Synthesis of Multi–Mode Memory Interfaces for FPGA Reconfigurable Computing Machine

Joonseok Park[††],

Inha University, School of Computer Science and Engineering,
Inchon, Republic of Korea

Summary
Multi-core reconfigurable architectures where soft-cores can be programmed over a reconfigurable substrate such as in an FPGA are a reality. For these target architectures it will become imperative that high-level mapping tools can synthesize and estimate the impact of high-level transformations in the overall design in terms of speed and area. The lack of support for external memory operations in current synthesis tools substantially increases the complexity and the burden on designers in the mapping of applications to FPGA-based computing engines. In this paper we address the problem of synthesizing and estimating the area and speed of memory interfacing for Static RAM (SRAM) and Synchronous Dynamic RAM (SDRAM) with various latency parameters and access modes. We describe a set of synthesizable and programmable memory interfaces a compiler can use to automatically generate the appropriate designs for mapping computations to FPGA-based architectures. Our preliminary results reveal that it is possible to accurately model the area and timing requirements using a linear estimation function. We have successfully integrated the proposed memory interface designs with simple image processing kernels generated using commercially available behavioral synthesis tools.

Key words:
FPGAs, Configurable Computing, Memory Access Protocols, Estimation.

1. Introduction

As the densities of current FPGA continue to grow it is now possible to generate System-On-a-Chip (SoC) designs where multiple computing cores are connected to various memory modules with customized topology with application specific memory access patterns. For example, Xilinx has recently introduced devices to which a paired down version of a PowerPC core can be mapped and connected to a set of internal memories.

Given the complexity and heterogeneity of these target architectures, it is very likely that high-level compilation tools will be required to perform a wide variety of high-level program transformations. In order to determine the impact of these transformations in the resulting designs in terms of area and speed these tools will have to make use of fast and accurate estimation data.

Unfortunately, most commercially available synthesis tools for FPGAs have mostly ignored system level issues when dealing with external memories. While some tools now incorporate internal RAM modules and the mapping of array variables to them, they have avoided all external memory interfacing issues and corresponding estimation of the appropriate interfaces. Typically, programmers must separately synthesize the datapath if they want to exploit the estimation capabilities of the tools and then integrate those designs with handcrafted memory interfaces.

In this paper we address these shortcomings by designing and validating the implementation of a family external of memory controllers for FPGA-based computing systems. The main objective is to allow compilers and other high-level synthesis tools not only to synthesize these interfaces taking advantage of a wide range of parameterization attributes, but also to estimate its area and timing in the overall design.

Specifically, this paper makes the following contributions:

- It describes a set of parameterizable memory controllers that are capable of signaling a wide variety of external memories with different latency parameters as well as number of channels.
- It presents concrete experimental results of the implementation of sample memory controllers for both Static RAM (SRAM) and Synchronous Dynamic RAM (SDRAM) memory modules.
- It presents the corresponding area and timing estimates and suggests a simple linear estimation function for this family of memory controllers compilers can use to estimate area and timing performance.
- It presents results validating the proposed approach for a selected set of digital signal processing kernels where the datapath were integrated with the proposed memory interfaces for an SDRAM memory with page mode operations.

Manuscript received September 5, 2007
Manuscript revised September 20, 2007

An important aspect of our approach is the decoupling of the datapath design, or application specific core design, with the development and synthesis of the external memories interfaces. To this effect we have designed a decoupled memory control scheme [9] that allows the developed of the actual computation core to be develop using commercially available synthesis tools. In this work we have extended our previous work with the addition of a more generic memory controller capable of both SRAM with and without pipelined memory accesses, with SDRAM with and without page-mode memory accesses.

The rest of this paper is organized as follows. In the next section we describe our overall memory controller architecture as an extension of our own previous work. Section 3 presents experimental results. Section 4 describes related work and we conclude in section 5

2. External Memory Controller

We now describe the basic architectural features of the family of memory controllers we have designed and evaluated for FPGA-based machines. The basic structure is depicted in Figure 1 where we have illustrated the application of the memory controller directly feeding a datapath core.

Associated with each data port in the datapath design there are FIFO queues. FIFO queues are an integral part of the design to help tolerate the latency of memory operations as well as isolate the issues of scheduling in the datapath from the implementation of the memory operations, in particular when dealing with strict timing constraints.

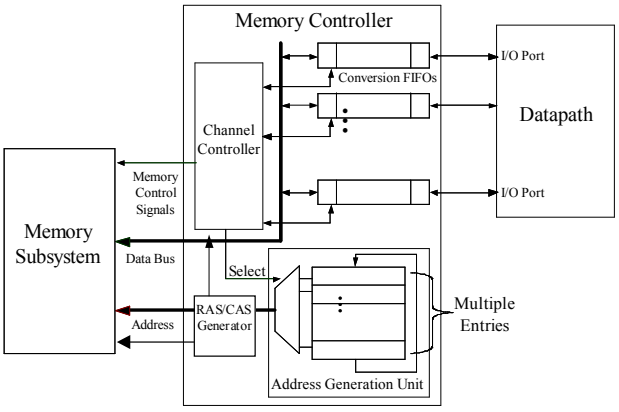

Fig. 1. Memory Controller Architecture.

Associated with each FIFO queue we define the notion of channels. A channel defines several other resources in the architecture, such as the setting in the Address Generation Unit (AGU) and which of the FIFOs is to store the data that is to be read/written for each of the datapath ports.

This architecture has a channel controller module that performs the external memory signaling for all of the channels and feeds the data into the FIFO queues. This design, while this is not the only design possibility for these data engines, it is a very modular approach that has been tested successfully in the context of a compiler that maps applications directly to FPGA [9].

The fundamental feature of this design is that is decouples the synthesis of the datapath with the synthesis of the memory controller and therefore insulates the physical memory access protocol with the issues of behavioral synthesis – including scheduling of memory accesses. These features come at the price of an extra virtualization layer by having the datapath perform a simple handshaking protocol to load and store data from external memories.

In Figure 1 we have also shown a RAS/CAS generator block that can be thought as part of the address generation unit. This component takes an isolated address and determines, should SDRAM page mode be used, whether or not the corresponding memory access is within the same page of the previous memory access. If so, it informs the channels controller, which in turn bypasses some of its internal states to perform a faster in-page memory access mode. Internally this component consists of a simple table lookup much like a cache block. Figure 2 below illustrates a timing diagram for the SDRAM page and non-page mode memory access as supported by our memory controller implementations.

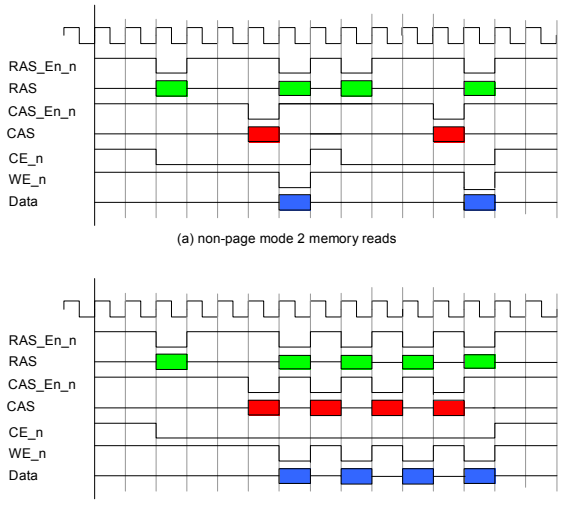

Fig 2. Typical Timing Diagram for SDRAM.

The current implementation can interleave the memory accesses of multiple S-DRAM channels, and with minor modifications support both SRAM (in pipelined and non-pipelined) modes with the S-DRAM memory accesses.

2.1 Interface Design Parameters

We now describe the attributes of the memory interface controller for two basic memory technologies, the SRAM and SDRAM. In both cases we describe the parameters required for both non-burst and burst-mode operations, i.e., pipelining in the RAM cases and page mode access in the SDRAM case, respectively:

- **Type**: SRAM or SDRAM memory access signaling with the corresponding physical signals.
- **Channels**: Number of input and output channels which dictates the number of FIFO queues and number of AGU entries.
- **Bit widths**: Whereas the memory interface word size if typically fixed, says at 32 bits, the bit width of the channels, i.e., the FIFO queues is programmable and application dependent.
- **Read/Write Latency**: Stipulates how many clock cycles for read/write cycles. This parameter also needs to be refined in the context of burst-mode operations.
- **Burst Mode and Burst Mode Latency**: Either pipelined or page-mode depending whether or not it is defined as an SRAM or SDRAM interface. The burst mode latency specifies the latency value for consecutive accesses, either pipelined or page mode.

We are developing simple library functions written in C that take these parameters and generate a complete synthesizable structural VHDL design, which has been successfully synthesized using Xilinx™ tools.

2.2 Discussion

The fundamental assumption of this design is that need to decouple the design of the datapath from the design and vagaries of the actual physical signaling for either of SRAM and SDRAM memories.

For the SDRAM interfaces the current implementation **cannot support burst-mode memory accesses**. The fundamental reason is that our designs want to preserve the control of the memory operations at the heart of the memory controller. A burst-mode operation would inherently destroy this approach making the design of the memory controller, and the corresponding opportunities for application specific scheduling much more complicated.

While the set of abstraction introduced in this design clearly insert some additional latency, we believe the abstractions of channels and the ability of generating application-specific memory scheduling operations (see [3][9]) will ultimately allow high-level compilation and synthesis tools to effectively and automatically target reconfigurable FPGA-based computing systems.

3. EXPERIMENTAL RESULTS

We have developed a multi-protocol memory interface unit as outlined in section 2 above in synthesizable VHDL specifications. Table 1 below illustrates the complexity and size implementation of these interfaces for the target Virtex™ XCV 1000 BG560 device using the Xilinx ISE™ 4.li toolset with logic synthesis tool. All of the designs in these experiments took less than 3 minutes to synthesize with a medium effort for place and routing on a PC with 800 MHz Pentium III processor with 756 Mbytes of memory.

We report on the actual implementation results for both SRAM and SDRAM interfaces. For each of the interfaces we report on the implementation resources (FPGA slices – the Virtex™ device has a maximum capacity of 12,288 slices) for different channels bit width, i.e., FIFO queue bit widths. We also distinguish between pipelined and non-pipelined memory access implementation for the SRAM and page and non-page mode for the SDRAM implementation.

These results are for a single memory controller with 2 input and 1 output channel. In the next section we explore the sensitivity of the designs to larger number of channels.

Table 1. VHDL Code and Complexity Implementation

Memory Interface	Transfer mode	Bit width	VHDL lines of code	FPGA Slices
SRAM	Non-pipe	8	1011	268 (2.2%)
		16	1005	235 (1.9%)
	Pipelined	8	1036	254 (2.0%)
		16	1033	226 (1.8%)
SDRAM	Non-page	8	1138	346 (2.0%)
		16	1129	263 (2.8%)
	Page	8	1169	299 (2.4%)
		16	1163	272 (2.2%)

Next we have used the generated VHDL from these libraries and simulated their performance. Table 2 below presents the performance and transfer rate for the various burst modes for each of the interfaces along with the base

clock rate specification. For each of these performance results we derived the number of cycles per transfer for each of the transfer modes through simulation. The maximum clock rate was obtained by running the place-and-route passes of the synthesis tool for the target device. Also, in the experiments we have assumed a predefined latency value of 3 clock cycles to access the external pins of the device. Typically these parameters are variable but know for each implementation of the vendor library. In our case we have used the WildStar™ [13] library for these experiments.

Table 2. VHDL Code and Complexity Implementation.

Memory Interface	Transfer mode	Bit width	Clock Rate (MHz)	Cycles per byte	Transfer Rate (Mbps)
SRAM	Non-pipe	8	36.7	1.25	29.4
		16	47.9	1.25	38.3
	Pipelined	8	39.6	0.5	79.2
		16	47.7	0.5	95.4
SDRAM	Non-page	8	30.3	2	15.1
		16	33.5	2	16.7
	Page	8	32.0	1	32.0
		16	36.0	1	36.0

These preliminary results indicate that sizes of the interfaces are fairly small for the tested FPGA Virtex devices (less than 3%). In terms of the clocks rates, these designs attain a minimum clock rate of 30 MHz and a maximum transfer rate of 16 Mbps (8bit channels) and 38Mbps (16 bit channels) for non-pipeline/page modes and 36 Mbps (8 bit channels) and 95 Mbps (16 bits channels).

As expected to clock rate results for the SDRAM page mode accesses are substantially slower than the non-page mode operations. We attribute this fact to the added complexity in the *hit-page* circuitry required to bypass the controller in the same page memory access operations.

We have successfully validated these designs via simulation for the case of the SDRAM and even on a WildStar™ multi-FPGA board for the SRAM interface. It will be impossible to test the SDRAM interface with this board we it does not have SDRAM modules.

3.1 Estimation Modeling

Based on these results we build a simple linear regression estimation models for multi-channel memory interfaces for both types of RAM memories. The results in table 3 below illustrate the various design area plots for various number of channels settings.

These results suggest, for this particular target architecture and place-and-route tool that the number of FPGA slides (area) and clock rate of a memory interface is given by the linear expression below obtained by linear regression and where N denotes the number of input channels.

$$\text{Area}(N) = 84.08\ N + 153.33$$
$$\text{Clock}(N) = 30.49 - 0.27\ N$$

Table 3. Sensitivity to Number of Channels for S-DRAM with page-mode accesses for 8 bit channels

Memory Interface	Transfer mode	Bit width	Number of Channels (in/out)	FPGA Slices (%)	Clock Rate (MHz)
SDRAM	Page	8	1/1	213 (1.7%)	30.9
		8	2/1	299 (2.4%)	32.0
		8	4/1	495 (4.0%)	28.2
		8	8/1	903 (7.3%)	25.3
		8	16/1	1463 (11.9%)	27.7

While the area metric tracks a linear interpolation fairly well, the clock rate data reveals a more disperse pattern. We attribute this to the fact that these are small designs in the overall FPGA devices (see table 1 above). As such the area typically grows according to a linear function. The clock rate behavior, however, tends to be more sensitive to the vagaries of the place-and-route steps use in logic synthesis.

These results reveal several simple aspects about the proposed memory interfaces. First, their size grows less than linearly with respect to the number of memory access channels. Expectedly, there is a slight degradation of clock rate for larger designs. This degradation. However, is not severe (about 10%) which we expect to worsen for large designs where place-and-route is unable to easily find available FPGA slice resources. Nevertheless, these results reveal that estimating the memory interface designs using simple linear function is a very good estimate of their overall area and timing characteristics.

While we expect to make a more comprehensive comparison between SRAM and SDRAM interfaces, in the final version of this paper, we expect a similar trend in the SRAM designs as fundamentally both have the same underlying architecture.

3.2 Application Experience

We have conducted an experiment with a concrete application to validate the correctness of our family of designs and assess the performance improvement and possible bottlenecks in the memory interface designs.

In this application experience we have used an SDRAM interface design and composed it with an existing behavioral datapath that implements the Sobel edge detection algorithm. For this particular experiment we have manually programmed the order in which the memory is to be accessed when defining the complete memory controller specification. In the final version of

the paper we expect to have more experimental result of other kernel such as the matrix-multiply and the FIR image processing kernel.

In this experiment with the Sobel application we have used the following specifications for the SDRAM memory controller: a page-mode latency of 2 clock cycles and a non-page mode latency of 6 clock cycles. From the perspective of the application datapath the memory interface introduces an extra latency for each memory operations to serve the request for read and write operations from the datapath and to generate the corresponding addresses (RAS and CAS signals). Table 4 below summarizes the actual values for the latency of the various memory operations used in our application simulation experiment.

Table 4. Memory operation latency values for a basic memory latency of page and non-page of 2 and 6 respectively. The added clock cycles over the latency of the actual memory interface is indicated in parenthesis

Non-Page Mode		Page Mode	
Read	Write	Read	Write
9 (+3)	7 (+1)	6 (+4)	3 (+1)

We simulated the overall design composed of the datapath implementing the Sobel edge detection and the SDRAM interface for a 16x16 image. The total simulated execution cycles were 2413 cycles of which 768 (32%) cycles were devoted to the datapath execution and 1645 (68%) cycles for the memory operations. To expose the impact of the memory operation latency we did not overlap any of the datapath computation with the memory operations.

Of the 1645 cycles in memory operations the application performed a total of 192 memory read operations of which 64 (33%) were in page mode and 128 (66%) in non-page mode. The application also performed a total of 43 non-page write operations. In terms of clock cycles, the non-page and page reads account for 576 (35%) and 768 (65%) of the 1645 cycles in memory operations respectively.

This experience shows that we were successful in integrated a datapath generated by a behavioral synthesis tool with an SDRAM memory controller. This controller was automatically generated using C template functions (although at this time the integrated was still done manually) and simulated correctly.

4. RELATED WORK

Other researchers have addressed the issues memory operations interfacing and estimation.

Weinhardt and Luk developed memory access optimizations for pipeline vectorization in RAM interface [12] using a scheme to reduce consecutive memory accesses of array data using shift registers but accessing only on-chip RAM modules. Gokhale and Stone [4] proposed an automatic array allocation compile-time algorithm for multi-level memory subsystem. They attempt to allocate array variables to memories based on the memory latency, data access frequency and execution schedule. Schmit [9] developed a mapping scheme for mapping datapaths with memory operations directly to hardware. His approach uses a centralized memory controller scheme where the scheduling of the operations is done in conjunction with the execution of the datapath computation. Panda et. al. [8,9] refined this approach by defining a time-constrained based specification of the a centralized scheduler for handling external memory operations. Catthoor, Balasa et. al. developed and evaluated memory optimizations for embedded systems for a particular application set [1,2,6]. This research focuses on optimizations to minimize memory area and power consumption. Catthoor also proposed a data packing scheme to reduce memory bandwidth requirements for dynamic data structure. Wuytack et al. suggested minimizing memory bandwidth requirements [14] by mapping highly accessed array variables to fast hierarchy storage.

From the viewpoint of estimation for high-level constructs for mapping designs to FPGAs in [5] researchers have developed regression-based models to capture the area of image processing operators. They combine these estimates hierarchically for larger designs to produce overall good quality estimates for a selected set of small designs. In our previous work for compiler guided transformations [11] we have also used estimates produced by behavioral synthesis for overall large designs. None of these effort, however, has taken into account the size and timing implications of the memory interfacing modules.

5. CONCLUSION

In this paper we have described a set of parameterizable memory interface designs for both SRAM and SDRAM memory technologies. The proposed designs present a set of abstractions a compilation and synthesis tools can use to automatically generate complete designs that interface with external memory modules. We have reported on the low area and fairly good timing for a wide variety of designs with pipelining and page-mode memory operations. The preliminary area and timing results for our designs reveal that it is possible to accurately model both area and timing of the proposed memory interfaces with linear functions. This estimation model will ultimately allow for tools to incorporate the memory

interface component in their estimates for complete designs.

REFERENCES

[1] F. Balasa, F. Catthoor, and H. De Man “"Dataflow-driven Memory Allocation for Multi-dimensional Signal Processing Systems", Proceedings of the IEEE International Conference on Computer Aided Design, Santa Jose, Calif., Nov. 1994, pp. 31-34.

[2] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. DeMan "Global communication and memory optimizing transformations for low power signal processing systems", IEEE workshop on VLSI signal processing, La Jolla, Calif., Oct. 1994.

[3] P. Diniz and J. Park, "Automatic Synthesis of Data Storage and Control Structures for FPGA-based Computing Machines", In Proc. of the IEEE Symp. on FPGAs for Custom Computing Machines (FCCM'00), IEEE Computer Society Press, Los Alamitos, Calif., Oct. 2000, pp. 91-100.

[4] M. Gokhale and J. Stone “Automatic Allocation of Arrays to Memories in FPGA Processors With Multiple Memory Banks” Proc. of IEEE Symp. on FPGAs for Custom Computing Machines (FCCM’99), IEEE Computer Society Press, Los Alamitos, Calif. Oct. 1999, pp. 63-69.

[5] D. Kulkami, W. Najjar, R. Rinker and F. Kurdah, “Fast Area Estimation to Support Compiler Optimizations in FPGA-based Reconfigurable Systems”, To appear in Proc. of the IEEE Symp. On FPGAs for Custom Computing Machines (FCCM’02), 2002.

[6] M. Miranda, F.Catthoor, M. Janssen and H. DeMan, “High-level address optimization and synthesis techniques for data-transfer-intensive applications”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 6(4), Dec. 1998, pp. 677 –686.

[7] MonetTM User’s and Reference Manual Software Release R42, Mentor Graphics Inc., 1999.

[8] P. Panda, F. Catthoor, N. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle and P. Kjeldsberg. “Data and memory optimization techniques for embedded systems”, ACM Transactions on Design Automation of Electronic System, 6(2), Apr. 2001.

[9] J. Park and P. Diniz, "Synthesis of Memory Access Controller for Streamed Data Applications for FPGA-based Computing Engines", In Proc. of the 14th Int. Symp. on System Synthesis (ISSS'2001), Oct. 2001, pp. 221-226.

[10] H. Schmit and D. Thomas, "Synthesis of Applications-Specific Memory Designs," IEEE Transactions on VLSI Systems, 5(1), Mar. 1997, pp. 101-111.

[11] Reference omitted.

[12] M. Weinhardt and W. Luk “Memory Access Optimization and RAM interface for Pipeline Vectorization”, In Proc. of Symp. on Field Programmable Logic (FPL’99), Springer-Verlag, 1999, pp 60-71.

[13] WildStarTM Reference Manual revision 4.0, Annapolis MicroSystems Inc., 1999.

[14] S.Wuytack, F.Catthoor, G.De Jong, and H.De Man. “Minimizing the required memory bandwidth in VLSI system realizations”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 7(4), Dec. 1999, pp. 433 –441.

[15] Xilinx, Inc. Virtex™ 2.5V Filed Programmable Gate Arrays Product Specification. DS003(v2.4), 2000.

Joonseok Park received the B.S.in Mathematics from Sogang University in 1997, M.S. degrees and Ph.D in Computer Science from University of Southern California in 2000 and 2004, respectively. During 2004-2006, he stayed in System On Chip Laboratory of Samsung Electronics, Inc. as a senior Engineer. He now full-time lecturer of Inha Univerisity, School of Computer Science and Engineering.