PIPELINED MEMORY CONTROLLERS FOR DSP REAL-TIME APPLICATIONS HANDLING UNPREDICTABLE DATA ACCESSES

Bertrand Le Gal, Emmanuel Casseau and Eric Martin

LESTER Laboratory, CNRS FRE2734
University of South Brittany, FRANCE
phone: + (0033), fax: + (0033), email: *name.surname*@univ-ubs.fr
web: web.univ-ubs.fr/lester

ABSTRACT

Multimedia applications such as video and image processing are often characterized by a large number of data accesses. In many digital signal-processing applications, the array access patterns are regular and periodic. In these cases, it becomes feasible and efficient to generate optimized Pipelined Memory Access Controllers. This technique is used to improve the pipeline access mode to RAM by creating specialized hardware components for generating addresses and packing and unpacking data items. In this paper we focus on the design, implementation and validation of external memory interfacing modules which can efficiently handle predictable address patterns as well as unpredictable (dynamic address computations) in a pipeline way. In a second time, we analyze the benefits of balancing dynamic address computation from datapath to dedicated units in the memory controller, optimizing bitwise of operators, and data locality (decreasing bus transfers for power efficient design).

1. INTRODUCTION

Multimedia applications such as video and image processing are often characterized by a large number of data accesses. In such applications, the memory accesses are often the limiting factor to the execution speed of Digital Signal Processing (DSP) hardware processors. Performances highly depend on the memory architecture: hierarchy, number of banks, data placement, etc. Memory design also affects the power consumption which is a critical feature in embedded applications. In the same time, custom hardware design increases the design time.

On one hand, actual researches in Multimedia applications try to reduce algorithm calculation complexity using ad-hoc solution composed of conditional computations (for example transformation of the *Full Search algorithm* for Block Matching to a *Three Step Search algorithm* [7]) leading to execution hazards that may occur with conditional computations and dynamic macro-bloc selection. On the other hand, other researches based on architectural implementations of these algorithms under real-time constraints try to exploit the parallelism of the computations. In that case optimized circuits are obtained for regular algorithms without execution hazard. This is also true for the memory architecture, which is more advantageous for data-flow applications where the sequence of accesses to the memory is predictable.

However, for most of the actual signal and image applications the entire access sequence to the memory is not known *a priori*. This prevents the designers to handle efficiently the repetitive sequences of the application. So according to the application class, the design flow for optimal area and power consumption design generation to use is different.

In this paper, we make the following contributions: we present a new address memory sequencer architecture which can perform pipeline memory accesses for static and dynamic access sequences. We show how to include dynamic memory access constraints in a Data Flow Graph (DFG). This technique will be extended in a second time with the dynamic address computation balancing between the datapath and the address sequencer for performance increase and data transfer reduction in a low-power perspective. The results presented in this paper support the claim that it is possible to exploit application specific information and integrate that knowledge in a custom access sequencer module for reducing the overhead associated with memory hazardous accesses.

2. RELATED WORK

A good overview about memory synthesis and optimization can be found in [5]. Most researches for control, intensive applications are based on a random access memory (RAM). In these kinds of applications, a common RAM based model is used; it accepts a binary coded address and decodes it using built-in decoders into row and column select signals because of un-predictable memory access.

In many digital signal-processing applications, the array access patterns are regular and periodic. In these cases, it becomes feasible and efficient to generate the necessary address patterns directly from memory address sequencer (using dedicated counter[2], etc.). Data processing can easily be speeded up through pipelining and other forms of parallelism. In the simplest case where the memory accesses can be determined statically, the compiler can schedule the order in which the accesses occur. The scheduler can be implemented by example in hardware using a Finite State Machine (FSM).

Sequencer architecture allows un-correlation between the datapath unit synthesis and its optimizations and the sequencer architecture synthesis and its optimizations. As shown in figure 1, only the produced or consumed data are transferred from one unit to the other (the address transfers are useless since their sequences are static *a priori*).

Studies have demonstrated that custom address generator created from the access patterns can be optimized to obtain optimal memory architectures. For example, these sequencers can be optimized by applying minimizing power consumption techniques on buses [3], accessing adjacent data in memory to limit the commutations on address buses [1]. Performance optimizations can also be performed. In

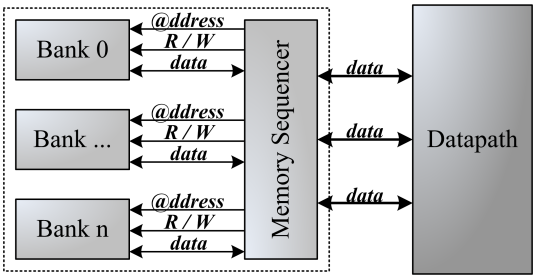


Figure 1: Sequencer-based Architecture for Predictive Data Transfer Sequence

[4], the author examines the impact on area and performance of memory access related circuitry by using simplified architectures for the address generation circuit. This technique improves the design frequency for important access sequences.

Nevertheless, these approaches are limited to predictive access patterns. The sequencer generation also allows the designer to decouple the concerns of memory interfacing and static scheduling of possible memory accesses for applications with streamed data [6]. This technique is used to improve the pipeline access mode to RAM by creating specialized hardware components for generating addresses and packing and unpacking data items (figure 2). Unfortunately, this approach and the sequencer implementation model do not support dynamic address calculation and the synthesis process constraints the controller generation without interaction for avoiding over constraints.

Our researches in this paper differ from these studies. Common approaches create a centralized memory scheduler that fetches data from an external memory into the datapath are based *only on predictable access patterns.* Whereas our approach relies on both, an architecture independent view of the data path and a memory controller which is decoupled from the datapath execution controller, thus allowing *unpredictable memory accesses*.

3. IMPLEMENTATION MODEL

Our goal is to generate an optimal memory sequencer supporting dynamic addressing access in a mainly deterministic data transfer sequence. We present in a first time an architecture allowing dynamic addressing, and then in a second time an extended architecture able to make internal bitwise optimized address computations in the sequencer, avoiding repetitive address transfers between the datapath and the memory sequencer.

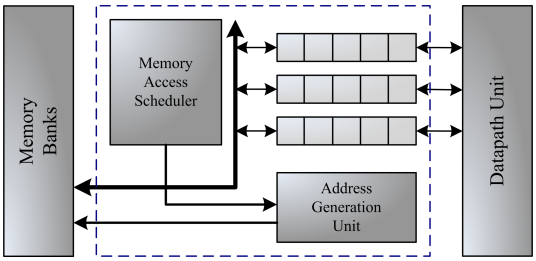


Figure 2: Park's Sequencer Architecture

3.1 Dynamic Address Access Sequencer

Our first sequencer architecture supposes that all the dynamic address computations are performed inside the data path unit and then transferred to the memory sequencer thought a data/address bus. The sequencer proposed in figure 3 is composed of 4 different units: a memory access scheduler, a dynamic address controller, an address generator and an address translation table.

Buses from the datapath are connected to the memory using a multiplexed crossbar. This crossbar is piloted by the *memory access scheduler*, which knows the memory access sequence. This scheduler controls the address generator progress in a synchronous manner from the datapath point of view. Dynamic address accesses to the memory will go thought the address translation table, this table will translate, the logical address or index to the couple (memory bank, address). This translation table allows the designer to bind pieces of a vector in different memories to exploit application access parallelism.

Depending on the targeted memory bank and the dynamic address access usage in the current cycle, the dynamic access controller will route the correct commands (read/write) and the physical address to the right memory bank.

This sequencer architecture allows dynamic memory accesses to be realized in a "static" sequencer approach. This also allows the designer to bind freely the split vectors in different memory banks.

3.2 Memory Sequencer Architecture for Dynamic Address Computation Balancing

The previously presented architecture allows dynamic address memory accesses using a sequencer architecture approach. This architecture allows local optimizations between this unit and the memory banks and it also reduces the memory address transfers from the datapath to the memory. We now extend it reduction by inserting computation units dedicated to the dynamic memory address in the sequencer unit.

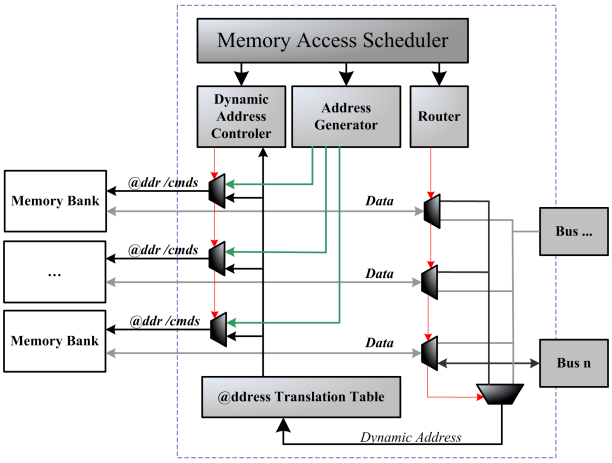


Figure 3: Predictive and Dynamic Data Transfer Sequences

Our second architecture is presented in figure 4. A dedicated datapath is added to the translation table for the address computations.

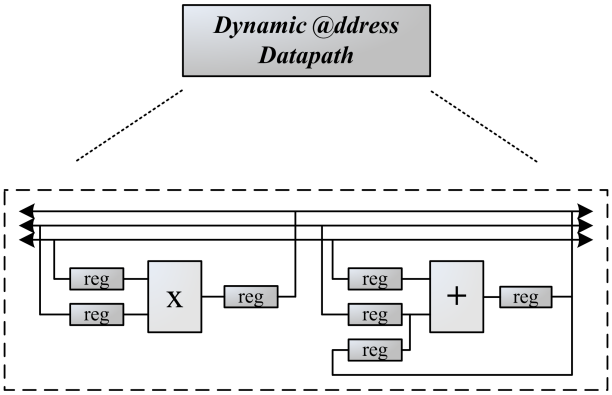


Figure 4: Extended Sequencer Architecture

This approach provides interesting gains for high performance circuit designs: in pipelined architectures, data transfers between the datapath and the sequencer can use more than one clock cycle (low-speed memories by example). In these cases, localizing address computations in the sequencer provides important latency gain by avoiding address transfers between the units. The address traffic reduction between the datapath and the sequencer also reduces the line switching, i.e. the power consumption. Furthermore this approach allows decreasing the bus requirement between the datapath and the sequencer.

This internal datapath is dedicated to the dynamic address computation (i.e. logical and arithmetic operations, like increment, shifts, etc.). These operators are optimized for address computation versus the datapath unit because address operator bitwise can be easily reduced. Another advantage from this approach comes from the locality of the constants or variables implied in the address computation.

The address computation unit is composed of operators and registers. The registers stores the addresses needed for dynamic addressing during computation and then transfer them to the address translation table when needed for the dynamic memory access.

4. DESIGN FLOW

In order to generate a memory sequencer architecture from a signal or image application, we present in figure 5 a design flow which allow the designer to handle the sequencer constraints during the circuit design process. The starting point of this design flow is a Data-Flow Graph modelling the behavioural of the application and optionally memory mapping.

The definition of the memory architecture (especially data mapping) can be performed in the first step of the overall design flow. To achieve this task, the designer can use advanced compilers such as Rice HPF compiler, Illinois Polaris or Stanford SUIF [5]. Indeed, these compilers automatically perform data distribution across banks, determine which access goes to which bank, and then schedule them to avoid bank conflicts. The Data Transfer and Storage Exploration (DTSE) method from IMEC and the associated tools (ATOMIUM, ADOPT) are also a good mean to determine a

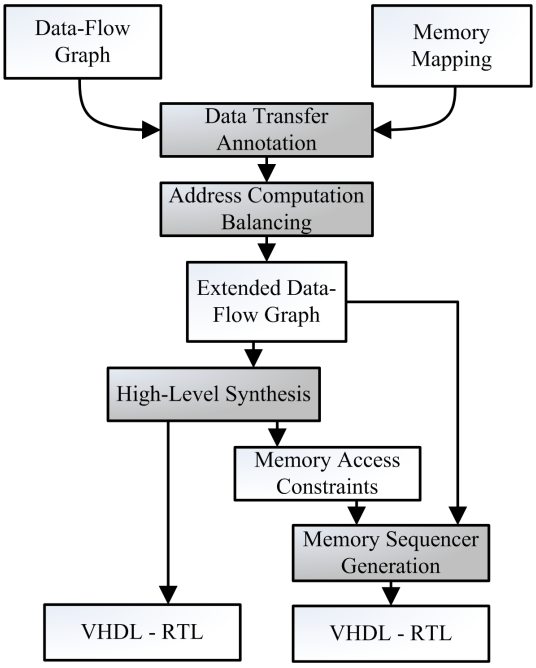


Figure 5: Design Flow

convenient data mapping. Otherwise the designer may decide to make this task later. Our methodology can independently handle the two approaches in a unique design flow.

Data transfer annotations

The first step of our design flow for dynamic address sequencer generation is the data flow graph annotation (fig. 6a, 6b): this annotation step aims at handling the timing requirements for data and address transfers from one unit to the other (Communication unit, Memory units and the Datapath unit). This also handles the dynamic address accesses requirements. This transformation can be guided by the memory mapping information given by the designer if the data placements were done before synthesis in the design flow. These annotations also contain information on the locality of the operations and data memorization (memory, register in datapath) In this first approach, all the dynamic address computations are allocated in the datapath unit.

Computation Balancing Metric

In a second time, the 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. The decision metric used to balance the address computations take different criteria into account: the number of data-transfer needed, the time increase/decrease of critical paths, addresses bitwise versus the datapath bitwise, etc.

The dynamic address computation balancing algorithm is applied in a static manner to the annotated graph. All the address computations are evaluated for balancing decision. If the balancing decision optimizes the system, then the graph is transformed: transferring nodes are added, others are deleted and the locality attributes for nodes are changed.

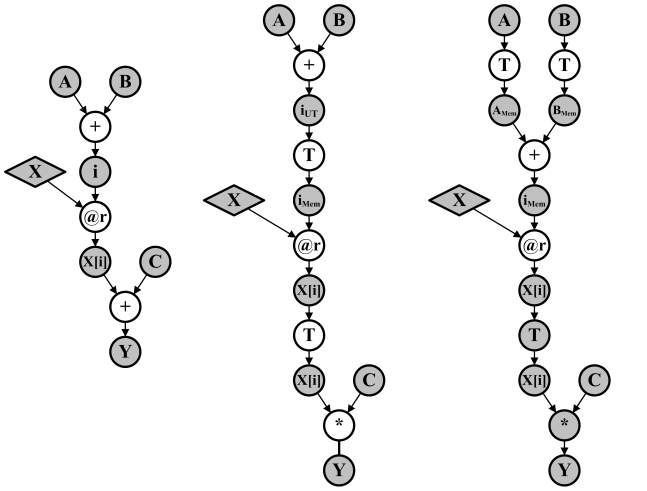
An example of these transformations is shown in figure 6 (b and c).

Datapath and Memory Sequencer Synthesis

Then the designer can implement its datapath by hand coding or using a high-level synthesis tool without regarding the memory implementation problems. At the end of the process, the datapath architecture is generated as sequencer datapath. In order to generate the entire sequencer, other information is generated as access pattern including dynamic address access time slot.

Memory Sequencer Hardware Generation

At the end, the entire sequencer is generated using the previously generated information on the access patterns. With this information the Memory Scheduler and the Memory Address Generators can be implemented using local optimizations. The internal datapath dedicated for dynamic address computation is then inserted in the sequencer architecture.



(a) Original graph, (b) Data Transfert Annotated Graph (c) Address Computations balanced Graph

Figure 6: Extended Data-Flow Graph Example

We are currently developing a tool that will automate this important system design step.

5. EXPERIMENTS

We applied our sequencer design approach on the Three Step Search algorithm [7]. In this algorithm, an un-deterministic access sequence is applied. In the control architecture, we consider the address transfer of the reference macro bloc plus the dynamic bloc accesses. In the second one all 5 first bloc transfers are know *a priori* as the reference bloc transfer. And in the second optimized sequencer architecture we only transfer the base address for the dynamic macro bloc access.

Results are presented in figure 7. A sequencer for deterministic memory access can be use in data-dominated application using low control operations. The results show the address transfers count from a normal approach using data-computation units for address computation in the sequencer. This is useful in a pipeline architecture when address transfers take more than one clock cycle. Moreover our sequencer

	Macro Bloc	Search Window	Addr. Transfert
Control Architecture	8X8	16x16	896
First Sequencer Architecture			512
Second Sequencer Architecture			8
Control Architecture	24x24	48x48	3584
First Sequencer Architecture			2048
Second Sequencer Architecture			8

Figure 7: Sequencers results versus Control Architecture

permits predictive and pipeline memory access as said in [6] which is not taking in account in these experiments.

6. CONCLUSION

In this paper we have presented a new sequencer architecture for DSP applications with dynamic memory accesses. We show that our design flow allows the designer to freely optimize each unit of his design. Future work will be to incorporate the sequencer generation in a High-Level Synthesis design flow to automatically generate the entire design. We will also handle conditional memory accesses, which are actually, converted to non-conditional accesses by speculatively fetching all of the potentially required.

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] P.B Denyer D. Grant and I. Finlay. Synthesis of address generators. In *Proceedings of ICCAD 89*, pages 116–119, 1989.

[3] A. Dasgupta and R. Karri. High-reliability, low-energy microarchitecture synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1273–1280, 1998.

[4] S. Hettiaratchi, P. Cheung, and T. Clarke. Performance-area trade-off of address generators for address decoder-decoupled memory. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, page 902. IEEE Computer Society, 2002.

[5] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 6(2):149–206, 2001.

[6] Joonseok Park and Pedro C. Diniz. Synthesis of pipelined memory access controllers for streamed data applications on fpga-based computing engines. In *ISSS '01: Proceedings of the 14th international symposium on Systems synthesis*, pages 221–226. ACM Press, 2001.

[7] B.Zeng R.Li and M.L.Liou. A new three-step search algorithm for block motion estimation. *IEEE Transaction on Circuits and systems for Video Technology*, 4(4):438–442, August 1994.