Onchip Interconnect Exploration for Multicore Processors Utilizing FPGAs

Graham Schelle and Dirk Grunwald
Deptartment of Computer Science
University of Colorado at Boulder
Boulder, CO

Dual core microprocessors are currently available and higher processor-count architectures will dominate the multicore market. A complex part of these higher order multicore designs will be the interconnection scheme that exists onchip and how exactly that interconnection is best used and configured. While FPGAs currently support a variety of onchip bus interconnects, there is a gap in the tools to provide Network on Chip (NoC) exploration. A Network on Chip is a onchip packet switched network that is used for computational elements (typically standard processors) to communicate with each other. In this paper, we present our NoC emulation tool (NoCem) and provide an example memory architecture exploration platform that can be created.

I. Introduction

Dual core processors are available from a variety of major vendors and provide many new research challenges. Notably, multicore processor design places a high importance on onchip and offchip communication, while the processors themselves need little modification from standard processor design. However, typical SMP (Symmetric Multiprocessing) challenges are present including cache coherency, process scheduling, and load balancing. The underlying architecture to support this communication is currently done using buses and switches. The communication fabric is utilized for memory, I/O, and processor-to-processor communication, all bottlenecks that cannot be ignored. Therefore, exploring the architecture of the onchip interconnect is a paramount concern for processor designers.

A large body of research predicts that packet switched networks will be necessary to replace buses for onchip communication [1], [2], [3]. With dual core designs currently on the market, onchip buses work and scale perfectly well. However, as more cores are added and clock speeds increase, single bus designs cannot support cross chip communication [4], [5]. Wire delay in high speed clock domains will dominate the timing budget, leading to wires that cannot reach across the chip in a single cycle. With this observation, pipelined wires and routed switching fabric will arise.

Two types of overhead occur when using a Network on Chip. First, communication latency is increased as components are forced to use a network in order to communicate. Often, NoCs provide fast and slow communication channels, depending on how far data must travel. Additionally, this latency is often deterministic, simplifying communication protocols. The second overhead comes from using transistors to implement the NoC. This overhead is justified by viewing transistors as "free", due to the exponential density increase seen in the scaling process. These NoCs will scale with higher numbers of chips on a single die and provide a higher bandwidth communication medium.

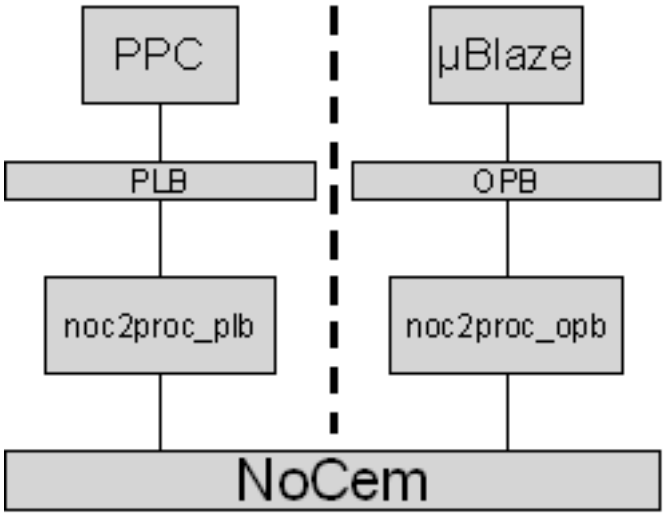

Fig. 1. Block Diagram of Processors Connected to NoCem

For researching multicore processor architecture, modern FPGAs have proven to be a valuable platform. For example, the largest Xilinx Virtex4 FPGAs hold up to 2 embedded PowerPC processors and a user can instantiate multiple soft processors in a design. For soft processors, the Xilinx Microblaze is a 32-bit Harvard RISC architecture. Accompanying these embedded and soft processors are a variety of buses and peripherals that can be utilized to communicate between processors and memories. With this base, many aspects of multicore architectures can be tested and explored. However, a NoC emulation tool does not exist that can be configured and quickly modified for a variety of testing. NoC implementations on FPGAs do exist [6], [7], but are often times nonsynthesizable, proprietary, or specific to an application utilizing it. NoCem (NoC Emulator) is a tool that is configurable and can quickly be modified to provide a variety of NoC implementations for use in multicore processor research.

II. NoCem Description

NoCem design goals: NoCem conforms to several goals in creating a tool that can emulate a variety of NoC configurations that is easily attached to current soft and embedded processors. These goals and how they are met are briefly enumerated here.

- Ease of configuration. This is done using VHDL generics and identifying what configurations are useful. NoCem currently allows configurations in terms of data width, network topology, channel FIFO depth, and packet length among other derivative parameters.
- Modularity of design. The NoC itself is modularized into channels and nodes. The nodes consist of a switching fabric and arbitration blocks. These blocks conform to simple interfaces and be swapped out with more complex internals if needed.
- Common External Interface. Each access point into the NoC has a similar interface regardless of the network internals. This will allow easy integration of the NoC into existing tools to connect FPGA logic to the processors.

NoCem Interface: The NoCem interface is very similar to that of a FIFO. Depending on the configuration of the NoC, the same lines are used for different purposes. NoCem does have a bus network configuration, though the standard Xilinx bus architectures are presumably more complete and EDK provides better tools to use them. The interface for both egress and ingress paths contain:

- *data path:* Datalines are connected to the channel FIFO for the access point into the NoC. Data is read or written using standard FIFO read and write enables.
- *packet control path:* Packet control is written to the channel FIFO for the access point into the NoC. Packet control content is dependent on the network itself. Typically, it will contain the packet header information such as source address, destination address, packet length, and other metadata. This information is used by the internal network to route the packet. NoCem currently supports destination based deterministic routing, but this can be extended in future versions. The only current restriction is the necessity for a destination address in the first word of the packet control. This restriction allows for fast routing, but NoCem could be extended to overcome this limitation.
- *channel metadata path:* These lines are used to arbitrate and grant access to the NoC by an access point. In the simplest case, these lines consist of various FIFO status signals (e.g. almost empty/full thresholds).

This interface hides any underlying network configurations that may exist. Only modifying the datawidth or packet control structure will require outside modifications to interface with NoCem.

Connection to Processors: NoCem is intended for Xilinx FPGAs and provides access point bridges (noc2proc_<bridge_name>) to both the PLB and OPB buses. These buses then connect to the embedded PowerPC and soft Microblaze processors. These bridges are easily integrated into Xilinx's EDK software, where the processors can be instantiated and connected to the NoC. Bus to NoC communication and vice versa is done using simple memory mapping of addresses which is the standard for Xilinx EDK projects. Figure 1 shows a block diagram of this bridging. An accompanying software driver is used to read and write the NoC from the processor. For sending a packet, the processor notifies the bridge of the packet destination and length. The processor can then write the packet into the NoC. For receiving a packet, the bridge interrupts the processor and allows the processor driver to "upload" the packet into the processor's memory space. A more complicated buffering and aggregation scheme could be added to NoCem, only requiring modification of the bridges themselves.

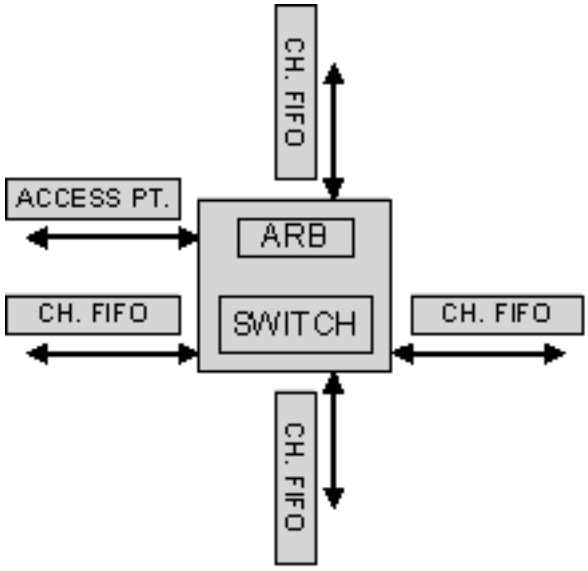

Fig. 2. Modular design of NoC nodes. Each node has an access point and a north, south, east, and west channel FIFO.

The noc2proc bridges can handle both the PLB and OPB datawidths. Both buses have a 32-bit address space, but the OPB and PLB have a 32-bit and 64-bit datawidth respectively. Since both use Xilinx's IPIF (IP Interface), this results in only minor differences in actual code.

Implementation: NoCem is implemented in standard VHDL with heavy use of generics throughout the code base To use with Xilinx's soft and embedded processors, we integrate NoCem and the accompanying noc2proc bridges into an EDK project. Using an EDK project requires some generation of files and metadata, but we use scripts wherever possible. An instantiation of NoCem can be used natively in any project as well, bypassing the scripts and bridges. However, for multiprocessor systems, integration into an EDK project is often a necessity. NoCem was created and simulated using Xilinx ISE 7.1.4, Xilinx EDK 7.1.2, and Mentor Graphics Modelsim 6.1.

NoCem was implemented with extensibility in mind. Figure 2 shows the block diagram of a common node in any

TABLE I
NUMBER OF LUTS USED BY NOCEM.

NoC Dimensions	Datawidth	LUTs	xc2vp30 LUTs used(%)
2x2	16b	4,086	14%
3x3	16b	11,693	42%
4x4	16b	21,570	78%
2x2	32b	5,822	21%
3x3	32b	16,394	59%
4x4	32b	34,370	125%

Network on Chip. Typically there are 5 entry points into the node (North, South, West, East, Access Point). We did not consider other node configurations (e.g. hexagonal networks have previously been suggested for parallel processing [8]). The channel FIFOs transmit and receive data and metadata to and from the node. How and what is transmitted is modularized to involve only the arbiter and the channel FIFO's channel_metadata signals (not shown in figure). The switch itself is an all-to-all mux that allows multiple paths of communication simultaneously. Any of these components can be switched out to add various NoC concepts (virtual channels, QoS, reservation systems) without affecting the remainder of the system.

The noc2proc bridges are stored as a Xilinx EDK peripheral, which consists of a variety of code and metadata files. Even though the bridges share almost identical code sources, they are kept separate for easy integration into EDK projects. Accompanying software drivers have been created to do simple packet receiving and transmitting.

Performance and Functional Measurements: NoCem supports buswidths up to 256b, but the larger the datawidth, the more FPGA fabric required to route wires and registers. Table I shows how for a 16b and 32b datawidth the size of NoCem increases. For a Virtex-II Pro xc2vp30, the percentage of fabric used can be seen. We chose to look at the xc2vp30, as it is the FPGA used in Digilent's XUP (Xilinx University Program) board [9] and Nallatech's XtremeDSP board [10]. These are two popular platform FPGAs and ones we possess. Interestingly, a 4x4 mesh with 32b lines cannot fit on this part. Of course, larger FPGAs can support this number of LUTs and even larger NoCs.

Currently NoCem uses no onchip BRAMs for buffering purposes on channel FIFOs, which could be seen as a way to alleviate LUT usage. This design decision was made for several reasons. Primarily, we wanted to allow for BRAM to be used for processors' onchip caches allowing for that valuable resource to be used for various memory research. Secondly, using BRAM for a channel FIFO is probably overkill, as each BRAM holds 2KB, there really is no need for that much in-network buffering on a per channel basis. The Xilinx provided LUT-based FIFOs can be configured to a variety of lengths and widths. This flexible FIFO structure allows for more complete onchip interconnect experimentation. For example, this flexibility in FIFO length could be used to confirm simulated NoC buffer sizes and its effect [11]. Delay is minimized using First-Word-Fallthrough FIFOs, which minimize the delay in valid data appearing on outputs of the channel FIFOs.

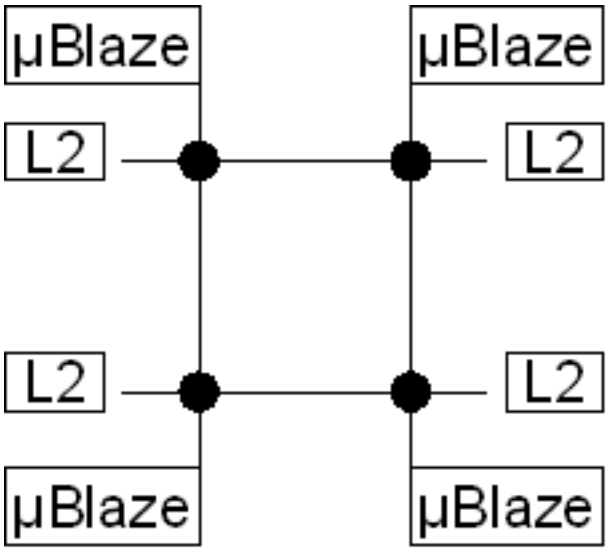

Fig. 3. Example Memory Exploration Platform Utilizing NoCem

With all the configurations, the highest sustainable clock-speed is consistently 140-150 MHz. This number would of course change depending on what is attached to NoCem. The clock used by NoCem is typically driven from the PLB or OPB generated buses, which is part of the noc2proc bridge interface.

Some latency is added in the bridging between the NoC and the PLB or OPB buses. While the bridging operations add a few cycles of delay in handshaking, it is unavoidable. Bus arbitration with other peripherals competing for the bus may also add nondeterministic delays. For this reason, extra bus-intensive peripherals should be avoided on buses communicating with NoCem if performance measurements are being taken involving the noc2proc bridge. Memory operations for data and instructions are a notable example of this observation, but cannot be avoided in some configurations. Various caching and alternative bus schemes can be used to alleviate these shared medium costs. Specifically, the PowerPC and Microblaze processors allow caches tied directly to the processor (i.e. only using bus for cache misses and not cache hits), and other buses such as the OCM can be used for processor memory.

III. EXAMPLE MEMORY ARCHITECTURE EXPLORATION

The Microblaze Processors can be configured to use onchip BRAM (Block RAM) as cache. One aspect of multicore processor research is to examine the variety of cache configurations that can exist onchip and how best they are laid out. This is an important area to examine as with the advent of NoCs, there are even more parameters to test in architecture exploration. For example, multicore research in [12] looks at how best to distribute cachelines in terms of possible process migrations, while other work looks at cache location and size tradeoffs over traditional interconnect architectures [13].

Our created platform is shown in figure 3 to examine cache configurations over NoC architectures. We took our existing NoCem tool, extended the bridge to handle memory requests, attached BRAM to the local node, and inserted the new code into a Xilinx EDK project. The steps to create this architecture are listed here to show how NoCem is used as a foundation to multicore processor research. The most time consuming aspects of the design are integrating NoCem into the EDK project flow, not configuring NoCem itself.

1) Set NoCem generics to provide a 2x2 mesh network topology with 4B words, with channel FIFOs of depth 16.
2) Create cache service layer that will sit between NoC and processor to handle local cache hits and service network and processor cache misses.
3) Create an EDK project to generate 4 Microblaze processors each on a separate OPB bus with a variable amount of L1 cache. Each OPB bus is then attached to a noc2proc_opb bridge. The EDK metadata files bridge for this exploration had to be modified slightly to be identified as a memory peripheral.
4) A toplevel file is needed to include the EDK project, the NoCem VHDL code, and our created L2 cache. This can be done using the Xilinx EDK or ISE tool flows. From either tool flow, a bitstream can be created and downloaded onto the FPGA.

In order to run a variety of tests, the NoC itself can quickly be modified by changing toplevel generics, as the interface into the NoC is held constant regardless of the underlying Network. The actual cache configuration of the entire chip takes reworking depending on what is being modified (shared vs. private, cache sizes symmetric vs. non-symmetric), but this is of course independent of the NoCem tool. The research goal of this example is to examine best ratios of memory in L1, L2 (shared or private) caches. The results of this exploration will be reported in future work, but here we present this example to show the usefulness of the NoCem tool.

IV. Conclusions

We have presented a tool that allows multicore processor designers the ability to treat NoCs as a first class citizen in architecture exploration. By providing common interfaces, modular components, and ease of parameterization, NoCem is a valuable tool in utilizing FPGAs to do multicore processor research. NoCem is easily extensible and future work will add further NoCem configurations, increasing its usefulness to the research community.

References

[1] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, N. Ranganathan, D. Burger, S. W. Keckler, R. G. McDonald, and C. R. Moore, "Trips: A polymorphous architecture for exploiting ilp, tlp, and dlp," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 62–93, 2004.

[2] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: a computational fabric for software circuits and general-purpose programs," in *Micro, IEEE*, vol. 22. IEEE Computer Society Press, 2002, pp. 25–35.

[3] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," in *Circuits and Systems Magazine*, vol. 4, no. 2. IEEE, 2004, pp. 18–31.

[4] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the Design Automation Conference*, Las Vegas, NV, June 2001, pp. 684–689.

[5] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Proceeding of the IEEE NorChip Conference*, November 2000. [Online]. Available: citeseer.ist.psu.edu/hemani00network.html

[6] L. Natvig, "High-level architectural simulation of the torus routing chip," in *Proceedings of 6'th Int'l Verilog HDL conference (IVC'97)*, 1997, pp. 48–55.

[7] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, "Interconnection networks enable fine-grain dynamic multi-tasking on fpgas," in *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2002, pp. 795–805.

[8] B. Parhami and D.-M. Kwai, "A unified formulation of honeycomb and diamond networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 1, pp. 74–80, 2001.

[9] "Virtex-ii pro development system," http://www.digilentinc.com/info/XUPV2P.cfm.

[10] "Xtremedsp kit-ii," http://www.xilinx.com/ipcenter/dsp/development_kit.htm.

[11] "A network on chip architecture and design methodology," in *ISVLSI '02: Proceedings of the IEEE Computer Society Annual Symposium on VLSI*. Washington, DC, USA: IEEE Computer Society, 2002, p. 117.

[12] P. Michaud, "Exploiting the cache capacity of a single-chip multicore processor with execution migration," in *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2004, p. 186.

[13] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multicore architectures: Understanding mechanisms, overheads and scaling," *SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 408–419, 2005.