Pipelined memory controllers for DSP applications handling unpredictable data acesses

Bertrand Le Gal, Emmanuel Casseau, Sylvain Huet, Eric Martin

Pipelined Memory Controllers for DSP Applications Handling Unpredictable Data Accesses

Bertrand Le Gal, Emmanuel Casseau, Sylvain Huet, Eric Martin
LESTER Laboratory - University of South Brittany
Lorient - FRANCE
name.surname@univ-ubs.fr

Abstract

Multimedia applications are often characterized by a large number of data accesses with regular and periodic access patterns. In these cases, optimized pipelined memory access controllers can be generated improving the pipeline access mode to RAM. We focus on the design and the implementation of memory sequencers that can be automatically generated from a behavioral synthesis tool and which can efficiently handle predictable address patterns as well as unpredictable ones (dynamic address computations) in a pipeline way.

1 Introduction

Actual researches in Multimedia applications try to reduce the computation complexity of algorithms using adhoc solution composed of conditional computations leading to execution hazards to appear with conditional and unbounded loop usages. On the other hand other architectural implementations (computation and memory architectures) are obtained for regular algorithms without execution hazard. For most of multimedia applications the entire memory access sequence is not known *a priori*. This prevents the designer and High-Level Synthesis tools to handle efficiently the application repetitive sequences for efficient area and power consumption design. In many digital signal-processing applications, the array access patterns are predictable, regular and periodic. In these cases, the necessary address patterns can be efficiently generated either directly from a memory address sequencer. The sequencer generation allows the designer to decouple the concerns of memory interfacing and static scheduling of possible memory accesses [2]. This technique is used to improve the pipeline access mode to RAM. Researches [1] have demonstrated the interest of an address sequencer utilization for memory access for power efficient circuit generation.

In this paper, we first present a design flow for the our sequencer which handles hazardous memory accesses providing the ability of accessing to a dynamically calculated addresses in a data-flow fashion. In a second time we present our sequencer architectures which can perform pipeline memory access for statically and dynamic access sequences. Experimental results are discussed in the conclusion.

2 The Design Flow

Our Design Flow (figure 1) starting point is a behavioral description of the application specifying the circuit functionality and a memory mapping specifying which data is in memory. In order to map an application onto the memory sequencer architecture, we define a graph model name *Extended Data-Flow Graph* which takes into account the required timing constraints due to dynamic address accesses.

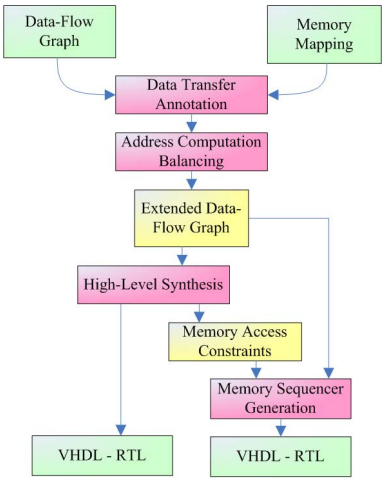

Figure 1. Design Flow

The first step of our design flow for dynamic address se-

quencer generation is the *Extended Data-Flow Graph* annotation : this annotation step aims to handle the timing requirements for data and address transfers from one unit to the other (Communication, Memory and Datapath unit).

In second step, a dynamic address computation balancing algorithm is applied to the previously annotated graph in order to move some dynamic address computations from datapath to the memory sequencer unit reducing transfer count, power consumption and delays. The decision metric used to select address computations which must balanced, takes into account different criteria: the number of data-transfer, the time increase/decrease for critical paths, the bitwise of the operators and the potential parallelism that can be exploited for computations and data-transfers. This dynamic address computation balancing script is applied in a static manner to the annotated graph. Depending on the application of this script, the sequencer architecture may need an internal datapath for address computations.

Then the designer can implement is datapath by hand or using a high-level synthesis tool. During this process, the HLS tool takes into account the timing constrains for dynamic data addressing and data transfers using the Extended Data-Flow Graph annotations.

Finally, the entire sequencer is generated using the previously generated informations on the access patterns.

3 Dynamic Address Sequencer Architecture

This sequencer supports dynamic addressing capabilities in a mainly deterministic data transfer sequence. In this architecture (figure 2, without the dynamic address datapath and its controller), we suppose that all the dynamic address computations are calculated inside the datapath unit and then transfered using data buses to the memory sequencer. The datapath access buses are connected to the memory using a multiplexed crossbar which is controlled through the memory access scheduler. This scheduler controls the address generator progress in a synchronous manner from the datapath point of view. A dynamic address access to the memory will go through the address translation table, this table will translate, the index of the data to a couple (memory bank, address). This translation allows the designer to bind noncontiguous pieces of a vector into different memories to better exploit access parallelism. The dynamic access controller will route the correct commands (read/write) and physical address to the right memory bank according to the dynamic access required.

Sequencer Architecture Improvement : We extend this architecture for massive transfers reduction by inserting dedicated computation units in the memory sequencer unit. These computation units perform internal address computations realizing the datapath unit constraints for eligible

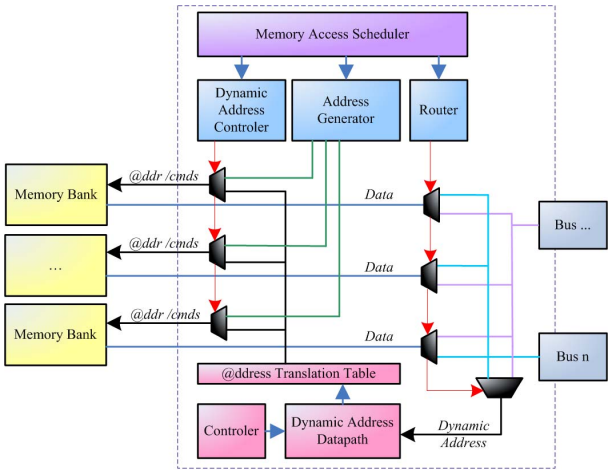

Figure 2. Dynamic Address Sequencer

address computations. This approach provides interesting gains for pipeline architecture design : the data transfers between the datapath and the sequencer can take more than one clock cycle. In these applications, localizing computations in the memory sequencer will reduce latency and avoid unnecessary address transfers. An internal datapath specialized in dynamic address computations is inserted before the translation table (figure 2). It is composed of operators and registers like a datapath. This dedicated hardware is shared between address computations during application execution.

4 Conclusion

Several experiments we have performed which exhibit the interest of this approach. This technique reduce the data transfer requirements for low-conditional high-computation applications exploiting data intensive techniques. This technique also allows performance increases by using an internal sequencer datapath for dynamic address computations reducing delay in pipeline architectural implementations.

References

[1] Taewhan Kim Chun-Gi Lyuh and Ki-Wook Kim. Coupling-aware high-level interconnect synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):157–164, January 2004.

[2] Joonseok Park and Pedro C. Diniz. Synthesis of pipelined memory access controllers for streamed data applications on fpga-based computing engines. In *Proc. of ISSS'01*, pages 221–226. ACM Press, 2001.