

# Memory Specification for Reconfigurable Computing Synthesis Tools

John Xue and Peter Sutton

*School of Information Technology and Electrical Engineering*

*The University of Queensland*

*Brisbane Australia 4072*

*{xue, p.sutton@itee.uq.edu.au}*

## Abstract

*To assist with the automatic synthesis and optimisation of the memory interface in a reconfigurable system, it is proposed that a memory specification language be developed that caters for a range of common memory architectures and informs a synthesis tool in making decisions on issues such as partitioning and scheduling. Such a modified design flow that allows a synthesis tool to manage the memory interfacing will allow designers to concentrate on higher level design issues.*

*A memory specification language approach differs from other approaches as the interface is not as constrained by a generic interface and more easily provides a wider range of supported memory systems.*

## 1. Introduction

One of the most important parts of the design of reconfigurable systems is interfacing to the memory and ensuring the design utilises the memory effectively. Current synthesis tools targeted at reconfigurable systems are lacking in the area of memory synthesis, often relying on the designer to manually handle the memory issues. Research in this area has yielded work that considers the number of memory banks and the various characteristics of those banks (see section 2.1); however, most of the research work has been confined to specific memory architectures and lacks flexibility in the platforms targeted.

Future synthesis tools that can automatically target multiple memory architectures would be a boon for designers, but will require a change in the design flow where the synthesis tools will handle the memory interfacing. The first step to achieving this is creating a specification language for the memory system. This is required as reconfigurable systems come with a wide range of attached memory systems.

Synthesis techniques for reconfigurable systems that consider the memory system during synthesis already

consider some attributes for the memory banks they cater for [1]. While these techniques allow for some bank attribute variation, variations in the overall memory architecture are usually not captured. As synthesis tools mature, they will attempt to resolve this problem.

To help support this, a specification language for memory systems is proposed that is designed to assist in automatic synthesis for memory interfacing in reconfigurable systems. It will do this by providing a flexible method of describing a wide variety of memories and architectures and providing it in a format that is relevant to the synthesis and optimisations currently in use.

This idea differs from other approaches that use a wrapper interface to the memory [2, 3], as the synthesis tool will be able to analyse the memory architecture and more tightly integrate optimisations to the memory architecture than a wrapper interface can hope to achieve.

The following sections describe the current progress of synthesis tool technology, why a memory specification language would be useful, and the significance of memory optimisation methodologies.

## 2. Memory Handling in Synthesis Tools

When automatic memory interfacing is used in the design process, there are often trade offs made with the amount of work handled by the designer, the flexibility in how the provided memory interfaces can be used and the flexibility in the platforms supported.

A memory specification language can help to automate more of the design process as the memory interface can be generated in response to design parameters. This allow for flexibility in the platform while keeping the flexibility in the memory interface.

### 2.1. Some Shortfalls in Automatic Synthesis

Current reconfigurable synthesis tools often leave the bulk of memory interfacing up to the designer. This

is because of the combined variability of both the application and the memory architecture. A common methodology is to provide the designer with simulation and verification tools that allow for faster analysis of various designs.

While the designer will often wish to manage the implementation of the memory there are times when allowing the synthesis tool to do it is preferred. One major reason is when a design needs to be ported to a different platform, as manually repartitioning the memory and adjusting the timing of the memory accesses is time consuming.

Also, as the technology develops it will be inevitable that designs will become more abstract as better automatic synthesis tools are created and time to market pushes for faster design time. Hiding much of the tedious task of memory interfacing would be one of the major issues in achieving better abstraction in hardware designs.

Some of the research synthesis tools such as DEFACTO [2, 4] and StreamsC [5] have looked more deeply into the issue of flexibility and their solution is a wrapper interface that connects to different memory architectures. This provides flexibility, but at the cost of possible optimisations as the synthesis tool is still not aware of the architectural information that may affect decision making. The interface effectively hides the memory architecture from the synthesis decision making.

Consequently, the synthesis tools which handle memory synthesis are restricted to only a subset of the possible reconfigurable platforms. To pave the way for synthesis of platform independent designs, synthesis tools need to work irrespective of variations in the memory architecture.

## 2.2. A Different Approach

The standard approach is that the memory architecture description and consequent interface is integrated into the system by the designer during the creation of the design shown in Figure 1. For such systems as DEFACTO and StreamsC a special interface is written and placed in a library that the synthesis tool uses.

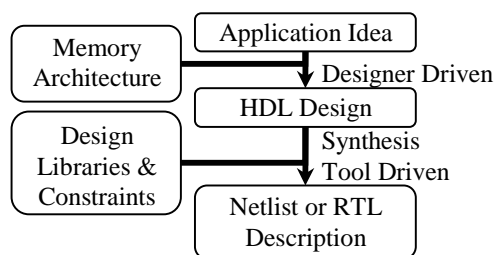


Figure 1 – Synthesis Process

A different methodology, shown in Figure 2, would be to capture the memory architecture description itself and allows the synthesis tool to synthesise the memory interface from the architectural description. This change in design flow allows for a greater level of automatic optimisation as the memory architecture is no longer hidden from the synthesis tool.

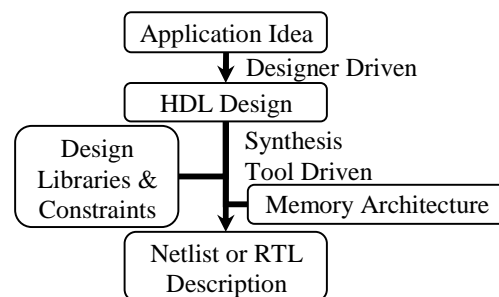


Figure 2 – Tool Driven Memory Synthesis

The first step to making synthesis tools independent of architecture is a memory specification language. There are already hardware description languages or graphical descriptions available that are used for simulations that include memories. However, these are not specifically designed for the purpose. Consequently, extraneous information will be present, or certain information useful to a synthesis tool may not be present. Furthermore, the descriptions are often not extendable or easily generated and parsed by synthesis tools. These problems render available languages or descriptions impractical for the purpose.

The specification language would mainly be used by board designers to describe their systems, with specifications packaged as synthesis tools libraries. It would also work for research boards, allowing a design to be easily mapped to custom memory architecture. An interface design would not be required, only a description of the custom memory architecture.

The main advantages of such a change in the design flow are:

- *Capturing information in one place* – The specification would capture both connectivity information as well as information about the specific memory banks in a standard format.
- *Board independence* – Designs that have their memory interfacing created by hand often require a time consuming complete redesign if the board is changed. By allowing automatic synthesis tools to handle the interfacing, designs can be made independently of architecture.
- *Higher level abstraction* – When creation of the memory interfacing is no longer the designer's

concern it will allow the designer to operate at a higher level of abstraction. Languages like C or Java would be much easier to synthesize from when architectural details do not have to be described.

- *Better than a generic memory interface* – Having architectural information available to the synthesis tool, allows for better optimizations than a fixed memory interface. This is because a fixed memory interface will not always be suited to the application or the architecture.

The main disadvantages of such a change in the design flow are:

- *Loss of control over details* – Although automatic synthesis is easier for the designer, it means that the designer has less control over the details. This often has a negative effect on performance as usually the designer can make design decisions better than an automatic synthesis tool.
- *Difficult to be fully independent* – It is difficult to make the memory access of the design fully independent of the platform. It is likely that some information about how the memory interface should be created is required and some of the more unique memory architectures may require designer intervention.

Of course, it will also be necessary to create or modify a synthesis tool which makes design decisions based on the memory architecture and the application. As described below, many of these techniques exist but have often only been applied to specific architectures or systems.

### 3. Design Requirements

To determine the requirements of a memory specification language, current memory optimisation techniques were studied to determine the information that is utilised by synthesis tools. Memory architectures were also assessed to determine what kinds of memory systems would have to be handled to achieve flexibility between common reconfigurable platforms.

#### 3.1. Input Required for Memory Optimisation

There has been significant work in automatic partitioning of memory and memory control in the area of reconfigurable computing. Various papers give techniques in automatic optimisation that take into account the attributes of various memory banks.

Schmidt and Thomas [6] have looked at partitioning of arrays to various memories considering dimensions, access times, and ports of memory banks. This

information is used to balance design decisions between the performance and connectivity and memory resource costs.

Ouais and Vemuri [7, 8] have described partitioning for a hierarchical memory structure with on-chip and external memory. They discuss the ILP (integer-linear programming) approach to memory mapping.

Weinhardt and Luk [9] have considered caching techniques, and parallelising data accesses through scheduling across multiple banks. They also mention ILP for memory allocation to help achieve these goals.

DRAM burst and page access modes have been studied by Panda [10, 11], leading to partitioning and scheduling techniques that are more efficient in DRAM usage.

Our work is targeted at capturing the information useful for memory interfacing optimisations that are completed during synthesis time. These optimisations are performed on the partitioning, the datapath, and the memory controller.

Most optimisations for memory ultimately come down to more efficient partitioning of arrays to memory banks, better bandwidth utilisation, and data reuse through hierarchical memory [12]. The information required to implement these optimisations is either to do with attributes of the memory banks or the connectivity information.

Memory bank attributes such as latencies, size of the bank, data width and address width are commonly used in optimisation techniques. This is mostly required for decision making in partitioning, although scheduling optimisations also make use of latency information and are integrated with the partitioning optimisations.

Connectivity information is useful for techniques such as data reuse and caching where the memory hierarchy is required for implementation and the way it is connected affects the hierarchy. Connectivity also comes into play with bandwidth estimation for optimisation techniques such as scheduling, interleaving and bandwidth balancing, which are techniques targeted at improving bandwidth utilisation. While those techniques can be implemented according to a raw data rate for each bank, this can be misleading if, for example, certain address or data buses to the memory banks are shared or multiplexed.

#### 3.2. Describing Memory Architectures

An memory specification language based on XML [13] was drafted. This was based on the important attributes that memory optimisation techniques focus on and the common internal and external memory configurations found on FPGA boards.

To allow for flexibility between different reconfigurable computing platforms the common architectural components were identified so they could be described. These components were divided into three main categories, memory, control signal or data transformations, and interconnection components, then numerous subcategories. For the memory specification the components were encapsulated in a highly hierarchical format. Space precludes a full description of the language, but Figure 3 shows a snippet of the developed memory specification.

```

<memory type="DDRSDRAM"
  template="Micron MT46V16M16">
  <include>Micron MT46V16M16_def.lib</include>
  <!--additional memory bank information can be added here-->
</memory>
<!--declaring the DDRSDRAM banks-->
<memory type="Micron MT46V16M16"
  label="DDRSDRAM_B">
</memory>

<!--template of the SRAM used-->
<memory type="SRAM"
  template="Micron MT55V512V36PF-10">
  <include>Micron MT55V512V36PF-10_def.lib</include>
</memory>
<memory type="Micron MT55V512V36PF-10"
  label="SRAM_A">
</memory>
</components>

<connectivity>
<signal>sigSDRAM_databus_bank1[15:0]</signal>
<signal>sigSDRAM_databus_bank2[15:0]</signal>
<connect>sigSDRAM_A[addressbus[15:0]]</connect>

```

Figure 3 - Snippet of a Memory Specification

## 4. Summary and Future Work

This paper has proposed a change in the design flow for automatic synthesis tools in the area of reconfigurable computing. The proposed change means that the details of the memory architecture are not required from the designer, while the details of the interface architecture are kept separate from the synthesis tool. This also helps provide board independence and presents a higher level of abstraction to a designer.

Some of the difficulties faced were modifications required to existing synthesis tools, the loss of control and hence performance that comes with automatic synthesis, and the difficulty in making the synthesis tool fully independent of designer intervention.

The solution was to use a specification language so that synthesis tools can understand the memory structure and can cope with different platforms thus allowing applications to be easily retargeted. This was different to other attempts at flexibility which have looked at memory bank attributes without looking at the memory system holistically, or have attempted to use wrapping interfaces to hide the differences in memory structure from the synthesis tool.

A specification language has been created based on XML to cater for the common memory structures found on FPGA boards and researched memory optimisations.

Future work includes construction of synthesis tool techniques that can actively use the specification language to drive design decisions, and further refinement of the specification language.

## 5. References

- [1] 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", in *ACM Trans. Des. Autom. Electron. Syst.*, vol. 6, pp. 149-206, 2001.
- [2] J. Park and P. C. Diniz, "Synthesis of pipelined memory access controllers for streamed data applications on FPGA-based computing engines", in *Proceedings 14th International Symposium on System Synthesis*, 2001, pp. 221-226.
- [3] M. S. Gokhale, J.; Arnold, J.; Kalinowski, M., "Stream-oriented FPGA computing in the Streams-C high level language", in *Proceedings of the Eighth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2000, pp. 49-56.
- [4] P. Diniz and J. Park, "Automatic synthesis of data storage and control structures for FPGA-based computing engines", in *Proceedings of the Eighth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2000, pp. 91-100.
- [5] M. B. Gokhale and J. M. Stone, "Automatic allocation of arrays to memories in FPGA processors with multiple memory banks", in *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 1999, pp. 63-69.
- [6] H. Schmit and D. E. Thomas, "Synthesis of application-specific memory designs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, pp. 101-111, 1997.
- [7] I. V. Ouais, R., "Hierarchical memory mapping during synthesis in FPGA-based reconfigurable computers", in *Proceedings of Design, automation and test in Europe*, 2001, pp. 650-657.
- [8] I. Ouais and R. Vemuri, "Global memory mapping for FPGA-based reconfigurable systems", in *Proceedings of the 15th Parallel and Distributed Processing Symposium*, 2001, pp. 1473-1480.
- [9] M. Weinhaut and W. Luk, "Memory access optimisation for reconfigurable systems", *IEE Proceedings of Computers and Digital Techniques*, vol. 148, pp. 2001, 105-112.
- [10] P. D. Ranjan Panda, N.D.; Nicolau, A., "Incorporating DRAM access modes into high-level synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 96-109, 1998.
- [11] P. R. D. Panda, N.D.; Nicolau, A., "Exploiting off-chip memory access modes in high-level synthesis", in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1997, pp. 333-340.
- [12] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology*. Dordrecht: Kluwer Academic Publishers, 1998.
- [13] D. Mercer, *XML : a beginner's guide*. New York: Osborne/McGraw-Hill, 2001.