture, XUMA reduces this penalty to 40x. Using the VTR (VPR6) academic tool flow, we have been able to compile the entire MCNC benchmark suite to XUMA. We invite authors of other tool flows to target XUMA as well.

Categories and Subject Descriptors

B.7.1 [**Types and Design Styles**]: Gate arrays

General Terms

Design, Performance, Standardization

Keywords

Overlay Architecture, Routing Architecture, Embedded FPGA

1. INTRODUCTION

FPGA devices are often used to replace ASIC designs or emulate them before they are manufactured, as well as implement designs during research and prototyping stages. However, due to their complexity, FPGA devices themselves are rarely implemented. Instead, the design of FPGA devices, and the FPGA CAD tools, is limited to a few companies that have deep pockets. Hence, access to low-level details about commercial FPGAs is often limited or sometimes 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 all major FPGA vendors. However, FPGA devices contain highly optimized and complex structures that should not always be exposed to end users. For example, PLL configuration may be too complex to document while some bitstream patterns have the capability to burn out the chip (eg, creating short circuits, or implementing massive numbers of ring oscillators in the interconnect). Also, FPGA vendors wish to preserve their competitive advantage by hiding details, and to retain their locked-in customers by preventing migration. Hence, it seems extremely unlikely that vendors will be motivated to produce a truly open and portable environment.

Despite the commercial resistance to such openness, an open flow could enable new types of CAD algorithms such as superfast place-and-route tools for productivity, or super-optimal tools that run very slowly but outperform all standard approaches. Or, it could lead to entirely new uses for FPGAs, e.g. as computational engines that are targeted by a JIT-like compilation process using a standard interface layer.

This paper presents the XUMA embedded FPGA architecture. XUMA is a free, open, cross-compatible eFPGA architecture that compiles onto Xilinx and Altera host FPGAs. It is designed as an open architecture for open CAD tools. It is intended for 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.

The key contributions of this paper are:

- adoption of configurable LUT-RAMs as the basis for implementing both programmable LUTs and routing MUXs
- design of a Clos-style 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 best parameters to map efficiently to a given base architecture.

The XUMA implementation makes use of the lowest-level programmable element common in modern FPGAs: the LUTRAM, which is a reprogrammable look-up-table configured to operate in a RAM mode. The LUTRAM is used for both the logic and routing structure of the XUMA overlay architecture. To further reduce area, an intra-cluster connection block is designed based on a modified Clos network. The configuration of XUMA is performed by a configuration controller which can be configured for either minimal resource usage or faster configuration time. The addition of heterogeneous elements such as multipliers and memories has not yet been done, but it could be enabled by the creation of multi-tile spanning blocks. Certain parameter sets map most efficiently to various hardware sets, and the optimal settings are explored and discussed. As a result, the overhead of XUMA can be as low as 40x.

Although an overhead of 40x might be considered high, it is roughly the same overhead of a circuit implemented on the FPGA compared to a custom ASIC implementation [12]. However, through careful studying of overheads in XUMA, we hope to motivate FPGA vendors to adopt more XUMA-friendly structures which can reduce this overhead significantly in the future. These types of changes may help FPGAs become more efficient at implementing other types of circuits as well.

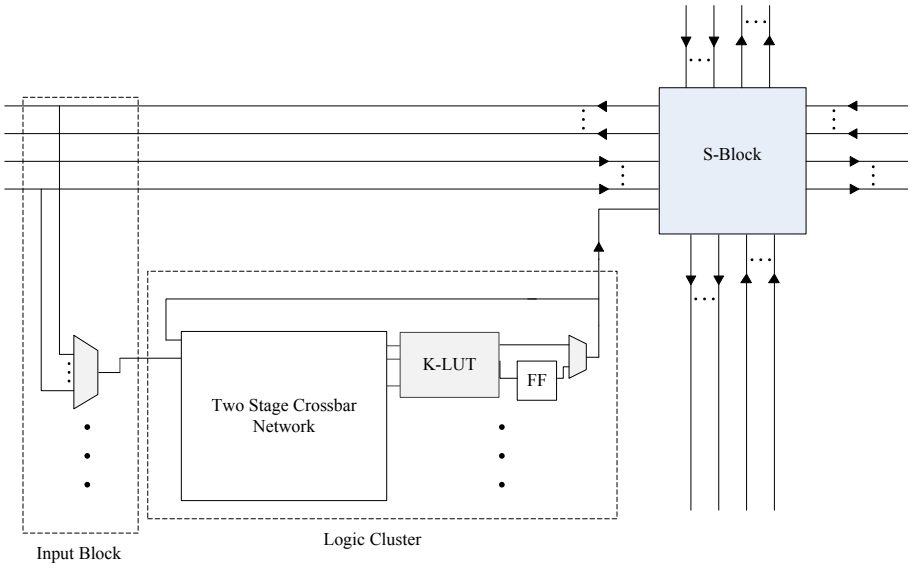

Figure 1: XUMA tile layout and logic cluster design (note: inputs are distributed on all sides in real design)

1.1 Motivation

The development and improvement of an FPGA-like overlay architecture is motivated by a variety of applications and research needs. As well, it is motivated by the strong need for improved designer productivity. XUMA can help address these needs by providing open access to all of the underlying details of an FPGA architecture, as an instrument for study and for implementation. An overlay architecture is probably too inefficient for performance-critical designs, but these are not the target application. Instead, the target applications are those which can benefit from the creation of new layer of functionality which is not available today with closed architectures.

XUMA can act as a compatibility layer, allowing interoperability of designs and bitstreams between different vendors and parts. In this manner, it can act in an analogous manner to a virtual machine in a computing environment. Identical embedded FPGAs can be implemented in many different systems, and designs can be compiled and run on all systems. In effect, a design becomes "route once, run anywhere."

XUMA 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 prospects of embedding an FPGA into an ASIC (in an SoC, for example) were not well received; for example, a promised partnership between IBM Microelectronics and Xilinx [3] did not have commercial success. Why was it not a success? Perhaps the area overhead of an eFPGA was just too large, or perhaps there was too much uncertainty selecting which portions of an SoC logic should be placed into the eFPGA. Finally, perhaps the cost of getting it wrong was just too great to risk. Instead, making small amounts of programmable logic available to FPGA users makes more sense. Unlike an ASIC, there is nearly zero risk in getting it wrong, and changing which part of the system is made reprogrammable can be made late in the design stage. Perhaps the area overhead remains as a significant barrier, but this is one we wish to address through this work and further research.

Much like an FPGA can be used as a reconfigurable processor, XUMA can act as a runtime reconfigurable accelerator for FPGA-based designs as well. 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.

XUMA is also a tool for FPGA research and education. XUMA's design is intended as a prototyping vehicle for future programmable logic architectures. Although incompatible with some architectural features such as bidirectional routing, XUMA is useful for testing features that go beyond simple density or speed improvements and offer new functionality. For example, by adding multiplier blocks to XUMA, a burgeoning architect can come to appreciate the increased routing demand that may be required from the large number of inputs and outputs, as well as the difficulty of spanning multiple layout tiles.

Some other possible areas of use for XUMA are: to implement superfast partial reconfiguration (eg. Algotronix/Xilinx XC6200 [9]), for reconfiguring only a small part of the FPGA on the fly, or for implementing hybrid architectures that make use of fine-grain LUTs alongside coarse-grain computational resources (eg, Malibu [8]).

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 already in place with VTR, a new approach to XUMA 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 XUMA (not implemented in XUMA, although that would be possible as well).

1.2 Prior Work

An FPGA overlay architecture is a design mapping an FPGA to another programmable architecture. Overlay architectures were first used to describe the design of time-multiplexed and packet-switched routing networks implemented on FPGAs [10]. Previous designs of overlay architectures for FPGAs have mapped the FPGA to a number of different categories.

A soft processor can be considered a form of overlay architecture, as it allows the user design in the form of a program to be run on

the FPGA. There are many soft processor designs for FPGAs that run the gamut from RISC processors (e.g. Nios II or MicroBlaze), to multiprocessor systems [18], to vector processors [4][21].

There has been work done on mapping coarse grain resource arrays (CGRAs) to FPGA systems. CGRAs are similar to FPGAs, but with a larger resource such as an ALU as its basic element, which usually is scheduled between various computations at different time slices. The QUKU CGRA overlay [19] is an example of a system designed to run on top of an FPGA, which runs user applications utilizing many coarse grain logic elements in parallel.

There has also been previous work on using FPGAs to implement custom FPGA architectures. VFPGA [17] is an FPGA overlay architecture that is meant to work with just-in-time (JIT) compilation of FPGA designs for acceleration of sequences extracted from the dataflow graphs of compiled programs. The virtual FPGA architecture uses a cluster comprised of 4 3-LUTs, and a routing width of 32. The design has a 100x LUT per LUT overhead (on 4-LUT Xilinx Spartan-IIe FPGA), which we improve upon by a significant factor.

Embedded FPGAs have been explored in research since the early 2000's. FPGAs synthesizable by ASIC CAD tools were explored in [1] and [20].

2. GENERIC BASELINE ARCHITECTURE

In designing the XUMA FPGA architecture, we decided to first design a purely generic 'vanilla' LUT-based FPGA architecture that is supported by VTR. Then, by studying the implementation results and overheads, the architecture and implementation can be tuned to utilize resources as efficiently as possible.

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 [2]. 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 [13]. This generic version contains configuration bits stored in FF-based shift registers.

2.1 Cluster Design

The generic logic cluster is comprised of a first stage depopulated C block for CLB 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 C block, while all I inputs and N feedback signals are available to any BLE input pin. The BLEs are assumed to be single K-LUTs, followed by a single flip-flop which can be bypassed using a MUX.

The input interconnect block, or IIB, connects the global routing wires and feedback signals to the inputs of the LUTs. The C block forms part of the IIB, while the rest of it is formed by the internal connections from the I+N signals to the k*N BLE inputs. Normally, these internal connections comprise a significant percentage of the area of a modern FPGA [14]. Many FPGA CAD tools, such as VPR, also expect fully populated internal connections. Depopulated clusters can reduce the area significantly, but complicate the placement and routing of designs, and can lead to much higher CAD runtimes [15]. To keep our tools simple, we chose to allow full routability internal to the cluster. Instead, other implementation optimizations were performed in the next section.

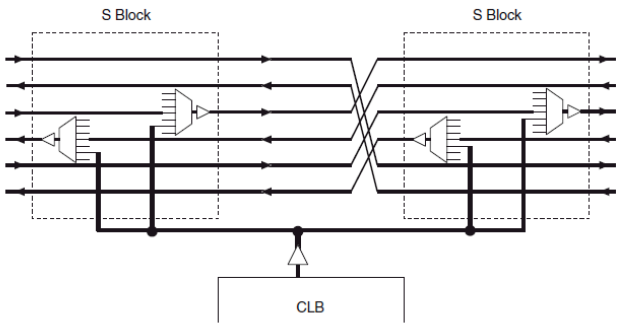

Figure 2: Global interconnect with width of 6 and length of 3

2.2 Global Interconnection Network

The routing architecture of the XUMA eFPGA is designed with a number of constraints introduced by the FPGA fabric on which it is built. Bidirectional wires are implementable from Verilog descriptions on FPGAs, but with significant overhead incurred as they are emulated with unidirectional wires. As wires in a standard FPGA can have a vast number of drivers, this approach is not feasible. In the XUMA architecture, a unidirectional interconnection network is constructed using the LUTRAMs as drivers.

The impact of unidirectional routing is explored in [13]. As each wire can have only one driver, one of the major impacts of unidirectional routing is the combination of the Switch Block and Output Connection Block. Each direction (north, south, east, and west) is driven by a single MUX located in the S-Block, which takes inputs from both other routing wires and neighboring logic blocks, as in Figure 2.

2.3 FPGA I/Os

FPGA I/Os are included at the periphery of the array. They follow roughly the same design as the inputs and outputs of the cluster, integrating with the switch block to drive wires, and connecting to a subset of wires to drive outputs.

2.4 Heterogeneous Blocks

Heterogeneous blocks, such as multipliers and RAMs, are extremely important to modern FPGAs, and capabilities to integrate them are included in the XUMA architecture. Features such as memories and multipliers provide challenges both in integration into the FPGA fabric and implementation from configurable blocks already in the FPGA.

As configurability of hard blocks such as memories and multipliers is often lost when compiled into a user design on the host FPGA, we are unable to offer the same degree of configurability in our architecture. Configurability can, however, be supported by combining multiple smaller blocks during compilation. One mode of the block will be fixed at the compilation time of the eFPGA. Different configurations can then be implemented by combining the blocks together using glue logic from XUMAs eLUTs. Configurability can also be achieved by synthesizing memories and multipliers that are configurable by design, which will make better use of the heterogeneous resources, at a higher fixed cost of host resources.

2.5 Initial Resource Usage

The generic architecture was created entirely from standard, portable, fully synthesizable Verilog. All configuration bits are stored in a shift register, which is inferred to use FFs of the host FPGA.

The results of synthesizing a single layout tile of the generic architecture on a Virtex 5 part for an architecture with cluster size of N=8, eFPGA LUT size of k=6, wire length of L=4, and channel width of W=112, are given in Table 1. The biggest use of resources is the embedded BLEs, or eBLEs, which contain the embedded LUTs, or eLUTs. Each requires almost 64 host LUTs and flip-flops per eLUT. The fully connected internal crossbar is the next-largest use of resources, at about 36 host LUTs per eLUT. Next, the C input block uses a lot of host FFs to hold its configuration data.

Table 1. Implementation of Generic Architecture on Xilinx Virtex 5 (for cluster size N=8)

Area	Host LUTs	Host FFs
Switch Block	121	156
Input Block	56	288
Internal Crossbar	288	248
eBLEs	528	520
Total	993	1212
per eLUT	124.125	151.5

3. XUMA ARCHITECTURE

The implementation of the XUMA FPGA takes the fairly standard architecture outlined in the previous section, and attempts to utilize the unique resources available on a modern FPGA to minimize the overhead of implementing the architecture. The design has been implemented for both Xilinx and Altera FPGAs, although current limitations with Altera's CAD introduce large overheads.

In this section, we will first present the use of LUTRAMs to be used as both LUT and MUX elements in XUMA. LUTRAMs are the biggest key to obtaining an efficient mapping to the host FPGA. Second, we will discuss the design of the IIB, and adopt a Clos network approach to reduce area. Third, we will present the configuration controller used by XUMA. Fourth, we will present an area modeling approach to find the best architectural parameter settings. Finally, we will discuss routing bottlenecks in the host FPGA.

3.1 LUTRAM Usage

The XUMA architecture attempts to utilize the re-programmability of the LUTs of the host architectures. In new generations of Altera and Xilinx FPGAs, the 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 LUT (figure 3). These are useful to directly implement the programmable eLUTs of the XUMA 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, as in figure 4, a k-input LUTRAM becomes equivalent to a

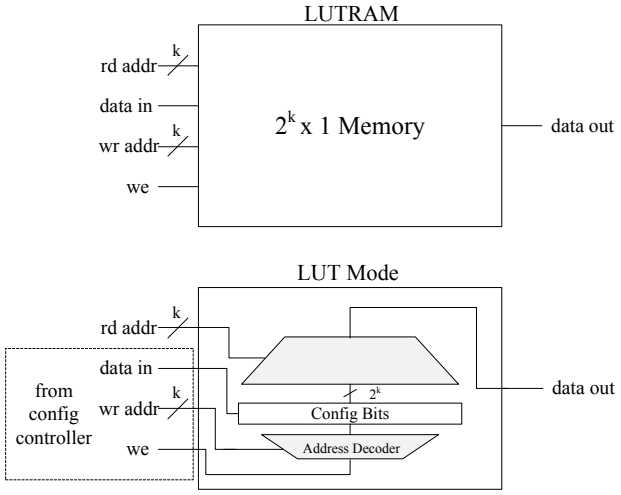

Figure 3: LUTRAM used as LUT

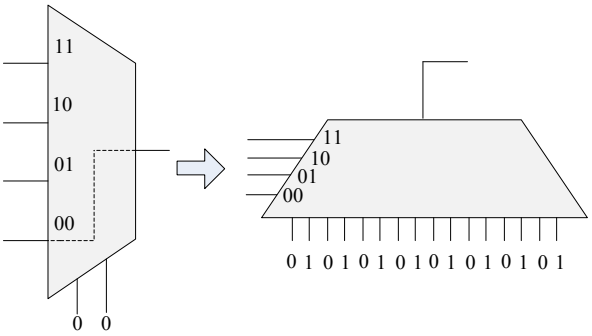

Figure 4: 16x1 LUTRAM configured as 4:1 MUX

k:1 routing multiplexer. These MUXs will be the backbone of the global and local interconnection networks of the XUMA FPGA.

To demonstrate the efficiency of a LUTRAM, consider the following. A 6:1 routing MUX synthesized from Verilog will require a minimum of two 6-LUTs and three flip-flops to be implemented. Plus, access to configure the flip-flops will be required. Instead, all of this can be provided using a single LUTRAM, which is essentially just one 6-LUT.

In a host FPGA with k-input LUTs, the k:1 MUX forms the best building block for the XUMA interconnect. As a result, an 11:1 MUX is just as efficient as a 7:1 mux in a 6-LUT host, as both require two LUTRAMs. As a result, larger muxes that are efficiently built from these LUTRAMs will dictate certain optimal sizes for the routing network.

Unfortunately, in addition to the eLUT, the rest of the eBLE will require a flip-flop and 2:1 bypass mux as well. In the worst case, each of these elements may consume an additional BLE of the host FPGA.

By proper selection of the network parameters in accordance with the LUT size of the host FPGA, the XUMA architecture can be optimized for increased area efficiency.

3.2 Clos Network Local Interconnect

The design of the internal connection block of the FPGA cluster is driven by a need for area efficiency, flexibility, and CAD compatibility. An IIB based on a Clos network was designed. It is similar in design to similar to the Type 3 IIB proposed by Feng

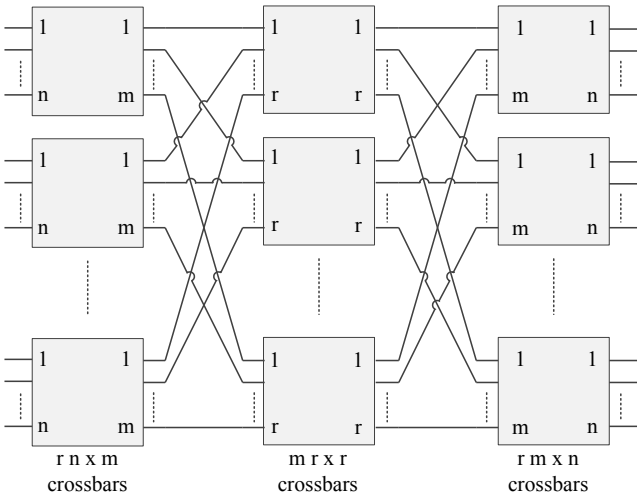

Figure 5: Clos network

and Kaptanoglu [7]. This 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, shown above, 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 [5].

Given that we have 6:1 MUXs as our basic switching element for a 6-LUT host FPGA, a first approach is to take 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 XUMA eLUTs, giving a total cost of 12 MUXs per eLUT for the IIB, with no reduction in routability. 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 if routing is difficult, giving an overall design that can be routed with a VPR without extensive modification.

If we decrease the number of inputs while preserving the output size, we can start with a 36 x 36 network and eliminate some of the second-stage MUXs that are unneeded. For example, if this network was used to drive just four 6-LUTs, the first stage would still need 36 MUXs, but the second stage would need 24 MUXs.

Similarly, a Clos network with an output width larger than 36 can be created, and then optimized for fewer inputs. Setting m=n=6 and letting r change depending on the width will still let us eliminate the last stage and satisfy the non-blocking properties of the network. The second stage will be composed of m r by r crossbars. Reducing the number of inputs (and eliminating the corresponding first stage crossbars) will also decrease the number of inputs to each of the second stage crossbars. As LUTRAMs are fixed at size 6, this will only result in an area reduction in the second stage if the first stage has six or less crossbars. Six first stage crossbars with 6 inputs each give us a network with 36 inputs in total. In this way, for example, a fully routable 36 to 48 network can be constructed out of a total of 84 MUXs, or 10.5 LUTRAMS per eLUT.

With Xilinx tools, we found that building a single 6-to-N memory from LUTRAMs is more efficient than N 6-to-1 memories, 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, as shown below.

This cluster design gives us both a modest overhead for area, as well as compatibility with a wide range of CAD tools.

3.3 Configuration Controller

The design of the eFPGA also requires the configuration controller to be able to rewrite the LUTRAMs and flip-flops used to configure the eFPGA. A configuration controller was designed to program all of the base elements of the eFPGA. Its implementation is parameterized for tradeoff between resource usage or speedier configuration. Other configuration styles, such as the use of wildcard addressing in [9], are candidates for future extensions to the architecture.

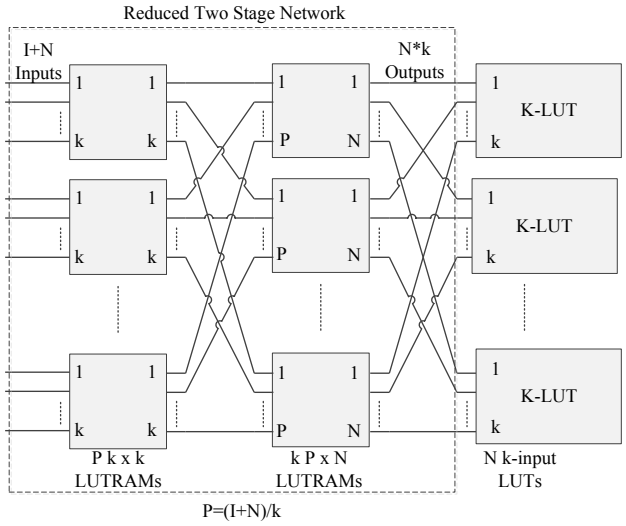

Figure 6: Modified two stage LUTRAM network

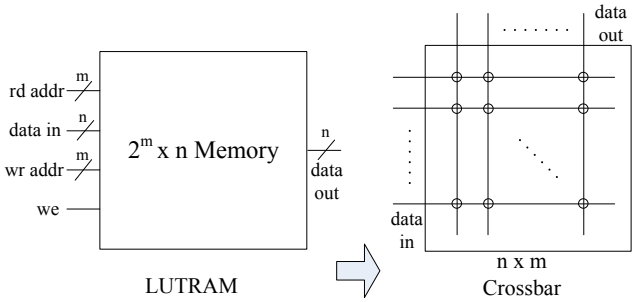

Figure 7: LUTRAM configured as crossbar

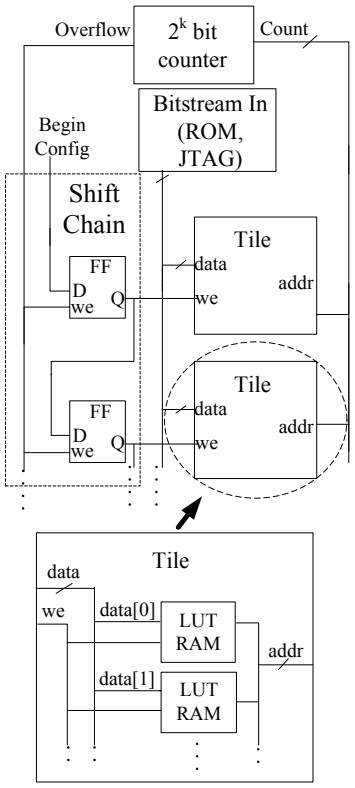

Figure 8: Configuration controller

For a k-LUT XUMA, k address signals, one write-enable, and one write-data signal are connected to each LUTRAM, and 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 another, but never both. Each unique write-data or write-enable also requires at least one additional LUT or flip-flop on the FPGA to drive the signal.

The controller design begins by partitioning the design into subsets of LUTRAM blocks driven by the same write signals so the configuration controller can easily scale with size. With the tiled design of the eFPGA, each distinct LUTRAM within a tile can be given an independent write-data signal, which it shares with the corresponding LUTRAM in the other tiles. Each tile is then given a separate write-enable signal, and all tiles are configured serially. This controller also needs at least one register for each LUTRAM in a tile, plus one register used to generate the write enable for each additional tile.

When the number of LUTRAMS in a tile, T, exceeds the number of tiles, the controller can be reduced by dividing the LUTRAMs in a tile into N roughly equal groups, fed by T/N signals, with a write enable for each group generated by the controller. By increasing the width of the data, faster configuration can be achieved at a higher area cost. This approach may also reduce the congestion in routing resources, as fewer signals will need to go into each cluster of the host FPGA when multiple LUTRAMs from a tile are shared.

In the generation of the write enable, address and data must also be carefully designed as to be area efficient with such a large number of outputs. The write enables will be written in serial, and therefore a shift chain of registers is used, which utilizes only one additional flip-flop per WE signal. A 6-bit counter is used to set the LUTRAM write addresses, and a shift chain of registers is used to generate the serial write enable signals for each group of LUTRAMs. After each 64-bit LUTRAM is written in a WE group, the shift chain is incremented by one. The design is shown in Figure 10. Currently the configuration bits are stored in a ROM in the FPGA, but will be extended for off-chip loading. On startup, the bit stream is read serially from the ROM, and used to configure the LUTRAMs.

3.4 Architectural Model for Area Efficiency

The correct choice of architectural parameters is very important for the efficiency of the eFPGA, due to the coarseness of the underlying fabric. Here, we perform modeling studies to explore optimal settings for the XUMA FPGA parameters as well as the impact of the host FPGA architecture on the overall mapping efficiency.

The careful selection of architectural parameters can have a large impact on the resource efficiency of the design. Given the base resource used in our design, for example a 6 input LUTRAM, parameters that fit exactly into these LUTs will be most efficient.

The internal connection network is most efficient for a 6-input 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 [2] as ((N+1)*k/2). Given that N inputs that must be feedback connections for each LUT, we can derive 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 routing width is fixed at 112 for these figures, that number was determined from experiments in VPR as allowing the routing of all MCNC benchmarks for this architecture.

The resource requirements for the XUMA architecture were modeled based on profiling each building block and plotted in Figures 7 and 8 for k=6 and k=4, respectively. For k=6, note that minimum area overhead is obtained at N=8 eLUTs per cluster. Likewise, for k=4, minimum area overhead is obtained at N=10 eLUTs per cluster. The actual number of host LUTs used will be higher as there will be other overheads on each platform, such as additional resource usage for LUTRAM write ports.

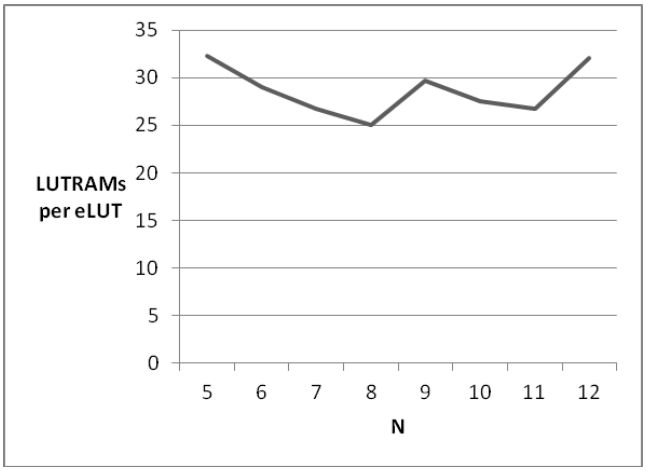

Figure 9: Density vs N for k=6

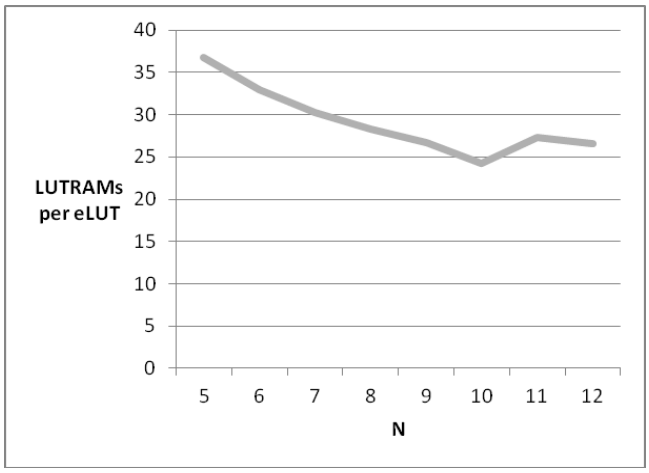

Figure 10: Density vs N for k=4

3.5 Host FPGA Routing Bottlenecks

The previous sections have focused on the usage of logic resources on the host FPGA, rather than on the routing resources. As the routing architecture will vary from FPGA to FPGA, and the final placement will be decided by proprietary CAD tools, this section will evaluate the design for suitability for placement in a modern FPGA architecture in terms of signal bandwidth requirements.

For simplicity, the analysis will focus on a ten 6-LUT per cluster host architecture. For an Altera Stratix III, the number of unique signals routed into to a cluster is given as 52. Assuming that we can fit 10 LUTRAMs into an Altera host cluster (LAB), this will require 6 address signals, 10 write data, and one write enable. As a result, there are only 35 free inputs to be shared among the ten LUTRAMs.

The Clos network, which consumes roughly half of the host LUTs in the final design, has a unique structure that will help the mapping. Each sub-crossbar of the network will use the same 6 inputs of the network for 6 or more LUTRAMs, depending on configuration. If the packing of the proprietary CAD algorithm is able to pack them in the same cluster, the cluster I/O capacity will be less stressed. In contrast, each LUTRAM used as an eLUT will require six independent inputs. The global routing MUXes will have fairly independent inputs as well. These eLUTs and MUXes are high-bandwidth LUTRAMs.

To reduce bandwidth requirements, the eLUTs and MUXes should be mixed with Clos LUTRAMs. Assuming the Clos network takes up half of the LUTRAMs (as is the case for most configurations) and assuming (conservatively) that all other LUTRAMS in a cluster have independent inputs, then 6 input signals will be required for all of the Clos LUTRAMS, and 30 input signals will be required for the eLUT and MUX LUTRAMs. Hence, a total of 36 inputs would be required, but the LAB only has 35 inputs remaining. To resolve this, only 4 or fewer eLUT and MUX LUTRAMs can be packed into the host cluster. Although this is pushing the limits of the host FPGA's cluster input bandwidth, there should still be enough Clos LUTRAMs and other stray LUTs to fill the host clusters.

4. XUMA SOFTWARE

This section details the mapping of the XUMA architecture to the host FPGA with vendor CAD tools, as well as the mapping of the user design from HDL to the bitstream that would be performed by the user of the XUMA architecture. The XUMA eFPGA will be integrated alongside a user's logic design or instantiated by itself in Verilog. In our early experimentation, compiling into the host FPGA required some care to ensure the unique requirements of the XUMA FPGA architecture are mapped efficiently using the vendor CAD tools. The user bitstream to be run on the eFPGA will also need to be generated using our open source toolflow based on VPR.

4.1 Vendor CAD tool limitations

The compilation of the XUMA system to the host FPGA may require additional configuration of the CAD tools used, as well as the overall architecture and features of the implemented overlay. Naturally occurring features such as combinatorial loops in FPGA structures are not handled well by FPGA CAD tools

In early tests with Altera tools, compilation failed after 2+ days when compiling designs that utilized 50%+ of the host FPGA. Similarly slow compilation time occurred with Xilinx tools. We expect that FPGA CAD tools are tuned to compile generic logic circuits, not the combinational loops and interconnect-intensive properties found in FPGA architectures. As a result, many features of the CAD tools must be bypassed to allow the XUMA design to be compiled to a host FPGA. Turning off timing analysis and compiler features reduced compile times to roughly one day, but was eventually successful. In Quartus, 'timing driven synthesis' was turned off, and CAD effort was reduced. In Xilinx ISE, 'timing driven placement' was turned off, the optimization target was set to area, and the 'Optimization effort' was set to 'fast'. Manually partitioning the XUMA design into smaller sub-arrays can also reduce compile time for larger arrays.

4.2 XUMA CAD tool capabilities

To synthesize circuits and generate configuration bitstreams for XUMA, we have adopted the open source VTR CAD Framework. The VTR project contains an entire toolchain and comprises the flow from "Verilog to Routing" [16]. The outputs of the technology mapping, packing, placement and routing stages are all required to generate the bitstream. Each element of the configuration is read in and entered into a data structure corresponding to the architecture of the eFPGA. This description is then serialized into a binary bitstream.

The configuration of the XUMA eLUTs is determined using the ABC technology mapping tool, after elaboration by the front end syntheses tool ODIN-II. These configurations are stored, and then packed into clusters from the output of AAPack. The cluster locations are determined from the output of the placement stage, in the VPR place file. The generation of the global routing LUTRAM bitstream is done from the output of VPR route file. The path of each net in the design is specified in the '.r' file by the sequence of channel locations the signal is routed on. The XUMA tool follows each path, and stores the switch setting for each MUX along the way in the data structure. Once all nets have been read, the drivers for all remaining (unused) wires are configured to logic 0. This way, unused wires don't toggle unnecessarily and waste power.

The local interconnect bitstream is then generated. The location of all input pins and LUTs is determined from the placement, and the Clos network configuration must be determined from the connectivity, from the crossbar configuration algorithm as outlined in [5].

Table 2. Track Utilization for XUMA architecture using VPR 6.0

Benchmark	6-LUTs	Array Size	Minimum Routing Track Width
alu4	1173	13x13	60
apex2	1478	14x14	102
bigkey	907	16x16	46
des	554	18x18	50
diffeq	1245	11x11	80
dsip	904	16x16	50
elliptic	3255	17x17	100
ex1010	3093	20x20	110
ex5p	740	10x10	98
frisc	3814	20x20	112
misex3	1158	13x13	78
pdc	3629	22x22	104
s298	1309	13x13	74
s38417	4501	21x21	78
Seq	1325	13x13	104
Spla	3005	20x20	106
Tseng	1182	10x10	76

Once all of the configurations have been generated, they must be packed into a serial bitstream. The packing is performed based on the configuration controller data width, and the ordering of the LUTRAMS in the design.

The latest version of VPR, 6.0, allows user specification of more complex logic blocks and architectures. However, the XUMA architecture does not currently utilize any advanced architectural features such as multipliers or block RAMs; these are left for future work.

The toolset is still in development. The data structures have been created, and the VPR placement and routing file decoding is currently functional, but the LUT configuration from the ABC output is still being written.

4.3 XUMA Timing

The current timing features used in XUMA are still primitive. The worst-case delay is extracted from the routing fabric and LUTs from the vendor timing tools, and used in the CAD and final timing of the user design. The worst-case delay for the configuration path of each component is also used for the configuration of all elements in the design.

Since the timing can differ from compilation to compilation, some re-computation of timing may be needed to run a bitstream between different host FPGAs. One way to eliminate this step is to assume a timing model common to all instances of a XUMA configuration, and scaling to the worst case deviation from it by the physical instance, thus ensuring the design will perform reliably on any architecture when this factor is known.

As the beta of VPR 6.0 is being used for placement and routing, which does not currently allow timing analysis, timing is not considered in the following sections.

5. RESULTS

The XUMA and generic architectures have both been implemented in both Xilinx Virtex 5 and Altera Stratix IIII FPGAs. The following section includes resource breakdowns for each version, results from the compilation to host FPGAs, as well as resource utilization from placement and routing with VPR 6.0.

5.1 CAD Results

The XUMA architecture was mapped to the VPR 6.0 architecture format. The 20 largest MCNC benchmarks were compiled onto the architecture, and the total number of tracks required to route the design was recorded. As show by the results in table 3, 112 tracks with a uniform wirelength of 4 is sufficient to route all designs. A small increase in needed tracks was incurred due to decreased input flexibility into the cluster compared to the default settings. VPR is run with a present overuse penalty factor of 1.1 and a maximum router iteration count of 200. Some benchmarks caused a segmentation fault in the latest version of VPR 6.0 upon parsing and are not included.

5.2 Xilinx Results

Given eight 6-LUTs per cluster, a routing width of 112 and routing wire length of L=4, a cluster input flexibility of 6, a cluster output flexibility of 3/8 and a switch block flexibility of 3, The resources consumed per XUMA tile are presented in table 3, as used on a Xilinx Virtex 5 FPGA.

Each 6-input LUTRAM in the design, when implemented, actually consumes 2 LUTRAMs on the architecture, making the design less dense. LUT-RAMS with multiple output bits are more dense, therefore the 6x6 and 6x8 crossbars in the local interconnect are implemented with 6 and 8 LUTRAMs for increased density.

Compared to the generic architecture, the XUMA architecture is 3x more dense. Compared to previous work where a 100x area reduction was achieved (though with a smaller LUT size than the host architecture), the design is 2.5x more dense.

Currently, on a Virtex 5 device with about 70,000 slices, a 15x15 tile design with eight 6-LUTs per tile has been implemented with only LUTRAMs at an efficiency of 40 LUTs and 1 flip-flop per eLUT. With the hybrid architecture, a 22x22 array has been implemented, at a reduced area efficiency of 42 LUTs and 55 flip-flops per eLUT.

5.3 Altera Results

The Altera version is currently less efficient than the Xilinx, for reasons we hope to resolve through co-operation with Altera developers. Currently 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. We are currently in discussion with Altera, and are hopeful that a more efficient LUTRAM mapping will be available in future software releases.

As in the Xilinx version, using a hybrid synthesized LUT and LUTRAM configuration allows a larger array to be implemented. The hybrid architecture area numbers are presented here. Given the same architectural details as given in the Xilinx results, the resources consumed per tile are presented in the table below, as used in an Altera Stratix III FPGA. Due to time constraints and compile time length, the largest array size that can implemented in a Stratix chip has yet to be determined.

6. CONCLUSIONS

This paper presented the XUMA overlay architecture. XUMA 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 XUMA 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 on top of the host FPGA were made to the implementation, while preserving the functionality and CAD tool compatibility with VPR. Through utilization of LUT-RAMs as reprogrammable MUXs and LUTs, we are able to reduce the resource overhead of implementing the overlay system. Our modified Clos network internal crossbar makes use of these elements, as well as taking advantage of the routing requirements of the LUT inputs, 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. Further density increases may be achieved through access to lower level elements of the host architecture

6.1 Future work

Much work can still be done to add features and capabilities both on the CAD and architecture elements of XUMA. Timing will be further considered, both to increase the speed of the underlying architecture, and the ability of the CAD tools to exploit timing variation. The addition of more architectural features, such as multiple clock networks, and asynchronous set and reset of registers, and further work on integrating heterogeneous elements into the CAD flow will further increase the applicability of the XUMA architecture.

7. ACKNOWLEDGMENTS

Our thanks to XXXXX for XXXXXX.

8. REFERENCES

[1] V.C. Aken'Ova, G. Lemieux and R. Saleh. An improved "soft" eFPGA design and implementation strategy. *Custom Integrated Circuits Conference, 2005. Proceedings of the IEEE 2005* , vol., no., pp. 179- 182, 18-21 Sept. 2005

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

[3] A. Cataldo. Hybrid architecture embeds Xilinx FPGA core into IBM ASICs. *EE Times*, Jun. 24, 2002.

[4] C.H. Chou, A Severance, A. Brant, Z. Liu, S. Sant, and . G.F. Lemieux. VEGAS: soft vector processor with scratchpad memory. In *FPGA*, 2011.

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

[6] J. Coole and G. Stitt. Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing, *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on , vol*., no., pp.13-22, 24-29 Oct. 2010

[7] W. Feng and S.Kaptanoglu. Designing Efficient Input Interconnect Blocks for LUT Clusters Using Counting and Entropy. *ACM Trans. Reconfigurable Technol. Syst*. 1, 1, Article 6 (March 2008), 28 pages.

[8] D. Grant, C.Wang and G. Lemieux. A CAD framework for Malibu: an FPGA with time-multiplexed coarse-grained elements. *In FPGA '11*.

[9] S. Hauck, Z. Li and E. Schwabe. Configuration Compression for the Xilinx XC6200 FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*1999.

[10] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M.J. Wilson, M. Wrighton, and A. DeHon. "Packet Switched vs. Time Multiplexed FPGA Overlay Networks*," Field-Programmable Custom Computing Machines,* pp.205-216, April 2006.

[11] J. Kingyens and J.G. Steffan. The Potential for a GPU-Like Overlay Architecture for FPGAs. *International Journal of Reconfigurable Computing,* vol. 2011, Article ID 514581, 15 pages, 2011.

[12] I. Kuon and J.Rose. Measuring the gap between FPGAs and ASICs. In *FPGA*, 2006.

[13] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and Single-Driver Wires in FPGA Interconnect", *IEEE International Conference on Field-Programmable Technology*, Brisbane, Australia, pp. 41-48, Dec. 2004.

[14] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*. Boston, MA: Kluwer, Nov. 2003.

[15] G. Lemieux and D. Lewis. Using sparse crossbars within LUT. In *FPGA*, 2001.

[16] J. Luu, I. Kuon, P.Jamieson, T. Campbell, A. Ye, W.M. Fang, and J. Rose. VPR 5.0: FPGA CAD and architecture

Table 3. Resource Breakdown per Cluster

	Xilinx Virtex 5 Implementation				Altera Stratix III Implementation			
	Generic Architecture		XUMA Architecture		Generic Architecture		XUMA 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

exploration tools with single-driver routing, heterogeneity and process scaling. In *FPGA*, 2009.

[17] R. Lysecky, K. Miller, F. Vahid, and K. Vissers. Firm-core Virtual FPGA for Just-in-Time FPGA Compilation. In *FPGA*, 2005.

[18] A. Patel, C.A Madill, M. Saldana, C. Comis, R. Pomes and P. Chow. A Scalable FPGA-based Multiprocessor. *Field-Programmable Custom Computing Machines,* pp.111-120, April 2006.

[19] S. Shukla , N.W Bergmann, and J.Becker. QUKU: a two-level reconfigurable architecture. *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, March 2006.

[20] S. Wilton, C. H. Ho, P. Leong, W. Luk, and B. Quinton. A synthesizable datapath-oriented embedded FPGA fabric. In *FPGA*, 2007.

[21] P. Yiannacouras, J. Steffan and J. Rose, Vespa: Portable, scalable, and flexible FPGA-based vector processors, *CASES'08: International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2008.