# A SOFTWARE DEFINED RADIO APPLICATION UTILIZING MODERN FPGAS AND NOC INTERCONNECTS

*Graham Schelle, Jeff Fifield, and Dirk Grunwald*

Dept. of Computer Science
University of Colorado at Boulder
Boulder, CO
email: schelleg,fifield,grunwald@cs.colorado.edu

## ABSTRACT

Network on Chips are becoming a common onchip interconnect for both FPGA and mainstream processor designs. At the same time, software defined radios (SDR) are a new application field that is gaining much attention. As SDR tasks are mapped onto Network on Chip architectures, the typically streaming nature of samples will stress the NoC itself and possibly hurt the performance of other applications using that NoC. In this paper, we present the results of our partitioning and placement of a SDR transmitter onto a NoC architecture using an FPGA. We use a 802.11a transmitter example partitioned across a NoC and compare it to a handcrafted design. Additionally, various placement schemes, runtime architecture loads and NoC access methods are examined to determine the feasibility of this application and architecture combination.

## 1. INTRODUCTION

Network on Chip Architectures are gaining momentum as the new onchip interconnect for processors (both mainstream and embedded processors). Intel recently prototyped a 80-core processor that contained a network on chip with a mesh topology. In the embedded world, IP reuse, bus scaling concerns, and power constraints have led to network on chip interconnects as well [1, 2]. FPGA designs benefit from a NoC architecture on top of reconfigurable logic as a method to support reusable IP blocks. NoCs on FPGAs also allow scalable designs when bus-based architectures fail to meet performance constraints.

At the same time, the processing power of general purpose computing platforms has increased to the point where it is possible to perform digital signal processing in software or, as in our case, in reconfigurable hardware. This has led to an increasing interest in software defined radio (SDR) and cognitive radio technologies. Today most implementations of software defined radios use some combination of general purpose processors, FPGA accelerators and digital signal processors connected together using custom hardware. The abstraction provided by a NoC can benefit SDR implementations by providing a standardized, scalable and reconfigurable interconnection mechanism.

In this work, we demonstrate an example combination of a NoC architecture combined with a SDR task to show the challenges and advantages of this platform. We focus on throughput loss due to the overhead of the Network on Chip, including resource usage of a modern FPGA. We also highlight the benefits of an application partitioned over the NoC on a FPGA, allowing for exploration of the best NoC configuration and application combination.

While various work has placed SDR designs onto FPGAs, to our knowledge, no attempt of placing SDR tasks using a NoC exists. In the past, FPGAs have been ill-suited for these applications, with needed components consuming large amount of logic. Specifically, NoCs require a large amount of buffering in the form of FIFOs while SDR applications require multipliers. Multipliers and FIFOs typically take up large amounts of logic resources, but modern FPGAs have embedded cores to handle those tasks. We examine how these architectural improvements of FPGAs benefit a SDR/NoC combination to push FPGAs as a platform for this application domain.

The main contributions of this paper are described below:

- Provide an example partitioning of a SDR task onto a NoC architecture.
- Show how modern FPGAs are more able to support NoC architectures and SDR applications with additions of embedded cores and FPGA architectural improvements.
- Examine the advantages and challenges of meeting performance constraints in a 802.11a example on a chip with multiple allocation possibilities.
- Examine how competing applications running in parallel with an SDR application can hurt performance and how this performance loss can be minimized.

Authorized licensed use limited to: University of Florida. Downloaded on March 3, 2009 at 17:56 from IEEE Xplore. Restrictions apply.

This paper is organized as follows: Section 2 discusses work related to this research. The 802.11a application and architectures are described in section 3. Performance results are discussed in section 4 followed by conclusions.

## 2. RELATED WORK

We are focusing on an application space (software defined radio) tied to a specific architecture for onchip communication (Network on Chip). There is no work that we are aware of that focuses on these two areas combined, but there is a good amount of related work on each individual research area. We highlight the research efforts that guided our efforts and gave insight into how to combine SDR and NoC architectures effectively.

There have been a number of FPGA and software solutions reported for software defined radio and OFDM systems. Cummings [3] and Reed [4] report on the architecture of FPGAs and the general applicability of FPGAs to software defined radio.

Work by Dick [5] demonstrated an OFDM physical layer implemented in an FPGA. Their design implements OFDM modulation/demodulation and receiver synchronization algorithms for 802.11a. As in this work, the Simulink and System Generator toolkits were used as fast implementation tools.

The WARP project [6] is another FPGA based platform for software defined radio. The project has chosen OFDM as the physical layer of choice, and they have implemented a full OFDM transmit and receive chain in the FPGA. They also make use of the PowerPC processors embedded in the Xilinx FPGAs. The WARP platform is programmed using Simulink and System Generator.

NoC architectures exist in a variety of forms supporting various processing elements. The RAW project [7] is an excellent example of standard processors communicating to each other onchip over a NoC. RAW used a variety of communication NoCs depending on the the proximity of the communication, giving dedicated bandwidth for neighboring tiles to use for communicating. Register mapping of the network interfaces into the processors allowed quick and easy interfacing to the communication medium.

Additionally, the PACT XPP architecture [8] utilized specialized processors connected by a NoC. The processors were most interesting here, reconfiguring themselves to a variety of processing models, but does use a communication network.

## 3. SOFTWARE DEFINED RADIO APPLICATION: 802.11A TRANSMITTER

In this section, we describe the two versions of a 802.11a transmitter that we created. One version is a handcrafted 802.11a transmitter that is made as a standalone FPGA design. Another version of the 802.11a transmitter is created and partitioned into 4 components; components that can be swapped out for various software defined radio functionality.

### 3.1. IEEE 802.11a

The IEEE 802.11a standard defines a physical layer based on orthogonal frequency division multiplexing (OFDM) for wireless networking in the 5GHz band [9]. IEEE 802.11g uses essentially the same physical layer for wireless networking in the 2.4GHz band. In OFDM systems, the data to be transmitted is split up into some number of parallel data streams which we will call subcarriers. These subcarriers are individually modulated at a low rate using modulation techniques such as phase shift keying (PSK) and quadrature amplitude modulation (QAM). A vector consisting of one modulated symbol from each of the subcarriers defines a frequency domain OFDM symbol. The inverse FFT operation is performed on the frequency domain symbols to transform them into time domain OFDM symbols for transmission over the air. Although the subcarriers are individually carrying data at a low rate, the aggregate throughput of all subcarriers can be high. By using the Fourier transform, the subcarriers are placed very close together with no interference between them.

After detecting a transmission and aligning itself to the stream of incoming OFDM symbols, the receiver recovers the original data by reversing the operations of the transmitter. Incoming time domain OFDM symbols are transformed into frequency domain symbols using the FFT operation. Then the modulated subcarriers are extracted from the OFDM symbols and demodulated to recover the original bits.

### 3.2. Handcrafted transmitter architecture

We have created a standalone FPGA design implementing a general purpose OFDM transmitter for software defined radio[10]. The design supports basic 802.11a/g transmission and allows many of the OFDM parameters to be changed dynamically. Using this transmitter, software running on the FPGA (e.g. on a Microblaze tile) or on a host PC can dynamically control the OFDM physical layer of the software radio. This lets the radio use physical layers other than 802.11a, although we only generate 802.11a signals for this work.

The modulator is organized as a pipeline with four stages. In the first stage, control information and the data bits to be transmitted are received from an application or medium access control (MAC) layer. The control information sets up the rest of the pipeline to appropriately process the data. The second stage of the pipeline modulates the subcarriers

Authorized licensed use limited to: University of Florida. Downloaded on March 3, 2009 at 17:56 from IEEE Xplore. Restrictions apply.

| OFDM Parameter | Allowable Values |
|---|---|
| Sample Rate | 80 MHz |
| Subcarrier Spacing | .3125 MHz |
| Number of FFT Points | 1 to 256 |
| Number of Data Subcarriers | 0 to 256 |
| Number of Pilot Subcarriers | 0 to 256 |
| FFT symbol period | 3.2 $\mu s$ |
| Guard Interval Duration | 0 to 3.2 $\mu s$ |
| OFDM Symbol Duration | 3.2 to 6.4 $\mu$s |
| Subcarrier Modulations | BPSK, QPSK, 16-QAM, 64-QAM |

**Table 1**. Software defined OFDM parameters

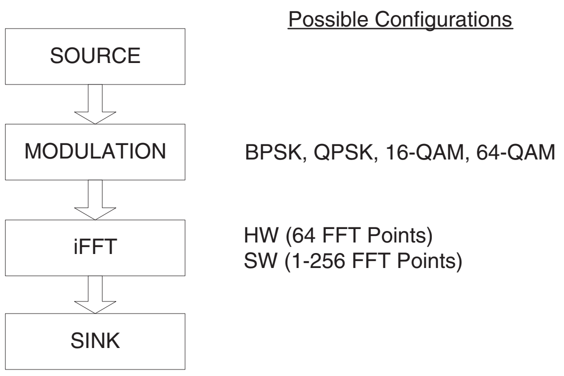

**Fig. 1**. Partitioning of NoC design.

using the specified PSK or QAM scheme. During the third pipeline stage, the inverse FFT transforms the data into time domain OFDM symbols. Finally, the fourth pipeline stage inserts a cyclic extension of the OFDM symbols to form guard intervals between symbols. The resulting samples are ready to be sent to the DAC. A summary of the configurable parameters of the OFDM modulator are shown in Table 1.

### 3.3. NoC based transmitter architecture

We use the NoCem Network-on-chip emulation tool [11] to create our NoC architecture on the FPGA. The NoC that we create is an 8x8 mesh network with 2-lane virtual channels. Connected to the NoC itself are the processing tiles with a front end of an operating system bridge.

Also for this design, we partitioned the SDR task into 4 different tiles. Figure 1 shows the breakout of the components. We partition the handcrafted design into components that may exist on a variety of implemented platforms in the future. These tiles include the signal generation source/sink, a modulation block, and a Discrete Fourier Transform block. The tiles' functionality and implementations are described here:

- Signal Generation Tile. The signal generator creates and consumes samples for the rest of the application (can act as a source and or a sink). This tile really emulates both the application layer that would create data bytes and the DAC (digital to analog converter) that would consume time-domain signals.
- Modulation Tile. The modulation tile takes in control information and data payload and emits modulated data corresponding to individual OFDM subcarriers. These subcarriers are modulated with the appropriate PSK or QAM modulation schemes and organized according to the 802.11a standard. The output of this block can be used as the input to an inverse FFT tile.
- FFT Tile (HW). Using Xilinx's Coregen, we are able to wrap an inverse FFT and make it an allocatable resource attached to the Network on Chip. This block uses clock enables to do flow control on the actual FFT when the network is either not ready to inject samples (empty ingress buffer) or collect samples (full egress buffer). We use the pipelined, streaming I/O version of the Xilinx Coregen FFT.
- FFT Tile (SW). This version of the FFT tile is actually a Xilinx Microblaze that runs a software implementation of the inverse FFT. The Microblaze runs at 100 MHz and collects all the samples needed (64 samples) before doing the transform.

For these experiments, we also created additional processing tiles needed to setup and load the chip at runtime. These tiles are:

- Allocation Tile. This tile is a Xilinx Microblaze tile that initializes the other processing tiles with their execution state. This state includes virtual table setup at the executing processing tiles and executable identifiers. These identifiers are used to tell the SDR application how to modulate the 802.11 packets. The allocation tile is presented with a task list and places those tasks on the chip.
- Exerciser Tile. The exerciser tile is a simple VHDL block that generate packets in a set pattern. Each exerciser is a simple state machine that communicates to area exercisers creating congestion on the network on chip. Each tile uses a random distribution of packet arrival times to generate a specific load on the network.

## 4. RESULTS

### 4.1. Development Software

For development, Xilinx's ISE and EDK 8.2 tools were used. We are currently using Mentor Graphic's Modelsim 6.2b for gathering simulation results. Hardware results are based on the Nallatech XtremeDSP development kit containing a Virtex-II Pro FPGA and the Xilinx ML505 Board containing a Virtex-5 FPGA.

Authorized licensed use limited to: University of Florida. Downloaded on March 3, 2009 at 17:56 from IEEE Xplore. Restrictions apply.

**Table 2**. Component Size Reduction Virtex-II Pro versus Virtex5 FPGA architectures.

| Component | V2P Logic (reg/LUT) | V5 Logic (reg/LUT) | Reduction |
|---|---|---|---|
| virtual channel 2-lane | 624/766 | 90/170 | 86%/78% |
| virtual channel 4-lane | 1,280/1,714 | 190/464 | 85%/73% |
| $\mu$Blaze tile | 567/1,114 | 157/296 | 72%/73% |
| 64-tap FFT | 2,280/1,568 | 1,772/1,581 | 22%/-1% |
| 256-tap FFT | 3,475/2,281 | 2,590/2,403 | 25%/-1% |

## 4.2. handcrafted versus NoC based implementations

The handcrafted design was verified and implemented on the Nallatech XtremeDSP board. This design was verified to have sent out a valid 802.11a frame using real hardware.

The NoC architecture, due to its size (a 8x8 mesh-topology NoC) does not fit on current boards (a 3x3 mesh is possible on the Xilinx ML505 platform containing a Virtex5 FPGA), but we use a cycle accurate simulator to gather results for that architecture. Its functionality was verified in simulation against the handcrafted design.

## 4.3. FPGA overhead

The newest lines of FPGAs include various components that greatly benefit NoC architectures and Software Defined Radios. Of course, with higher transistor densities, the amount of programmable logic will increase, but we focus on dedicated embedded cores here. Specifically, the Virtex-5 Family of Xilinx FPGAs contain dedicated FIFO controllers and DSP48E slices. As NoCs and packet switched networks consist of a variety of FIFOs for packet storage, the FIFO controllers greatly reduce logic usage. For Software Defined Radio applications (even more appropriate – DSP applications), the dedicated DSP48E blocks on Virtex5 FPGAs help in the discrete Fourier transform tasks. While the Virtex2 Pro has dedicated multipliers, the Virtex5 DSP48E's hold much more dedicated logic including a larger multiplier and accumulation blocks. As FPGAs evolve, it is expected that these DSP cores will grow even more complex to accommodate the digital signal processing markets.

We briefly list out the various size comparisons in table 2 simply to show that FPGA architectures themselves are increasing their ability to support this combination of architecture and application. It is notable that the Virtex2 Pro architecture uses 4-input LUTs as compared to the Virtex5 having a 6-input LUT.

**Table 3**. Comparing Performance of a software based FFT and a hardware based FFT in the software defined radio application.

| SDR application | sample rate (Msamples / second) |
|---|---|
| software iFFT | 0.080 |
| hardware iFFT | 14.286 |

## 4.4. Performance Results

### *4.4.1. Software versus Hardware implementations of the FFT*

We wanted to first see the feasibility of a software based version of the FFT. From the results in table 3 it is clear that a software FFT is only useful for "offline" tasks. By offline, we are referring to packet creation that can be slowly stored in a main memory before being sent into a DAC at a later time. This is feasible, but not desirable in 802.11a communications where delays in packet sending may lead to timeouts.

It is important to see this performance hit which is caused by the actual implementation of the partitioned application (as opposed to NoC placement strategies). While software might be more flexible than reconfigurable hardware, the performance of software based applications on FPGAs cannot meet the requirements of SDR applications (at least in this one example). This should be expected, as SDR tasks on mainstream processors only became a possibility once processor clock speeds reached into the gigahertz with complicated memory hierarchies, while FPGA based processors currently are in the 100 MHz range. The remainder of the architectures presented in this paper will utilize the hardware version of the FTT.

### *4.4.2. Affect of Distance Between Partitioned Blocks*

We wanted to determine if the Network on Chip had an adverse effect on this applications performance due to distance between partitioned blocks. Various placement schemes or runtime state of the processing tiles could lead to placing this particular SDR task at non-adjacent tiles. Interestingly, when the SDR application is the only application running on chip (all other processing tiles are idle), the NoC acts as a natural pipeline, holding samples in the NoC itself as processing occurs. There is no performance decrease when the application is placed onchip at any distance apart. With this observation, we then wanted to examine the SDR application running in parallel with other applications.

### *4.4.3. Affect of Parallel Processing and Competing Applications for NoC Resources*

We are emulating a 64 node NoC, where the actual SDR application only consumes 4 processing tiles. It would seem

Authorized licensed use limited to: University of Florida. Downloaded on March 3, 2009 at 17:56 from IEEE Xplore. Restrictions apply.

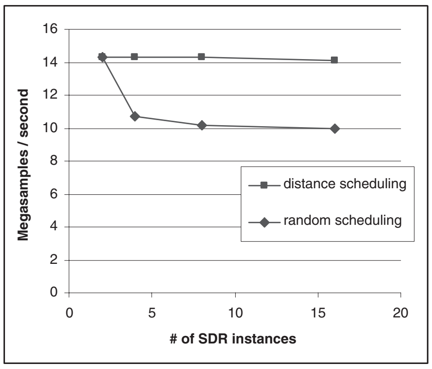

**Fig. 2**. As more instances of the application are placed onchip, performance declines as competition for NoC increases. Two scheduling policies are examined.

intuitive that other applications may be also running onchip in parallel with the SDR application. Figure 2 shows the performance that can be sustained with multiple instances of the application running in parallel.

We show two scheduling policies in this figure. Randomly allocating instances of the SDR application leads to higher contention onchip. Distance scheduling is done by placing the applications close together onchip minimizing communication costs and contention.

There is a reasonable performance hit on a per-instance basis of the SDR application once packets start contending for resources. Whereas previously, the SDR task could treat the NoC as a natural pipeline, those semantics are broken once other applications compete for that pipeline. This competition occurs as the applications compete for virtual channels and router switch allocation.

### *4.4.4. NoC Datawidth Configuration Effect on SDR Performance*

With a HW iFFT and a distance-based scheduling algorithm, we next examined how the NoC could best be configured to handle this specific SDR application and its dataflow patterns. The initial NoC had a 8b datawidth, which is 1/4 of the resulting sample size (16b imaginary and 16b real component of sample). By increasing the NoC datawidth to 32b, we are able to better push samples through the network without any resizing tasks. 4 placements are examined (see figure 3) that while the task is placed tightly in a cluster, there still will be contention for NoC resources. Figure 4 shows the comparison of these datawidths where the NoC is being accessed in parallel by competing applications. We show a loading of 0-20% on the network while an SDR application executes.

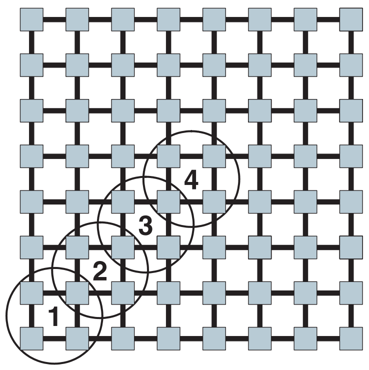

**Fig. 3**. The 4 placements on a 8x8 NoC that will be used to show how the SDR application performs under various NoC load conditions . Placement 1 (in the lower left corner) will have much less contention for NoC resources than placement 4 (located in the middle of NoC).

### *4.4.5. Streaming Interface Addition to NoC Access Points*

Our final and last improvement to the NoC bottleneck in streaming performance was to add a streaming interface to the NoC access point. This access point takes advantage of the virtual channel implementation of the Network on Chip in order to secure a dedicated path through the network utilizing 1 of the 2 virtual channels at each physical link. No modifications were made to the NoC itself and due to the up-down routing protocol within the network, deadlock avoidance is still maintained. Table 4 shows that the streaming interface to the NoC provides roughly 50 megasamples per second of throughput for the SDR application with any of the four placements shown previously in figure 3. While we measured the applications running across all NoC loading scenarios as previously shown in figure 4, regardless of the placement or loading scenario each scenario resulted in 49-51 megasample per second throughput. This improvement to the packet based interface of the NoC clearly had the largest affect on performance.

## 4.5. Results Summary

From these incremental improvements to the NoC architecture, it is very clear that an FPGA-based Network on Chip can support streaming software defined radio applications. Using an FPGA allowed for the NoC, the NoC access point and the actual processing elements to be configured correctly. In a non-reconfigurable environment, none of this would not be possible, leaving designers a much small architectural search space.

Interestingly, all these numbers are based on a 100 MHz clockrate, which meets timing constraints using this NoC

Authorized licensed use limited to: University of Florida. Downloaded on March 3, 2009 at 17:56 from IEEE Xplore. Restrictions apply.

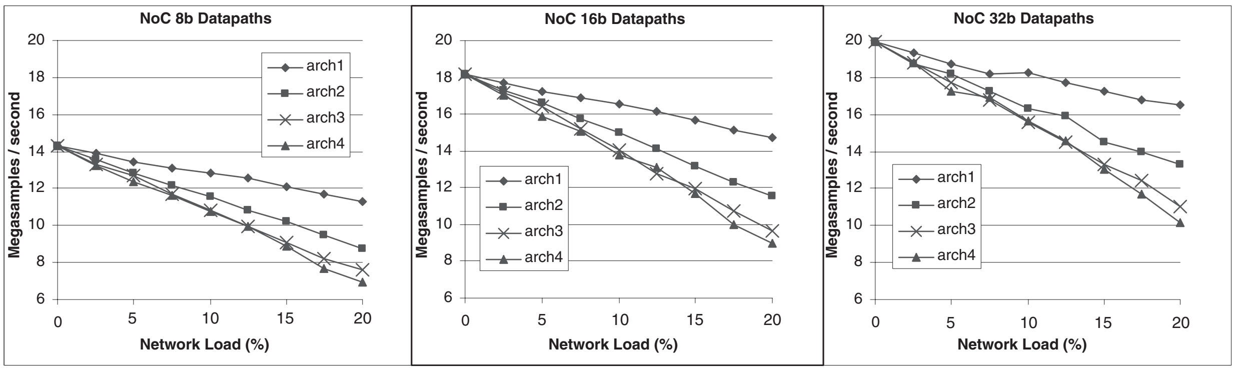

**Fig. 4**. Increasing datapath size to match sample size across various network loads. The arch1-4 labels refer to the placement location shown in figure 3

**Table 4**. The 32b datapath NoC with a streaming interface to the actual processing elements in the SDR application. The arch1-4 labels refer to the placement location shown in figure 3

| Architecture | sample rate (Msamples per second) |
|---|---|
| arch1 | 50.50 |
| arch2 | 50.44 |
| arch3 | 50.23 |
| arch4 | 50.32 |

architecture (observed in timing analysis of a 2x2 design), and is a clockrate provided on several Xilinx platform FPGA boards. We hope to implement this design on larger Virtex-5 boards and use a higher clockrate to meet the 80 megasample / second threshold for 802.11a communication.

## 5. CONCLUSIONS

In this paper, we have presented a software defined radio application mapped onto a Network on Chip. Taking advantage of FPGA architectural improvements, NoC architectures are more easily placed onto FPGAs and better use platform resources. While the challenges of efficiently using the NoC on these streaming applications was also examined, we found and explored several improvements that take advantage of reconfigurable hardware. With applications competing for NoC resources, it will be very important to have configurable processing tiles, freedom to reconfigure the network, and the ability to access the NoC in a streaming manner.

## 6. REFERENCES

[1] 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.

[2] 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

[3] M. Cummings and S. Haruyama, "FPGA in the software radio," in *IEEE Communications Magazine*, 1999, pp. Vol. 37, pp.108–112.

[4] J. H. Reed, *Software Radio: A Modern Approach to Radio Engineering*. Prentice Hall, 2002.

[5] C. Dick and F. Harris, "FPGA implementation of an OFDM PHY," in *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 1, 9-12 Nov. 2003, pp. 905–909Vol.1.

[6] R. University, "Warp - wireless open-access research platform," http://warp.rice.edu/.

[7] 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.

[8] V. Baumgarte, G. Ehlers, F. May, A. N, M. Vorbach, and M. Weinhardt, "Pact xpp: A self-reconfigurable data processing architecture," *J. Supercomput.*, vol. 26, no. 2, pp. 167–184, 2003.

[9] *Wireless LAN medium access control (MAC) and physical layer specifications: High speed physical layer in the 5 GHz band*, IEEE Std. 802.11a, 1999.

[10] J. Fifield, "A software defined ofdm modulator," Master's thesis, University of Colorado, Boulder, 2006.

[11] G. Schelle and D. Grunwald, "Onchip interconnect exploration for multicore processors utilizing FPGAs," in *2nd Workshop on Architecture Research using FPGA Platforms*, 2006.

Authorized licensed use limited to: University of Florida. Downloaded on March 3, 2009 at 17:56 from IEEE Xplore. Restrictions apply.