POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Acceleration by Separate-Process Cache for Memory-Intensive Algorithms on FPGA via High-Level Synthesis

Supervisor
Prof. Luciano Lavagno

Candidate
Giovanni Brignone
ID: 274148

Academic year 2020-2021

Abstract

Nowadays integrated circuit technology is approaching to its limits: transistors pitch is reaching the physical limit and power density is higher and higher. In this scenario it is more and more difficult to advance the general purpose processor based technology, therefore specialized components are taking place. These specialized components allow better performance and lower power consumption, but require much higher design costs, due to two main reasons: producing a dedicated IC takes a huge amount of time and money and hardware design is more complex than general purpose software design.

The first issue have been addressed by the introduction of FPGAs, which allow to implement a special-purpose hardware module on a general-purpose underlying architecture, without having to build an integrated circuit from ground-up.

The second issue have been addressed by the High Level Synthesis, which allows to translate general-purpose software into special-purpose hardware.

One of the most precious resources available on board an FPGA are Block-RAMs, which should be used with care in order to fully take advantage of them. Therefore this work is aimed at designing and implementing a cache module which takes care of transferring data between external DRAM and internal BRAM, letting the designer to focus on his design, without having to worry about the memory management.

Contents

Chapter 1

Background

1.1 Cache memory

Memory devices are usually the performance bottleneck in the execution of memory-bound algorithms. The ideal memory should be fast, large and cheap, but current technology forces the designer to choose a tradeoff between the metrics.

A common solution to this problem is to setup a memory hierarchy in which fast but small memories are paired with large but slow memories, which allows to get good performance on average while containing costs.

This hierarchy can be managed by two main approaches:

- *Scratchpad*: different memories belongs to different addressing spaces: the user is in charge of manually choosing what memory to access: this approach allows to optimally exploit the hierarchy at the cost of high design effort.
- *Cache*: different memories belongs to the same addressing space: the system automatically uses the whole hierarchy exploiting spatial locality (accessed data is likely physically close to previously accessed data) and temporal locality (accessed data has likely recently been accessed), which are typical of many algorithms.

Structure

A cache memory is logically split into *sets* containing *lines* which are in turn made up of *words*. Whenever a word w is requested there are two possibilities:

- *Hit*: w is present on the cache: the request can be immediately fulfilled.
- *Miss*: w is not present on the cache: it is necessary to retrieve it from lower level memory before fulfill the request.

During the data retrieving a cache line is filled with a block of contiguous words loaded from the lower level memory, trying to exploit spatial locality of future accesses, while

mapping policies and replacement policies determine which cache line to overwrite, trying to exploit temporal locality.

If the cache memory is writable, data consistency is ensured by a consistency policy.

Policies

Mapping policy

The mapping policy is in charge of statically associating a lower level memory line to a cache set.

The *set associative* policy is the most common mapping policy: given a cache memory with s sets of w words, the word address (referred to the lower level memory) bits are split into three parts:

1. $\log_2(w)$: offset of the word in the line.
2. $\log_2(s)$: set.
3. remaining MSBs: tag identifying the specific line.

Special cases of this policy are:

- *Direct mapped* policy: each set is composed of a single line: the set bits identify a specific cache line, therefore there is no need for a replacement policy.
- *Fully associative* policy: there is only a single set, therefore the line is fully determined by the replacement policy.

Replacement policy

The replacement policy is in charge of dynamically associating a lower level memory line to a cache line of a set.

Multiple solutions of this problem have been developed, trying to maximize the temporal locality exploitation. Among the most commonly used solutions there are:

- *First-In First-Out*: the line to be replaced is the first one that has been inserted to the cache.
- *Least recently used*: the line to be replaced is the one that has least recently been accessed.

Consistency policy

The consistency policy is in charge of ensuring data consistency between memories belonging to different hierarchy levels.

The most common solutions to this problem are:

- *Write-back*: write accesses are performed to the highest level memory and lower level memories are updated when the cache line is replaced only.
- *Write-through*: each write access is propagated along the whole hierarchy.

1.2 High-Level Synthesis

Multi-process modeling

Process modeling

Communication modeling

1.3 Previous work

Chapter 2

Thesis contribution

2.1 Design

2.2 Implementation

2.3 Results

Chapter 3

Conclusion