METAWIRE: USING FPGA CONFIGURATION CIRCUITRY TO EMULATE A NETWORK-ON-CHIP

M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, R. Fong [*]

Configurable Computing Lab
Bradley Department of Electrical & Computer Engineering
Virginia Tech
Blacksburg VA 24061
{mshelbur,cdp,athanas,mtj,bm92,rfong}@vt.edu

ABSTRACT

While there have been many reported implementations of Networks-on-Chip (NoCs) on FPGAs, they have not seen the same acceptance as NoCs on ASICs. One reason is that communication on an FPGA is already costly due to the die resources and time delays inherent in the reconfigurable structure. Layering another general-purpose network on top of the reconfigurable network simply incurs too many performance penalties. There is, however, already a largely unused, global network available in FPGAs. As a proof-of-concept, we demonstrate that the Xilinx FPGA configuration circuitry, which is normally idle during system operation, can function as a relatively high-performance NoC. MetaWire performs transfers through an overclocked Virtex-4 Internal Configuration Access Port (ICAP) and is shown to provide a bandwidth exceeding 200 MBytes/sec.

1. INTRODUCTION

An FPGA's ability to implement efficient application- and algorithm-specific computational structures is as much due to abundant, custom routing as to configurable logic, arithmetic and memory blocks. This is both a great strength of the technology and a great burden for the designer. The performance of architectures with less flexible routing, such as course-grained reconfigurable arrays and multicore processors, is more often limited by communication bottlenecks than insufficient computational resources. Network-on-Chip (NoC) offers designers less complexity at the expense of less flexibility. While this tradeoff has seen success in the ASIC community, NoC implementations in programmable logic have not received widespread use. One reason is that an NoC, implemented on top of the reconfigurable routing of the FPGA, places one general-purpose routing layer on top of another general-purpose routing layer; the resulting resource and performance overheads represent a level of inefficiency not typically accepted in hardware designs.

There is a need, however, to manage the complexity and performance requirements of increasingly large FPGA designs, a task that is increasingly difficult given the current approach to communication on FPGAs. A reasonable approach is to combine current FPGA routing architectures with NoC, mapping communication within a hardware block to the routing architecture and communication between blocks to the NoC. Data transfers within a hardware block are often single-word transactions, while high-level communication between blocks tends to transfer buffers, lines, packets or frames. We propose that the configuration circuitry on an FPGA is well-suited to use as a NoC because it is a global, addressable high-speed network that is currently underutilized. As a proof-of-concept, we demonstrate that the existing configuration circuitry on a Xilinx FPGA can be used as a high-speed NoC, achieving an aggregate speed of over 200 MBytes/sec. We further show that an application can multiplex node-to-node communication links over this NoC. We then point out modifications that can be made to this configuration circuitry that will improve performance and usability without devoting significant resources.

In Section 2, we survey related work. Section 3 gives a design overview of the MetaWire controller with the implementation and testing described in Section 4. A demonstration in a signal processing application is given in Section 4.4. Conclusions are drawn in Section 5.

2. PRIOR WORK

Typical networks, such as multiprocessor shared memory systems, inter-chip networks, and LANs, are limited in the number of pins on their connectors, and hence the number of wires per link. To compensate, large buffers are implemented in the nodes of these networks to handle the reassembling of smaller width data. The constraints are reversed for NoCs [2]. The number of wires per link is much

[*]This material is based upon work supported by the United States Army under Contract Number W31P4Q-06-C-0498. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Army.

less restricted, allowing for faster and wider link transfers, but silicon area for buffers becomes a large concern. In addition, power and network reliability become important. Network links are predicted to dominate NoC power dissipation, requiring four times as much power compared to NoC logic [1]. Communication between cores in a chip cannot tolerate errors or dropped packets because cores are designed with the assumption that communication is error-free when using traditional wires. Another interesting characteristic of NoC is that all network objects (routers and network interfaces), can be synchronized together, due to the small area of the network. Global synchronization allows for routing algorithms that guarantee throughput and latency performance of network traffic [5, 3]. NoCs are also scalable in that additional links, routers and network interfaces can be added without changing the electrical properties of an existing NoC. This is especially true when networks have allocated space in a layout. Such is the case in NoCs where logic cores reside in rectangular areas, with network links and routers placed between edges of adjacent logic cores.

To reduce the penalties of FPGA interconnect scaling, Fong proposed the concept of "wire emulation" as a means for cross-chip communication in Xilinx FPGAs [4]. Communication between cores is realized through packetized data, replacing the dedicated wiring and buses once required. Logical channels can be established on the network to describe a collection of dedicated wires or buses. Wire reuse is an advantage of packet communication over dedicated wiring schemes because multiple logical channels can be transmitted over the same network link. Fong demonstrated how two or more cores can communicate within a Virtex-II FPGA without FPGA programmable interconnects between them. These data transfers are accomplished using self-reconfiguration, which is a combination of configuration memory readback and partial-reconfiguration of distributed RAM blocks. The self-reconfiguration driver, consisting of ICAP control and debug logic, consumes 7 percent of the total resources in an XC2V1000-4 FPGA. Data transfer throughput was measured to be 11.1 MBytes/sec from one source to one destination.

3. METAWIRE FRAMEWORK

MetaWire extends the approach taken in Fong's work with an emphasis on robustness and applicability, as well as by exploring the advancements offered by current and future FPGA architectures. The Virtex-4 family introduces an ICAP supporting higher bandwidth than that of Virtex-II/Pro. Additional bandwidth gain results from changes in the configuration fabric which permit the use of BRAM instead of distributed RAM as the readback/configuration target. Lastly, configuration frames in Virtex-4 and Virtex-5 span a fraction of the full device height, allowing the designer to dedicate fewer RAM resources to communication and locate those resources more flexibly.

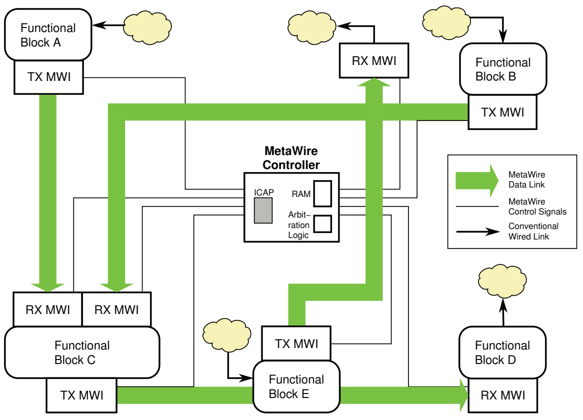

Fig. 1. Abstract MetaWire application.

Whereas Fong's prototype was demonstrated with a single point-to-point link (single-source multicast was modeled but not implemented), MetaWire's objective is to distribute the bandwidth of the ICAP across any number of point-to-point links. No provision was made in Fong's work for rate negotiation and interfacing with functional blocks, and the prototype was not demonstrated in the context of an application. Testing was performed on the basis of manually initiated transfers consisting of one readback/configuration, and data errors were not resolved. In contrast, MetaWire has been developed for sustained data transfer suitable for integration in streaming applications.

As Figure 1 illustrates, a link consists of two MetaWire Interfaces (MWIs): a transmitter and a receiver. MWIs may attach to a functional block or to a traditional wired link. The central MetaWire Controller (MWC) fulfills three roles: interfacing with the ICAP, multiplexing the configuration fabric's bandwidth, and providing up- and downstream flow control for each link. Whereas the data are moved "wirelessly" from transmitter to receiver, a small number of low-bandwidth, wired handshaking signals connect each MWI to the MWC. These signals allow the controller to support independent, time-varying data rates across each link.

The objective of MetaWire is to explore the limits of the FPGA configuration fabric in an active, data-transfer role; there are a number of refinements that could be made to the framework presented here. The hardware in the following sections is defined in Verilog.

3.1. Interface Units

The responsibility of a transmitting MWI, illustrated in Figure 2, is to collect data from the wired domain and store it to a set of BRAMs which form the transmitting RAM (TXRAM). Because data that are stored in a Virtex-

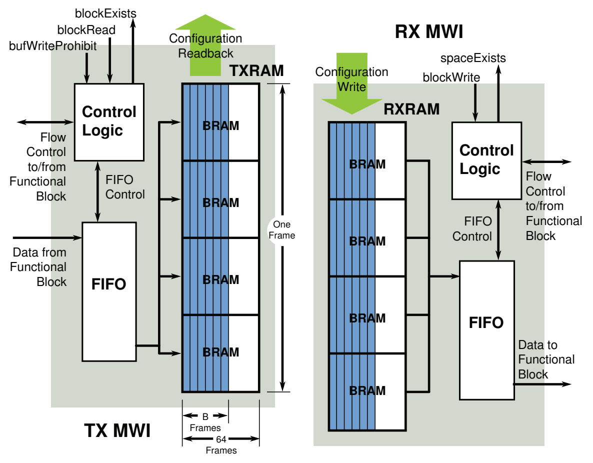

Fig. 2. MetaWire Transmitter and Receiver Interfaces.

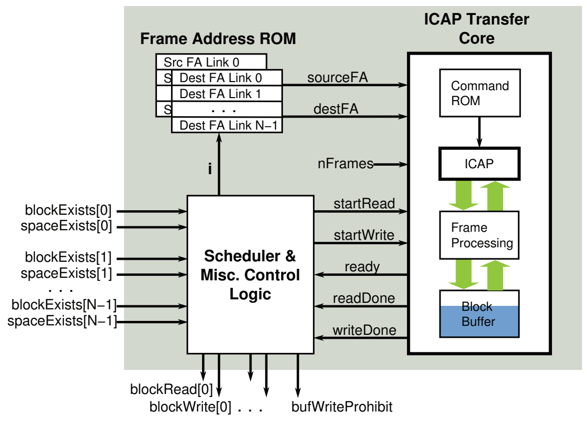

Fig. 3. MetaWire Controller.

4 BRAM via user-circuit ports are immediately reflected in the frames obtained from readback, the TXRAM serves as the data's entry point into the FPGA configuration fabric. When a transmitting MWI has accumulated a unit of data, termed a *block*, it notifies the MWC by asserting `blockExists`. The MWC performs readback via the ICAP and responds by pulsing `blockRead`, informing the transmitter that the next block may be written to the TXRAM. The `bufWriteProhibit` control is a global signal requesting all MWIs to suspend writing to TXRAM, which accommodates certain conflicts between the configuration plane and user circuits (see Section 4.1).

Figure 2 also shows the receiving MWI's structure. Frames loaded into a BRAM via active partial configuration immediately update the BRAM's contents, allowing the MWI's RXRAM to act as the data exit point from the configuration plane into the wired domain. The control signals `spaceExists` and `blockWrite` indicate that the MWI can accept a block and that the MWC has just written a block, respectively.

To fit the maximum data payload into each configuration frame, the RXRAM and TXRAM utilize all four vertically adjacent BRAMs (termed a *cluster*) that share a row of frames in Virtex-4, as shown in the figure. The size of a data block is defined by the number B of contiguous configuration frames within a cluster, which are transferred from MWI to MWI in an atomic operation. Parameter B is a system-wide constant defined at compile time, and may range from 1 frame to 64 frames (the width of one BRAM column). Blocks are treated as unstructured arrays of 16-bit words, with no packet-like header; however, the MWIs and MWC could, as an alternative to the fixed, point-to-point model, support a destination address field in each block, providing application-level access to the frame-addressable, switch-like capability of the underlying configuration fabric.

3.2. Controller

As shown in Figure 3, the central component of the controller is the ICAP Transfer Core, which performs a readback of B frames, stores them in a local buffer, then executes a partial configuration consisting of these frames. Readback and configuration are performed using the specified source and destination frame addresses (FAs), which denote the physical locations of the TXRAM and RXRAM. FAs are used generically by the configuration circuitry to identify the type and location of every resource in the FPGA.

A ROM within the Transfer Core stores the sequence of configuration commands which must be issued to the device to initiate and conclude every readback and configuration [7]. These commands include specifying the type of operation to perform (readback or configuration), the starting FA for the operation, and the amount of data to readback or configure. Writing these commands, along with the requirement to read and write a certain amount of flush data, constitute overhead in the transfer process.

Frame processing logic performs two types of operations on readback data: turning off the four save-data bits found in each BRAM frame, and conditional frame reversal. The save-data bits (which are always asserted in readback frames) indicate that existing data in the bit's respective BRAM should not be overwritten by the new frame. Frame reversal accounts for the fact that frames in the upper and lower halves of a Virtex-4 FPGA are bitwise reversed with respect to one another [7].

3.3. ICAP Utilization and Link Scheduling

The MetaWire architecture provides maximum aggregate bandwidth when the ICAP experiences minimum idle time. Figure 4 illustrates the events within the Controller and MWIs when the ICAP sees full utilization, in this case ex-

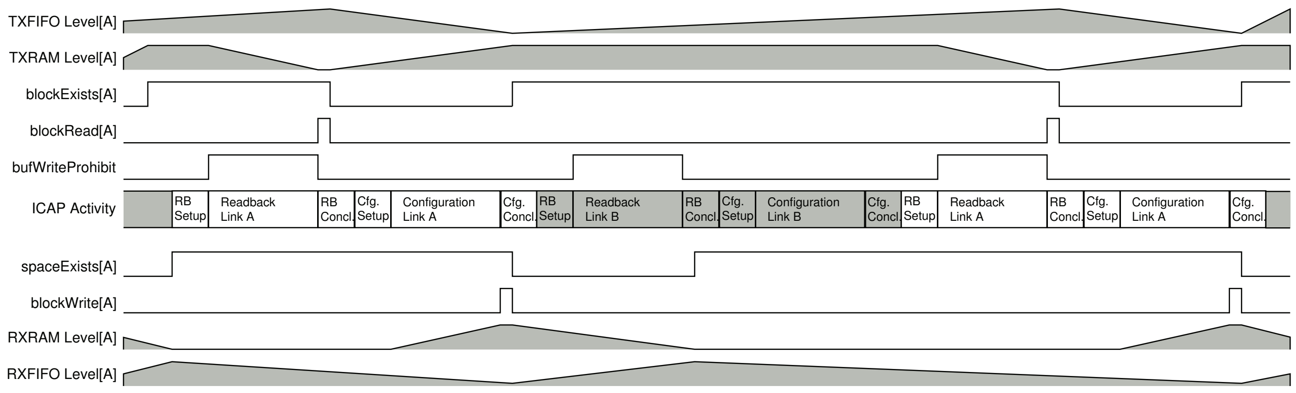

Fig. 4. Interface and controller activity with full ICAP utilization.

ecuting two block transfers on Link A, separated by one transfer on Link B. The purpose of the FIFOs, evident in the figure, is to permit the functional blocks at the links' end-points to consume and produce data steadily, without the need to be periodically halted—reducing a functional block's effective throughput—while waiting for the Controller to empty or fill the RXRAM or TXRAM.

In an N-link system, the Controller allocates block transfers in a simple round-robin fashion, sequentially scanning the status of each link. When the scheduler determines that link i meets the readiness condition, `blockExists[`i`]` `AND spaceExists[`i`]`, the Transfer Core moves one block across link i, after which the scheduler resumes the scan at link $(i+1) \bmod N$, thereby preventing a starvation condition. (When $N = 1$, readback must begin before `spaceExists[i]` is asserted in order to achieve full utilization.)

3.4. Performance Model

If the overhead introduced by the scheduler and the exchange of handshaking signals is neglected, MetaWire's aggregate throughput T is obtained from the number of payload bytes in one block, $(PL \cdot B)$, divided by the time needed to transfer one block. Table 1 describes the parameters in Equation 1 and gives their values in the current implementation.

$$T = \frac{PL \cdot B \cdot f}{\frac{WW}{PW} \cdot (2 \cdot FL \cdot B + FD + C) + MC} \tag{1}$$

The latency L is modeled as the time from when a transmitting node asserts `blockExists` to when the receiving node's `blockWrite` is asserted. The expected throughput and latency are plotted in Figure 5.

$$L = \frac{\frac{WW}{PW} \cdot (2 \cdot FL \cdot B + FD + C - WCC) + MC}{f} \tag{2}$$

Param.	Description	Value
B	Block size (frames)	1 - 64
f	Config. port clock frequency (MHz)	144
PW	Config. port width (bytes)	4
WW	Config. word width (bytes)	4
PL	Payload per frame (bytes)	128
FL	Config. frame length (words)	41
C	Config. commands, total (words)	33
WCC	Write conclusion cmds. (words)	1
FD	Flush data, read + write (words)	84
MC	Misc. wait and transition cycles	10

Table 1. MetaWire performance parameters.

4. IMPLEMENTATION

This section describes the challenges encountered in exploiting the Virtex-4 configuration fabric, followed by verification results and resource requirements. Finally, we present a signal processing application.

4.1. ICAP Limitations

Limited documentation and tool support are available for the ICAP. The following observations reflect our findings with an XC4VLX60 part in speed grade 10 and "engineering sample" silicon revision. Because MetaWire's use of the configuration fabric steps outside Xilinx's characterization and documentation, it should be noted that the following observations could conceivably vary even among chips of identical part number, speed grade, and revision.

Overclocking the ICAP beyond Xilinx's 100 MHz specification is possible. With the ICAP configured for 32-bit width, reliable operation has been obtained at 48, 96, and 144 MHz. Higher clock frequencies have not been attempted.

The setup and hold times for the ICAP's pins appear to be uncharacterized by the ISE tools as of version 10.1. At 96

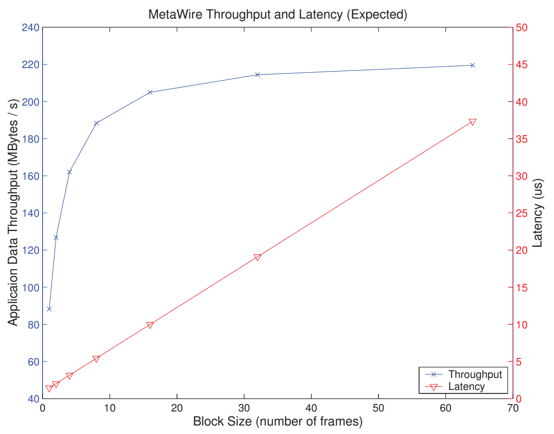

Fig. 5. Expected throughput and latency.

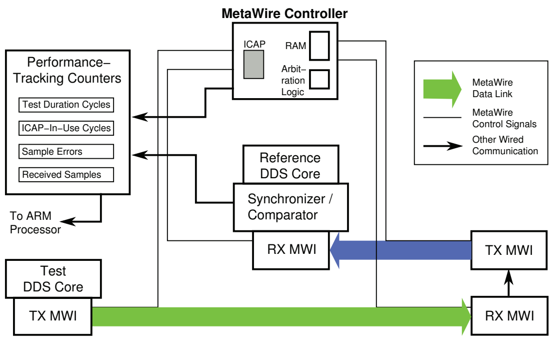

Fig. 6. Performance testbench.

MHz and higher, the ICAP Transfer Core worked intermittently when placement was left up to ISE's placer. Adding placement-constrained registers between our logic and the ICAP "pins" proved necessary to obtain stable performance.

The Virtex-4 Configuration Guide warns that BRAM readback should not be performed while the user's design is accessing a BRAM [7]. While page limitations prohibit a detailed description of the conflict scenarios we observed, readback of a given BRAM cluster conflicts with user-circuit reads and writes on the same cluster, as well as a limited number of clusters elsewhere on the chip. In contrast, configuration of a cluster only interferes with user-circuit access to that cluster.

4.2. Integrity and Performance Verification

The targeted hardware platform includes a Virtex-4 XC4VLX60 part which connects via a shared bus to an ARM9-based DaVinci processor from Texas Instruments. In order to validate Equation 1, the on-chip testbench shown in Figure 6 was developed to saturate and measure the ICAP/MetaWire bandwidth, while at the same time verifying data integrity.

Block size (frames)	$T_{expected}$ (MBytes/sec)	$T_{measured}$ (MBytes/sec)
1	88.19	86.54
2	126.68	124.96
4	162.04	160.63
8	188.32	187.37
16	204.94	204.37
32	214.40	214.09
64	219.47	219.31

Table 2. Throughput measurements.

Resource	**TX MWI**	**RX MWI**	**MWC**
Flip-flops	50	56	265
4-input LUTs	78	104	466
18-kbit BRAMs	5	5	4

Table 3. MetaWire resource requirements.

The test data source is a Direct Digital Synthesis (DDS) core–a sinusoid generator–from Xilinx CoreGen. The core produces 16-bit samples at a rate of up to one sample per clock cycle, and an identical DDS acts as a reference for sample-by-sample verification of the links. Flow control synchronizes the reference DDS with the rate of the incoming test data. The period of the sinusoid is a non-integer that results in long runs of unique binary words.

Two or more concurrent MetaWire links are needed to saturate the ICAP bandwidth. A single link proves inadequate due to a 16-bit path between the FIFO and TX/RXRAM in the MWI design. The ICAP and all test circuits are clocked at 144 MHz, and the test runs indefinitely with zero data errors. The results given in Table 2 are obtained from 25-second tests with two links in series, and are shown with the expected throughput from Equation 1. Positioning MWIs at opposite ends of the chip has confirmed that, on the XC4VLX60, performance is independent of link distance.

The measured ICAP utilization, an indicator of scheduler and handshaking overhead, ranges from 98.1 percent (B=1) to 99.9 percent (B=64). When the testbench is extended to eight links in series, the same aggregate throughput and utilization are measured, to the precision reported here. Data transfer remains error-free. More than eight links have not been tested due to placement constraints arising from the conflicts described in Section 4.1.

4.3. Logic Resource Requirements

The logic overhead incurred by MetaWire is presented in Table 3. These values represent the hardware when configured for B equal to 32 frames, which dictates the size of the frame buffer and hence the controller's BRAM requirement. Both MWIs use a 2 kByte (one BRAM) FIFO.

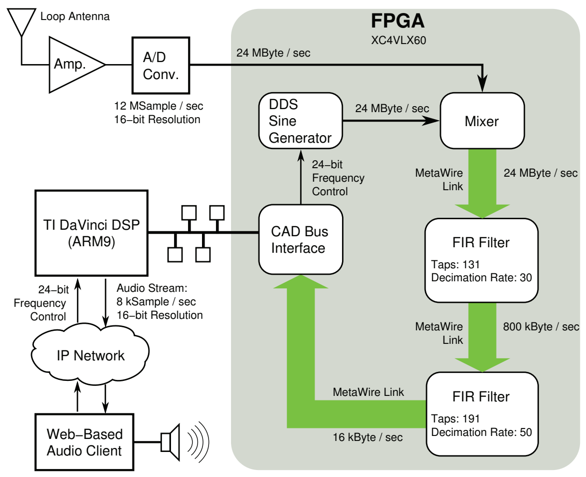

Fig. 7. Digital AM radio receiver application.

4.4. AM Radio Application

In order to demonstrate compatibility with non-trivial functional blocks in an application with real-time requirements, we have applied MetaWire to the digital AM radio receiver shown in Figure 7. This datapath, a very basic form of receiver, implements direct digital down-conversion of any desired channel in the AM spectrum. While AM radio itself is not an exciting application of modern FPGAs, the receiver consists of common elements in software defined radios.

MetaWire is delegated three links of differing rates with a combined throughput of 24.816 Mbytes/sec. The MetaWire controller and the ICAP are clocked at 144 MHz, while the remainder of the system, including the MWIs, is clocked at 96 MHz. ICAP utilization is observed to be 11.61 percent, which agrees with the bandwidth load/capacity ratio for this clock frequency and block size (B=32). The MetaWire hardware, in concert with the other links, autonomously negotiates data flow; only the A/D converter and the audio playback client define a data rate. Audio output is clear and audibly indistinguishable from that of the reference receiver (in which MetaWire links are replaced by conventional FIFO-based connections), suggesting error-free transfer; see Section 4.2 for objective link verification.

Finally, to make a rough assessment of the power footprint of the configuration fabric, we evaluated the current consumed by the MetaWire-based receiver and the reference receiver, implemented with similar floorplanning. Power consumption was measured to be 500 mW higher with MetaWire, a 61 percent increase over the FPGA's 822 mW consumption with the reference design. In contrast, the increase predicted by Xilinx's XPower tool, which presumably does not include the configuration fabric in its model, is only 84 mW, a 10 percent increase.

5. CONCLUSIONS

MetaWire may benefit designs in which offloading long-haul, streaming communication would alleviate local routing congestion. However, while page limitations prevent a concrete assessment of congestion reduction, such net-benefit scenarios are likely to be rare given the configuration fabric's modest bandwidth, which is currently orders of magnitude less than the aggregate bandwidth offered by the dense interconnect matrix in modern FPGAs. Nevertheless, MetaWire may offer a solution to inter-module communication in reconfiguration flows that support dynamic module instantiation and placement [6], by simplifying or eliminating run-time routing.

More significantly, MetaWire prompts the argument that if a configuration fabric not intended for a role in data transfer can offer the feasibility and performance demonstrated here, a dedicated NoC fabric or a combined NoC and configuration plane might yield an efficient layer of FPGA communication hierarchy that would enhance silicon utilization and reduce the compile times of large systems-on-chip. Improvements to the resource overhead, inter-node connectivity, and system-wide bandwidth shown with MetaWire would be achieved through extensions to existing configuration architectures, such as support for direct, concurrent frame transfers, and dedicated, cluster-local ports for flow control signaling and destination address specification.

6. REFERENCES

[1] D. Pamunuwa *et al*. Layout, performance and power trade-offs in mesh-based network-on-chip architectures. In *Proc. of the 12th IFIP International Conference on Very Large Scale Integration (VLSI-SoC 2003)*, pages 362–367, Dec. 2003.

[2] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proc. of 38th Conference on Design Automation (DAC 2001)*, pages 684–689, June 2001.

[3] E. Rijpkema *et al*. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE03)*, pages 10350–10355, Mar. 2003.

[4] R. Fong. Improving field-programmable gate array scaling through wire emulation. Master's thesis, Virginia Tech, Blacksburg Virginia, Sept. 2004.

[5] M. Millberg *et al*. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04)*, pages 20890–20895, Feb. 2004.

[6] P. Athanas *et al*. Wires on demand: Run-time communication synthesis for reconfigurable computing. In *Proc. of the 17th International Conference on Field Programmable Logic and Applications (FPL 2007)*, pages 513–516, Aug. 2007.

[7] Xilinx, Inc. *Virtex-4 Configuration Guide*, Oct. 2007.