CHALMERS

Evaluation of synthesizable CPU cores

DANIEL MATTSSON
MARCUS CHRISTENSSON

Master's Thesis
Computer Science and Engineering Program
CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Engineering
Gothenburg 2004

Abstract

The three synthesizable processors: LEON2 from Gaisler Research, MicroBlaze from Xilinx, and OpenRISC 1200 from OpenCores are evaluated and discussed. Performance in terms of benchmark results and area resource usage is measured. Different aspects like usability and configurability are also reviewed.

Three configurations for each of the processors are defined and evaluated: the comparable configuration, the performance optimized configuration and the area optimized configuration. For each of the configurations three benchmarks are executed: the Dhrystone 2.1 benchmark, the Stanford benchmark suite and a typical control application run as a benchmark.

A detailed analysis of the three processors and their development tools is presented. The three benchmarks are described and motivated. Conclusions and results in terms of benchmark results, performance per clock cycle and performance per area unit are discussed and presented.

Sammanfattning

De tre syntetiserbara processorerna: LEON2 från Gaisler Research, MicroBlaze från Xilinx och OpenRISC 1200 från OpenCores utvärderas och diskuteras. Prestanda i form av resultat från benchmarkprogram och areautnyttjande mäts. Olika aspekter som användarvänlighet och konfigurerbarhet undersöks också.

Tre konfigurationer av varje processor definieras och utvärderas: den jämförbara konfigurationen, den prestandaoptimerade konfigurationen samt den areaoptimerade konfigurationen. För var och en av de tre konfigurationerna exekveras tre benchmarkprogram: Dhrystone 2.1, Stanford samt en typisk styrapplikation vilken körs som ett benchmarkprogram.

En detaljerad analys av de tre processorerna och deras utvecklingsverktyg framförs. De tre benchmarkprogrammen beskrivs och skälet till att de används motiveras. Slutsatser och resultat i form av resultat från benchmarkprogram, prestanda per klockcykel och prestanda per areaenhet diskuteras och framförs.

Conclusions

The purpose of the thesis work was to produce a report containing a qualitative and quantitative comparison between the three processors: LEON2, MicroBlaze and OpenRISC 1200.

LEON2 yields the best results for both the Dhrystone 2.1 benchmark and the Stanford benchmark, for all three configurations compared. MicroBlaze performs nearly as well as LEON2 for the fastest configuration, where LEON2 operates at significantly lower clock frequency.

LEON2 yields best performance per clock cycle for for all benchmarks and all configurations. The OpenRISC 1200 processor shows better performance per clock cycle than MicroBlaze in the Stanford benchmark and is therefore considered a more efficient architecture than the MicroBlaze architecture. MicroBlaze is significantly more efficient per area unit than the other two processors, but it is highly optimized for Xilinx FPGAs. The LEON2 shows better efficiency per area unit than the OpenRISC 1200 processor.

The MicroBlaze area usage is less than half the area usage of the other two processors. For the area optimized configuration LEON2 and OpenRISC 1200 utilizes approximately the same area.

The opinion of the authors regarding usability is that LEON2 is less difficult than the others to manage. MicroBlaze has the best documentation and the best support for adding user defined IP-blocks. The OpenRISC 1200 documentation is insufficient and the processor is in general more difficult to manage than both the MicroBlaze and LEON2 processors.

We are convinced that this report fulfills the scope and requirements of this Master's thesis. All three processors have been implemented on the target FPGA circuit. For all three processors, three configurations have been defined and evaluated. Two benchmarks have been executed on the processors implemented in hardware on the FPGA development board. The results have been documented and further analyzed.

Acknowledgements

First of all we would like to thank our supervisor Jiri Gaisler at Gaisler Research for invaluable support and help with technical questions.

We would also like to thank the other employees at Gaisler Research and especially Edvin Catovic for help with technical problems and giving feedback on the final report.

Further we would like to thank our examiner Lars Bengtsson at the department of Computer Engineering at Chalmers for undertaking our Master's thesis.

Daniel Mattsson and Marcus Christensson

Gothenburg, 21st of December 2004

Table of Contents

1 Introduction

In this chapter the background and description of this thesis work is presented. The project goals are formulated and motivated.

1.1 Background

Gaisler Research develops and supports the LEON SPARC V8 processor, a synthesizable processor for embedded applications. In this context, it is of interest to make a comparative analysis with synthesizable processors from other providers.

1.2 Project description

The work will consist of comparing three different processors: LEON2 from Gaisler Research, MicroBlaze from Xilinx and OpenRISC 1200 from OpenCores. The work will consist of three parts: initial analysis, implementation and benchmarking.

During the initial analysis, the processor architectures shall be analyzed and compared. Characteristics such as pipeline depth, cache architecture, instruction set, configurable options, etc should be described and evaluated.

Each processor shall be synthesized and implemented on a Virtex-II FPGA breadboard. Characteristics such as gate count, maximum clock frequency, and performance shall be measured. It is foreseen that two implementations of each processor will be done: minimum area and maximum performance.

Performance of the implemented processors shall be measured with a set of standard benchmarks. For this, the installation of a cross-compiler toolchain for each of the processors will be necessary.

A final report shall be produced, containing the description of the work, the findings, and the conclusion.

1.3 Project goals

The goals with this Master's thesis is to compare the synthesizable processors LEON2 from Gaisler Research, MicroBlaze from Xilinx and OpenRISC 1200 from OpenCores.

1.3.1 Measures

The three processors will be compared on the following aspects:

- Implementation aspects
  - Processor clock frequency
  - Processor area
  - Instruction Set Architecture (ISA)
- Performance
  - Benchmark results
- Usability
  - Documentation
  - Tools
  - Hardware configuration
  - Portability
  - Adding user defined IP-blocks
- Configurability

1.3.2 Motivation

Clock frequency is not directly comparable between different instruction set architectures, since the amount of work accomplished by the processor during one clock cycle can differ a lot between different ISAs. Even so the clock frequency can be rather interesting because lower clock frequency utilizes less power.

Usability is of great importance since customers do not want to spend money on learning complex tools. The comparison of usability is somewhat difficult and should be considered as the personal opinions of the authors.

An adaptable system is desirable and therefore the ability to configure as much as possible is important.

Portability is the ability to port the hardware to different platforms (e.g. FPGAs, ASICs). A widely portable processor attracts more customers.

2 Analysis and methods

In this chapter the three synthesizable processors will be analyzed. Three configurations of each processor will be defined and further evaluated.

2.1 FPGA development board

The three processors were evaluated in hardware on the GR-PCI-XC2V development board from Pender Electronic Design GmbH [PEWEB]. The development board includes a Xilinx Virtex-II FPGA (XC2V3000fg676-4), which is clocked by an onboard oscillator operating at 40 MHz. The hardware designs were downloaded to the FPGA via the onboard JTAG interface.

The following onboard features were used in the evaluations:

- 1 Mbyte of SRAM for storing programs.
- UART interface for output.

2.2 Analysis of the processors

In this section a detailed review of the three processors to be evaluated is presented. Parameters and data for each processor will be presented as well as a brief overview.

2.2.1 LEON2

2.2.1.1 Overview

LEON2 is a 32-bit RISC SPARC V8 compliant architecture, and uses big endian byte ordering as specified in the SPARC V8 reference manual [SPA92]. An overview of the LEON2 processor architecture can be seen in Figure 1 below.

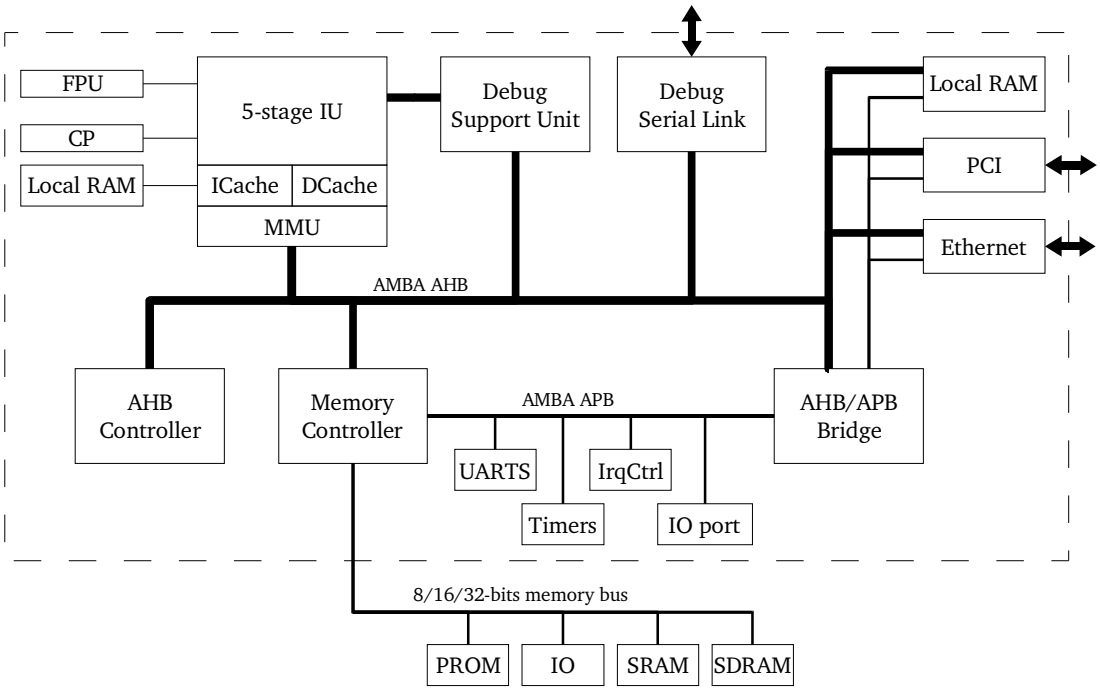

Figure 1: Overview of the LEON2 processor architecture.

LEON2 is a synthesizable processor developed by ESA and maintained by Gaisler Research. The processor was originally developed as a fault-tolerant processor for space applications. This report covers the non fault-tolerant version licensed under the GNU LGPL license, which is freely available as a VHDL model from the Gaisler Research website [GRWEB]. LEON2 targets both the ASIC and FPGA markets.

2.2.1.2 ISA

LEON2 utilizes the SPARC V8 instruction set architecture. The design goals of the SPARC V8 ISA were to make software optimization done by the compiler easy and to ease implementations of pipelined hardware.

The ISA has three different instruction formats and three addressing modes: immediate, displacement and indexed. Branch instructions have one annulled delay slot, see [SPA92].

The ISA includes instructions for multiply and accumulate (MAC) operations, multiplication and division. LEON2 implements optionally the MAC as 16x16-bit input with a 40-bit accumulator. The multiply operation can be implemented as six different hardware implementations as seen in Table 1.

Configuration	*Latency (clock)*	*Approx. area (Kgates)*
iterative	35	1000
M16x16 + pipeline reg	5	6500
m16x16	4	6000
m32x8	4	5000
m32x16	2	9000
m32x32	1	15000

Table 1: Multiplier configurations.

The divide instruction can be implemented in hardware as a radix-2 divider which results in a latency of 35 clock cycles. [GR04]

The ISA includes support for two coprocessors, one FPU and one custom user defined coprocessor. The FPU can be chosen from one of the following: GRFPU provided by Gaisler Research, Meiko FPU from Sun Microsystems and the incomplete open source LTH FPU developed at Lunds Tekniska Högskola in Sweden.

2.2.1.3 Integer unit

The integer unit implements the SPARC V8 ISA as a single issue 5-stage pipeline. Multipliers and dividers are configurable as described in 2.2.1.2 ISA.

The LEON2 register file is windowed with 2-32 register windows. Each register window has eight in registers, eight out registers and eight local registers. The in and out registers are shared between adjacent windows. In addition to the mentioned registers there are eight global registers. This means that the programmer at every instant can access 32 registers. The total number of registers can be calculated as 8+16*NWINDOWS, where NWINDOWS is the number of windows. This gives a total of 40-520 registers depending on the configured amount of register windows.

2.2.1.4 Cache system

The cache system is a Harvard architecture with 1-64 Kbyte per way for both the instruction and the data cache. Each cache line can contain between 4 and 8 sub-blocks, each containing 4 byte.

The caches can be configured as either one way direct-mapped or 2-4 way set associative. In a multi-way configuration three replacement policies can be used: least recently used (LRU), last recently replaced (LRR) and pseudo-random. When using LRR, only two way set associativity is available.

For reduced cache miss latency the instruction cache uses streaming during line-refill, which means that data is sent to the processor at the same time as it is written to the cache. On a cache miss in the data cache, only the requested sub-block is fetched.

The data cache uses write-through policy. To minimize pipeline stalls caused by store instructions, a double-word write buffer is used.

The caches have support for individual cache line locking in multi-way configurations. To prevent a specific memory address to be blocked from the cache, the last line in each set can not be locked.

In order to sustain cache consistency when there are several units on the AHB (AMBA High Speed Bus) bus (section 2.2.1.6 System interface) capable of writing to the memory, the data cache can perform bus snooping on the AHB bus. Bus snooping is only available when the MMU is disabled, since the cache is virtually addressed.

2.2.1.5 MMU

A memory management unit can be enabled providing support for memory protection mechanisms required for advanced operating systems. The MMU can be configured to either use a shared or split translation lookaside buffer (TLB) for instruction and data memories. The TLB is fully associative and the number of entries can be configured between 2 and 32. The MMU supports page sizes of 4 Kbyte, 256 Kbyte and 16 Mbyte.

2.2.1.6 System interface

The integer unit interfaces to memory and other peripherals via the AMBA Advanced High performance Bus (AMBA-2.0 AHB) and the AMBA Advanced Peripheral Bus (AMBA-2.0 APB).

The AMBA-2.0 AHB bus connects to high-speed peripherals, DMA controllers, on-chip memory and interfaces. The bus has pipelined operation and supports burst transfers, multiple masters and split transactions. The AMBA-2.0 APB bus is optimized for minimal power consumption and uses an interface with reduced complexity to support peripheral functions. The protocol is designed for ancillary or general-purpose peripherals.

2.2.1.7 Power management

A power-down mode is supported from which the processor wakes up when an unmasked interrupt with a specified priority becomes pending. In power-down mode the integer unit is halted.

2.2.1.8 Memory controller

The memory controller supports several memory types including:

- PROM
- SRAM
- SDRAM (up to two banks of PC100/PC133)

2.2.1.9 Additional units

In addition to the above mentioned units, several other units can be connected:

- Debug Support Unit (DSU), available for easier debugging.
- PCI interface.
- Ethernet MAC.
- On-chip RAM for fast memory accesses.
- GRFPU and MEIKO FPU (both fully IEEE-754 compliant) and the incomplete LTH FPU.

2.2.2 MicroBlaze

2.2.2.1 Overview

MicroBlaze is a 32-bit RISC synthesizable processor which uses big endian byte ordering. An overview of the MicroBlaze processor architecture can be seen in Figure 2. MicroBlaze is developed and maintained by Xilinx Inc. The processor is designed specifically for Xilinx FPGAs and therefore highly optimized for their FPGA circuits. [XIL04a]

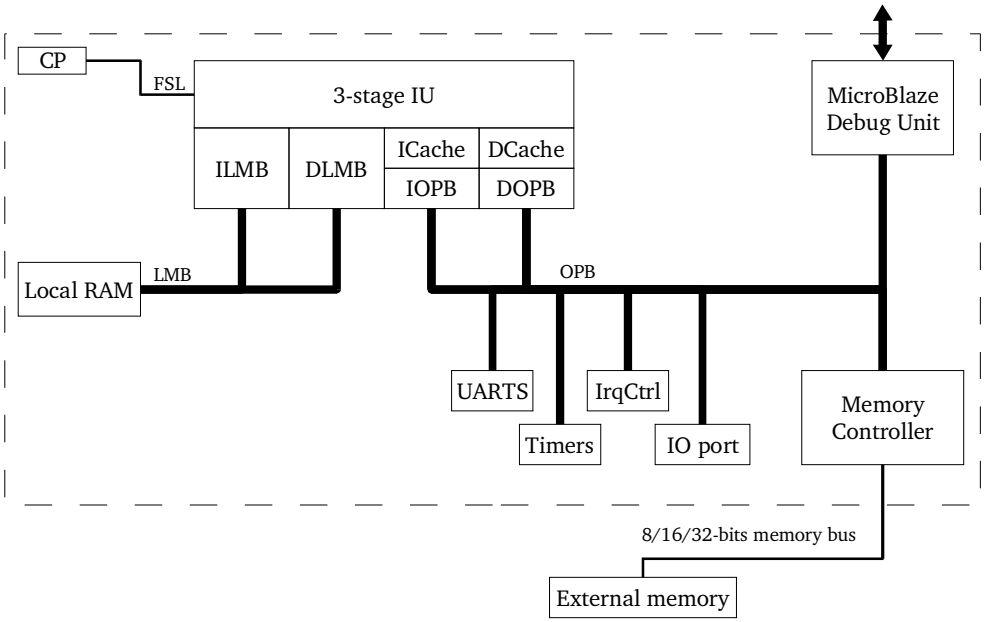

Figure 2: Overview of the MicroBlaze processor architecture.

MicroBlaze is distributed with the Xilinx Embedded Development Kit (EDK) as a parameterizable netlist, but the VHDL source code can be obtained from Xilinx at a higher cost. [XILWEB] The version of MicroBlaze evaluated here is version 2.10.a supplied with Xilinx EDK version 6.2.

2.2.2.2 ISA

MicroBlaze has its own ISA specially designed for MicroBlaze. The ISA has two different instruction formats and two addressing modes, immediate and displacement.

The ISA includes instructions for multiplication and division, which are optionally implemented in hardware. The multiplication, which has a latency of 3 clock cycles when implemented in hardware, can only be implemented if the FPGA has built-in hard multipliers. The division has, if implemented in hardware, a latency of 34 clock cycles. [XIL04a]

The ISA includes instructions for blocking and non-blocking reads and writes to the FSL (Fast Simplex Link) bus for fast communication with a custom unit. Branch instructions have one delay slot which is always executed.

2.2.2.3 Integer unit

The MicroBlaze integer unit is a single issue 3-stage pipeline implementing the MicroBlaze ISA. Hardware multiplier and divider are available if configured and supported by target FPGA architecture as described in 2.2.2.2 ISA. The register file has 32 general purpose registers, each 32-bit wide. MicroBlaze uses a simple technique called a branch history buffer (BHB) to reduce branch miss rate.

2.2.2.4 Cache system

The cache architecture is a Harvard architecture where both instruction and data caches can vary in size between 2 and 64 Kbyte. A single cache line contains 4 byte, resulting in not exploiting the spatial locality of a program.

The caches are direct-mapped and supports individual cache line locking. Since the MicroBlaze cache is direct-mapped, locking one line can render other memory addresses blocked from the cache, possibly affecting performance negatively. The data cache operates as a write-through cache and implements allocate-on-write. The MicroBlaze cache is limited to only allow caching of a continuous subspace of the total memory.

2.2.2.5 MMU

No memory management unit available.

2.2.2.6 System interface

The system interface for MicroBlaze consists of the Local Memory Bus (LMB), the IBM CoreConnect On-chip Peripheral Bus v2.0 (OPB) and the Fast Simplex Link bus (FSL).

The integer unit interfaces to the internal block RAM with a single-cycle access through the LMB. The OPB, which is a multi-master multi-slave bus,

provides a handshake interface to both on- and off-chip peripherals and memory. The FSL is available at the ISA level, capable of point-to-point transfers of data with a latency of two clock cycles.

2.2.2.7 Power management

No power management available.

2.2.2.8 Memory controller

MicroBlaze has support for a number of memory types, including:

- Intel StrataFlash
- SRAM
- SDRAM
- DDR SDRAM

2.2.2.9 Additional units

MicroBlaze can interface to an extensive amount of IP-cores including:

- Xilinx Microprocessor Debug Module (MDM) for debugging.
- Ethernet MAC.
- Quixilica IEEE-754 single precision FPU.
- UART

2.2.3 OpenRISC 1200

2.2.3.1 Overview

OpenRISC 1200 is a synthesizable processor developed and managed by a team of developers at OpenCores [OCWEB]. OpenRISC 1200 is a 32-bit RISC processor implementing the 32-bit OpenRISC 1000 architecture. An overview of the OpenRISC 1200 processor architecture can be seen in Figure 3 below. The processor uses big endian byte ordering. The processor is intended for embedded, portable and network applications. OpenRISC 1200 is an open source IP-core freely available from the OpenCores website as a Verilog model, licensed under the GNU LGPL license.

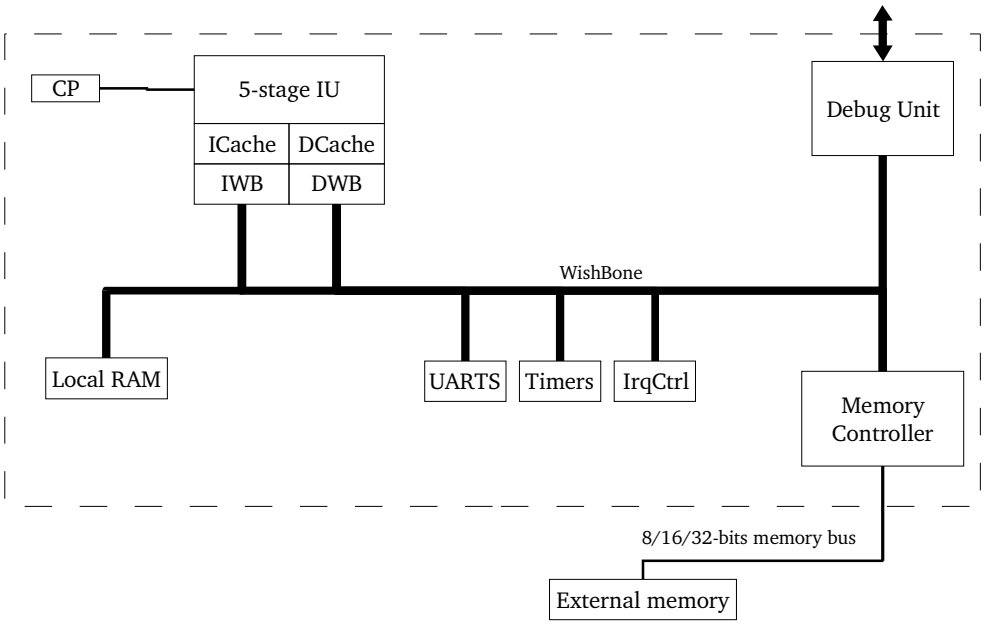

Figure 3: Overview of the OpenRISC 1200 processor architecture.

2.2.3.2 ISA

OpenRISC 1200 implements the ORBIS32 instruction set architecture. ORBIS32 has five instruction formats and three addressing modes: immediate, displacement and pc-relative.

The ISA includes instructions for multiplication and division, which are optionally implemented in hardware. The hardware multiplication has a latency of 3 clock cycles. The division has, if implemented in hardware, an estimated latency of 64 clock cycles[1]. [ORSRC]

A MAC instruction with two 32-bit operands and a 48-bit accumulator is included in the ISA. The ISA can be extended with custom instructions and an additional coprocessor can be attached. Branch instructions have one delay slot which always is executed.

2.2.3.3 Integer unit

The OpenRISC 1200 integer unit is a single issue 5-stage pipeline implementing the ORBIS32 ISA. Hardware multiplier and divider is available if configured as described in 2.2.3.2 ISA. The register file has 32 general purpose registers, each 32-bit wide.

1 The value of 64 is an estimation obtained from source code studies, since the latency is undocumented.

2.2.3.4 Cache system

The cache architecture is a Harvard architecture where the instruction cache can vary in size between 512 byte and 8 Kbyte and the data cache size can vary between 1 and 8 Kbyte. A single cache line contains either 8 or 16 byte data. [ORSRC]

Both caches are direct-mapped. The data cache operates in a write-through mode and optionally implements allocate-on-write. Cache locking is implemented as locking the cache on a one way basis. Critical-word-first is implemented as cache fetch technique in both caches.

2.2.3.5 MMU

A memory management unit is implemented and can be enabled providing support for memory protection mechanisms required for advanced operating systems. The MMU uses a split TLB for instruction and data memories. The TLBs are direct-mapped and has 64 entries each, with a fixed page size of 8 Kbyte.

2.2.3.6 System interface

OpenRISC 1200 interfaces to memory and peripherals via two Wishbone[2] compliant 32-bit bus interfaces. The Wishbone interface supports point-to-point, shared bus, crossbar switch and data flow interconnections. The multi-master multi-slave bus also supports both single cycle data transfers and burst transfers.

2.2.3.7 Power management

OpenRISC 1200 implements three different power modes. The slow and idle mode is implemented in software and reduces power usage by adjusting the clock frequency. The doze mode disables all operations in the integer unit. The clocks to the internal modules of the processor are disabled, except for the timer unit. In the sleep mode all internal units are disables and all clocks are gated. The sleep and doze modes will be leaved when a pending interrupt occurs and normal operation mode is entered.

2.2.3.8 Memory controller

The OpenRISC 1200 processor supports a number of memories including:

- SDRAM
- SSRAM
- FLASH
- SRAM

2 Wishbone SoC Interconnection specification Rev. B

2.2.3.9 Additional units

A lot of third party IP-cores are available from OpenCores. Some examples are listed below:

- FFT cores
- Ethernet MAC
- I2C controller core
- Cryptographic cores

2.2.4 Summary

All three processors presented in 2.2 Analysis of the processors are 32-bit RISC big endian synthesizable processors with single issue pipelines. LEON2 and OpenRISC 1200 has 5-stage pipelines in contrary to MicroBlaze, which has a 3-stage pipeline.

LEON2 and OpenRISC 1200 are available for free under the LGPL license, but MicroBlaze on the other hand is commercially available from Xilinx for use in their FPGA circuits.

LEON2 has a large windowed register file, while the others use a flat register file with less registers. The windowed register file results in a worst case execution time which is more difficult to analyze compared to the case when a flat register file is used. The use of a flat register file can not result in a window overflow in contrary to the windowed register file in LEON2, where window overflows may result in large overheads.

All three processors have Harvard caches with similar values for the effective cache size. LEON2 extinguishes itself from the others by, besides direct-mapped support, also providing support for a 2-4 way set associative cache configuration, in which three replacement strategies can be used. MicroBlaze only allows caching of a continuous address range, while the others can cache multiple address ranges.

MicroBlaze is optimized for Xilinx FPGAs and offers an efficient use of the FPGA resources for improved performance and low area resource usage. The key parameters for the three processors are summarized in Table 2 below.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
License	GNU LGPL	Ships with Xilinx EDK	GNU LGPL
Platform	FPGA, ASIC	Xilinx FPGA	FPGA, ASIC
Distributed file format	VHDL	EDIF[3]	Verilog
General			
Architecture	32-bit RISC	32-bit RISC	32-bit RISC
Byte ordering	Big endian	Big endian	Big endian
Pipeline depth	5	3	5
Issue type	Single	Single	Single
Branch prediction	N/A[4]	BHB[5]	N/A
Register file			
Organization	Windowed	Flat	Flat
# of global registers	8	32	32
# of windows	2-32	N/A	N/A
# of registers/window	16	N/A	N/A
Total # of GPRs[6]	40-520	32	32
ISA			
Type	SPARC V8	MicroBlaze ISA	ORBIS32[7]
Addressing modes	Immediate, displacement, indexed	Immediate, displacement	Immediate, displacement, pc-relative
MAC	16x16-bit, 40-bit Acc	N/A	32x32-bit, 48-bit Acc
MUL latency	1-35 cycles[8]	3 cycles[9]	3 cycles
DIV latency	35 cycles	34 cycles[10]	~64 cycles[11]
Branch delay slots	1[12]	1	1
Branch latency	0-1	1-3	Unknown
Load delay	1, 2[13]	2	Unknown[14]
Custom instructions	No	No	Yes
Custom coprocessor	Yes	Yes (via FSL)	Yes
Hardware floating-point support	GRFPU, Meiko FPU, LTH FPU[15]	Quixilica FPU[16]	N/A[17]

3 The VHDL source code is commercially available.
4 N/A indicates that the feature is not available.
5 Branch History Buffer.
6 Number of General Purpose Registers
7 32-bit OpenRISC Basic Instruction Set.
8 Six different hardware multipliers available with different latencies and area requirements.
9 When the target FPGA has built-in hardware multipliers.
10 When the optional hardware divider is implemented.
11 Estimated by looking at the OpenRISC 1200 Verilog source code. 64 cycles is only an approximate value.
12 The delay slot can be “annulled” for some branch instructions.
13 Only used for ASIC.
14 The OpenRISC 1200 documentation does not include information regarding the load delay.
15 GRFPU is available from Gaisler Research, Meiko FRU is available from SUN, and LTH FPU is an

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Software floating-point support	IEEE-754 Single and double precision	Single precision[18]	IEEE-754 Single and double precision
Cache			
Hierarchy	Harvard	Harvard	Harvard
Instruction cache size	1-256 Kbyte	2-64 Kbyte	512 byte-8 Kbyte
Data cache size	1-256 Kbyte	2-64 Kbyte	4-8 Kbyte
Line size	16-32 byte	4 byte	8-16 byte
Sub-block size	4 byte	N/A	N/A
Placement scheme	Direct-mapped, 2-4 way set associative	Direct-mapped	Direct-mapped
Replacement strategies	LRU, LRR[19], Random	N/A	N/A[20]
D$ write policy	Write-through	Write-through	Write-through
Store Buffer	1 double-words	N/A	0,4,8 words[21]
I$ refill policy	Streaming during line-refill	Line refill	Critical-word-first line refill
D$ read refill policy	Only requested sub-block	Line refill	Critical-word-first line refill
Allocate-on-write	No	Yes	Optional
Read hit wait states	0	0	0
Write hit wait states	0	0	0
Valid bits	One per sub-block	One per cache line	One per cache line
Line-locking	Individual	Individual	Set basis
Bus snooping	Data cache[22]	N/A	N/A
System Interface	AMBA-2.0 AHB, AMBA-2.0 APB	LMB, IBM OPB v2.0, FSL	Wishbone SoC rev. B 32-bit
Power management	Power-down mode	N/A	Slow and idle mode, sleep mode, doze mode
Memory			
On-chip RAM	Configurable	Configurable	Configurable
External Memory Controller	PROM, SRAM, SDRAM, memory-mapped I/O devices	DDR SDRAM, SDRAM, SRAM, external FLASH	SDRAM, SRAM, SSRAM, FLASH
MMU			

incomplete open source FPU. Both the GRFPU and the Meiko FPU implements full IEEE-754 floating-point support.

16 Quixilica FPU is a single precision IEEE-754 third-party IP core commercially available.

17 There exists an IEEE-754 single-precision compliant FPU which works in simulation, but the implementation is incomplete for synthesis.

18 Double precision has not been confirmed. Single precision does not follow the IEEE-754 standard.

19 LRR can only be used in a 2-way set associative cache configuration.

20 OpenCores claim to use LRU, but LRU seems unnecessary in a direct-mapped cache.

21 The OpenRISC 1200 source code reports that only a store buffer of size 4 or 8 words has been tested properly, a test run with Stanford confirmed that a store buffer length of 16 words did not work.

22 Not available when the MMU is enabled.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Shared or split TLB	Configurable	N/A	Split
# of TLB entries	2-32	N/A	64
TLB placement scheme	Fully associative	N/A	Direct-mapped
Page size	4 Kbyte, 256 Kbyte, 16 Mbyte	N/A	8 Kbyte
Operating system support	eCos, Thumbpod (Java), SnapGear Embedded Linux (Linux and uClinux), OAR RTEMS RTOS, VxWorks	AT Nucleus Plus RTOS, Express Logic ThreadX, Micriμm μC/OS-II RTOS, uClinux, VxWorks	Linux, uClinux, OAR RTEMS RTOS

Table 2: Summary of the synthesizable processors' parameters.

2.3 Development tools

This section will present development tools used for software and hardware development. The purpose of this section is to cover the available tools for each one of the three processors.

2.3.1 LEON2

A number of make scripts are used to configure and implement the LEON2 processor. There is a graphical TCL/Tk based configuration tool which works similar to the TCL/Tk graphical configuration tool for the Linux kernel.

Simulation is done through TSIM which is a commercially available cycle accurate instruction set simulator, capable of simulating LEON-based computer systems. An evaluation version is also available. The simulator can be executed standalone or connect remotely to the GNU debugger, GDB.

Synthesis is available through both Xilinx XST and Synplicity Synplify. Hardware simulation can be done with Mentor Graphics Modelsim, Cadence NCsim and the GNU HDL simulator (GHDL).

There are two cross-compiler toolchains for the LEON2 processor, one for bare-C applications and one for RTEMS applications. Both of them use the GNU compiler toolchain and the GNU debugger. There is a graphical IDE for C/C++ development available as a plugin for Eclipse 3.0.

There is a graphical user interface for GRMON, which acts as a front-end for LEON2 debugging. This user interface is also provided as an Eclipse 3.0 plugin.

In order to debug a LEON2 microprocessor system, GRMON can be used. GRMON can connect to a LEON2 system implemented on a physical FPGA

development board via a hardware debug unit, or execute the instruction set simulator TSIM. To debug the software the GNU debugger (GDB) can connect to GRMON.

2.3.2 MicroBlaze

MicroBlaze hardware and software development is done using Xilinx Embedded Development Kit (EDK). EDK is a development environment where the hardware is instantiated as different IP-blocks connected via buses and signals. The software is developed on top of the generated libraries derivated from the hardware design. EDK focuses on system development closely integrated with a microprocessor. Both Xilinx's soft processor MicroBlaze and IBM's PowerPC 405, available as a hard macro in some FPGA circuits, are supported. EDK includes an integrated development environment (IDE) named Xilinx Platform Studio (XPS), which is a graphical GUI on top of the EDK.

The design is mainly specified in the Microprocessor Hardware Specification file (MHS) and the Microprocessor Software Specification file (MSS). The MHS file instantiates the different hardware IP-blocks and connects them together. A make script is used to synthesize and route the hardware, compile the software libraries and applications, generate simulation models and bitstreams etc.

XPS acts as a graphical front-end for the make scripts when compiling the software and implementing the hardware. The compiler used in the compilation is a modification of the GNU Compiler Collection tools (GCC). For synthesis the Xilinx XST is used.

In order to debug software Xilinx Microprocessor Debugger (XMD) is included. XMD can connect to a MicroBlaze or PowerPC processor implemented on a physical FPGA development board or execute an instruction set simulator. To debug the software the GNU debugger (GDB) can connect to XMD.

The Xilinx Embedded Development Kit v6.2 is available for the Windows, Solaris and Linux platforms.

2.3.3 OpenRISC 1200

The hardware configuration is done by manually editing the Verilog files. Most of the configuration options are changed from within one file, containing numerous define statements.

For the OpenRISC 1200 processor there is a GNU toolchain including the GCC compiler and GNU debugger (GDB). An instruction set simulator called or1ksim is also available.

To connect to the debug unit implemented on the target FPGA, a tool called JP1 can be compiled. JP1 connects via a dedicated JTAG port on the FPGA circuit and serves as a server for GDB. GDB can connect to JP1 providing access to the FPGA board. From within GDB the user can download software programs into the processor memory, modify registers, execute programs etc.

2.4 Processor configurations

This section aims to describe the differences between the three processor configurations, the comparable, the performance optimized and the area optimized configuration. Parameters and configuration options of interest for the different configurations are presented.

2.4.1 Parameters common for all configurations

For a better overview of the differences between the three configurations, parameters and configuration options specific for each configuration are listed in separate tables. Parameters and configuration options that remains constant for all configurations are listed in Table 3 below. Note that some of the parameters in Table 3 are not configurable and are there merely for greater understanding.

	LEON2	***MicroBlaze***	***OpenRISC 1200***
Branch prediction	Not needed[23]	BHB	N/A
Register file			
Organization	Windowed	Flat	Flat
# of global regs	8	32	32
# of windows	8	N/A	N/A
# of registers/window	16	N/A	N/A
Total # of GPRs	136	32	32
Reg-file RAM bits	4352	1024	1024
ISA			
MAC	No	N/A	Yes[24]
Load delay	1	2	Unknown
Branch latency	0-1	1-3	Unknown
Custom instructions	No	N/A	No
Custom coprocessor	No	No	No
Floating-point support	SW	SW	SW
Cache			
Hierarchy	Harvard	Harvard	Harvard
Line size	32 byte	4 byte	16 byte
Sub-block size	4 byte	N/A	N/A

23 No branch prediction needed since branch is always resolved during the delay slot.

24 The compiler does not use the MAC instructions however the area will be affected negatively. Unfortunately the system did not work properly with the MAC unit turned off.

	LEON2	*MicroBlaze*		*OpenRISC 1200*
D$ write policy	Write-through	Write-through		Write-through
Store buffer	1 double-word	N/A		4 words
I$ refill policy	Streaming during line-refill	Line refill		Critical-word-first line refill
D$ read refill policy	Only requested sub-block	Line refill		Critical-word-first line refill
Allocate-on-write	No	Yes		No
Bus snooping	No	N/A		N/A
Line-locking enabled	No	Yes[25]		N/A
Valid bits/line	8	1		1
Lock bits/line	0	1		0
On-chip RAM	No	8 Kbyte		8 Kbyte
Memory Controller[26]	SRAM	SRAM1	SRAM2[27]	SRAM
Read latency	2	1	3	1
Read cycle time	3[28]	3	5	3
Word write latency	2	1	2	1
Word write cycle time	3	3	4	3
Byte write latency	5	4	5	4
Byte write cycle time	6	6	7	6
External memory	1 Mbyte SRAM	1 Mbyte SRAM		1 Mbyte SRAM
MMU	No	N/A		No

Table 3: The basic parameters which are common between all three configurations.

The software for the LEON2 processor is written for the use of eight register windows. Therefore the LEON2 processor is kept at the standard configuration of eight register windows.

The OpenRISC 1200 and the MicroBlaze processors use on-chip RAM to execute a bootloader, in which the caches are initiated and enabled. After the caches are enabled a jump is performed to the start of the program in external memory. This initialization is not needed in the LEON2 processor, since the DSU performs all the initialization needed. The execution can therefore start directly in external memory.

The cycle time for the memory controller denotes the smallest number of clock cycles between the start of two consecutive memory accesses.The latencies of the memory controllers are defined as the time in clock cycles

25 Line-locking is enabled in hardware but not used in software.
26 All latencies and cycle times are measured in clock cycles.
27 In order to satisfy the timing constraints for the 80 MHz configuration, see 3.1.2 Performance optimized configuration, the memory controller had to be rewritten.
28 During a read burst transaction the lead-out cycle can be left out for all reads but the last.

from the clock cycle when the memory controller recognizes a bus memory access to the clock cycle when the memory controller acknowledges the transfer.

Since the FPGA development board used [PE03] does not have individual byte write signals to the external SRAM chips, a memory controller with read-modify-write support had to be used for byte writes. The OpenRISC 1200 memory controller and the MicroBlaze memory controller did not have such support. Therefore such a memory controller had to be written.

The timing constraints for the performance optimized configuration of the MicroBlaze processor required a timing optimization of the SRAM controller, denoted SRAM2 in Table 3.

2.4.2 Comparable configuration

The comparable configuration aims to be as fair as possible with respect to available resources. The three processors are configured to run with the same amount of cache memory and at the same clock frequency. Multiplication and division are configured to be handled in hardware with roughly the same latencies among the different processors. SRAM is used as external memory for all the processors, and the MicroBlaze and the OpenRISC 1200 uses the same memory controller, which is similar to the LEON2 memory controller, see Table 3 for more details. Floating-point arithmetics are configured to be performed in software for all three processors.

Table 4 below includes only parameters specific for the comparable configuration of all the processors, refer to Table 3 for options common for all configurations. Note that some of the parameters in Table 4 are not configurable and are there merely for greater understanding.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Clock frequency	30 MHz	30 MHz	30 MHz
ISA			
MUL	4 cycles[29]	3 cycles	3 cycles
DIV	35 cycles	34 cycles	~64 cycles
Cache			
Instruction cache size	8 Kbyte	8 Kbyte	8 Kbyte
Data cache size	8 Kbyte	8 Kbyte	8 Kbyte
Placement scheme	2-way LRU	Direct-mapped	Direct-mapped
Replacement bits/line	1	0	0
Tag bits/line	20	7	19
Address width	32	20	32
Total cache line size[30]	285	41	148
Number of cache lines	256	2048	512
Cache RAM bits[31]	72960	83968	75776
Total cache RAM bits	145920	167936	151552
Memory Controller	SRAM	SRAM1	SRAM

Table 4: Parameters for the comparable configurations.

The three processors have been written with different cache parameter settings in mind. The cache organizations are not configured to be similar since the cache are considered part of the processor. The cache size is kept constant, but the cache line size, the degree of associativity, and the replacement strategy differ between the processors.

Considering the vast amount of cache configuration possibilities, a better parameter to use as a comparable measure is the total number of RAM bits used for memory in each processor. For example, in spite of the large LEON2 register file, the processor still utilizes less RAM bits than the others because of the larger cache line size.

29 Multiplier m32x8 is used yielding a latency of 4 clock cycles.
30 Size in bits of a cache line, including data, tag, replacement bits, valid bits and lock bits.
31 Calculated using Formula i and Formula ii in appendix A Information on caches.

2.4.3 Performance optimized configuration

Table 5 below includes only parameters specific for the performance optimized configuration of all the processors, refer to Table 3 for options common for all configurations. Note that some of the parameters in Table 5 are not configurable and is there merely for greater understanding. The configuration options in Table 5 was achieved during an iterative process. Different configurations were synthesized and benchmarked, and the best was chosen as the configuration defined in Table 5.

	LEON2	***MicroBlaze***	***OpenRISC 1200***
Clock frequency	53.3 MHz	80 MHz	40 MHz
ISA			
MUL	4 cycles[32]	3 cycles	3 cycles
DIV	35 cycles	34 cycles	~64 cycles
Cache			
Instruction cache size	16 Kbyte	8 Kbyte	8 Kbyte
Data cache size	16 Kbyte	8 Kbyte	8 Kbyte
Placement scheme	4-way LRU	Direct-mapped	Direct-mapped
Replacement bits/line	5	0	0
Tag bits/line	20	7	19
Address width	32	20	32
Total cache line size[33]	289	41	148
Number of cache lines	512	2048	512
Cache RAM bits[34]	147968	83968	75776
Total cache RAM bits	295936	167936	151552
Memory Controller	SRAM	SRAM2	SRAM

Table 5: Parameters for the performance optimized configurations.

In order to get MicroBlaze running at 80 MHz, the memory controller had to be replaced by a version with better timing, since it was part of the critical path. Another reason was that the SRAM chip read latency required an extra wait state.

An eight word store buffer for the OpenRISC 1200 processor was evaluated, but did not yield any significant improvement in the benchmark results for the evaluated cache configurations. Therefore also the performance optimized configuration has a four word store buffer.

32 Multiplier m32x8 is used yielding a latency of 4 clock cycles.
33 Size in bits of a cache line, including data, tag, replacement bits, valid bits and lock bits.
34 Calculated using Formula i and Formula ii in appendix A Information on caches.

2.4.4 Area optimized configuration

Table 6 below only includes parameters specific for the area optimized configuration of all the processors, refer to Table 3 for options common for all configurations. Note that some of the parameters in Table 6 are not configurable and is there merely for greater understanding.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Clock frequency	26.7 MHz	26.7 MHz	26.7 MHz
ISA			
MUL	Software	3 cycles	3 cycles
DIV	Software	Software	Software
Cache			
Instruction cache size	4 Kbyte	4 Kbyte	4 Kbyte
Data cache size	4 Kbyte	4 Kbyte	4 Kbyte
Placement scheme	Direct-mapped	Direct-mapped	Direct-mapped
Replacement bits/line	0	0	0
Tag bits/line	20	8	20
Address width	32	20	32
Total line size	284	42	149
Number of lines	128	1024	256
Cache bits[35]	36352	43008	38144
Total cache RAM bits	72704	86016	76288
Memory Controller	SRAM	SRAM1	SRAM

Table 6: Parameters for the area optimized configurations.

For the area optimized configuration the same clock frequency was used for all processors to give the synthesis tool the same prerequisites. MicroBlaze and OpenRISC 1200 are configured to use hardware multiplier since the Virtex-II FPGA used includes dedicated hard multiplier resources. The LEON2 processor includes support for enabling or disabling hardware multiplier and divider individually, but since the compiler (GCC) does not provide individual settings for hardware multiplication and division, the options are not separated in the graphical configuration tool. There is no benefit of having individual enabling of hardware multiplication and division, because the generated code either has instructions for both or for none of them.

The caches in LEON2 are configured to use a one way (direct-mapped) cache architecture because it has a lower area overhead compared to a multi-way configuration. The lowest area usage would be obtained if the processors disabled all caches. However the area cost for using caches is low since the Virtex-II includes lots of hard RAM resources, and the amount of logic inferred is small compared to the gain in performance.

35 Calculated using Formula i and Formula ii in appendix A Information on caches.

2.5 Benchmarks

In this section the different benchmarks used are presented and further discussed. Pros and cons regarding benchmarking in general are discussed.

2.5.1 Pros and cons regarding benchmarking

When choosing a benchmark, the system's intended area of usage should be considered. If the system is intended for automotive applications, the benchmark should try to benchmark parameters important in such applications.

The ideal benchmark is measuring the performance of all the applications the system will ever run, but such a benchmark is difficult to construct. Most benchmarks include fragments from real applications, or algorithms comparable to algorithms in real applications, in an attempt to behave comparable to real applications.

It is of great importance to know the differences between the underlying hardware and software when comparing benchmark results from different processors.

2.5.2 Dhrystone 2.1

The Dhrystone benchmark was created back in 1984 by Dr. Reinhold P. Weicker. Today Dhrystone 2.1 is the current version, which was written in 1988. Weicker's intention with writing Dhrystone was to measure the performance of computer systems, and since the computer systems of that era were focused on integer performance, Dhrystone primarily targets integer performance. [AW04]

Dhrystone is a synthetic benchmark composed of a, of that time, “typical” application mix of mathematical and other operators. Dhrystone is written in the C language which makes it highly portable but there are some drawbacks:

- The size of the code is very small, not stressing the memory system of todays machines.
- The small size of the code makes it possible for compiler writers to write a compiler that recognizes the code and optimizes it.
- A large amount of the execution time is spent in basic library functions, rendering the benchmark, really measuring the performance of the library functions of different compilers.
- Compiler optimizations may render unrealistic results.

The Dhrystone benchmark basically consists of a main loop executed a number of times. The output of the benchmark is the time spent in the main loop.

2.5.3 Stanford

Stanford is a small benchmark suite gathered by John Hennessy and modified by Peter Nye. The suite contains the following programs:

- Perm Heavily recursive permutation program.
- Towers Program solving the Towers of Hanoi problem.
- Queens Program solving the Eight Queens problem 50 times.
- Intmm Program multiplying two integer matrices.
- Mm Program multiplying two floating-point matrices.
- Puzzle Compute-bound program.
- Quick Program sorting an array using Quicksort.
- Bubble Program sorting an array using Bubblesort.
- Tree Program sorting an array using Treesort.
- FFT Program calculating a Fast Fourier Transform.

Stanford measures the execution time in milliseconds for each one of the ten small programs included in the benchmark suite. Two weighted sums are calculated. One reflecting the execution times of the fixed-point programs and one reflecting the execution times of the floating-point programs. The weighted sums are calculated based on the program execution times and a number of, in Stanford, predefined constants. The fixed-point weighted sum includes execution times for all programs except Mm and FFT, and the floating-point weighted sum includes all the execution times.

Note that, since MicroBlaze and OpenRISC 1200 does not include hardware floating-point support, the Stanford benchmark is compiled with the GCC option -msoft-float. This options forces all floating-point operations to be done in software with integer arithmetics. Thus the floating-point applications in this benchmark actually reflects integer performance.

2.5.4 Typical control application

The third benchmark consist of a typical control application. The benchmark uses floating-point operations and thus it is compiled with soft floating-point support for all processors to get a fair comparison.

3 Results

This chapter presents the results obtained from the benchmarks and the synthesis, the usability in terms of available tools, available documentation and how easy the processors are to port and configure are discussed. The configurability of the processors is also discussed.

3.1 Benchmarks

This section presents the benchmark results for the Dhrystone 2.1 benchmark, the Stanford benchmark and the typical control application benchmark. Three configurations are evaluated, the comparable configuration, the performance optimized configuration and the area optimized configuration. The results are discussed and for some cases further explained.

3.1.1 Comparable configuration

The comparable configurations of the processors run at the same clock frequency and have the same amount of cache memory. Hardware multipliers and dividers are used with roughly the same latencies among the processors. For a complete specification of the different configuration parameters see Table 4 and Table 2 for the configuration possibilities and other non-configurable options.

3.1.1.1 Dhrystone 2.1

The Dhrystone 2.1 benchmark measures only integer performance and does not stress an 8 Kbyte cache much. The significance of the Dhrystone 2.1 benchmark results should not be taken too seriously since the benchmark is considered unreliable for today's processor architectures as stated in chapter 2.5.2 Dhrystone 2.1.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	30	30	30
Time for one Dhrystone iteration (us)	22.5	32.7	50.5
Dhrystone iterations/second	44444.4	30611.4	19808.6
Dhrystone iterations/second/MHz	1481.48	1020.38	660.29

Table 7: Dhrystone 2.1 benchmark results for the comparable configuration.

The Dhrystone 2.1 benchmark results, see Table 7, are best for LEON2, followed by MicroBlaze and OpenRISC 1200.

3.1.1.2 Stanford

Stanford is a larger benchmark stressing the memory subsystem more than the Dhrystone 2.1 benchmark. The benchmark suite contains a number of

small programs benchmarking different aspects of the processors. Refer to 2.5.3 Stanford for further descriptions of the included programs and more information on the Stanford benchmark suite.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	30	30	30
Run times (ms)			
Perm	50	129	122
Towers	67	144	130
Queens	50	77	74
Intmm	34	54	54
Mm	1050	633[36]	1749
Puzzle	367	374	380
Quick	33	66	56
Bubble	50	112	70
Tree	67	109	99
FFT	1450	375[36]	2692
Fixed-point composite	88	156	140
Floating-point composite	1039	507[36]	1846

Table 8: Stanford benchmark results for the comparable configuration.

The Stanford benchmark suite results are shown in Table 8. The most noticeable result is that the LEON2 processor performs best in all fixed-point subtests. In all but the Puzzle test, LEON2 has more than 32 percent shorter execution time than the other processors. OpenRISC 1200 performs slightly better than MicroBlaze in all tests but the Puzzle test and the Intmm test.

For the floating-point tests, an interesting result is that the floating-point results for the MicroBlaze processor are nearly unrealistically good. The results can not be explained by the configuration parameters. One possibility could be that the MicroBlaze processor does not implement the full IEEE-754 floating-point standard, which LEON2 and OpenRISC 1200 does. To confirm this, Kahan's floating-point test program Paranoia [PASRC] was compiled and executed on the MicroBlaze processor.

The result from Paranoia, see appendix C Paranoia, shows that the MicroBlaze GCC library does not implement proper IEEE-754 floating-point operations. The GCC library floating-point functions seems to have serious faults. Paranoia showed that the LEON2 GCC library behaves correctly. Unfortunately no results were obtained for the OpenRISC 1200, see appendix C Paranoia for more details. The OpenRISC 1200 seems to implement proper floating-point operations, since the floating-point routines in the GCC library are compiled from the same source code as the LEON2 floating-point routines.

36 The MicroBlaze GCC library software routines for floating-point arithmetics does not comply with the IEEE-754 floating-point standard.

Because of the above mentioned shortages in the MicroBlaze GCC library, the results for the floating-point programs in Stanford can not be compared to the other processors results for those programs.

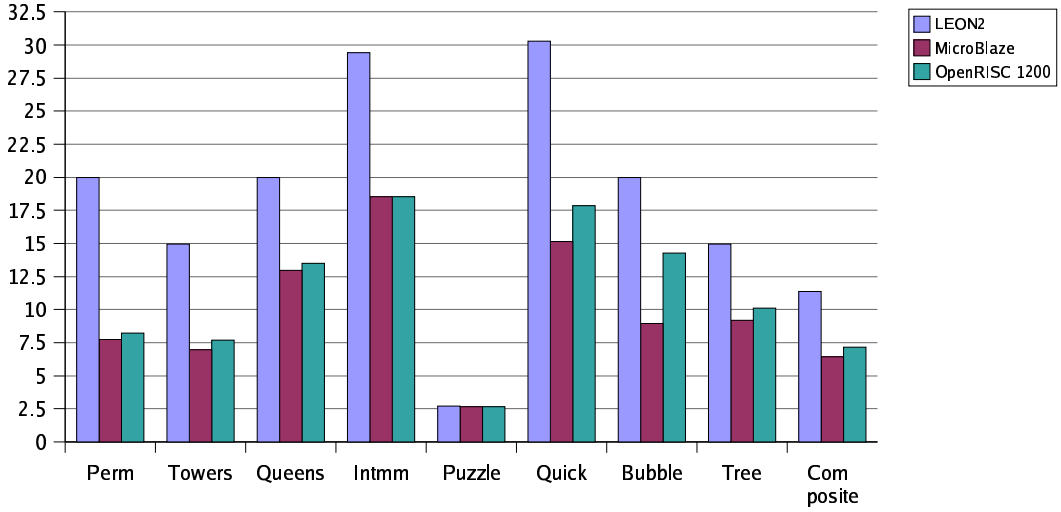

Figure 4: Stanford integer results for the comparable configuration. The result is presented in iterations/second.

One interesting observation in Figure 4, showing the fixed-point results for Stanford, is that the Perm program is much faster in the LEON2 case compared to the other processors. This could be explained by the fact that Perm is a recursive program, with a small recursion depth, and that LEON2 has a windowed register file, which accelerates function calls compared to a flat register file. With a greater recursion depth, the LEON2 processor could have shown worse results since greater depth increases the probability of window overflows.

Both the LEON2 processor and the MicroBlaze processor only fetch the requested word on a data read. The OpenRISC 1200 processor fetches a whole cache line using the critical word first policy on a data cache read. This gives the OpenRISC 1200 an advantage in programs where data has a high spatial locality such as the Quick and Bubble sorting algorithms. In these two tests the OpenRISC 1200 processor performs significantly better than the MicroBlaze processor.

3.1.1.3 Typical control application

The typical control application is a real application, larger than both Dhrystone 2.1 and Stanford. The typical control application stresses the memory system significantly. Refer to 2.5.4 Typical control application for more information.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	30	30	30
Run time (s)	168	~212[37]	N/A[38]

Table 9: Typical control application benchmark results for the comparable configuration.

The results from the typical control application benchmark, see Table 9, for MicroBlaze can not be compared with the results for LEON2, since the typical control application uses floating-point arithmetics and MicroBlaze does not implement working IEEE-754 floating-point arithmetics. For more details refer to 3.1.1.2 Stanford. In spite of the fact that MicroBlaze uses a simpler and thus faster algorithm, the LEON2 processor still performs better. A possible explanation could be the less advanced cache architecture in MicroBlaze.

The typical control application did not execute correctly on the OpenRISC 1200, and therefore did not yield any useful results. This could be the effect of some bug in the toolchain or even in the processor, but more realistically it is an incomplete port of the newlib C library [NLWEB]. The OpenRISC 1200 port of the library was downloaded from the OpenCores CVS server, also available via the OpenCores website [OCWEB], and the versions tried were 1.8.2 and 1.11.0.

3.1.1.4 Reflections

The LEON2 processor performs best for both Dhrystone 2.1 and Stanford. Its time for one Dhrystone 2.1 run is more than 31 percent shorter than the other processors and the Stanford fixed-point composite shows more than 37 percent better results than the other processors. The OpenRISC 1200 processor performs slightly better than the MicroBlaze processor when running Stanford at the same clock frequencies with the configuration defined in 2.4.2 Comparable configuration, mainly due to the cache architecture in MicroBlaze.

The MicroBlaze's incomplete support for IEEE-754 floating-point, combined with the non-existing results for the OpenRISC 1200 from the typical control application (see 3.1.1.3 Typical control application), raises the question whether this benchmark is motivated or not. The benchmark results are presented anyway since it is a real application and since it really shows the importance of a good implementation of the memory subsystem.

37 Time measured with a stopwatch since the MicroBlaze timer overflows.

38 No result available since the control application did not execute properly because of a possible bug somewhere, or an incomplete port of the newlib library.

3.1.2 Performance optimized configuration

The goal with the performance optimized configurations is to achieve maximum performance. In other words to achieve as good results as possible for the benchmarks. The three processors are individually tuned for optimal performance. For a complete specification of the different configuration parameters, see Table 2 and Table 5.

3.1.2.1 Dhrystone 2.1

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	53.3	80	40
Time for one Dhrystone iteration (us)	12.8	13.1	37.8
Dhrystone iterations/second	78431.4	76189.6	26454.9
Dhrystone iterations/second/MHz	1471.51	952.37	661.37

Table 10: Dhrystone 2.1 benchmark results for the performance optimized configuration.

The Dhrystone 2.1 benchmark results, see Table 10, are best for LEON2, followed by MicroBlaze and OpenRISC 1200. The efficiency of the architecture is reflected by the Dhrystone iterations/second/MHz value in Table 10. LEON2 shows more than two times the OpenRISC 1200 value for Dhrystone iterations/second/MHz. When not taking efficiency into consideration, LEON2 still yields the best result, even though MicroBlaze shows nearly the same result.

Better results for the LEON2 processor was achieved with another configuration, which was running with a clock frequency of 66.7 MHz with soft multiplication and division, and a smaller cache with a simpler architecture. However it showed worse result for larger and more advanced programs. It also showed lower efficiency when taking the clock frequency into consideration.

3.1.2.2 Stanford

	LEON2	MicroBlaze	OpenRISC 1200
Processor frequency (MHz)	53.3	80	40
Run times (ms)			
Perm	34	52	91
Towers	50	58	97
Queens	34	31	55
Intmm	34	22	41
Mm	600	241	1313
Puzzle	200	150	285
Quick	16	25	42
Bubble	33	44	53
Tree	50	44	79
FFT	733	148	1981
Fixed-point composite	60	62	106
*Fixed-point composite*MHz*	3180	4976	4232
Floating-point composite	561	198	1369
*Floating-point composite*MHz*	29733	15864	54748

Table 11: Stanford benchmark results for the performance optimized configuration.

The Stanford benchmark suite results are shown in Table 11. The MicroBlaze processor results for the floating-point programs can not be taken into consideration, see 3.1.1.2 Stanford for more information.

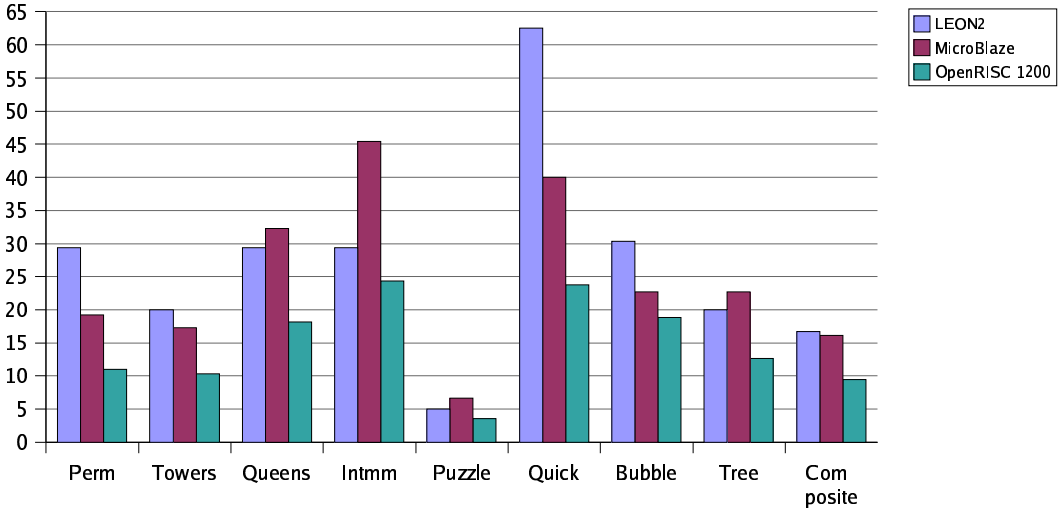

Figure 5: Stanford integer results for the performance optimized configuration. The result is presented in iterations/second.

The results shown in Figure 5 shows that the LEON2 processor and the MicroBlaze processor ties in the subtests with four wins each. When weighted into the composite value, the LEON2 processor wins with a small marginal. Despite the less advanced cache architecture for MicroBlaze, it still performs

well thanks to the higher clock frequency. The OpenRISC 1200 processor shows better results than MicroBlaze for the product of the frequency and the fixed-point composite in the Stanford benchmark. The result could indicate that the OpenRISC 1200 processor has a more efficient architecture than the MicroBlaze processor. Even so it fails to perform as well as the other two processors because of the poor implementation, which prevents it from reaching higher clock frequencies.

When comparing the sorting algorithms, Quick and Bubble are algorithms that operates on arrays, which are data with high spatial locality. Tree on the other hand uses a binary tree data type, which in this case is implemented with pointers. This results in a data structure where the data has low spatial locality. The OpenRISC 1200 data cache performs better than the MicroBlaze data cache on data with high spatial locality, in contrary to data with low spatial locality, where it yields worse results due to high overheads for fetching data never to be used. Therefore it yields much worse results than the MicroBlaze processor in the binary tree sorting algorithm, compared to the Quick and Bubble algorithms, where the results are good when taking clock frequency into the calculation. In the tree sorting algorithm, the OpenRISC 1200 processor does not benefit as much from its larger cache lines, with critical word first, as in the other algorithms, where the data has higher spatial locality.

3.1.2.3 Typical control application

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	53.3	80	40
Run time (s)	93	~110[39]	N/A[40]
*Run time*MHz*	4929	~8800	N/A

Table 12: Typical control application benchmark results for the performance optimized configuration.

Even though the MicroBlaze does not implement proper IEEE-754 floating-point arithmetics, the LEON2 processor performs better in this benchmark. The 67 MHz version of the LEON2 processor, with soft multiplication and division, and smaller and simpler cache, performed worse results in this benchmark. The execution time was approximately 17 percent longer than the execution time for the 53 MHz version.

39 Time measured with a stopwatch since the MicroBlaze timer overflows.

40 No result available since the control application did not execute properly because of a possible bug somewhere, or an incomplete port of the newlib library.

3.1.2.4 Reflections

Compared to the results from the comparable configuration, the outcome was more even in the performance optimized configuration. LEON2 and MicroBlaze showed almost the same results for Dhrystone 2.1 and Stanford. MicroBlaze is optimized for high clock frequencies, which can be seen in the benchmarks for this configuration. The fact that MicroBlaze reaches higher clock frequencies than the other two processors, can be the result of a high optimization for Xilinx FPGA circuits. The results from the OpenRISC 1200 on the other hand were more moderate, which likely is due to the lower clock frequency.

The 67 MHz version of the LEON2 processor yielded approximately 15 percent shorter execution time than the 53 MHz version for both Dhrystone 2.1 and Stanford. For the typical control application benchmark, the 53 MHz version showed approximately 15 percent shorter execution time than the 67 MHz version. Since the typical control application is considered to be the most realistic benchmark of the three, the result from this benchmark is the reason why the 53 MHz configuration were chosen as the performance optimized configuration for LEON2. The efficiency per clock cycle for the 67 MHz version is lower than for the 53 MHz version, which can be confirmed by the fact that the relative decrease in clock frequency is higher than the relative decrease in execution time for the 67 MHz version compared to the 53 MHz version.

3.1.3 Area optimized configuration

The goal with the area optimized configurations is to achieve moderate performance with low area cost. The three processors are individually tuned for low area cost.

To give the synthesis tools the same prerequisites, the three processors run at the same clock frequencies. Since increased cache size does not increase the used logic as much as the used block RAMs, the cache size has not been considered as a vital area parameter. Therefore all processors are configured with the same cache size of 4 Kbyte for instruction and data caches respectively. All three processors were also synthesized with software division. Since LEON2 does not support individual parameter settings for multiplication and division, hardware multiplication was also turned off for this processor. For a complete specification of the different configuration parameters, see Table 2 and Table 6.

3.1.3.1 Dhrystone 2.1

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	26.7	26.7	26.7
Time for one Dhrystone iteration (us)	32.6	43.1	93.9
Dhrystone iterations/second	30690.5	23188.3	10653.8
Dhrystone iterations/second/MHz	1149.46	868.48	399.02

Table 13: Dhrystone 2.1 benchmark results for the area optimized configuration.

Table 13 shows the Dhrystone 2.1 benchmark results. As in the comparable configuration, LEON2 shows the best results, followed by MicroBlaze and OpenRISC 1200.

3.1.3.2 Stanford

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	26.7	26.7	26.7
Run times (ms)			
Perm	50	145	137
Towers	83	162	146
Queens	50	86	83
Intmm	150	92	131
Mm	2550	769	3852
Puzzle	467	421	443
Quick	50	122	68
Bubble	67	131	80
Tree	117	172	139
FFT	2216	449	4049
Fixed-point composite	133	201	176
Floating-point composite	1861	625	3098

Table 14: Stanford benchmark results for the area optimized configuration.

The Stanford benchmark suite results are shown in Table 14. The MicroBlaze processor results for the floating-point programs can not be taken into consideration, see 3.1.1.2 Stanford for more information.

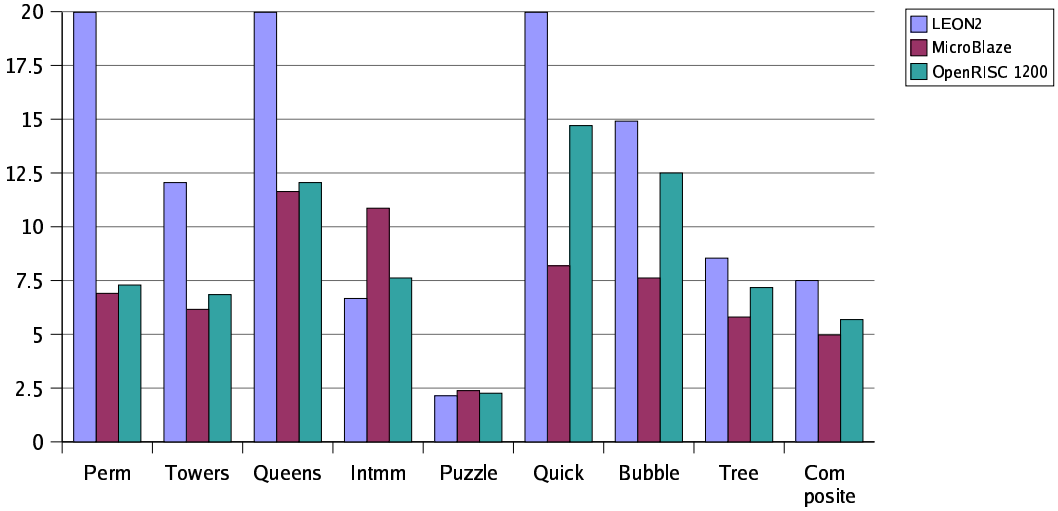

Figure 6: Stanford integer results for the area optimized configuration. The result is presented as iterations/second.

Figure 6 shows that the LEON2 processor yields best results followed by OpenRISC 1200 and MicroBlaze. In the Puzzle and Intmm tests, where MicroBlaze performed so well, LEON2 yielded less good results and finished last in both tests. Intmm is a program doing an extensive amount of multiplications and since LEON2 is configured with software multiplication

and division, the results are expected to be worse than for the other processors. The same reasoning could be applied in the Puzzle case, since Puzzle also uses multiplication.

3.1.3.3 Typical control application

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Processor frequency (MHz)	26.7	26.7	26.7
Run time (s)	374.5	~363[41]	N/A[42]

Table 15: Typical control application benchmark results for the area optimized configuration.

MicroBlaze performs better than LEON2, but since it does not implement proper IEEE-754 floating-point arithmetics, no conclusions can be made from the result.

3.1.3.4 Reflections

LEON2 showed better results than MicroBlaze and OpenRISC 1200 for Dhrystone 2.1 and Stanford. Due to the software multiplication, LEON2 did not perform as well compared to the other processors for some subtests in Stanford.

41 Time measured with a stopwatch since the MicroBlaze timer overflows.

42 No result available since the control application did not execute properly because of a possible bug somewhere, or an incomplete port of the newlib library.

3.1.4 Benchmark summary

For reasons mentioned in 3.1.1.4 Reflections, only two of the benchmarks yielded useful results. The results from these benchmarks are presented in Figure 7 and Figure 8.

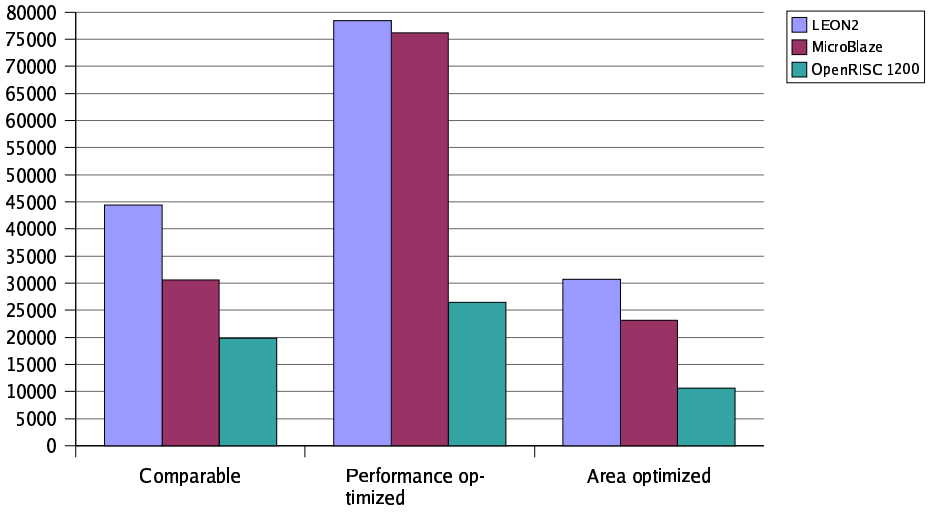

Figure 7: Iterations/second for Dhrystone 2.1 for the three configurations.

The fact that the 67 MHz version of LEON2 performs better than the 53 MHz version in both Dhrystone 2.1 and Stanford, but yields worse result for the typical control application, shows how difficult it is to find usable benchmarks which reflect the execution of real programs. Thus, when only considering Dhrystone 2.1 (see Figure 7) and Stanford (see Figure 8), MicroBlaze and LEON2 performs roughly the same for the performance optimized configuration. However, since both Dhrystone 2.1 and Stanford are small synthetic benchmark programs, that not fully stress the cache, which the typical control application does, the result from the typical control application must be taken into account. In spite of the fact that MicroBlaze uses a simplified algorithm for floating-point arithmetics, LEON2 still performs better in the typical control application benchmark for all configurations but the area optimized configuration, where it uses soft multiplication and division.

The previous mentioned conclusions and the fact that LEON2 yields best results for nearly all benchmarks for all three configurations, must render LEON2 a better processor than both MicroBlaze and OpenRISC 1200, with respect to performance.

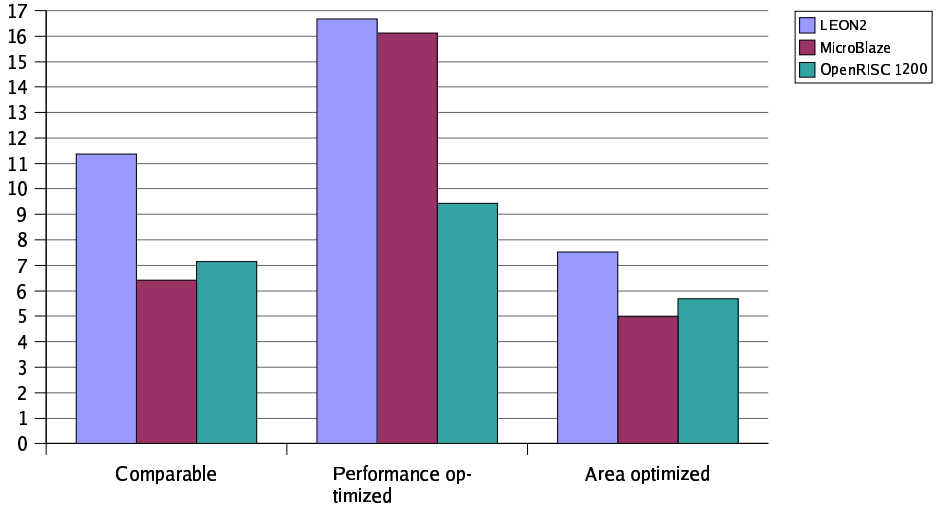

Figure 8: Stanford fixed-point composite/second for the three configurations.

MicroBlaze performs well in most benchmarks and must be considered a worthy opponent to LEON2. MicroBlaze is optimized with high clock frequencies in mind, which can be seen from the more moderate benchmark results for the comparable configuration and the area optimized configuration, which both run at fairly low clock frequencies. The main drawbacks of the MicroBlaze processor must be the cache subsystem and the floating-point implementation.

OpenRISC 1200 performs well for the comparable configuration and the area optimized configuration, where it yields better results than MicroBlaze in the Stanford fixed-point benchmark, see Figure 8. The major drawback of the OpenRISC 1200 processor must be the poor implementation which prevents it from reaching higher clock frequencies. This can be seen in the results from the benchmarks for the performance optimized configuration, see 3.1.2 Performance optimized configuration.

3.2 Synthesis results

Synplicity Synplify Pro is used for the synthesis of both the OpenRISC 1200 and the LEON2 processor. Xilinx XST is used for the synthesis of MicroBlaze. The Synplicity Synplify Pro synthesis tool generates better synthesis results in most cases compared to the synthesis results when Xilinx XST is used. The reason why Synplicity Synplify Pro is not used for the synthesis of the MicroBlaze processor is the fact that Xilinx XST is tightly integrated into the Xilinx EDK implementation flow, and since the MicroBlaze processor is distributed as a parameterizable netlist, i.e. it is already synthesized. In reality, only the peripherals in a MicroBlaze system are synthesized.

Since both the MicroBlaze processor and the peripherals are highly optimized for Xilinx FPGA circuits, the FPGA resource usage is expected to be much lower than the resource usage of the other processors. OpenRISC 1200 and LEON2 are written not only for Xilinx FPGA circuits, the two processors are mostly written in general Verilog and VHDL code. In some critical parts, low-level cells are instantiated for the intended target technology. The consensus is that it is very difficult for a synthesis tool to synthesize both OpenRISC 1200 and LEON2 with the same low FPGA resource utilization as MicroBlaze.

All three processor systems are synthesized with instruction cache, data cache, integer unit, timer unit, debug unit, and UART. The MicroBlaze and OpenRISC 1200 systems are synthesized with 8 Kbyte of on-chip RAM, which is needed for initialization of the caches. All initialization for the LEON2 processor is handled by the debug support unit.

A processor core is in this section defined as the integer unit and both caches, and since MicroBlaze and LEON2 have this hierarchical distinction of the processor core, the synthesis of them is straight-forward. OpenRISC 1200 on the other hand instantiates all processor units inside a toplevel Verilog file. In order to produce comparable results, that is synthesize the same units, the debug unit and the timer unit had to be turned off when synthesizing the OpenRISC 1200 processor core.

The synthesis results for both the whole system and the synthesis results for the processor cores are presented. The results for the whole systems are the results after the mapping stage, where the design has been mapped to the target FPGA circuit. The results for the processor cores are the results after synthesis.

The synthesis results presented are the total amount of lookup tables (LUT), the number of RAMB16 cells and the number of MULT18X18 cells. In a Xilinx Virtex-II FPGA circuit the LUT cells can implement any logic function with up to four input variables. The smallest block RAM resource in a Xilinx Virtex-II FPGA circuit is the RAMB16 cell, which is an 18 Kbit dual-port RAM where

16 Kbit are used to store data and 2 Kbit can be used to store parity information. The MULT18X18 cell is a hardware signed multiplier with two 18-bit inputs and one 36-bit output.

3.2.1 Comparable configuration

All three processors are synthesized with a clock frequency of 30 MHz for the comparable configuration. MicroBlaze and OpenRISC 1200 are configured to use 8 Kbyte of on-chip RAM. All three processors have an effective cache size of 8 Kbyte. The LEON2 processor uses a 2-way set associative cache architecture with LRU replacement policy, while MicroBlaze and OpenRISC 1200 use a direct-mapped cache architecture. A 2-way LRU implementation of the cache results in a higher area usage than for a direct-mapped cache implementation. For more information on parameters and configuration options for the comparable configuration refer to Table 3 and Table 4. The FPGA resource usage for the whole systems, including all peripherals, and for the processor cores are presented in Table 16.

	LEON2	***MicroBlaze***	***OpenRISC 1200***
System			
Number of 4-input LUTs	7460	2441	6197
Number of RAMB16	14	14	16
Number of MULT18X18s	1	3	4
Processor core			
Number of 4-input LUTs	5516	1679	4626
Number of RAMB16	14	10	12
Number of MULT18X18s	1	3	4

Table 16: FPGA resource usage for the comparable configuration.

MicroBlaze has as expected the lowest FPGA resource usage of the three processors, primarily since it is highly optimized for Xilinx FPGA circuits. LEON2 has the highest resource usage among the three processors, but it also has for example the largest register file and the most advanced cache subsystem among the three processors.

MicroBlaze and OpenRISC 1200 use four additional RAMB16 cells for the whole system compared to the processor core, this is because the on-chip RAM is not part of the processor core. The great variety of used MULT18X18 cells between the processors can be explained by the latency of the hardware multiplication and the implementation strategy, which differ among the three.

3.2.2 Performance optimized configuration

For the performance optimized configuration, each processor are individually tuned with the highest performance in mind. Low FPGA resource usage is not considered the primary goal for this configuration of the three processors.

The tuning resulted in a LEON2 system running at 53 MHz, a MicroBlaze system running at 80 MHz and an OpenRISC 1200 system running at 40 MHz. MicroBlaze and OpenRISC 1200 are configured to use 8 Kbyte of on-chip RAM, for more information on parameters and configuration options for the performance optimized configuration refer to Table 3 and Table 5. The FPGA resource usage for the whole systems including all peripherals and for the processor cores is presented in Table 17.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
System			
Number of 4-input LUTs	8974	2442	6443
Number of RAMB16	26	14	16
Number of MULT18X18s	1	3	4
Processor core			
Number of 4-input LUTs	7178	1679	4888
Number of RAMB16	26	10	12
Number of MULT18X18s	1	3	4

Table 17: FPGA resource usage for the performance optimized configuration.

As seen in Table 17 above MicroBlaze has the lowest resource usage among the three processors. The LUT resource usage for the OpenRISC 1200 processor core and the LEON2 processor core are roughly three and four times larger than the LUT resource usage of the MicroBlaze processor core.

3.2.3 Area optimized configuration

The goal with the area optimized configuration for each processor is to optimize each processor individually for low area usage, i.e. low resource utilization with reasonable performance. To give the implementation tools the same prerequisites, each system is configured to run at a clock frequency of 27 MHz.

MicroBlaze and OpenRISC 1200 are configured to use 8 Kbyte of on-chip RAM, for more information on parameters and configuration options for the area optimized configuration refer to Table 3 and Table 6. The FPGA resource usage for the whole systems including all peripherals and for the processor cores are presented in Table 18 below.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
System			
Number of 4-input LUTs	5781	2325	5865
Number of RAMB16	8	12	14
Number of MULT18X18s	0	3	4
Processor core			
Number of 4-input LUTs	3820	1566	3909
Number of RAMB16	8	8	10
Number of MULT18X18s	0	3	4

Table 18: FPGA resource usage for the area optimized configuration.

MicroBlaze still has the lowest area usage, but LEON2 now has a lower area usage than the OpenRISC 1200. This is primarily because the LEON2 processor is configured without hardware multiplication and the OpenRISC 1200 is configured to support MAC instructions. A synthesis of the OpenRISC 1200 core without multiplier and MAC unit yielded a resource usage of 3541 LUTs. A synthesis with only the MAC unit turned of yielded 3665 LUTs. This results in an approximate cost of 244 LUTs for hardware multiplication support in OpenRISC 1200.

3.2.4 Synthesis discussion

The difference, in number of RAMB16 cells, between the whole system and the processor core for MicroBlaze and OpenRISC 1200 is equal to four for all three configurations. Four RAMB16 cells corresponds to 8 Kbyte, which is the size of the on-chip RAM in both processors.

There is no difference in the resource usage between the MicroBlaze processor core for the performance optimized configuration and the comparable configuration. This is because the core is a parameterizable netlist, and therefore not synthesized, and it is instantiated with the same parameters in both cases. The resource usage for the area optimized configuration of the MicroBlaze processor core is lower compared to the other two configurations. This is because the area optimized configuration is configured to use smaller caches and software division.

LEON2 has the largest difference in LUT resource usage between the performance optimized and the area optimized configuration, compared to the other two processors. This is because the LEON2 processor is much more configurable, for example the cache associativity can be configured from one up to four ways. Different replacement policies can be used with different resource utilization impact.

3.2.5 Synthesis summary

The total number of LUTs as well as the number of LUTs used for the processor core and additional peripherals are summarized in Figure 9 below. Both the MicroBlaze system and its core have a very low LUT usage, which remains almost constant among the three configurations of the processor. LEON2 is the most configurable processor resulting in very different resource usage between the three configurations. The OpenRISC 1200 processor keeps at almost the same resource usage for all three configurations. This could be explained by the fact that the difference in settings between the three configurations is fairly small.

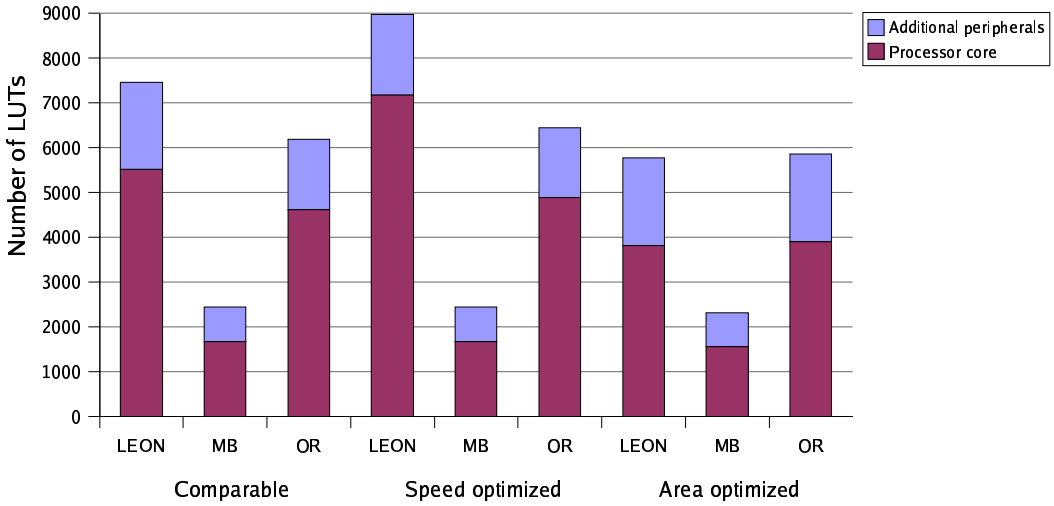

Figure 9: FPGA LUT usage for all processor cores and additional peripherals for all configurations. LEON, MB and OR are abbreviations for LEON2, MicroBlaze and OpenRISC 1200 respectively.

When looking at a specific processor, in Figure 9, the amount of LUT cells is constant for the additional peripherals over the three configurations. This is because each processor has the same peripherals, with almost the same configuration, in all three configurations.

MicroBlaze has the lowest area usage, but has no portability to for example other FPGA vendors than Xilinx, like the other two processors have. For the comparable and the performance optimized configuration, LEON2 has the highest area usage, but it also yields the highest performance. OpenRISC 1200 utilize a bit less area resources than LEON2 for the same two configurations. However it yields a lower performance than LEON2, and has a simpler cache architecture. For the area optimized configuration, where all processors are tuned for low area usage, LEON2 and OpenRISC 1200 have almost the same area usage.

3.3 Performance

Since there is no obvious unit for measurement of performance, an attempt to present measurement units of interest will be done in this section. Parameters as clock frequency, area resource usage and benchmark results will be taken into consideration.

3.3.1 Performance per clock cycle

To get a fair comparison between different processors and configurations, the relation performance per clock cycle can be used. This gives an estimate on how efficient a processor architecture is. The estimate for the different configurations has been calculated for the Dhrystone 2.1 benchmark and for the fixed-point composite of the Stanford benchmark. Figure 10 below presents the results for the Dhrystone 2.1 benchmark. LEON2 has the best efficiency results for all three configurations. Notable is the lower efficiency of the area optimized configuration, where LEON2 has a smaller cache and uses software division and multiplication. The reduction in efficiency for the area optimized configuration of LEON2 compared to the other two configurations is greater than for the other two processors, when looking at the same case. This is probably due to the use of hardware multiplication in the area optimized configuration for the other two processors. The efficiency for the MicroBlaze processor shows little difference between the three configurations.

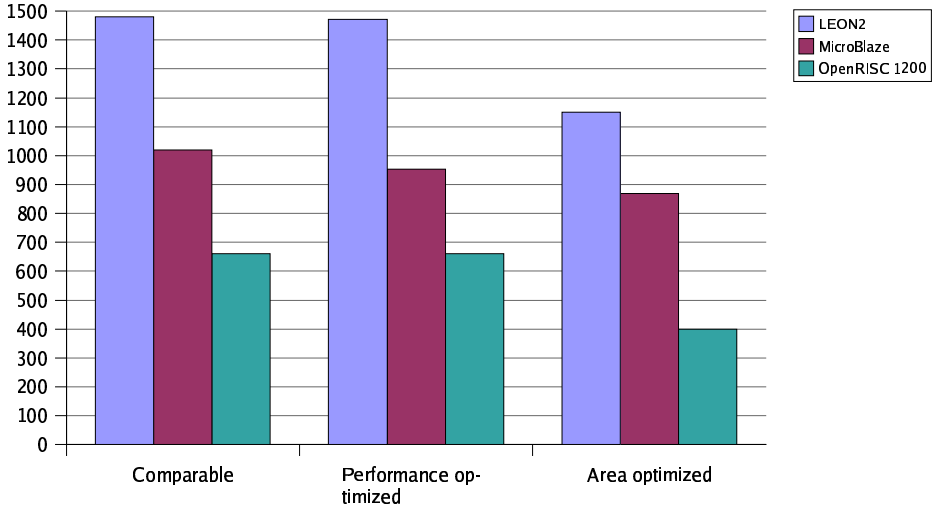

Figure 10: Dhrystone 2.1 result for the three configurations. The result is presented as Dhrystone iterations/second/MHz, which gives the relation performance per clock cycle for the different processor configurations.

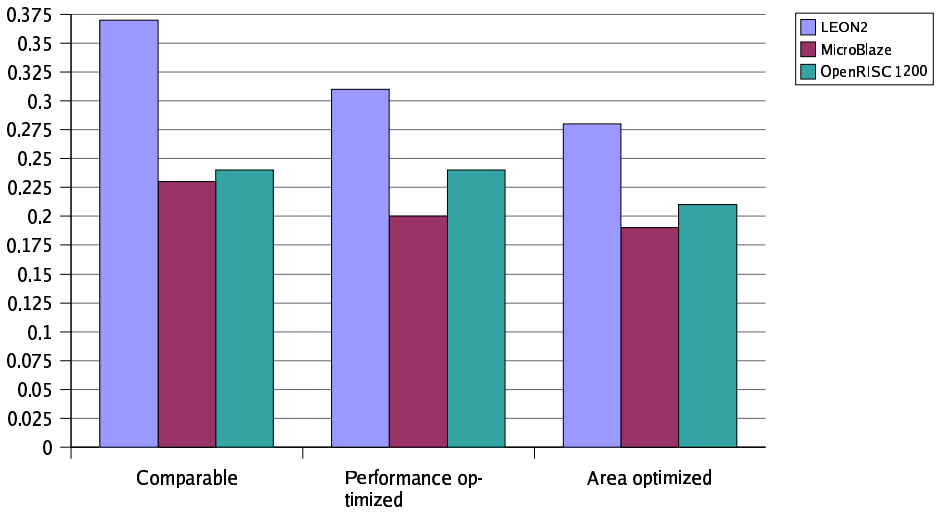

Figure 11: Stanford fixed-point composite result for the three configurations. The result is presented as Stanford fixed-point composites/second/MHz, which gives the relation performance per clock cycle for the different processor configurations.

The efficiency for the configurations for the Stanford fixed-point composite is visualized in Figure 11. The main difference compared to the Dhrystone 2.1 result presented in Figure 10 must be the efficiency gain for the OpenRISC 1200 processor, which yields better results than MicroBlaze for the Stanford efficiency test. As seen in Figure 10 and Figure 11, the OpenRISC 1200 shows nearly constant efficiency for the comparable configuration and the performance optimized configuration. This indicates that the two configurations are nearly identical, besides the increase in clock frequency, which can be confirmed from Table 4 and Table 5. Notable is that the comparable configuration shows best efficiency per clock cycle for all three processors.

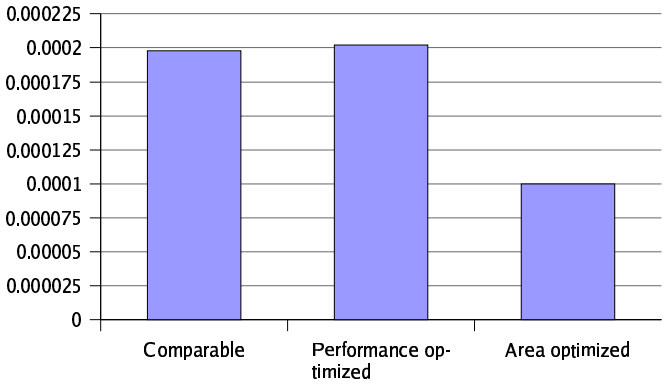

Figure 12: Typical control application iterations/second/MHz for LEON2 for the three configurations.

The performance optimized configuration of the LEON2 processor is expected to show a higher performance per clock cycle compared to the comparable configuration when executing larger applications. This is because the cache size for the performance optimized configuration of LEON2 is twice the size of the cache for the comparable configuration. The difference is even larger compared to the area optimized configuration, which besides a smaller cache also uses a direct-mapped cache architecture. This can be seen from the results for the typical control application benchmark, see Figure 12.

3.3.2 Performance per area unit

When designing a system, two important aspects are area usage and performance in terms of throughput. To draw conclusions about both area usage and performance for the processors discussed in this report, a very simple model is used. A high value of benchmark performance per area resource is considered a system with good performance with respect to its area usage.

Both the Dhrystone 2.1 and the Stanford integer benchmarks have been executed on all three processors and the performance from those benchmarks is measured in Dhrystone 2.1 iterations/second and Stanford fixed-point composites/second. In a Xilinx FPGA circuit the most important resource is the LUT cells, which therefore is used as an estimate of a processors area. The benchmark performance per area resource is calculated as Dhrystone 2.1 iterations/second/LUT for Dhrystone 2.1 and Stanford fixed-point composites/second/LUT for the Stanford fixed-point benchmark. These data are shown in Table 19 below.

	LEON2	*MicroBlaze*	*OpenRISC 1200*
Comparable configuration			
Number of 4-input LUTs	5516	1679	4626
Dhrystone iterations/second	44444.4	30611.4	19808.6
Stanford integer composite/second	11.36	6.41	7.14
Dhrystone iterations/second/LUT	8.06	18.23	4.28
Stanford fixed-point composite/second/LUT	2.06E-3	3.82E-3	1.54E-3
Performance optimized configuration			
Number of 4-input LUTs	7178	1679	4888
Dhrystone iterations/second	78431.4	76189.6	26454.9
Stanford fixed-point composite/second	16.67	16.13	9.43
Dhrystone iterations/second/LUT	10.93	45.38	5.41
Stanford fixed-point composite/second/LUT	2.32E-3	9.61E-3	1.93E-3
Area optimized configuration			
Number of 4-input LUTs	3820	1566	3909
Dhrystone iterations/second	30690.5	23188.3	10653.8
Stanford integer composite/second	7.52	4.98	5.68
Dhrystone iterations/second/LUT	8.03	14.81	2.73
Stanford fixed-point composite/second/LUT	1.97E-3	3.18E-3	1.45E-3

Table 19: Performance values for all processors in all configurations. The performance values are weighted by the area resource usage of the cores. A large value is a good value.

As expected Table 19 shows great results for the MicroBlaze processor, which is highly optimized for Xilinx FPGAs. To visualize the differences in Table 19, see Figure 13 and Figure 14 below.

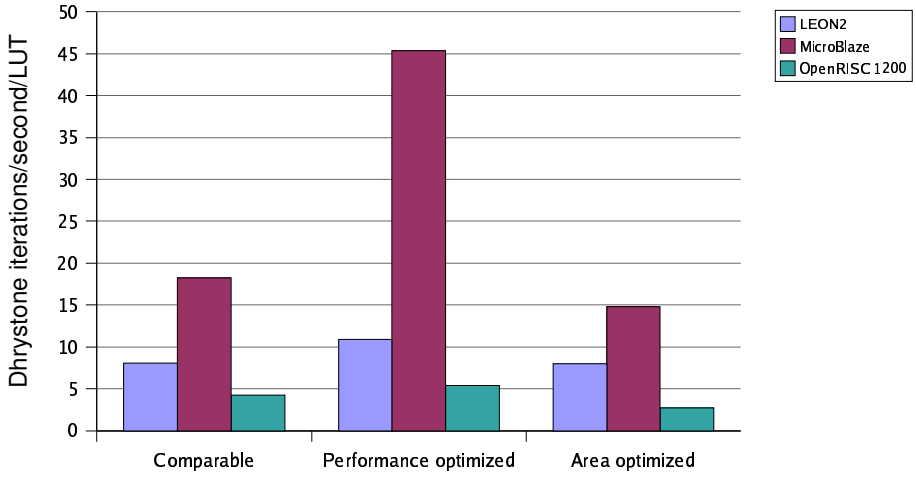

Figure 13: Dhrystone 2.1 iterations/second/LUT for all configurations and processor cores.

As Figure 13 shows, the variance in result for MicroBlaze between the three configurations is roughly proportional to the performance difference among them. This is to be expected since the area usage for the MicroBlaze processor

core is almost constant for the three configurations. Common for the three processors in Figure 13 is that the performance optimized configuration yields best results. Even the simplest configuration of a processor needs a certain amount of area. By adding extra performance enhancing logic, like store buffers, the area increases of course. However the relative increase in area is smaller than the relative increase in performance, which explains why the performance optimized configuration renders best results.

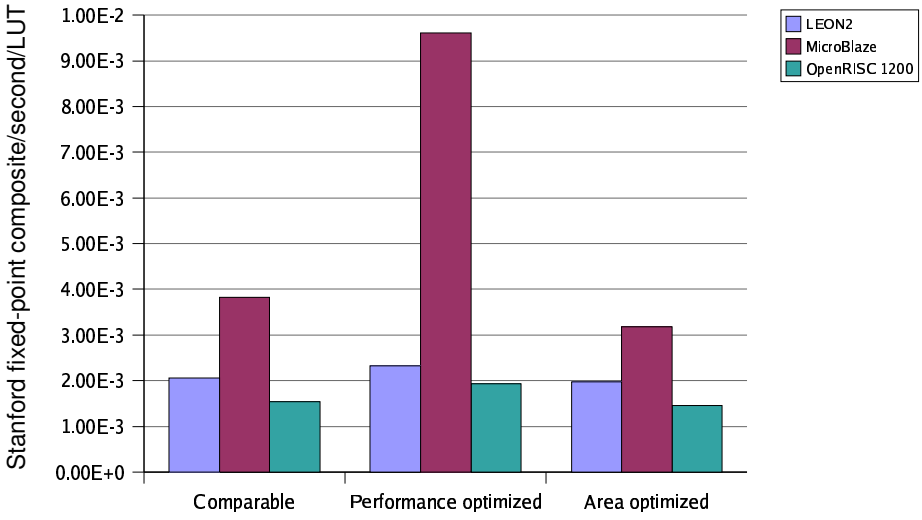

Figure 14: Stanford fixed-point composite/second/LUT for all configurations and processor cores.

As in the Dhrystone 2.1 benchmark (see Figure 13), the MicroBlaze processor renders much better results than the other processors for the Stanford benchmark (see Figure 14). Despite the larger area of LEON2, when compared to OpenRISC 1200, it still yields better performance per area unit.

3.3.3 Performance summary

The LEON2 processor shows highest efficiency per clock cycle for all the configurations for both the Dhrystone 2.1 benchmark and the Stanford fixed-point composite benchmark. Since Stanford is considered a more reliable benchmark than Dhrystone 2.1 and OpenRISC 1200 shows better results than MicroBlaze for the Stanford benchmark, the OpenRISC 1200 processor can be considered more effective per clock cycle than the MicroBlaze processor.

The performance optimized configuration yields best performance per area unit for all processors. MicroBlaze reaches a significantly better result than the other two processors, but is highly optimized for Xilinx FPGA circuits and therefore not as portable as the others. When looking at the more portable processors, the LEON2 processor achieves better performance per area unit than the OpenRISC 1200 processor.

3.4 Usability

This section is intended to give a description of how easy the system is to use. Aspects like how easy the available tools are to use, how much documentation there is and how well it is written will be reviewed together with other important matters like how much effort needed to add a new IP-block.

3.4.1 LEON2

3.4.1.1 Tools

The tool evaluated for LEON2 is the configuration and implementation environment, which consists of make scripts. A TCL/Tk based configuration GUI can be invoked from the make scripts for generating a working LEON2 system. The GUI is very similar to the Linux 2.4 kernel graphical configuration tool and is very simple to use. To help the user, there are tool tip boxes for all configuration options. The output of the configuration tool is a configuration file, which is used by the mkdevice program to generate a VHDL package containing constants defining the configuration of the LEON2 processor.

Everything from configuration to implementation and simulation is handled by make scripts, which works well. The use of make scripts is positive since it is commonly used in software development and is shown effective there.

Debugging possibilities are considered adequate since the GRMON with the DSU back-end easily connects to the hardware DSU in a LEON2 system. Debugging in GRMON is simple and works reliably. Downloading software applications to the target FPGA is fast and all initialization is performed automatically by the DSU. With the simulation back-end (TSIM) for GRMON, debugging works similarly to debugging in real hardware. The TSIM simulator is commercially available from Gaisler Research, but there is also an evaluation version. GRMON supports the GDB interface through which GDB can connect.

The simulation models provided, works fine with HDL simulation. Simulation models for common components like SDRAM and SRAM are included. The UART provides text output in for example Modelsim. The UART has a configuration option to speed up UART communication in simulation, providing faster simulation.

3.4.1.2 Documentation and support

The available documentation for the LEON2 processor and the included tools is good. There is a mailing list providing LEON2 support at Yahoo, see [LMWEB], which is frequently visited by Gaisler Research employees. There is also commercial support available.

3.4.1.3 Hardware configuration

Configuring the LEON2 processor hardware is easy. The configuration is done with a graphical tool which has help available for all options. To add user defined IP-blocks, the VHDL files have to be altered manually. Adding user defined IP-blocks was not necessary for running the benchmarks and has therefore not been evaluated.

3.4.1.4 Portability

Since the LEON2 is already ported to the GR-PCI-XC2V development board, the actual porting process will not be reviewed here. The LEON2 processor includes support for several development boards and the configuration files for the supported boards can easily be altered to support other development boards. Target technology dependent HDL code is instantiated from technology independent components, which uses the intended target technology[43].

3.4.2 MicroBlaze

3.4.2.1 Tools

The tools evaluated for the MicroBlaze processor are the ones included in the Embedded Development Kit v6.2, abbreviated EDK. EDK includes the GNU toolchain for compilation and debugging, a graphical user interface named Xilinx Platform Studio (XPS) for developing FPGA systems with a MicroBlaze or PowerPC processor, and additional tools.

From within XPS one can configure MicroBlaze, add peripherals and user defined IP cores, add software and finally synthesize, place and route and combine the hardware and software into a single bitstream. The bitstream can be downloaded to the FPGA circuit.

The hardware IP-blocks can be instantiated as VHDL, Verilog or a synthesized netlist. To include an IP-block, the interface, and possibly synthesis order, must be defined in some additional files. If the proper directory structure for the IP-block is obtained, the IP-block can easily be instantiated from the XPS GUI. Instead of using the XPS GUI the more advanced user can edit the underlying files.

A possible downside with the underlying configuration files is that they do not follow any standards. A better approach would be to use a standardized HDL such as VHDL to define the interfaces and toplevel instantiation of the IP-blocks.

43 The LEON2 processor is ported to several technologies: Xilinx (Virtex, Virtex-II), Actel (Proasic, Axcel), TSMC-0.25, Atmel (ATC35, ATC25, ATC18) and UMC (FS90, 0.18).

A positive aspect of XPS is that it includes a user friendly wizard for building a base system. This wizard works fine for the supported FPGA development boards, but if the user wants to use an unsupported FPGA development board like the GR-PCI-XC2V [PE03], some additional modification has to be done. The user does not necessary have to use the wizard for building a base system, but this requires deeper knowledge of the hardware being used.

The software can be simulated with an instruction set simulator (ISS) started from the Xilinx Microprocessor Debugger (XMD). The user can use a graphical debugger, like the included version of GDB, to remotely connect to the ISS started by XMD. A shortcoming is that no access to peripherals can be done with this ISS, only the correctness of the software algorithms without input and output can be tested.

The system can be simulated from a HDL simulator like Modelsim. Unfortunately some libraries[44] and models must be compiled before generating the simulation. The documentation lack some information regarding this. A shortcoming with the simulation is that it does not include a behavioral model for neither an external memory or a UART. To simulate a system executing code located in an external memory, the user must add a simulation model of the memory, and instantiate it together with the rest of the system in a toplevel entity.

When running the system on the FPGA development board, XMD can connect to the MicroBlaze processor via a debug unit using the JTAG interface. From XMD the user can download software applications into the MicroBlaze memory, control the execution flow, and inspect registers etc.

The source code for the MicroBlaze processor is proprietary and is not included[45] in the EDK. The processor being closed source requires that the documentation follows a high standard. The closed source code makes the simulation and debugging more difficult, because it is difficult to know what is really happening inside the processor.

3.4.2.2 Documentation and support

The available documentation for MicroBlaze and for the included tools is very informative. There is also an extensive answer database available at the Xilinx website [XILWEB], where users can submit questions. The documentation regarding how to generate a working simulation of the system is unfortunately inadequate.

44 The Simprim library, the Unisim library, the Xilinx core library and the MicroBlaze processor simulation model.

45 The source code for MicroBlaze can be bought from Xilinx.

3.4.2.3 Hardware configuration

The configuration of the MicroBlaze hardware is handled smoothly through the “Add/Edit Hardware Platform Specification” dialog. The user can add, remove and configure different IP-blocks from a simple dialog. For more experienced users, all configuration can be done by manually editing plain text files.

3.4.2.4 Portability

The porting of the MicroBlaze system to the GR-PCI-XC2V FPGA development board [PE03] simply consisted of editing the FPGA pin configuration in the user constraints file. To support the external SRAM memory supplied on the board, a memory controller had to be written with read-modify-write support for byte write operations.

In order to start the execution in external memory, a bootloader had to be implemented in on-chip RAM. The bootloader initiates and enables the caches, followed by a jump to the program in external memory.

The MicroBlaze processor was easy to port to the GR-PCI-XC2V development board. The problems that occurred mostly depended on the interface to the external memory.

3.4.3 OpenRISC 1200

3.4.3.1 Tools

The tools used for compilation and debugging are the standard GNU toolchain, which works well. Compilation is straight-forward, the only matter is that the toolchain port does not include a complete, working C-library. Debugging through GDB is easy, the connection to the development board is handled well by the JP1 program included. A graphical front-end for the debugger is available, DDD, which provides a more user friendly environment.

3.4.3.2 Documentation and support

The OpenRISC 1000 architecture [OC04] is well documented compared to the other available documentation for the OpenRISC 1200 processor. Especially the OpenRISC 1200 specification [OC01] lacks some information. The main problem with the OpenRISC 1200 specification is that it does not reflect the current state of the implementation. The specification seems to document an implementation that will be available in the future. For example the authors claims that the data cache implements an LRU replacement scheme within each set. The source code does not implement other data cache configurations than a one way set associative cache of either 4 Kbyte or 8 Kbyte. One way associative cache means one cache line per set, which is the same as a direct-mapped cache, this makes an LRU implementation useless. In fact, the source

code does not implement an LRU scheme anyway. The same corresponds to the instruction cache, except that it can be configured to an additional size of 512 byte. With the documentation being unsatisfiable it is in some cases vital that the source code is open source. Though it is of great importance that the source code is easy to read, and unfortunately, this is not the case with the OpenRISC 1200 source code.

The OpenRISC 1200 source code is written in almost uncommented Verilog code. This and the fact that the Verilog code is written with a low level of abstraction, makes it difficult to understand the code. The lack of a generate construct combined with the lack of a record data type in Verilog, both available in VHDL, makes the source code larger and more difficult to grasp.

The documentation of the toolchain is really good, since it is a port of the well documented GNU toolchain. Unfortunately we have not found any documentation regarding port specific parameters.

Fortunately there exists a reference design called OpenRISC Platform System-On-Chip (ORPSOC), which shows how to connect some IP-blocks together via the Wishbone bus. This reference design was used as a starting point, when the OpenRISC 1200 was ported to the GR-PCI-XC2V development board.

Support and help with an OpenRISC 1200 system can be found in the web forums on the OpenCores website [OCWEB], where users can submit questions. The Embedded Systems Group at the De Nayer Instituut in Belgium has made a good tutorial [DN04a] regarding how to get an OpenRISC 1200 system implemented on a FPGA development board. The group has also made a tutorial [DN04b] regarding how to build the required software tools. Both these tutorials are available at the groups website [DNWEB].

3.4.3.3 Hardware configuration

The configuration of the OpenRISC 1200 processor is done by manually editing the Verilog source code. All configuration options for the processor are gathered together into a single file containing numerous define statements. The configuration file contains comments for the different options, explaining their effects.

Adding user IP-blocks is somewhat more difficult, the user has to manually edit the Verilog source code for the Wishbone bus, plus instantiate and connect the IP-block. This requires a deep understanding of the bus architecture, and the underlying language. An easier approach, could be to use the ORPSOC reference design and modify it. The advantage with this approach is that the bus interconnect is already implemented.

3.4.3.4 Portability

When porting the OpenRISC 1200 system to the GR-PCI-XC2V FPGA development board [PE03], the user constraints file had to be edited to achieve the correct FPGA pin configuration. Since the supplied SRAM controller only had support for read-modify-write on a 16 bit basis, it had to be rewritten to support read-modify-write on a 32 bit basis.

In order to start the execution in external memory, a bootloader had to be implemented. The bootloader initiates and enables the caches followed by a jump to the program in external memory.

The OpenRISC 1200 processor must be considered fairly easy to port to the GR-PCI-XC2V FPGA development board even though there were some obstacles encountered during the porting process. One problem worth mentioning was the fact that the debug unit requires a dedicated JTAG interface. The problem was solved by dedicating some FPGA pins to a JTAG interface for the debug unit. This lead to the annoying procedure of having to move the JTAG cable between the interfaces when switching from programming to debugging and the other way around. The most troublesome obstacle was a bug in the cache, which produced Wishbone bus burst accesses that did not comply to the Wishbone standard, which led to erroneous execution. The bug has been reported to the OpenRISC 1200 development team. A workaround was used in the cacheable slave units.

3.4.4 Usability summary

LEON2 must be considered the processor which is most easy to configure and implement of the three processors. It is also easier than the others to simulate and debug on real hardware. MicroBlaze has the best documentation together with the best support for adding user defined IP-blocks. Both OpenRISC 1200 and MicroBlaze are easy to port, but MicroBlaze must be considered the easiest of the two since it was straight-forward to add user defined IP-blocks.

3.5 Configurability

The intention of this section is to describe how configurable the three processors are. The word configurable is here intended to be interpreted as the amount of configuration options for a processor and not as how easy the processor is to configure, which is evaluated in 3.4 Usability.

3.5.1 LEON2

The LEON2 processor has an extensive amount of configurable parameters. This can easily be reviewed by the number of configurable parameters in Table 2, which contains some of the more interesting parameters. Besides the many functional options, one can also find several options for improved timing, simulation and debugging.

3.5.2 MicroBlaze

The amount of configurable parameters for MicroBlaze is less than for the LEON2 processor. Most notable is the configurability of the caches, which is much less configurable than for LEON2. A limitation is that the caches only can be configured to cache a continuous subset of the total memory.

3.5.3 OpenRISC 1200

The OpenRISC 1200 processor is quite configurable. There are options for specific parameters like whether to use software or hardware multiplication and some options for the ALU. The main difference in the configuration options between the OpenRISC 1200 and the MicroBlaze processor is that in OpenRISC 1200 there are also options regarding how to physically implement something, rather than just whether to implement it or not.

3.5.4 Configurability summary

The LEON2 processor is the most configurable among the three. The other two processor have options for many desirable parameters like cache size, while LEON2 besides the high-level configuration options also can configure more low-level options like replacement strategies and specific options regarding the windowed register file. OpenRISC 1200 is more configurable than MicroBlaze and have options like configurable size of the cache lines and the store buffer, which MicroBlaze is missing.

3.6 Summary

The three processors have been compared on performance, area usage, configurability and usability. The LEON2 processor shows the best performance both in terms of benchmark results and in terms of performance per clock cycle. MicroBlaze has the smallest area for both the core and for the whole system, for all configurations. It also has the best performance per area unit. The small area usage for the MicroBlaze processor origins from a high optimization for Xilinx FPGA circuits, which results in less portability compared to the other two processors.

LEON2 is the most configurable processor with the most advanced cache architecture, while MicroBlaze and OpenRISC 1200 have less configuration options and more simple cache architectures.

MicroBlaze has the most extensive documentation, both for the processor and for available IP-blocks. The documentation for LEON2 is adequate, while the documentation for OpenRISC 1200 is insufficient. Both MicroBlaze and LEON2 have qualified support. The support for the OpenRISC 1200 have not been tested in the same extent.

Both LEON2 and MicroBlaze have satisfying configuration tools, which makes configuration easy. The LEON2 configuration tool has help readily available for all configuration options, while the options in MicroBlaze are well documented but more difficult to access. The OpenRISC 1200 configuration is done by manually editing the Verilog files.

HDL simulation and debugging are considered less difficult for LEON2 compared to the other two processors. The MicroBlaze processor does not include simulation models for external memory and simulation is therefore not as trivial as for LEON2. Adding user defined IP-blocks showed to be very simple for MicroBlaze, but required more work for OpenRISC 1200.

4 Discussion

4.1 Obstacles

One of the more difficult parts of this thesis work was to define three configurations. It is not obvious how to choose which parameters to use and how to assure that the resulting configurations are comparable. Finding an optimal configuration for a specific purpose is a non-trivial iterative process which can result in configurations like the 53 MHz and the 67 MHz version of the LEON2 processor. The second showed best result for the Dhrystone 2.1 and the Stanford benchmarks, while the other one showed best result for the typical control application benchmark and yielded better efficiency per clock cycle. In this particular case, the typical control application and the efficiency per clock cycle were considered more important aspects.

Interpreting results is far from trivial. In the case of comparing benchmark results between different processors it is important that the software library routines are similar and use the same amount of optimization. The benchmarks must be examined thorough in order to find specific characteristics which can explain differences in results for different processors. To be able to explain the differences, an extensive knowledge of the internal structure and implementation of the processors is essential.

When comparing results between different processors, care must be taken to assure that the results are comparable, otherwise the benchmark result is useless. This was the case with the software floating-point operations for the MicroBlaze processor, where the software routines did not comply to the IEEE-754 floating-point standard. If the Paranoia program had not been executed, the wrong conclusions regarding the result from the floating-point part of the Stanford benchmark would have been made.

Beside the main scope of this Master's thesis, considerable time was spent on debugging an erroneous execution in the OpenRISC 1200 processor, which resulted in the finding of a bug in the internal bus communication. MicroBlaze was also configured to run software on SDRAM, which was the first choice of external memory. SDRAM was later discarded due to incomplete support from the OpenRISC 1200 processor combined with the internal nature of the target FPGA circuit.

4.2 Future improvements

There are several improvements which can be done to achieve more comparable results. A possible improvement could be to use memory controllers with identical number of latency and delay cycles for all processors. Another improvement could be to get the typical control application to execute properly on the OpenRISC 1200 processor. To do this a working C library must be compiled for the OpenRISC 1200.

An extension of the number of benchmarks could be done to increase the reliability of the results. The development board of choice could had been changed into a development board which is currently unsupported by all processors. This could result in a comparison of the portability for all three processors, including LEON2.

5 Glossary

AHB	Advanced High-speed Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASIC	Application Specific Integrated Circuit
BHB	Branch History Buffer
CVS	Concurrent Versioning System
DDR SDRAM	Double Data Rate SDRAM
DSU	Debug Support Unit
EDIF	Electronic Design Interchange Format
Ethernet MAC	Ethernet Media Access Controller
FFT	Fast Fourier Transform
FIFO	First In First Out
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FSL	Fast Simplex Link
GCC	GNU Compiler Collection
GDB	GNU Debugger
GNU	GNU's Not UNIX
GPR	General Purpose Register
GUI	Graphical User Interface
HDL	Hardware Definition Language
I2C	Inter-IC
IDE	Integrated Development Environment
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
JTAG	Joint Test Access Group
LGPL	Lesser General Public Licence
LMB	Local Memory Bus
LRR	Last Recently Replaced
LRU	Least Recently Used
LUT	LookUp Table
MAC	Multiply and ACcumulate
MMU	Memory Management Unit
OPB	On-chip Peripheral Bus
PCI	Peripheral Component Interconnect
PROM	Programmable ROM

RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	Real-Time Operating System
SDRAM	Synchronous Dynamic RAM
SPARC	Scalable Processor ARChitecture
SRAM	Static RAM
SSRAM	Synchronous SRAM
TCL/Tk	Tool Command Language/Toolkit
TLB	Translation Lookaside Buffer
UART	Universal Asynchronous Receiver-Transmitter
VHDL	Very High Speed Integrated Circuit HDL

6 References

[AW04]: Alan R. Weiss (October 1, 2002), "Dhrystone Benchmark - History, Analysis, "Scores" and Recommendations", EEMBC Certification Labratories, "http://ebenchmarks.com/download/ECLDhrystoneWhitePaper2.pdf"

[DN04a]: Patrick Pelgrims, Tom Tierens and Dries Driessens (2004), "Basic Custom OpenRISC System Hardware Tutorial, Xilinx, Version 1.00", De Nayer Instituut, "http://emsys.denayer.wenk.be/empro/openrisc-HW-tutorial-Xilinx.pdf"

[DN04b]: Patrick Pelgrims, Tom Tierens and Dries Driessens (2003), "Basic Custom OpenRISC system Software Tutorial, Linux, Version 1.00", De Nayer Instituut, "http://emsys.denayer.wenk.be/empro/openrisc-SW-tutorial.pdf"

[GR04]: Gaisler Research (2004), "LEON2 Processor User's Manual, XST edition, version 1.0.24", Gaisler Research, "http://www.gaisler.com/doc/leon2-1.0.24-xst.pdf"

[OC01]: OpenCores (Sep 6, 2001), "OpenRISC 1200 IP Core Specification, Preliminary draft, Rev. 0.7", OpenCores, "http://www.opencores.org/cvsget.cgi/or1k/or1200/doc/or1200_spec.pdf"

[OC04]: OpenCores (August 31, 2004), "OpenRISC 1000 Architecture Manual", OpenCores, "http://www.opencores.org/cvsget.cgi/or1k/docs/openrisc_arch.doc"

[PE03]: Gaisler Research and Pender Electronic Design GmbH (May 8, 2003), "LEON-PCI-XC2V Development Board User Manual, Rev. 1.0", Pender Electronic Design GmbH, "http://www.pender.ch/docs/GR-PCI-XC2V_User_Manual_rev1-0.pdf"

[SPA92]: SPARC International Inc. (1992), "The SPARC Architecture Manual, Version 8, Rev. SAV080SI9308", SPARC International Inc., "http://www.sparc.org/standards/V8.pdf"

[XIL04a]: Xilinx Inc. (June 14, 2004), "MicroBlaze Processor Reference Guide, Embedded Development Kit, Version 6.2", Xilinx Inc., "http://www.xilinx.com/ise/embedded/edk6_2docs/mb_ref_guide.pdf"

[XIL04b]: Xilinx Inc. (June 16, 2004), "Embedded System Tools Guide v3.0, Embedded Development Kit, Version 6.2", Xilinx Inc., "http://www.xilinx.com/ise/embedded/edk6_2docs/est_guide.pdf"

[ORSRC]: OpenCores (2004), The OpenRISC 1200 Source Code available at OpenCores.org

[PASRC]: W. M. Kahan (1995), A C version of Kahan's Floating Point Test 'Paranoia'

[DNWEB]: De Nayer Instituut, "http://emsys.denayer.wenk.be", [accessed: Oct 2004]

[GRWEB]: Gaisler Research, "www.gaisler.com", [accessed: Sept 2004]

[LMWEB]: LEON2 mailing list, "http://groups.yahoo.com/group/leon_sparc/", [accessed: 2004]

[NLWEB]: Red Hat Inc., "http://sources.redhat.com/newlib/", [accessed: Nov 2004]

[OCWEB]: OpenCores, "www.opencores.org", [accessed: Sept 2004]

[PEWEB]: Pender Electronic Design GmbH, "www.pender.ch", [accessed: Sept 2004]

[XILWEB]: Xilinx Inc., "www.xilinx.com", [accessed: Sept 2004]

[PH97]: David A. Patterson and John L. Hennessy (1997), "Computer Organization & Design, second edition", Morgan Kaufmann, ISBN: 1-55860-428-6

7 Index of tables

8 Index of figures

A Information on caches

A.1 Cache overview

The cache contains copies of small memory blocks residing in a lower hierarchy memory. Since the cache is smaller than lower hierarchy memories, accessing the cache can be done faster than accessing the lower hierarchy memories.

A.2 Cache organization

The cache is organized as a number of sets, each consisting of a specified number of cache lines. The number of cache lines in each set is determined by the degree of cache associativity (i.e. the number of ways).

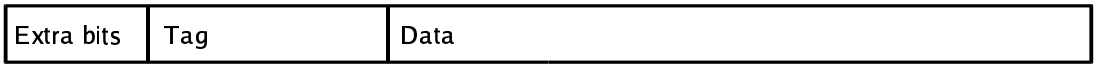

Figure 15: Cache line structure.

Figure 15 visualizes the structure of a cache line, which consists of the following fields:

- Data, a number of sub-blocks.
- Tag, which together with the set index forms a unique identifier for the cache line.
- Extra bits such as:
  - Valid bit(s), specifies whether the cache line (or sub-blocks) is a valid copy of the cache line (or sub-blocks) in the lower hierarchy memory.
  - Replacement algorithm bits. Some replacement algorithms requires additional information bits.
  - Lock bit, which when set prevents the cache line or the individual sub-blocks from being thrown out of the cache.
  - Dirty bit, used in a write-back cache to specify whether or not the cache line needs to be written back to lower hierarchy memory or not. [PH97]

A.3 Cache operation

On a memory access to a specified sub-block in a cacheable memory area, a request for the specified sub-block is sent to the cache. If the sub-block resides in the cache, the sub-block is returned to the requester, a so called cache hit. If the sub-block on the other hand is not in the cache, a cache miss occurs, which means that the cache line containing the sub-block has to be fetched by the cache from a lower hierarchy memory. A replacement algorithm decides

where to place the cache line in the cache, and writes the cache line to that location. If the location is not free, the cache line occupying the location is written back to lower hierarchy memory if needed before being overwritten.

A.4 Cache access

To find a sub-block in the cache, the index field of the sub-block memory address (see Figure 16) is used to address a certain set in the cache. To find the correct cache line in the set, the address tag field is compared with the tag of each cache line in the set. If a tag match occurred and if the cache line valid bit indicates that the cache line is valid, the correct cache line is found. The correct sub-block in the cache line data field (see Figure 15) is then addressed by the address line offset field (see Figure 16).

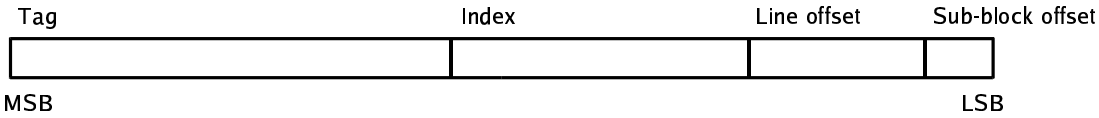

Figure 16: Memory address.

The size in bits of the sub-block field, the line offset field and the index field in Figure 16 is derived from the logarithm of two of the number number of bytes in each sub-block, the number of sub-blocks in each cache line, and the number of sets in the cache. The tag field is calculated according to Formula ii.

If a write miss occurs in the data cache, an implementation can either implement allocate-on-write or not. Allocate-on-write means that the cache line from which the miss occurred is fetched from main memory into the cache.

A.5 Replacement policies

A.5.1 FIFO

The FIFO (First In First Out) algorithm replaces the oldest cache line in a set specified by the address of the to be written sub-block. The FIFO algorithm requires storing extra temporal information bits to each cache line.

A.5.2 LRU

The LRU (Least Recently Used) algorithm replaces the least recently used cache line in a set specified by the address of the to be written sub-block. The LRU algorithm requires storing extra history bits to each cache line.

A.5.3 LRR

The LRR (Last Recently Replaced) algorithm replaces the last recently replaced cache line in a set specified by the address of the to be written sub-block. The LRR algorithm requires extra bits to each cache line.

A.5.4 Random

The Random algorithm replaces a random cache line in a set specified by the address of the to be written sub-block. The replacement is almost always pseudo-random, which means that it is not completely random. The Random algorithm does not require any additional bits.

A.6 Calculating cache size

$$TotalCacheSize = (Extra + Tag + Data)\frac{CacheSize}{LineSize}$$

Formula i: The total size of the cache in bits.

Formula i above calculates the total number of bits in the cache from the following parameters:

- Extra is the number of additional bits added to each cache line, e.g. valid bit, replacement policy bits, lock bits etc.
- Tag is the number of bits in the cache line tag field, see Formula ii below.
- Data is the number of bits of data in each cache line.
- CacheSize is the total number of data bytes in the cache.
- LineSize is the total number of bytes in each cache line.

$$Tag = \log_2 \frac{MemorySize \cdot NumberOfWays}{CacheSize}$$

Formula ii: The size of the tag field in bits.

Formula ii above calculates the size of the tag field in bits from the following parameters:

- MemorySize is the total cacheable address range size in byte. The second logarithm of MemorySize is referred to as address width in Table 4.
- NumberOfWays is the number of ways in the cache, which corresponds to the degree of associativity of the cache.
- CacheSize is the total number of data bytes in the cache.

B Implementation procedure

B.1 General

The main problem with porting a system is to adapt it to the new development board. The FPGA pin configuration must be set correctly to comply with the development board's units. The system must be able to communicate with board components as external memories and interfaces.

The problems specific for the GR-PCI-XC2V development board are the fact that the development board only supports SRAM write accesses of 32-bit words and the fact that the chip select signals to components attached to the data bus must be configured correctly to avoid several units driving the bus simultaneously.

In order to prevent incorrect behavior of the system, which could damage the development board, simulation in Modelsim was used before the actual hardware implementation. The simulation caused additional problems like simulating external memories and UARTs.

B.2 LEON

B.2.1 Generating a working system

Since the LEON2 processor already is ported to the GR-PCI-XC2V development board, only the configuration of the processor was necessary. The configuration was made with the TCL/Tk GUI distributed with the LEON2 processor.

B.2.2 Software applications

Since the DSU takes care of all initialization, no on-chip RAM was necessary. The 24-bit timer in the LEON2 processor uses a 10-bit prescaler. The DSU configures the prescaler to increment the timer every microsecond. For the typical control application, this implementation could cause an overflow in the timer. To resolve this problem, the prescaler was adjusted in software for this application.

B.2.3 Simulation in Modelsim

Simulation in Modelsim is straight-forward since all behavioral models are provided for both UARTs and external memories. The VHDL model was configured for faster UART output, which made simulation of programs with UART output possible.

B.2.4 Synthesis and bitstream generation

Synthesis is done with either Synplicity Synplify Pro or Xilinx XST. Synplicity Synplify Pro yielded the best results and was therefore used for synthesis.

B.2.5 Running on physical hardware

Impact was used to program the FPGA circuit with the generated bit file over the JTAG interface. To be able to run a program in SRAM, GRMON with the DSU back-end was used to initialize the memory and start execution.

B.3 MicroBlaze

B.3.1 Generating a working system

To generate a working MicroBlaze system for the GR-PCI-XC2V development board, a basic system was created with the Xilinx XPS. To address the problem with driving the board chip select signals, a simple IP-core was designed and instantiated in the MHS file describing the toplevel structure of the system. Since the existing SRAM controllers included in Xilinx EDK did not have support for read-modify-write operation when writing single bytes, a new memory controller had to be designed.

The memory controller is attached as a slave on the OPB bus and implements a read-modify-write scheme for single byte writes. The memory controller was instantiated in the MHS file. Since adjustable frequency of the system clock was desired, an IP-block including a Virtex-II DCM was designed and instantiated. The development board RS232 interface was used for standard output.

B.3.2 Software applications

Since the benchmarks will be executed in SRAM, a straight-forward implementation is to put a bootloader in on-chip RAM and download and start execution of the software from the debug software. This implementation requires no initialization in SRAM, since all initialization like enabling caches is done in on-chip RAM.

The bootloader invalidates both caches and enables them. The bootloader finishes with a jump to the lowest address in the SRAM. To verify correctness of connections and timing to SRAM, a simple memory test was executed in on-chip RAM during the implementation phase.

B.3.3 Simulation in Modelsim

Before running a simulation, the libraries Unisim, Simprim and XilinxCoreLib had to be compiled. The compilation was done with the compxlib tool. The MicroBlaze simulation model also had to be compiled, which was done with the compedklib tool.

The simulation model of the system was generated with the Xilinx XPS. A shortcoming is that the simulation model does not include a behavioral model of neither SRAM nor UART. A model for the UART is not vital, since the software can be simulated without output to standard output.

To be able to simulate the system with a SRAM memory model, the memory model and the MicroBlaze system must be instantiated from a toplevel HDL file.

B.3.4 Synthesis and bitstream generation

Synthesis is done with the default synthesis tool in Xilinx XPS, which is the Xilinx XST. When generating a bitstream the system is first synthesized after which it is mapped, placed and routed. A final bitstream is produced and initialized with the program for the on-chip RAM.

B.3.5 Running on physical hardware

Impact was used to program the FPGA circuit with the generated bit file over the JTAG interface. To be able to run a program in SRAM, XMD was used to initialize the memory.

XMD had problems with identifying the physical ethernet device on the JTAG chain when connecting to the MicroBlaze debug target. To resolve this problem, the length of the instruction register of the FPGA circuit and the physical ethernet device had to be specified as shown below.

```
XMD% mbconnect mdm \
-configdevice devicenr 4 irlength 6 \
-configdevice devicenr 5 irlength 16 \
-debugdevice devicenr 4
```

Instructions for debugging with XMD can be found in the Embedded System Tools Guide, [XIL04b].

B.4 OpenRISC 1200

B.4.1 Generating a working system

The ORPSOC reference design was used as an initial design, which was modified to comply with the GR-PCI-XC2V development board. There is a guide for generating a basic working system, which was very helpful, see [DN04a].

There is a bug in the OpenRISC 1200 processor, which leads to bus accesses that do not fully comply to the Wishbone specification. Finding this bug took several days, because it was really difficult to reproduce the error in simulation. Once reproduced in simulation, a workaround was simple to implement and only took a few hours. A workaround is to modify the read acknowledge process in the on-chip RAM implemented by the research group at De Nayer Instituut [DNWEB]. A working version of the read acknowledge process could look like the following lines of code:

```
always @ (posedge wb_clk_i or posedge wb_rst_i)
     begin
        if (wb_rst_i)
          ack_re <= 1'b0;
        else if (wb_cyc_i & wb_stb_i & ~wb_err_o &
                  ~wb_we_i & ~ack_re & (|wb_sel_i[3:0]))
          ack_re <= #1 1'b1;
        else ack_re <= #1 1'b0;
     end
```

The toplevel Verilog file had to be edited to support the GR-PCI-XC2V development board, to address the problem with driving the board chip select signals. Since the existing SRAM controller only had support for read-modify-write operation on a 16-bit basis, a new memory controller with read-modify-write support on a 32-bit basis had to be designed. The memory controller is attached as a slave on the Wishbone bus. The development board RS232 interface was used for standard output.

The OpenRISC 1200 processor only instantiates Virtex primitives. In a Virtex-II the RAMB4 primitive does not exist and is therefore replaced by the RAMB16 primitive, resulting in an approximate loss of RAM bits by 75 percent. This problem was solved by adding support for more efficient use of RAMB16 cells in all components instantiating RAMB4 cells.

B.4.2 Software applications

To execute software, the same procedure as for the MicroBlaze processor was done, see B.3.2 Software applications. Special care had to be taken to make sure that the benchmark stacks were placed in the SRAM.

B.4.3 Simulation in Modelsim

The OpenRISC 1200 system and a behavioral model for the SRAM was instantiated in a toplevel Verilog file. To simulate the on-chip RAM, a behavioral model was written.

B.4.4 Synthesis and bitstream generation

Synthesis is done with the Synplicity Synplify Pro. When generating a bitstream the netlist is mapped, placed and routed. A final bitstream is produced and initialized with the program for the on-chip RAM.

B.4.5 Running on physical hardware

Impact was used to program the FPGA circuit with the generated bit file over the JTAG interface. To debug an OpenRISC 1200 system the JP1 tool was used to provide an interface to GDB. For downloading and executing the software, GDB was used. In order to support the JP1 tool, the system had to be configured with an additional JTAG interface, used only for debugging.

B.5 Discussion

The first intention was to implement all systems with SDRAM as external memory. MicroBlaze, which was the first system implemented, had support for SDRAM and was therefore implemented with SDRAM as external memory. The simulation with SDRAM was rather troublesome, since the behavioral model for the SDRAM did not have support for the Motorola SREC S1 format[46], which the MicroBlaze GCC toolchain program, mb-objdump produced in some cases. A lot of effort was spent on this problem and resulted in support for the S1 format in the memory model. The memory model was also rewritten to support different configurations of row and column sizes.

The memory controller for the OpenRISC 1200 processor required that system bus clock frequency was running at twice the rate of the memory clock frequency. The lowest allowable output frequency of the instantiated DCM was 24 MHz, which required that the system clock had to be running at 48 MHz, which was simply not possible on a Xilinx Virtex-II FPGA with the intended configuration. The two above mentioned problems rendered the use of SDRAM impossible for this configuration of OpenRISC 1200. Therefore SRAM was used instead. To be able to compare the results between the OpenRISC 1200 and the MicroBlaze processor, the MicroBlaze processor was reconfigured to use SRAM instead.

46 http://www.amelek.gda.pl/avr/uisp/srecord.htm

C Paranoia

C.1 About

Paranoia, or Kahan's floating-point test program, is a small program to test the compliance with the IEEE-754 floating-point standard. The original version was written in BASIC by W. M. Kahan in 1983.

In the test of the processors, version 1.1 of the Paranoia program was used, which is written in the C programming language.

C.2 Compilation

The Paranoia floating-point test program was compiled with the NOSIGNAL and BATCHMODE defined. For the single precision tests SINGLE_PRECISION was also defined.

C.3 LEON2

C.3.1 Single precision floating-point

```
Paranoia version 1.1 [cygnus]
...
No failures, defects nor flaws have been discovered.
Rounding appears to conform to the proposed IEEE standard P754,
except for possibly Double Rounding during Gradual Underflow.
The arithmetic diagnosed appears to be Excellent!
END OF TEST.
```

C.3.2 Double precision floating-point

```
Paranoia version 1.1 [cygnus]
...
No failures, defects nor flaws have been discovered.
Rounding appears to conform to the proposed IEEE standard P754,
except for possibly Double Rounding during Gradual Underflow.
The arithmetic diagnosed appears to be Excellent!
END OF TEST.
```

C.4 MicroBlaze

C.4.1 Modifications

The C library function printf in MicroBlaze does not handle floating-point operations properly. To work around this error, the precision used in all printf calls with floating-point numbers as arguments, had to be reduced from seventeen decimals to six decimals. This was done with a search replace in the Paranoia source code. All occurrences of the string “%.17e” was replaced with the string “%.6e”. This could be done with the UNIX command sed:

```
sed "s/\%\.17e/\%\.6e/g" paranoia.c > mb_paranoia.c
```

Some characters in the Paranoia output are garbled and the printf function also fails on some prints resulting in whole sentences and words being left out. This is because the C library printf function uses floating-point operations and MicroBlaze does not handle them properly.

C.4.2 Single precision floating-point

```
Paranoia version 1.1 [cygnus]
...
The number of  FAILUREs  encountered =       1.
The number of  SERIOUS DEFECTs  discovered = 5.
The number of  DEFECTs  discovered =         3.
The number of  FLAWs  discovered =           1.

unacceptable Serious Defects.
 program's subsequent diagnoses.
END OF TEST.
```

C.4.3 Double precision floating-point

```
Paranoia version 1.1 [cygnus]
...
The number of  FAILUREs  encountered =       2.
The number of  SERIOUS DEFECTs  discovered = 3.
The number of  DEFECTs  discovered =         6.
The number of  FLAWs  discovered =           2.

unacceptable Serious Defects.
 program's subsequent diagnoses.
END OF TEST.
```

C.5 OpenRISC 1200

Unfortunately no results could be obtained for the OpenRISC 1200 processor since the absence of a working C library led to erroneous execution.