

Efficient use of Communications Between an FPGA's Embedded Processor and its Reconfigurable Logic

Joshua Noseworthy
Mercury Computer Systems
Chelmsford, MA
Email: jnosewor@mc.com

Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA
Email: mel@coe.neu.edu

Abstract—Increasing device densities allow chip manufacturers to integrate more functionality onto a single piece of silicon. FPGA manufacturers, such as Xilinx and Altera, use these additional resources to further diversify and improve the processing capabilities of their architectures. One of the more recent architectural enhancements that has been made to Xilinx's Virtex family architecture is the introduction of the PowerPC405 hard-core embedded processor. In this paper we present a Software Defined Radio Application that serves as a vehicle for investigating effective communications between the PowerPC405 Processor and the surrounding FPGA fabric..

A challenging aspect of developing applications that target the PowerPC is the interfacing of the processor with the surrounding reconfigurable logic. We have implemented a dozen different versions of a Software Defined Radio (SDR) application to exercise the various interfaces that enable communication between the processor and the surrounding FPGA fabric. The implementations differ only in the interfaces used. Our results indicate that the performance of the SDR application application can be increased by as much as 60 percent just by choosing the interfaces that are most appropriate for the different types of data in the implementation. This demonstrates that the performance of FPGA applications that use the embedded processor are dramatically effected by the mechanisms chosen to enable communication between the processor and its surrounding resources.

I. INTRODUCTION

Advancements in silicon technologies continue to increase the number of transistors that can be integrated into a single device. Many chip manufacturers use this new integration potential to fabricate complete systems on a single silicon fabric. FPGA manufacturers are using this expanded capacity to implement embedded DSPs, multipliers, and memory blocks along with the reconfigurable fabric. At Xilinx, FPGA designers have developed new generations of FPGA architectures that contain a variety of embedded resources. One of several recent additions to Xilinx's Virtex family architecture is the embedded PowerPC405 core. The motivation for including the processor core is that most FPGAs contained within an embedded system require some level of interaction with an external processor. Moving this processor onto the fabric of the FPGA eliminates bottlenecks associated with communicating through off-chip interfaces. These are replaced by on-chip interfaces that must

provide efficient communication between the processor and the reconfigurable resources.

We have investigated various mechanisms that interface the Virtex II Pros PowerPC405 with the surrounding FPGA fabric. These mechanisms utilize the On-Chip Memory (OCM) and the Processor Local Bus (PLB) interfaces of the PowerPC405. Both of these interfaces enable communication between the PowerPC405 and its surrounding reconfigurable resources. However, the mechanisms used by each interface are very different. For instance, the OCM interface provides a dedicated interface to the surrounding FPGA fabric, while the PLB provides a shared interface. A dedicated interface provides higher performance relative to the shared interface; however, the share interface offers greater flexibility in interfacing peripherals to the processor. We investigated the use of different types of buses, the enabling/disabling of caches, and the best location for different types of data, including instructions, program data, and application data. The results provide an accurate characterization of the advantages and disadvantages of the interfaces that enable the communication between the Virtex-II Pro's PowerPC405 and its surrounding resources. FPGA. To serve as a basis for comparison we present multiple implementations of an application that is representative of most Software Defined Radio Applications. These implementations differ only in the interfaces that communicate data between the PowerPC and the reconfigurable fabric.

The rest of this paper is organized as follows. First we present related work. In the next section, we present background on the Virtex-II Pro architecture and on the FM3TR application. In Section III we describe our experiments using the FM3TR application to exercise the different interfaces on the Virtex-II Pro. In Section IV we present our results. We conclude with a discussion of future directions.

A. Related Work

Several studies that were previously done on the Virtex-II Pro's embedded PowerPC were limited to implementations that perform all of the application's processing using the PowerPC. Those studies that did examine implementations that processed application data in both the PowerPC and

reconfigurable logic were based on results that were extracted from simulation models. Our study is the first to perform real-time analysis of an application that uses both the FPGA's PowerPC and reconfigurable logic to process data.

Gordon Brebner has developed an SOC application intended for use with computer networks that contains a mixture of IPv4 and IPv6 workstations. The platform makes extensive use of the PowerPC405 core on the Virtex-II Pro FPGA [1]. This research occurred prior to the actual availability of Virtex-II Pro devices and is based on results that were obtained exclusively through simulation models. The paper shows the advantage of a logic centric approach over a processor centric approach when mapping designs to the Virtex-II Pro. Our design treats the processor and logic on an equal footing, with the processor feeding data to the logic.

In a more recent paper, the authors describe a logic centric design methodology where processing tasks are offloaded from the logic to the processor of a Virtex-II Pro FPGA [4]. This technique is described by the term "software decelerator," implying that a processing task offloaded from logic to a processor may result in an implementation that uses the FPGAs resources more efficiently, even if it exhibits no gain in performance.

The Xilinx application note, "PLB vs. OCM Comparison Using The Packet Processor Software," uses a packet processing application to compare the performance tradeoffs of the PowerPC405 Processor Local Bus and On-Chip Memory interfaces [5].

II. BACKGROUND

A. The PowerPC405 Block

The Virtex-II Pro's processor block [2] contains the PowerPC405 core, logic to interface the core with its surrounding resources, and general-purpose routing resources. The number of processor blocks that are present is specific to the Virtex-II Pro part.

The two interfaces that allow the processor block to communicate data to and from the FPGA fabric are the PLB and OCM interfaces. The On-Chip Memory (OCM) interface serves as the interface between the Block RAMs and the OCM signals that are contained within the PowerPC405 processor core. The primary advantage of the OCM interface is that it guarantees a fixed latency of execution. This provides a higher level of determinism, making communication over the OCM interface a good choice for processor applications that must guarantee a specific rate of communication with the fabric. The disadvantage of the OCM interface is its ability to address a relatively small amount of memory efficiently.

The PLB interface is designed to communicate with a third party on-chip bus architecture, known as CoreConnect. The CoreConnect Architecture is a hierarchical bus topology that has been designed to provide efficient on-chip communications. The CoreConnect topology that we use consists of 2 buses: the Processor Local Bus (PLB) and the On-Chip Peripheral Bus (OPB).

The PLB is designed to be used by the processor to service peripherals that require high performance. The PLB is a fully synchronous 64-bit data bus that supports read and write transfers between master and slave devices. Each PLB master is attached to the PLB through separate address, read-data, and write-data buses. PLB slaves are attached to the PLB through shared but decoupled address, read-data, and write-data buses and a plurality of transfer and status control signals. Devices that wish to communicate over the PLB must first contact the PLB arbiter. After considering the bus's activity level, the PLB arbiter will either grant or deny the device access to the bus.

The PowerPC405 uses the OPB to communicate with low speed devices, such as a Universal Asynchronous Receiver Transmitter (UART). The OPB is a fully synchronous 32-bit data bus that functions independently of the PLB on a different level of the bus hierarchy. The OPB does not interface directly to the PowerPC405 core. Instead an OPB to PLB bridge provides the interface between the two levels of hierarchy. Therefore, if an OPB master needs to communicate with the processor, it must do so by using the OPB to PLB bridge to generate the appropriate transaction on the PLB. Similarly, if the processor wants to communicate with a device on the OPB, it must do so by using the PLB to OPB bridge to generate the appropriate transaction on the OPB.

B. Software Defined Radios

The high performance and high reconfigurability requirements of SDR make FPGA architectures a good implementation fabric for many SDR applications. SDR processing tasks such as modulation, de-modulation, up-conversion, and down-conversion benefit from the high performance that is offered by modern FPGA architectures.

In addition to FPGAs, general-purpose processors (GPPs) play an important role in most software defined radio systems. As a result, the introduction of FPGA architectures that contain embedded processors has prompted some SDR system designers to integrate the functionality of the two separate entities onto a single piece of silicon.

C. The FM3TR Reference Waveform

Future Multi-Band, Multi-Waveform, Modular, Tactical Radio Waveform (FM3TR) is a relatively simple and unclassified waveform that can be used to demonstrate interoperability between various software defined radios [3]. The processing requirements of the waveform itself is relatively low when compared to industrial waveforms such as WCDMA. The simplicity and low processing requirements of FM3TR are appropriate for our application because they allow us to focus on studying the architectural issues associated with FPGA development using embedded processors and not implementation issues.

III. EXPERIMENTAL SETUP

A. The FM3TR Waveform Application

Our study aims to demonstrate the advantages and disadvantages of the different interfaces that enable communication

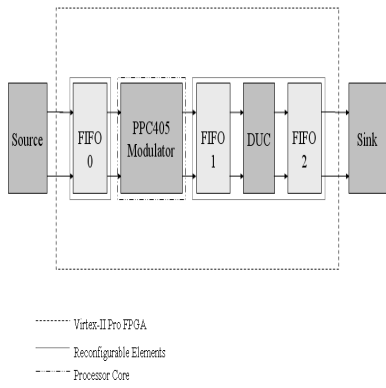


Fig. 1. FM3TR Application Architecture Overview

between the Virtex-II Pros FPGA fabric and its embedded processors. Our application, shown in Figure 1, implements the modulation and digital up conversion that is associated with a FM3TR transmitter. The implementations uses both the reconfigurable logic and the embedded processors that are contained within a Virtex-II Pro Platform FPGA. Data is placed into the initial First In First Out (FIFO) Queue 0 via a source that is external to the FPGA, such as a waveform encoder. The modulator then pulls the data out of the FIFO 0 and performs Minimum Shift-Key (MSK) modulation. The signal values that result from the modulation are then pushed into FIFO 1.

B. The FM3TR Modulator Implementation

The FM3TR modulator is implemented in software using the embedded PowerPC processor that is contained within the Virtex II-Pro FPGA. The functionality of the FM3TR modulator is described in the C Programming language. This description is then compiled with a GNU C compiler and the resulting code is downloaded into the BlockRAMs of the FPGA. This compiled code is then executed on the embedded processor.

C. The FM3TR Digital Up-Converter Implementation

The implementation of the Digital Up-Converter is provided by v1.4 of the Digital Up-Converter IP Core from Xilinx. Symbol values are read out of FIFO 1 and presented to the in-phase and quadrature inputs of the DUC for processing. When processing is complete, qualified values of the passband signal will appear on the output of the DUC on successive clock cycles. The values that appear on the output of the DUC are pushed into FIFO 2.

D. Experiments

We use 12 different implementations of the FM3TR waveform processing application as a vehicle to study the aforementioned methods of communications. These implementations differ *only* in the interfaces used. The modulator and DUC implementations are identical in all designs. Our experiments examine the movement of three types of data: instruction,

program, and application. Instruction data refers to the information that communicates instructions to the PowerPC. All data that exists for the purpose of maintaining the state of the PowerPC is referred to as Program Data. Finally, application data is the term used to describe the data that is actually processed by the processor. We separate our implementations into three classes. Each differs in the mechanisms that are used to communicate data between the PowerPC and its surrounding resources. The first class uses the OCM, the second class uses the PLB, and the third class uses the OPB.

1) *Implementation Class 1:* Implementation class 1 uses the OCM interface to communicate modulated data to BlockRAM. Data samples and processor instructions are loaded into the appropriate memory locations at the time of configuration. Data samples residing in DSOCM are modulated by the PowerPC according to the instructions being stored in ISOCM. After modulation, the data is written back into the DSOCM (FIFO 1). The DUC removes the modulated signal values from Port B of the DSOCM. The modulated signal is processed by the DUC and pushed into FIFO 2. Implementation class 1 contains three implementations, a, b and c, shown in Figures 2 and 3. Implementations a, b, and c use the same memory to store the application's instructions and the modulated data. The difference between the three implementations is the storage locations of the program data that is responsible for maintaining the processors stack and heap. Implementation subclass 1.b stores the stack/heap data in OCM BRAM. Implementation subclass 1.a store the stack/heap data in PLB BRAM. Implementation 1.c is identical to 1.a except that the cache is enabled in 1.c, but not in 1.a. Implementation subclass 1.b demonstrates the performance that one can expect from an application that runs completely out of OCM BRAM. Implementation subclasses 1.a and 1.c demonstrate how the relocation of the program data can effect the performance of an application. Note that the OCM is not cacheable.

2) *Implementation Class 2:* Implementation class 2 uses the PLB to transfer data between the modulator and FIFO 1. Data samples and instructions are loaded into the appropriate on-chip memory at the time of configuration. The modulator removes the sample values from DSOCM and performs the modulation. The sample values of the modulated signal are pushed into FIFO 1 through the PLB interface. The DUC removes the modulated signal values from FIFO 1. The modulated signal is processed by the DUC and pushed into FIFO 2. Implementation class 2 contains 6 implementations. Implementation 2.a, shown in Figure 4, stores its program data in DSOCM BRAM. Implementation 2.b, shown in Figure 5, stores its program data in PLB BRAM. Instructions and program data for both implementations are stored in ISOCM BRAM and PLB BRAM respectively.

Implementations 2.c, 2.d, 2.e, and 2.f, seen in Figure 6, demonstrate the performance consequences of storing both data and instructions in PLB BRAM. The sharing of a single PLB between multiple memories will cause contention if the processor attempts to communicate data and instructions over the PLB at the same time. This contention will need to be

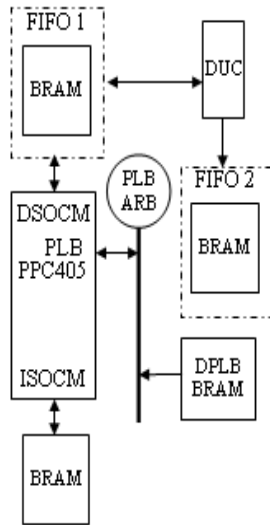


Fig. 2. Architecture Overview for Implementation Subclass 1.a and 1.c

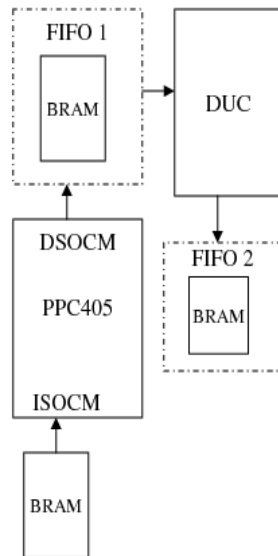


Fig. 3. Architecture Overview for Implementation Subclass 1.b

resolved by the PLB arbiter before either communication can proceed. The cache can be used to speed up the performance of these implementations. By enabling the cache, communication over the PLB can be reduced significantly. This communication reduction is dependent on the locality of the data stored in PLB BRAM.

3) *Implementation Class 3*: Implementation class 3 demonstrates the performance of the application when the modulated data is communicated to memory over the OPB. Data samples and instructions are loaded into the appropriate on-chip memory at the time of configuration. The modulator removes the sample values from either PLB or ISOCM BRAM and performs the modulation. The sample values of the modulated

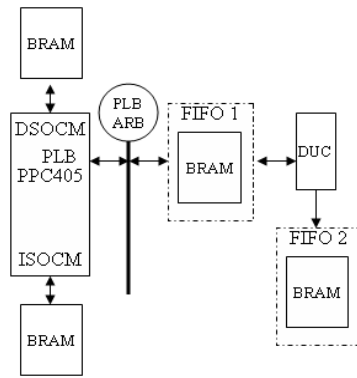


Fig. 4. Architecture Overview for Implementation Subclass 2.a

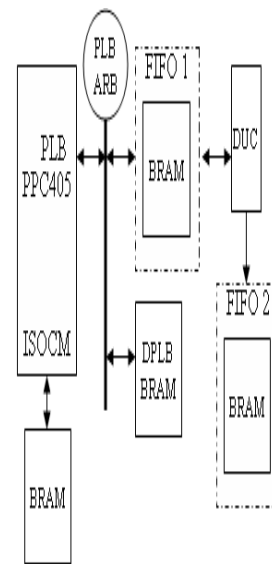


Fig. 5. Architecture Overview for Implementation Subclass 2.b

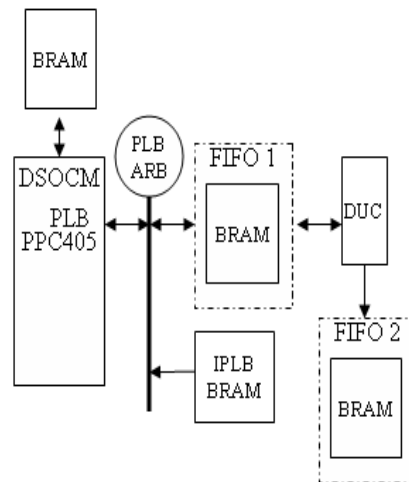


Fig. 6. Architecture Overview for Implementation Subclass 2.c, 2.d, 2.e, and 2.f

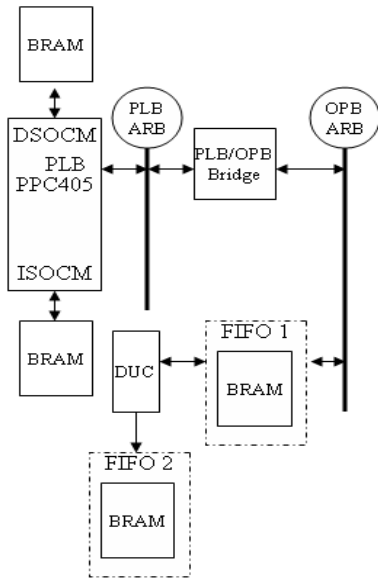


Fig. 7. Architecture Overview for Implementation Subclass 3.a

signal are pushed into FIFO 1 through the PLB interface. The PLB/OPB Bridge recognizes the destination address of the modulated data and generates the appropriate transaction on the OPB. The DUC removes the modulated signal values from FIFO 1. The modulated signal is processed by the DUC and pushed into FIFO 2. Implementation 3.a, shown in Figure 7, stores the instructions in ISOCM BRAM, the program data in DSOCM BRAM, and the modulated data in OPB BRAM. Implementations 3.b and 3.c, shown in Figure 8, demonstrates the consequences of sharing the PLB while communicating modulated data over the OPB. In Implementation 3.a, all of the memories are connected to separate interfaces, allowing for concurrent memory transfers. This is the best possible scenario given the implementation specification requiring the modulated data to be communicated over the OPB. The relocation of instruction storage to the PLB in Implementation 3.b will create contention for the PLB bus. This contention is resolved in Implementation 3.c by enabling the i-cache.

IV. RESULTS

A. Implementation Results

We compared the number of clock cycles each implementation consumed during its execution. Since the computation in each implementation is identical, differences in execution cycles are due to the interfaces used. By comparing the execution cycles that are consumed by each implementation, we are able to determine which interfacing techniques work best. Tables 1, 2, and 3 show the performance results for all of the implementations. It should be noted that since the OCM interface is a non-cacheable interface, the enabling or disabling of the cache effects the PLB interface only.

The values in the column titled Computation Cycles are obtained by measuring the number of clock cycles consumed by the execution of a modulation code version that does not save

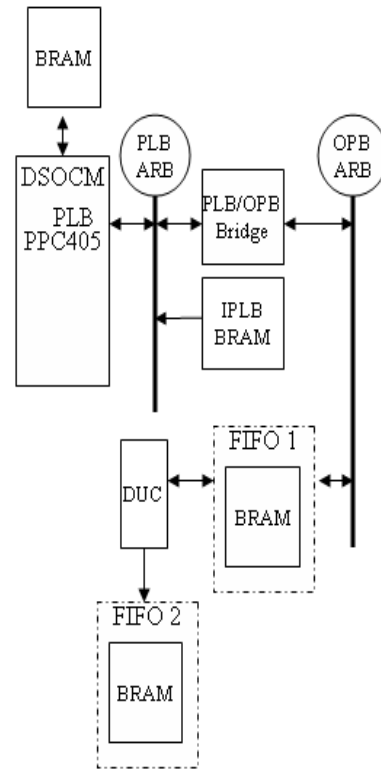


Fig. 8. Architecture Overview for Implementation Subclass 3.b and 3.c

the results to memory. The values in the column titled Total Cycles are obtained by measuring the number of clock cycles consumed by the execution of a modulation code version that saves the result of each computation to memory. The values of the Communication Cycles are the difference between the two. It should be noted that the values in the Communication Cycles column do not reflect the communication costs associated with instruction fetches. The values indicate the number of clock cycles required to communicate the modulated data to memory. Instruction transfers are required for computation and thus are reflected by the values in the Computational Cycles column. Differences in the number of computational cycles between implementations are due to differences in instruction fetch times.

B. Analysis of Results

The first thing to note is the large range of performance achieved simply by changing the way the interface is used. There is a factor of three difference between the number of cycles needed for the slowest vs. the fastest implementation. Recall that all that has changed between different implementations is the interfaces. Clearly, using interfaces intelligently is essential to the performance of a design using the embedded PowerPC.

1) *The OCM Interface Analysis:* The best performing designs are in Class 1, shown in Table 1, which use the On-Chip Memory (OCM) interface to communicate modulated data to the DUC. Implementation 1.b communicates both

TABLE I
CLASS 1: CONFIGURATION AND RESULTS

SubClass	1.a	1.b	1.c
Instruction Data	ISOCM	ISOCM	ISOCM
Program Data	DPLB	DSOCM	DPLB
Application Data	DSOCM	DSOCM	DSOCM
I-Cache	N/A	N/A	N/A
D-Cache	DISABLED	N/A	ENABLED
Computation Cycles	7828	7818	7816
Communication Cycles	20566	13468	13462
Total Cycles	28394	21286	21278

TABLE II
CLASS 2: CONFIGURATION AND RESULTS

SubClass	2.a	2.b	2.c
Instruction Data	ISOCM	ISOCM	IPLB
Program Data	DSOCM	DPLB	DSOCM
Application Data	DPLB	DPLB	DPLB
I-Cache	N/A	N/A	DISABLED
D-Cache	ENABLED	ENABLED	DISABLED
Computation Cycles	8430	8430	16638
Communication Cycles	17752	27716	36887
Total Cycles	26182	36146	53525

instruction and data through the processors OCM interfaces. This implementation executes the entire processor application in under 22000 clock cycles. The fastest overall design is implementation 1.c. Implementation 1.c differs from 1.b in that stack and heap data are communicated using the Processor Local Bus (PLB). This is comparable to implementation 1.c, which The performance delivered by 1.c is dependent on the processors ability to use its internal cache. When the cache is disabled, as in case 1.a, the processor must use the fabric of the FPGA to communicate with PLB BRAM. The need for this communication is the reason why implementation 1.a's performance is poorer than 1.b and 1.c. Implementations 2.a and 3.a deliver the best performance in their respective classes. In both 2.a and 3.a, the instructions are communicated through the OCM bus. This leads us to conclude that the OCM interface provides the fastest communications between the processor and the fabric. It is interesting to note that the communication performance of a cache enabled PLB interface is comparable to the performance of the OCM interface. This observation may depend on the fact that the cache is capable of storing all of the data communicated through the PLB interface. If the size of the stored data is larger than the cache, the need to fetch data from BRAM may widen the performance gap between the OCM and the cache enabled PLB interface. Furthermore, if the locality of the data that is being communicated through the PLB interface is poor, the cache will not be of any use.

2) *The Processor Local Bus*: The results in Table 2 indicate that using the PowerPCs cache can improve the performance

SubClass	2.d	2.e	2.f
Instruction Data	IPLB	IPLB	IPLB
Program Data	DSOCM	DSOCM	DSOCM
Application Data	DPLB	DPLB	DPLB
I-Cache	DISABLED	DISABLED	ENABLED
D-Cache	ENABLED	ENABLED	DISABLED
Computation Cycles	15561	6519	6519
Communication Cycles	40400	17659	17742
Total Cycles	55961	24178	24261

of implementations that use the Processor Local Bus (PLB) interface to communicate to the FPGAs fabric. However, this is only true for data that exhibits good locality. Implementations that illustrate this fact include 2.c, 2.d, 2.e, 2.f, 3.b, and 3.c. When we compare implementations 2.c, 2.d, 2.e, and 2.f to each other we see that the enabling of the I-Cache results in a reduction in execution time by more than 50%. With the I-Cache disabled (designs 2.c and 2.d), the IPLB and the DPLB were fighting for control of the PLB. Enabling the I-Cache allows the instructions to be fetched almost exclusively from the processor cache. This increases the performance of the application. Furthermore, other investigations [6] show that using a cache enabled PLB interface consumes five times less power than an equivalent implementation that uses a PLB interface with the cache disabled, and three times less power than an implementation that uses the OCM interface. Though this analysis was done using a different application, it is evident that using the cache to reduce communication over the FPGA fabric can result in a substantial power savings.

Performance improvements that result from the I-Cache are a consequence of the instructions temporal locality. Temporal locality is a consequence of the frequent execution of one instruction within a given window of instruction code. The PowerPC code for this application performs modulation using a single loop that iterates over the entire application data set. During each loop iteration several instructions are executed to produce the appropriate pulse values. Since the modulation of large data sets requires a substantial number of loop iterations, the application's performance is improved by using the I-Cache to store instructions contained within the loop. of many applications that will be run on the embedded PowerPC.

In cases where the data being cached does not exhibit temporal locality, use of the PowerPC's cache can hurt the application's performance. This is illustrated by implementations 2.d and 2.f, which show slightly worse performance than implementations 2.c and 2.e. This performance loss is a result of the fact that streaming data usually does not exhibit any temporal locality since the current output is almost always a function of the current input. Caching application is of no value since a data value gets used only once. As a result, the data value will need to be copied to BRAM soon after being placed in the cache in order to make room in the cache for data that has been accessed more recently. Therefore, enabling the D-Cache in this instance actually hurts performance because it introduces the unnecessary step of writing the data value into

TABLE III
CLASS 3 : CONFIGURATION AND RESULTS

SubClass	3.a	3.b	3.c
Instruction Data	ISOCM	IPLB	IPLB
Program Data	DSOCM	DSOCM	DSOCM
Application Data	OPB	OPB	OPB
I-Cache	DISABLED	DISABLED	ENABLED
D-Cache	DISABLED	DISABLED	DISABLED
Computation Cycles	7779	17891	16766
Communication Cycles	29225	46238	21312
Total Cycles	37004	64129	38078

the D-Cache.

3) *The On-Chip Peripheral Bus:* Table 3 summarizes the results for our OPB experiments. The On-Chip Peripheral Bus (OPB) has the poorest performance of the three available interfaces. Our experiments included it for completeness. Our experiments show that the On-Chip Peripheral Bus (OPB) is capable of offering reasonable performance when there are no devices that have to fight for control of the PLB. Implementations 3.a and 3.c, indicate scenarios where the OPB offers performance that is comparable to the PLB. However, neither implementation exhibits any bus contention since only one type of data is on the OPB. In 3.a, all data except for the applications data is store in OCM BRAM. As a result, traffic appearing on the PLB and the OPB is always destined for OPB BRAM. In implementation 3.c, the instruction data has been moved to PLB BRAM. However, implementation 3.cs enabling of the cache nearly eliminates any traffic on the PLB related to the communication of processor instructions. This is a consequence of the fact that all of the processors instructions are capable of fitting inside the PowerPCs cache.

Implementation 3.b is identical to 3.c except for the instruction cache in 3.b being disabled. This disabling of the cache forces both the processor instruction and applications data to be fetched through the PLB interface of the processor. This creates contention on the PLB, drastically lowering the performance of the design. Using the results of the execution of Implementation Class 3, we conclude that the performance OPB is reasonable provided that contention for the PLB is minimized.

V. DISCUSSION

We have presented a study that investigates the performance of all three interfaces on the Virtex-II Pro: the On-Chip Memory bus (OCM), the Processor Local Bus (PLB) and the On-chip Peripheral Bus (OPB).

Our results show that the interfaces chosen and the use of the cache can have dramatic effect on the performance of a design. The best performing designs make use of all the available bandwidth to the processor by communicating only one type of data through each interface. This also minimizes contention on the PLB. In addition, while caching instruction and program data improves performance, using the D-Cache

in applications that process streaming data can actually hurt performance.

This investigation describes the performance of the PowerPCs interfaces in terms of the number of clock cycles required to communicate data from the processor to the surrounding FPGA fabric. In the future, we plan to investigate the amount of FPGA resources that are consumed by the use of a specific interface. Such a study would enable application designers to weigh real estate costs against the performance of a specific interface.

VI. CONCLUSIONS

This investigation shows that the performance of an application that uses the Virtex-II Pros embedded processor is strongly effected by the types of interfaces that are chosen to communicate data between the processor and the fabric of the FPGA. The OCM and PLB are both capable of providing high performance if used appropriately. Specifically, using all types of available interfaces provides high performance. Streaming data, which requires a high performance interface but does not exhibit temporal locality, should be communicated through the OCM interface. Data that exhibits temporal locality, such as program data, should be communicated through a cache-enabled PLB interface. Application designers that have reasons for communicating data with poor locality over the PLB should consider disabling the cache to eliminate the overhead associated with the maintenance of cache coherency. The result will be designs that make use of the embedded PowerPCs and exhibit good performance.

ACKNOWLEDGEMENTS

This research was conducted when the first author was a graduate student at Northeastern Unversity. This research was supported in part by Mercury Computer Systems and donations from the Xilinx Corporation.

REFERENCES

- [1] G. Brebner. Single-chip gigabit mixed-version ip router on a virtex-ii pro. *Proceedings of the 10th Annual IEEE Symposim on Field-Programmable Custom Computing Machines*, 2002.
- [2] T. X. Corporation. PowerPC block reference guide, July 2005. Last Accessed July 2005. <http://direct.xilinx.com/bvdocs/userguides/ug018.pdf>.
- [3] S. D. R. Forum. Faq. Last Accessed August 2005. <http://www.sdrforum.org/faq.html>.
- [4] G. B. E. Keller and P. James-Roxby. Software decelerators. *12th Annual IEEE Symposium on FCCM*, pages 385–395, 2004.
- [5] K. Lund. PLB vs. OCM comparison using the packet processor software. <http://direct.xilinx.com/bvdocs/appnotes/xapp644.pdf>, October 2004. Last Accessed July 2005.
- [6] J. Ou and V. K. Prasanna. A methodology for energy efficient application synthesis using platform fpgas. In T. P. Plaks, editor, *ERSA*, pages 280–283. CSREA Press, 2004.