Stream Memory Subsystem in Reconfigurable Platforms

Sek M. Chai, Nikolaos Bellas, Malcolm Dwyer, Dan Linzmeier
Embedded Systems Research Center, Motorola Labs,
(sek.chai@motorola.com)

Abstract— **High performance computing platforms require an efficient memory subsystem to keep processors busy. This paper proposes a memory hierarchy using stream units to move stream data between memory and processors. The stream units prefetch and align data based on stream descriptors, a mechanism that allows programmers to indicate data movement explicitly by describing their memory access patterns. The memory hierarchy is configured on reconfigurable logic based on application needs. This paper presents an example stream unit design with preliminary synthesis results.**

Index Terms—**Stream processing, memory hierarchy, memory burst.**

I. INTRODUCTION

Reconfigurable systems offer flexible platforms in which to optimize a memory subsystem for single application or a class of applications. While architectural research on FPGA have been partial to processor designs, the same flexibility and performance offered by today's FPGAs are equally suitable for the memory subsystem design. As the performance disparity between processor and memory intensifies [1], high performance or real-time application performance continues to be limited by the memory subsystem [2]. Consequently, studies on efficient memory subsystems should be considered alongside the processor design as memory performance must be scrutinized on new reconfigurable architectures.

While FPGA platforms continue to provide a larger number of configurable logic blocks that can be mapped to processing elements to satisfy computing demands, the interconnect delays and relatively slower memories maintain an imbalance between processor and memory performance. Traditional approaches to compensate for poor memory performance such as caches are not effective due to poor temporal locality of data for streaming data [3], especially when large memory buffers are not available on FPGA platforms. Data duplication on distributed memory buffers is also not effective as the chip area can be better utilized for processing.

This paper presents a flexible memory subsystem for stream computation. The memory subsystem builds upon configurable stream units that move data while computation is performed. The stream units are specialized DMA units that are optimized for stream data transfer. They rely on a set of stream descriptors, which defines the memory access pattern, to prefetch and align data in the order required by the computing platform. In using the stream units in the memory subsystems, the architecture effectively decouples the communication from computation, and allows hardware designers to address their implementation and optimization individually. The stream units take advantage of available bandwidth by prefetching data before it is needed, and consequently, the system performance becomes dependent on average bandwidth of the memory subsystem with less sensitivity to peak latency to access a data element.

II. RELATED WORK

The streaming programming model separates communication from computation, allowing either programmer or compiler to specify each portion independently [4]. Properties of streaming model of computation include:

- Computations kernels are independent and self contained
  Computation kernels are localized such that there are no data dependencies between other kernels. A programmer can annotate portions of a program that exhibit this behavior for mapping onto a stream processor or accelerator.
- Computation groups are relatively static
  The processing performed in each computation group is regular or repetitive, which often come in the form of a loop structure. There are opportunities for compiler optimization to organize the computation as well as the regular access patterns to memory.
- Explicit definition of communication
  Computation kernels produce an output stream from one or more input streams. The stream and other scalar values which hold persistent application state are identified explicitly as variables in a communication stream or signal between kernels.
- Data movement exposed to programmer
  A programmer can explicitly define movement of data from memory or to other computation kernels. Hardware mechanisms such as a DMA or stream unit provide this capability without interrupting the processor. The stream processing model seeks to either minimize data movement by localizing the computation, or to overlap computation with data movement. Furthermore, the programmer can retune the application memory access as memory bottlenecks arise.

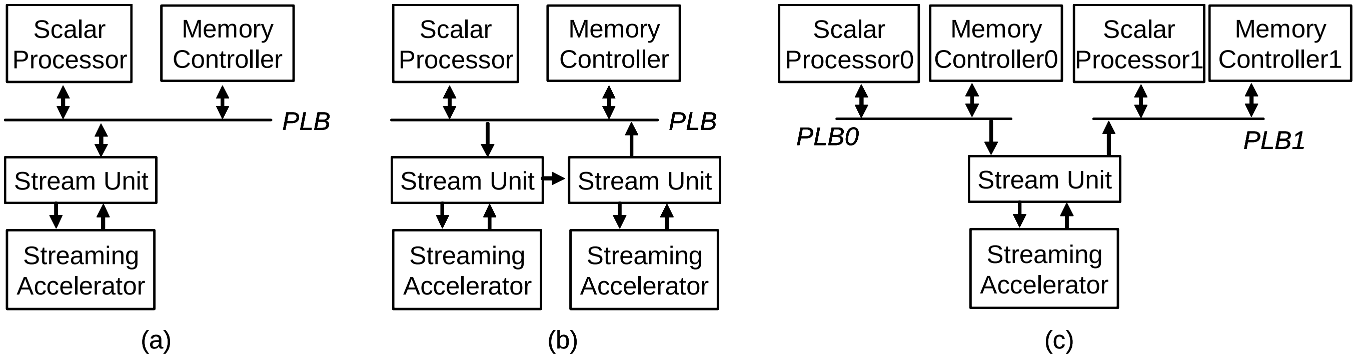

Figure 1. Stream memory subsystem, (a) single accelerator, (b) multiple accelerators, (c) alternative configurations

There is a number of streaming processor architectures developed over recent years. Examples of stream processors include RAW [5], Imagine [6], Merrimac [7], and the RSVP™ architecture [8,9]. There is also another class of streaming architectures with origins from reconfigurable platforms such as FPGA. These architectures rely on the flexibility of the platform to synthesize streaming accelerators based on programmer definition. In comparison to the above mentioned architectures, a set of compiler tools create optimized hardware configurations rather than map computation onto existing design. They are associated with the programming language or compiler tool that allows software developers to configure hardware for stream computation. Examples include SCORE [10], ASC [11], and Streams-C [12].

While each approach is different, stream architectures provide hardware mechanisms that can configure their datapaths for different types of parallelism in stream computation. Furthermore, they include programmable communication infrastructure to move data based on programmer defined API. In this paper, we propose the use of stream descriptors [8,9] for use in a reconfigurable FPGA platform to generate an optimized memory subsystem. Stream descriptors are a language extension to specify memory access patterns, which is used by dedicated stream units to prefetch and assemble data. The programmer describes the computation independently from stream descriptors, and then a compiler synthesizes the proper hardware for stream processing.

III. Stream Memory Hierarchy

A design framework is being developed to automatically generate synthesizable streaming accelerators [13]. Using stream programming languages [9,14,15,16] which includes programmer's explicit definition of streams and their movement, an integrated memory subsystem can be built. This approach selects designs from well-engineered framework consisting of accelerators and network rather than generating hardware from a generic representation of a high level language [17].

The memory subsystem builds upon stream units that moves data based on stream descriptors, as shown in Figure 1. Single or multiple accelerators in various configurations can be built. Furthermore, systems with multiple scalar processors, bus, peripherals or memory controllers can be configured such that the stream unit and accelerator are placed appropriately according to the flow of data. Stream descriptors have been recently applied to stream processors [8,9] and peripherals [18,19] to leverage on the deterministic movements of data from memory. In this paper, the stream descriptors are applied to the entire memory subsystem so as to enable stream data movement throughout the computing platform.

The goal of this research is to generate an optimized memory subsystem based on stream programming input. As data stream type and movement are explicitly defined, there are opportunities to optimize the memory subsystem by prefetching and overlapping movement with computation. By distributing stream units throughout the memory subsystem, the design framework avoid large cache mechanisms that are not efficient for streaming computation and are difficult to synthesize on FPGAs.

This following section describes the stream descriptors used to capture stream access patterns in memory. Furthermore, an example stream unit design is described with preliminary results from synthesis.

A. *Stream Descriptors*

Stream descriptors are mechanisms to allow the programmer to describe the shape and location of data in memory. Dedicated stream units can then utilize the stream descriptors to prefetch data from memory for the computing platform. Each stream unit handles all issues in loading/storing of data: address calculation, byte alignment, data ordering, and memory bus interface. A compiler can also schedule the loading of a stream descriptor that is dependent on run time values.

A stream descriptor is represented by the tuple (Type, Start_Address, Stride, Span Skip, Size) where:

- *Type* indicates how many bytes are in each element (Type is 0 for bytes, 1 for 16-bit half-words, etc.)
- *Start_Address* represents the memory address of the first stream element.

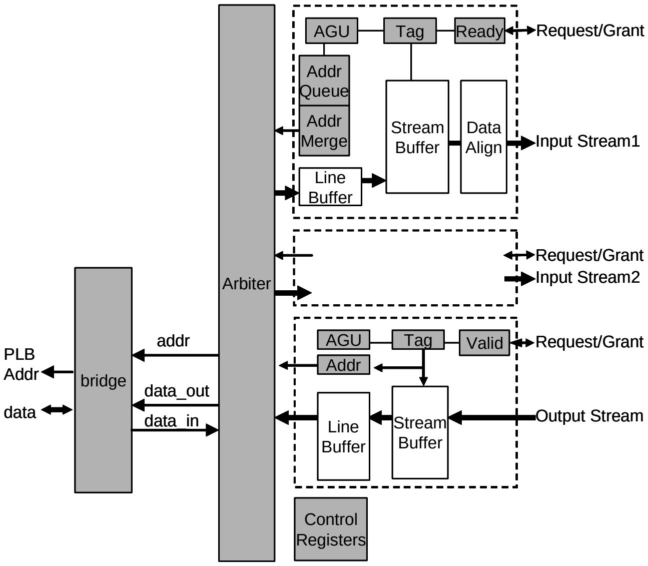

Figure 2. Stream unit block diagram

- *Stride* is the spacing, in number of elements, between two consecutive stream elements.
- *Span* is the number of elements that are gathered before applying the skip offset
- *Skip* is the offset is applied between groups of span elements, after the stride has been applied
- *Size* is the number of elements in the stream

The *Stride*, *Span*, *Skip*, and *Type* fields define the shape of a data object. The grouping and order in which data is accessed defines a *Stream Record* and corresponds to the preferred alignment of the computation kernel. Stream records can be processed in parallel by hardware accelerators and this explicit alignment of the data facilitates their hardware implementation by eliminating packing and unpacking instructions. Multidimensional or even non-regular spaces can be created by extending the defined semantics of each stream descriptor field. More details are available in [8,9].

B. Stream Unit

The stream unit consists of one or more input and output stream modules, which are generated to match the needs of the streaming accelerators. In Figure 2, there are two input and one output stream modules. The stream unit is used to transfer data from a system memory or peripheral, and present them in-order to the streaming accelerator. It also transfers processed data back to other memory locations.

The following paragraph describes internal operations of the input stream module. The address generation unit (AGU) generates bus addresses based on stream descriptor values and stores pending requests in a queue (Addr Queue). The AGU has similar functionality to [20] but with more robust stream descriptors that allows for different bit-widths and more complex access patterns. The Addr Merge unit then selects the next bus address to issue, while removing duplicate bus addresses. Data is then stored in the line buffer when the PLB bus returns data from memory. A Tag unit selects stream elements from the line buffer for storage into a stream buffer queue. Data is then presented to the streaming accelerator as aligned data, in the order defined by the stream descriptor.

The output stream module consists of similar internal components, but data flows in the opposite direction. Processed data is first stored in stream buffers, which are selected for transfer by the Tag unit. A line buffer holds the set of selected stream data which can be stored at a specified bus address, stored in the Addr unit.

The stream unit can be configured to match application requirements based on stream descriptor values, and characteristics of the bus-based system and streaming accelerators. For example, the number of storage elements (*stream buffers*) and their sizes (*bit-width*) are selected based on the stream descriptor values and requested bandwidth of the streaming accelerator (*stream bandwidth*) so that the stream module can provide the maximum number of stream elements requested per cycle. Furthermore, the *Address Queue buffer* size is selected based on the maximum number of pending requests supported by the bus. The bus line buffer size is set based on bus bandwidth and bursting schemes. This would allow maximum saturation of the bus that can pipeline transfer requests from the memory controller or peripherals. Finally, the address generation unit can be hardwired to generate the memory access patterns based on stream descriptors.

Table 1 shows the preliminary synthesis results for different configurations of the stream unit. The resulting clock speeds is about 130MHz on the selected Xilinx FPGA device. In general, the larger the buffer sizes, the larger the stream unit. For larger bit-width parameter, the stream unit gate count can actually decrease due to reduced logic to handle multiple bytes within a 32bit word. The current logic circuits can be further optimized by restructuring the logic in Tag unit which compares against the bus address in Addr Queue unit when accessing the line buffer.

IV. Conclusion and Future Work

This paper presents a configurable stream unit for use in a stream memory hierarchy. The stream unit prefetches and aligns data for streaming accelerators based on a set of stream descriptors, which defines the data shape and location. Preliminary synthesis results on the different configurations are shown.

Future work for the stream unit includes further optimization and use of the design in a full memory subsystem. Benchmarking on applications can be performed with a suite of applications. Integration with streaming peripherals[18] and a high performance memory controller would improve performance. There are additional research areas in integrating streaming descriptors within a streaming language and compiler infrastructure.

Table 1. Preliminary implementation results[1]

Address queue buffer[2]		Input stream buffers[3]		Input stream bit-width[4]		Output stream bit-width[5]		Input stream bandwidth[6]		Output stream bandwidth[7]	
2	1779	8	1862	8	1862	8	1862	1	1862	1	1862
4	1862	16	2350	16	1744	16	1863	2	1878	2	1869
8	2050	32	3209	32	1605	32	1863	4	1915	4	1930
16	2072							8	1956		

[1] All data given in number of slices in the Xilinx 4vfx140ff1760-11 device

[2] Input/output buffer = 8, input/output bit-width = 8, input/output bandwidth = 1

[3] Address queue buffer = 4, input/output bit-width = 8, input/output bandwidth = 1

[4] Address queue buffer = 4, input/output buffer = 8, input/output bandwidth = 1

[5] Address queue buffer = 4, input/output buffer = 8, input/output bandwidth = 1

[6] Address queue buffer = 4, input/output buffer = 8, input/output bit-width = 8

[7] Address queue buffer = 4, input/output buffer = 8, input/output bit-width = 8

Acknowledgment

The authors acknowledge previous contributions by RSVP™ design team at Motorola Labs. Furthermore, the authors extend thanks to Erica Lau for her invaluable feedback that has greatly improved the different aspects described in this paper.

References

[1] David A. Patterson, “Latency Lags Bandwidth,” Communications of the ACM, vol.47, no.10, pp.71-75 October 2004.

[2] W.A. Wulf, S. A. McKee, “Hitting the memory wall: implications of the obvious,” ACM SIGARCH Computer Architecture News, Vol. 23, No. 1, March 1995, pp. 20-24.

[3] Parthasarathy Ranganathan, Sarita Adve, Norman P. Jouppi, “Performance of image and video processing with general-purpose processors and media ISA extensions,” *Proceedings of the 26th Annual International Symposium on Computer Architecture* (ISCA’99), May 1999, pp. 124-135

[4] Saman P. Amarasinghe; William Thies, “Architecture, languages and compilers for the Streaming Domain,” PACT 2003 Tutorial., http://cag.lcs.mit.edu/wss03/.

[5] Michael Bedford, et al, “Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams,” *Proceedings of the 31st Annual International Symposium on Computer Architecture* (ISCA’04), June 2004, pp. 2-14.

[6] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucek Khailany, Abelardo López-Lagunas, Peter R. Mattson, John D. Owens, “A bandwidth-efficient architecture for media processing,” *Proceedings of the 31st annual ACM/IEEE International Symposium on Microarchitecture*, November 1998, pp. 3-13.

[7] William J. Dally; Patrick Hanrahan; Mattan Erez; Timothy J. Knight; François Labonté; Jung-Ho Ahn; Nuwan Jayasena; Ujval J. Kapasi; Abhishek Das; Jayanth Gummaraju; Ian Buck, "Merrimac: Supercomputing with streams", *Proceedings of the SuperComputing SC’03 Conference*, November 2003, Phoenix, Arizona, pp. 35-43.

[8] S. Chiricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Saidi, "The Reconfigurable Streaming Vector Processor (RSVP™)," *Proceedings of the 36th International Symposium on Microarchitecture*, December 2003, pp. 141-150.

[9] S. M. Chai, S. Chiricescu, R. Essick, A. López-Lagunas, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, “Streaming Processors for Next Generation Mobile Imaging Applications,” to appear in IEEE Communications Magazine, Special Topics in Circuits for Communication Series (Mobile Multimedia), Dec 2005

[10] Eylon Caspi, Michael Chu, Randy Huang, Joseph Yeh, John Wawrzynek, André DeHon. Stream Computations Organized for Reconfigurable Execution (SCORE). Field Programmable Logic (FPL), August 2000, pp. 605-614

[11] Oskar Mencer, David J. Pearce, lee W. Howes, Wayne Luk, “Design Space Exploration with a Stream Compiler”, IEEE International Conference on Field Programmable Technology (FPT), Tokyo, December 2003

[12] Maya Gokhale, Jan Sone, Jeff Arnold, Mirek Kalinowski, “Stream-Oriented FPGA Computing in the Streams-C High Level Language”, IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), pp 49-56, 2000.

[13] N. Bellas, S. Chai, M. Dwyer, and D. Linzmeier, “FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators,” submitted to Reconfigurable Architecture Workshop 2006.

[14] Ian Buck, “Current Brook Specification (0.2),” October 31, 2003. http://merrimac.stanford.edu/brook/brookspec-v0.2.pdf

[15] William R. Mark; R. Steven Glanville; Kurt Akeley; Mark J. Kilgard, “Cg: a system for programming graphics hardware in a C-like language,” July 2003 ACM Transactions on Graphics (TOG), Volume 22 Issue 3.

[16] P. Mattson, B. Thies, L. Hammond, M. Vahey “Streaming Virtual Machine Specification,” Morphware Forum, Version 1.0 July 19, 2004

[17] Handel-C. Language Reference Manual, /www.celoxica.com /

[18] S. M. Chai and A. López-Lagunas, “Streaming I/O for Imaging Applications”, *IEEE International Conference on Computer Architecture for Machine Perception*, July 2005, pp. 178-183.

[19] A. López-Lagunas and S. M. Chai, “Memory Bandwidth Optimization through Stream Descriptors”, Memory Performance: Dealing with Applications, Systems and Architecture (MEDEA) Workshop, St. Louis, Missouri, September 2005, pp. 59-66.

[20] Kjetil E. Vistnes, Oddvar Sorasen, “Reconfigurable Address Generators for Stream-Based Computation Implemented on FPGAs,” Reconfigurable Architecture Workshop 2005.