

An External Memory Interface for FPGA-Based Computing Engines*

Joonseok Park and Pedro Diniz

University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, California 90292
{joonseok, pedro}@isi.edu

1. Introduction

Stream-based computations, common in image-processing applications, are a natural match for Field-Programmable-Gate-Arrays (FPGAs). These computations typically manipulate large volumes of fine-grain data that, despite recent increases in FPGA capacity, must be mapped to external memories and then streamed through the FPGAs for processing.

Commercially available synthesis tools offer limited capabilities when dealing with external memories. This problem is exacerbated by the diversity of each vendor-specific interface. Often vendor interfaces are specified in structural VHDL and where timing and its latency models are not directly exposed to the synthesis tools. As a result designers are faced with the tenuous tasks of manually matching the design specification to the vendor-provided interface.

In this abstract we address issues relating to the interface of external memories in FPGA-based designs by proposing an architecture that interfaces with external memories. The proposed architecture relies on two interfaces and a set of parameterizable abstractions that can be integrated with existing behavioral and structural synthesis tools. In addition, the proposed architecture allows the designer to exploit application-specific data access patterns by providing support for pipelined memory access as well as allowing the designer to define specific memory access controllers.

2. Decoupled Architecture

The proposed architecture aims at decoupling target-architecture dependent characteristics from the datapath that implements the application. This decoupling is achieved by defining two interfaces. One interface generates all the external memory control signals matching the vendor-specific interface signals – *physical address*, *enable*, *write_enable*, *data_in/out*, etc.

* Funded by the Defense Advanced Research Project Agency under contract number F30603-98-2-0113

A second interface communicates with the datapath and implements the transfer of data to the various data ports of the design. We have implemented these two interfaces over an architecture as depicted in Figure 1. This architecture consists of Conversion FIFO Queues, Stream Channels, Address Generation Unit, and a Memory Channel Controller which we describe next.

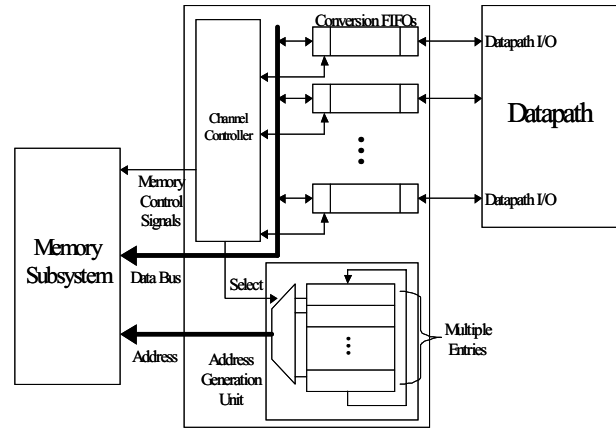


Figure 1. Decoupled Memory Architecture Diagram.

Conversion FIFO Queues

The current design uses FIFO queues for several purposes. First the datapath implementation is decoupled from the timing issues relating to memory accesses. The memory controller simply needs to test if each input/output queue is empty/full to retrieve/store the following data item as specified by the stream channels parameters (see below). Second, there is a substantial simplification of the execution control for the core datapath. Getting data from the FIFO queues amounts to an handshaking protocol with an *empty/full* and *push/pop* set of signals. Lastly, the conversion FIFO queues allow for the implementation of data packing/unpacking operations. Image processing applications that operate on pixel-based images using 8 bit values can have a substantial reduction in the number of memory accesses as a single 32 bit memory transfer can retrieve/store 4 consecutive pixel values.

Stream Channels

They are defined by the tuple (*queue*, *base*, *offset*, *stride*, *dir*) indicating a sequence of consecutive address for the data to be fetched for a particular FIFO queue, where *queue* indicated the identity of the source/destination queue, *base*, *offset* and *stride* are the typical memory address calculation parameters and *dir* indicates either a *read* or a *write*.

Address Generation Unit (AGU)

Because stream-based applications make heavy use of fine-grain streamed data access patterns, we have developed hardware to directly support stream-based data access modes. The AGU generates physical addresses for stream channels. The current AGU design implementation (depicted in Figure 2) has auto-incrementing features to generate subsequent addresses for a given stream channel. The design also allows for a host program to directly write or read (via a specific set of addresses in the host memory map) the current values of the internal registers, thereby allowing changing the base and offset values for a specific stream channel.

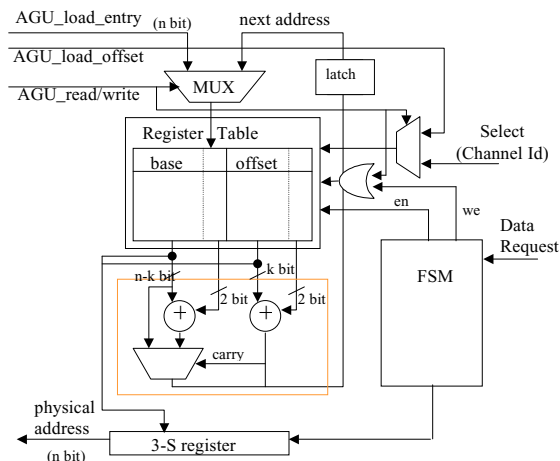


Figure 2. Address Generation Unit (AGU) Implementation.

Memory Channel Controller

The memory channel controller serializes the multiple memory requests for the various active stream channels. Internally it consists of three FSMs. One FSM is responsible for computing and emitting the address for each channel that requires data. Another FSM generates the hardware signals according to the vendor specific memory interface timing specifications. The third FSM synchronizes the other two FSMs.

The current implementation allows for application-specific scheduling by defining a distinct orderings of the memory accesses for the different stream channels. We have also incorporated support for pipelined memory accesses across multiple channels

to reduce latency when these features are supported by the specific vendor-interface as is the case with our current target FPGA-board – the WildStarTM [2].

3. Discussion and Summary

The proposed architecture and related interfaces presents several advantages over the current design implementation practices. First, we provide a target independent view of the core datapath design and in particular execution control. The simple full/empty and pop/push set of signals protocol for retrieving/storing data from the channels allows for a smooth integration of behavioral design specifications with the structural descriptions that interface with memory. Second, given that the components of the proposed architecture and interfaces are parameterizable, we developed several code generation functions that can be integrated as part of a compilation system. Third, the decoupling of the scheduling of the computation in the core datapath with the memory accesses exposes several opportunities for memory operation optimizations, which are beyond the scope of current behavioral tools - memory operation pipelining and grouping[3].

The main disadvantage of the proposed approach is the incurred overhead of memory operations by the additions of an extra layer of abstractions (e.g., the stream channels) and their interfaces. While this is a potential disadvantage for increasing the latency of memory accesses, throughput and also clock rates can potentially be improved due to the simpler (and therefore possibly shorter) connections between the core datapath and the FIFO queues. Also, deeper conversion FIFO queues will allow overlapping of computation with communication.

We believe the advantages of simpler interfaces and amenability for automation overall outweigh the disadvantages from additional latency. We have successfully integrated the external memory architecture presented here in the context of the DEFACTO[1] for a set of simple image processing kernels without a substantial performance sacrifice.

References

- [1] K. Bondalapati, P. Diniz, P. Duncan, J. Granacki, M. Hall, R. Jain, "DEFACTO: A Design Environment for Adaptive Computing TechnOlogy", In Proc. of the 1999 Workshop on Reconfigurable Computing, Puerto Rico, April 1999.
- [2] WildStarTM Reference Manual revision 4.0, Annapolis MicroSystems Inc., 1999.
- [3] John P. Elliot Understanding Behavioral Synthesis, A practical guide to high-level design, Kluwer Academic Publishers 1999.