KTH Information and
Communication Technology

Performance tests of an open source multiprocessor system

Master's Thesis at KTH/ICT
Stockholm, October 2009

GUSTAV KYNNEFJÄLL

Supervisor: Sandro Penolazzi
Examiner: Ingo Sander

TRITA-ICT-EX-2009:164

Abstract

Multicore processors for embedded systems is a growing field, and there is a need for better knowledge about implementation and performance of already existing systems. This study focuses on the RISC processor LEON3 in a single AMBA AHB-bus architecture. It shows how to implement the LEON3 processors and the operating system RTEMS. The performance in the system is measured and the result illustrates what can be expected from this type of system. The results from the experiments show that the cache in the LEON3 processor is a key issue for the system's overall performance. Furthermore, it shows that the choice of algorithm, Round Robin or Fixed Priority, for the bus controller is important to consider during configuration of a multiprocessor system. The performance measurement with the RTEMS shows that high-time resolution affects the performance more than the context switch between different Tasks.

Sammanfattning

Multiprocessorsystem för inbyggda system är ett växande område, och det finns ett behov av ökad kunskap om implementation och prestandan av redan existerande system. Den här studien fokuserar på RISC-processorn LEON3 i en enkel AMBA buss-AHB bussarkitektur. Rapporten visar hur man installerar LEON3-processorn och operativsystemet RTEMS. Prestandan i systemet mäts och resultaten visar vad man bör kunna förvänta sig av den här typen system. Resultaten från experimenten visar att cachen i LEON3-processorn är en nyckelparameter för prestandan, både för den enskilda processoren och för systemet som sådant. Det visas också att de olika algoritmerna, Round Robin eller Fixed Priority, för busskontrollen är viktiga att ta i beaktning vid konfigurering av ett multiprocessor system. Vidare visar prestandamätningen med RTEMS att en för hög tidsupplösning i operativsystemet påverkar prestandan mer än kontext-bytet mellan olika Tasks.

Acknowledgement

I would like to express sincere gratitude to my examiner professor Ingo Sander and my supervisor Sandro Penolazzi. In addition, I wish to thank Daniel Hellström from Aeroflex Gaisler AB and Roger Dahlqvist for their technical support, and Jennifer McConville for her literary support. Special thanks to my parents and sisters for their support during my studies.

Most of all, I would like to thank Karen, my wife, for her support, understanding and love during these years.
Quiero agradecer a mi esposa Karen por la ayuda y el tiempo que dedicó, para que yo pueda avanzar en mis estudios y a mi hija Valeria por ser la alegria en cada momento de mi vida.

Table of Contents

List of Abbreviations

AHB	Advanced High-performance Bus
ALUT	Adaptive LookUp Table
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASIC	Application Specific Integrated Circuit
BSP	Board Support Package
ESA	European Space Agency
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
GPL	General Public License
GRLIB	Gaisler Reusable LIBrary
HDL	Hardware Description Language
IRQMP	Interrupt ReQuest MultiProcessor controller
LUT	LookUp Table
IP	Intellectuel Property
ITRON	Industrial The Real-time Operation Nucleus
JTAG	Joint Test Action Group
USB	Universal Serial Bus
MMU	Memory Management Unit

MPCI	MultiProcessor Communication Interface
NoC	Network on Chip
POSIX	Portable Operating System Interface for uniX
OAR	On-line Applications Research
RCC	RTEMS Cross Compiler
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTEMS	Real Time Executive Multiprocessor Systems
SHM	SHared Memoy driver
SMP	Synchronous MultiProcessing
SPARC	Scalable Processor ARChitecture V.8
UART	Universal Asynchronous Receiver Transmitter
VHDL	VHSIC Hardware Description Language

Chapter 1

Introduction

Performance measurement of different systems has always been of great interest in the struggle to balance performance versus the cost for designs. For computer systems there are several ways to improve the calculation power, i.e. the performance for the system. The most common way to cost-effectively improve the system has been to increase the systems working frequency, since the relation between frequency and performance generally is proportional.

However, there are limits to how high frequency can be used in processor systems. As the working frequency and the quantity of gates increase in the circuit there are growing problems and new limitations are reached. For example, at higher frequency the leakage power loss becomes a factor that has to be taken in account. Therefore, the Cubic rule approximation comes into play at high frequencies. It is shown by this approximation that eight times more power is needed to reduce the processing time by half, this is called the Power Wall. Other issues are the Memory Wall and the Frequency Wall. The Memory Wall is the gap between the processor frequency and the speed which data can be fetched from the memory. The Frequency Wall is determind by the maximum pipeline segment since it is difficult to reduce the segment beyond a certain point [10, p. 6]. Together these walls limit how high frequency can be used in computer systems.

One way to increase the speed of the system beyond these limits is to use more than one processor and move towards multiprocessor systems. Parallel computing and multiprocess systems have been known for decades but have attracted little interest over the last twenty years, apart from the exotic art of developing mainframes. However, since the process for production of circuits has developed a lot, it is now possible to put several processor cores on the same chip and therefore there is a rising interest in the world of designing multicore systems.

In addition to speed improvement, a multiprocessing system under the right conditions can bring significant power savings [17]. Power savings in electronic systems have lately become a hot issue both for environmental politics and embedded systems. The power consumed by computers has become so high that it has reached a significant degree and is now an environmental issue. Embedded systems is a field that takes special interest in performance at low power and therefore also in multicore processors. For the process of designing an embedded system the aspect of time to market is a key issue. Therefore, the knowledge of multiprocessor system characteristics in existing designs is very important and the subject for research in this study. Together, these limits at higher frequencies, development of production technologies and new demands, are driving our world to a paradigm shift, from a single processor world to a multiprocessor world.

1.1 Background

The focus in this study was to investigate how to implement and do performance measurements of an existing and commonly used multicore processor system with a single bus interconnection. Of special interest was the performance for different scheduling algorithms used by the bus controller. The selected system was the LEON3 processor with a single bus architecture and the operating system Real Time Executive Multiprocessor Systems (RTEMS).

The LEON3 processor is widely accepted as a reliable hardware with a minimum amount of bugs. A lot of research has been done with the LEON3 processor and it is part of a living community. Furthermore, The LEON3 processor is well documented and it is relatively easy to find solutions for problems on the Internet. An example of an earlier study with the LEON processor's performance in focus was done by Westlund. Westlund showed a performance comparison between the LEON2 processor and the NIOSII processor [21]. His study indicates which performance can be be expected from the LEON2 processor in a single processor system. However, there is a lack of information about performance measurement with the LEON processor in a multiprocessor environment.

The operating system RTEMS is not as common and widely used. The documentation that is available is well written but more of a specification for development of RTEMS than instructions for a user to develop applications. However, some research has been done on RTEMS as a single processor operating system. For example, Colin and Puaut did measurements and static analysis for worst-case execution in RTEMS [7]. Their work shows expected worst-time for semaphores and Task managing. However, performance measurement for the time resolution in the operating system RTEMS in combination with the overhead for context switch between Tasks is also not available.

The difference between this study and the others is the overall approach and the test of the performance in a complete multiprocessor system. Performance measurement with the LEON3 processor in a multicore environment, with and witout cache will show the effect of the use of single bus architecture for embedded system. This increased knowledge will help to shorten the time to market for embedded systems and enhance their performance at low power.

1.2 Purpose and objective

The general purpose for this study is to enhance the knowledge of an existing multiprocessor systems performance and show how to put such a system into practice.
The specific goals for this study were to:

- Show how to manage the LEON3 processor in a multiprocessor environment.
- Measure Performance of the LEON3 processor in a multiprocessor environment.
  - With and without Cache.
  - With different scheduling algorithms for the AMBA bus controller, i.e. Round Robin and Fixed Priority.
- Investigate how the operating system RTEMS is implemented in a multiprocessor environment.
- Performance measurement with the operating system RTEMS and investigate how the overhead to handle Tasks versus the time resolution affects the overall performance for the system.

This study's measurement results should be interpreted as an indication of performance from this type of system and not an optimization of the system.

Chapter 2

Overview of multiprocessor systems

The idea of multiprocessing is to solve tasks and problems faster and cheaper than with a single processor. It could be expected that doubling the amount of processors would double the performance, but the increase of performance in the new system depends on its ability to be parallelized. Parallelism is simultaneous execution of program instructions on multiple processors. Parallelism of problems is a key issue for multiprocessor system. The ability of the system to be parallelized can in this context be equated to performance and speedup of a system.

The overall speedup of a multiprocessor system can be modelled with Amdahl's Law. Gene Amdahl proposed in 1967, a theoretical model for behavior of performance in multiprocessor systems. This model later became Amdahl's Law [6]. Amdahl's Law expresses the "Law of Diminishing Return" and shows that expected improvement of a system will never be better than the performance of the parts in the system that are not enhanced. As can be seen in the Equation below, no matter how many processors are added to the system, the performance will never be better than 1/F [16, p. 40].

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \dfrac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$= \left\{ \begin{array}{l} F = 1 - Fraction_{enhanced} \\ N = Speedup_{enhanced} \end{array} \right\} = \frac{1}{F + \dfrac{1 - F}{N}}$$

N is equal to the quantity of processors and F the portion of the system that not can be parallelized.

Amdahl's Law is particularly useful for comparing speedup of the overall performance in multiprocessor system. In this study it will be used as a tool to analyze the performance results.

Generally, the performance for a multiprocessor system is less than a linear relation to the quantity of processors. However, there can be cases when the performance is above linear to the quantity of processors, this is called "super linear" performance. For example, a reason for "super linear" performance can be that, when the amount of processors increases, at the same time the total cache in the system also increases so that at some point the whole benchmark will be kept in the cache memory. At this point the scaled up benchmark will indicate "super linear" speedup. However, Amdahl's Law still holds, relative to true speed up, when the benchmark is correctly scaled [16, p. 638].

The question about a system's ability to be parallelized concerns all levels, both hardware, software and type of application. The Figure 2.1 illustrates these different aspects in the process of designing multiprocessor systems. An important part of the design process is defining which kind of problems the system has to solve. Cluster computing proposes four different categories of different problems [19, p. 18].

- Asynchronous Problems.
- Loosely Synchronous Problems.
- Synchronous Problems.
- Embarrassingly Parallel Problems.

This study only solves problems of the type Asynchronous, i.e. different processors solve different problems which have no relation. Figure 2.1 also shows how multiprocessing software can be divided in two main categories: data parallelism and functional parallelism. Data parallelism is when a solution algorithm for a problem fragments to partly independent fractions that can be processed in parallel. The fractions are then distributed equally to all processors in the system to get as high throughput as possible. Functional parallelism is when tasks with different functionality are dedicated to certain processors. Multiprocessing hardware can be divided in several categories. The two main groups are Homogeneous and Heterogeneous multiprocessor system and these two groups are divided between Shared or Distributed memory (Figure 2.1).

Two important concepts within multiprocessing are SMP and message passing. Synchronous MultiProcessing (SMP) is a system where all processors are identical and connected to a shared main memory. All processors can access the whole memory and the workload of tasks is divided equally to

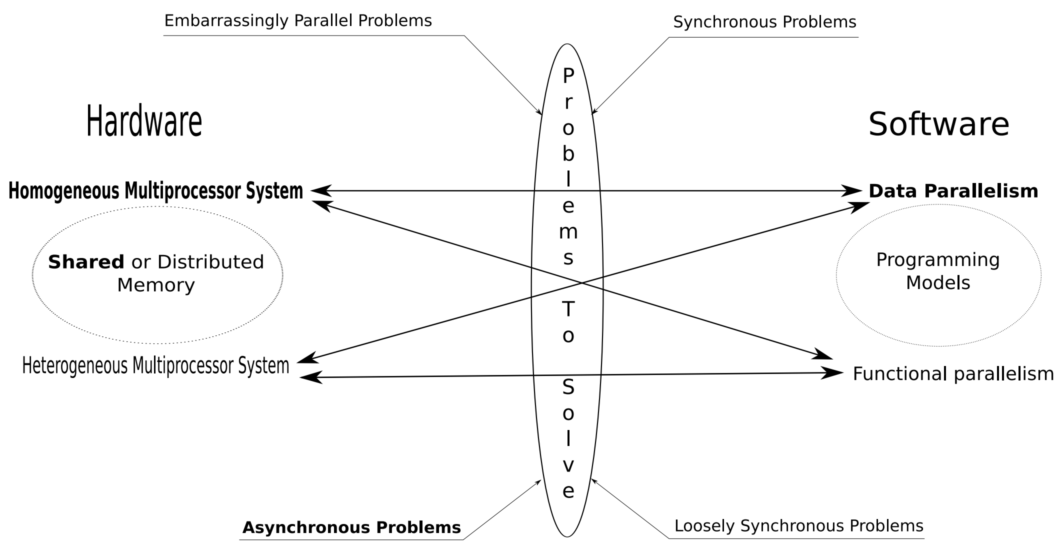

Embarrassingly Parallel Problems
Synchronous Problems
Hardware
Software
Problems To Solve
Homogeneous Multiprocessor System
Data Parallelism
Shared or Distributed Memory
Programming Models
Heterogeneous Multiprocessor System
Functional parallelism
Asynchronous Problems
Loosely Synchronous Problems

Figure 2.1. An outline of the different combinations of hardware, software and four different types of problems.

all processors. Since all processors are identical, the model for the software is simple, it is just a question of how to distribute the workload to different processors. A single task will not be run faster than others but the overall performance for the whole system will be higher. SMP is very common for multiprocessor systems and examples of operating systems that support SMP are Linux, Windows NT, Mac OS and Sun Solaris. The RTEMS does not support SMP but only message passing. The message passing technique for communication between processors gives the operating system a simple and flexible design that is easy to develop. Each processor in the system has its own dedicated memory area. For interaction with other processors sends the application on the processor messages to a common memory that is available for all processors. The hardware for such a system can use Homogeneous or Heterogeneous processors and Shared or Distributed memory.

There are some basic interconnection topologies that are worth mentioning. For this study the most important is the bus architecture. It is very common and used in many designs. The bus architecture has a high throughput of data and a low complexity. However, when several units use the same bus at the same time it can be saturated, since it will be occupied by other units in the system. This is especially the case when using a SMP multiprocessor system with Reduced Instruction Set Computer (RISC) processors. One processor without cache will almost immediately occupy the bus all the time. A way to overcome this problem is to use more than one bus with a bridge between them. However, the communication between different units will be slower than if they were on the same bus. Another drawback for bus architectures is that it is difficult to scale-up to bigger multiprocessor designs.

A development of the bridge strategy is to separate the processor cores with switches instead of bridges. This arrangement would move architecture towards Network on Chip (NoC) designs. NoC interconnections designs have higher complexity than a bus architecture but they are scalable to bigger designs.

This study used a homogeneous multicore system with four LEON3 processors and a shared main memory. The system was configured to solve Asynchronous problems with data Parallelism. The list below summarizes this study's multiprocessor system:

- Homogeneous Multiprocessor System
- Single Bus Architecture
- Shared Memory
- Asynchronous Problems solves with Data Parallelism.

Chapter 3

An open source multiprocessor system

This study focuses on multiprocessor systems with bus architecture and shared main memory. Such a multiprocessor system is supplied by Aeroflex Gaisler AB in an open source library. This Intellectuel Property (IP) library contains reusable components such as the LEON3 processor and the Advanced Microcontroller Bus Architecture (AMBA). In the frame of the library from Aeroflex Gaisler AB, one can configure an open-source multiprocessor system. This study also introduces the operating system RTEMS by On-line Applications Research (OAR) Corporation. The LEON3 processor has also been ported to several other operating systems such as eCos, Snapgear Embedded Linux, ThreadX and LynxOS. However, this study is limited to the operating system RTEMS.

3.1 The LEON3 processor

The LEON processor was developed by Jiri Gaisler as a project in the European Space Agency (ESA). Later, the company Gaisler Research was founded and continued to develop the LEON processor. In 2002, they released the LEON2 processor and in 2004, the LEON3 processor. There is also a fault tolerant version available of the LEON processor, LEON-FT. The LEON3 is a soft processor which means that the design is described in Hardware Description Language (HDL), for example VHSIC Hardware Description Language (VHDL) or Verilog. A design in HDL can easily be changed before it is synthesized to a netlist. The netlist in its turn can be transfered either to an Application Specific Integrated Circuit (ASIC) design, or implemented on a flexible Field Programmable Gate Array (FPGA) circuit. The LEON3 processor is a design that has been implemented both as ASICs and in various FPGAs.

The LEON3 processor design is built on two different standards. The core follows the Scalable Processor ARChitecture V.8 (SPARC) standard from Sun and the bus architecture is constructed according to the AMBA standard from ARM Holdings. Gaisler Research has added some features to the system, such as a plug-and-play functionality for the AMBA bus, but mainly the SPARC and AMBA standards are fully followed.

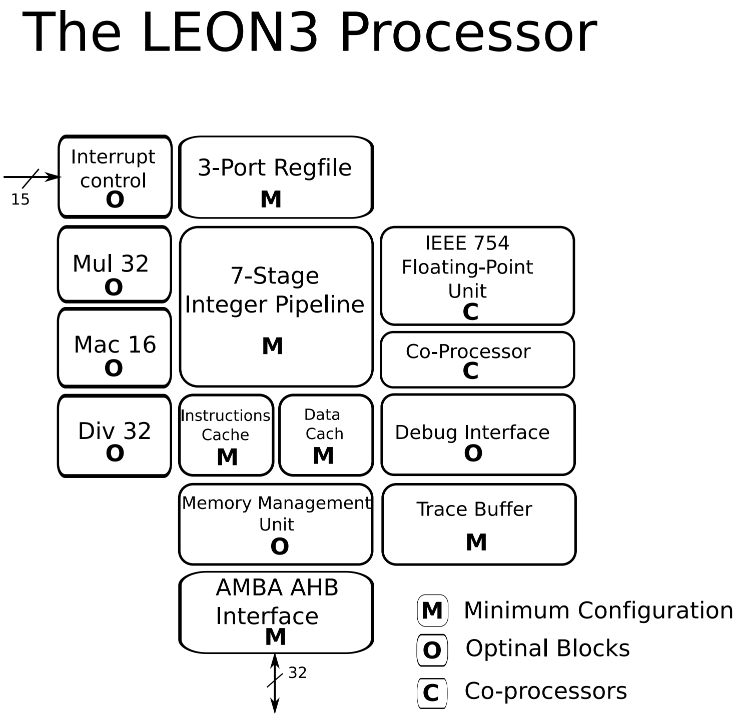


Figure 3.1. An outline of the LEON3 processor.

The LEON3 is a 32-bit processor of RISC architecture, and has a seven-stage pipeline. The cache design is of Harvard architecture, with a separate Data and Instruction cache. The LEON3 processor uses write-through policy and cache snooping to maintain cache coherency. The LEON3 processor stores words in the memory in Big Endian order, i.e. the most significant byte is put at the address with the lowest memory value. As can be seen in Figure 3.1, the LEON3 processor architecture is highly reconfigurable. Parts can be added or taken away and the processor can be configured for a specific application or reconfigured if the conditions for the application change. Figure 3.1 also illustrates that a floating point unit and other co-processors can be added to the LEON3 processor.

The instruction set in LEON3 is similar to an ordinary RISC processor. The time to execute one instruction generally takes one clock cycle, if it is a cache hit. There are some exceptions of instructions that need extra clock cycles to be executed. For example, an integer multiplication can take about one, two, four or 35 clock cycles to execute, which depends on the configuration of the multiplier. The SPARC Atomic Instructions, “ldstub”, “swap” and “cas” are worth a little more attention in a multiprocessor system. The SPARC Atomic load-store instructions take three clock cycles and perform several tasks at once without being interrupted. For example, the “ldstub” instruction copies a byte from the memory into a register, then rewrites the data on the address in the memory, on the same instruction. When a processor executes a SPARC Atomic load-store instruction in a multiprocessor system it is guaranteed that the whole instruction will be carried out without being broken off by other processors. This is especially useful when processes or processors need to block restricted memory areas. For example, if several processors try to access the same semaphore at the same moment, the SPARC Atomic load-store instructions will guarantee that only the first processor gets the semaphore [20, p. 101].

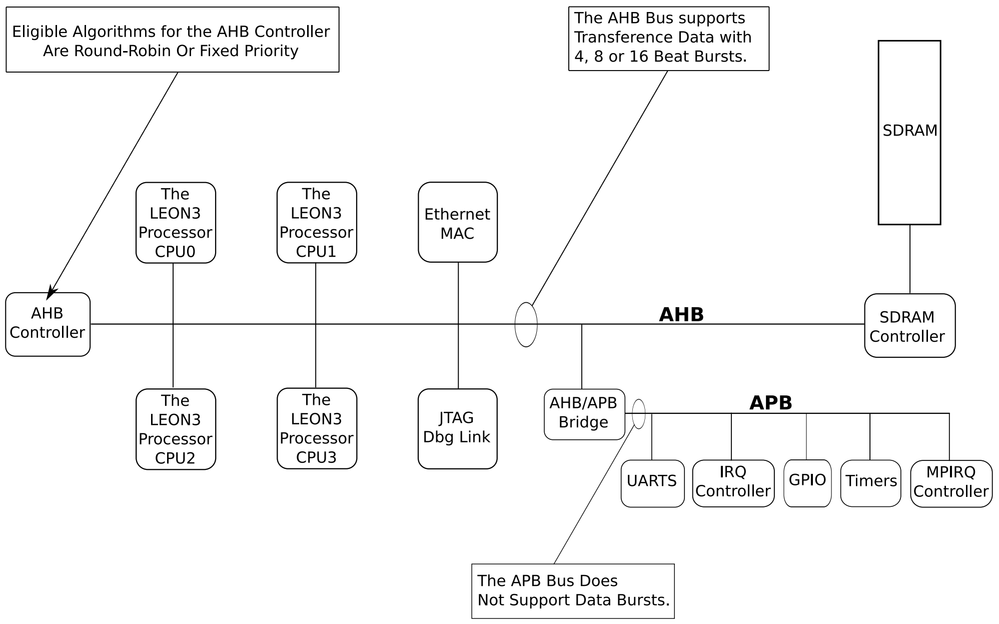


Figure 3.2. An outline of the LEON3 in a multiprocessor environment implemented with the AMBA bus.

The AMBA bus serves as a backbone for the whole system. All parts of the system are connected and communicate through the AMBA bus. As can be seen in Figure 3.2, the AMBA architecture is divided into the Advanced High-performance Bus (AHB) and the Advanced Peripheral Bus (APB). Units that need high speed communications are connected to the Advanced High-performance Bus (AHB); for example processors, Universal Serial Bus (USB), Ethernet and the Joint Test Action Group (JTAG) debug link. The AHB bus supports transfers with four, eight and 16 Beat Bursts. Default burst mode is incremental but can be changed to fixed-length bursts. Units that use low speed communication are connected to the APB bus; for example, Universal Asynchronous Receiver Transmitter (UART), timers and IRQ control registers. The APB bus does not support burst operations [8, p. 1-6]. The units that connect to the bus are divided into Master and Slaves. Masters send requests to the bus controller for access to bus and the bus controller grants access according to an algorithm. Slaves have to send an interrupt to the masters and then wait for a master to respond, they never access the bus themselves. The LEON3 processor is an example of a master unit and a USB is an example of a slave unit. The AHB controller can support up to 16 masters and slaves [9, p. 44]. In order to get access to the bus the LEON3 processor, or another master in the system, must send a request to the AMBA AHB bus controller. The AHB controller has two different algorithms to grant master access to the bus, Round Robin and Fixed Priority. As can be seen in

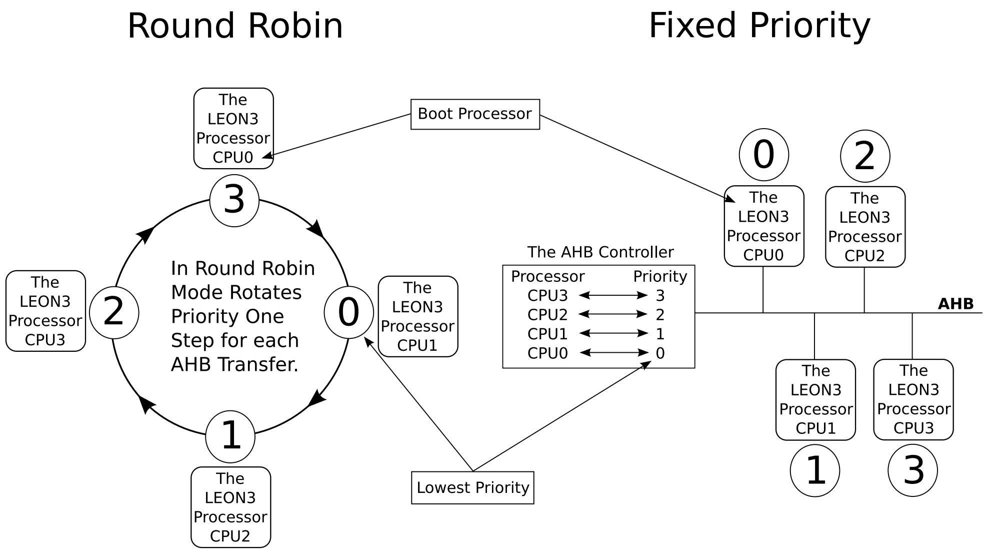


Figure 3.3. A sketch of the Round Robin and the Fixed Priority algorithm.

Figure 3.3, the Fixed Priority algorithm prioritizes masters in the same order as theirs bus index. The boot processor has bus index zero and will hence be given, in Fixed Priority mode, the lowest priority [15, p. 42]. Figure 3.3 also illustrates how the Round Robin algorithm rotates the priority for the masters, one step after each AHB transfer. If no other master sends a request for access to the bus the last master that accessed the bus, will be given the highest priority [9, p. 44].

3.2 The LEON3 in a multiprocessor environment

It is important to be familiar with and understand how to configure a multiprocessor system and its different properties. This study takes an extra interest in the two features that are described below. These features have a direct impact on the system configuration and the test results.

3.2.1 The multiprocessor interrupt controller, IRQMP

The Interrupt ReQuest MultiProcessor controller (IRQMP) is a set of control registers. As can be seen in Figure 3.2, it is connected as a slave to AMBAs APB bus. The IRQMP contains 11 different registers that all have their own purpose. As an example, the status register shows if a processor is halted or running. The status register can also be used to reset, restart and halt a processor.

These registers are configured during the start up process, but can also be reconfigured while the system is running. Initially all processors are halted except for the first processor, the boot processor. The boot processor writes to the status register, which wakes up those other processors that initially were halted [9, p. 526]. The configuration of the IRQMP is a key issue for a multiprocessor system. Therefore IRQMP is of extra interest in this study and it is used in implementation, chapter 4.3.

3.2.2 Cache coherence with the LEON3 processor

The LEON3 processor utilizes write-through policy and cache snooping in order to maintain cache coherence in a multiprocessor system. With the write-through policy the processor writes both to the D-cache and the main memory at the same time. Hence the main memory will always contain the correct data. The write-through policy will slowdown the system a little, but it has a high reliability [9, p. 533].

Data Cache snooping insures that all data in the cache is marked valid or invalid. If the same part of the memory is used by several processors simultaneously, cache snooping will maintain that when a processor writes to that cache line, it will be marked as invalid for all processors [9, p. 545].

Together the write-through policy and cache snooping secure the cache coherence. The write-through policy keeps the main memory up to date and cache snooping transfers the information to all other processors when a cache line becomes invalid. If a processor then tries to read from an invalid cache line it will be forced to fetch the correct data from the main memory. The effect of cache snooping is important to understand in this study, because it will affect the result for some of the tests.

3.3 The GRLIB - a library of reusable IP components

The GRLIB is an open-source code library with reusable IP components. It is used to construct on-chip designs with the LEON3 processor and other peripheral equipments. The configured design can be assembled to a working processor system within minutes.

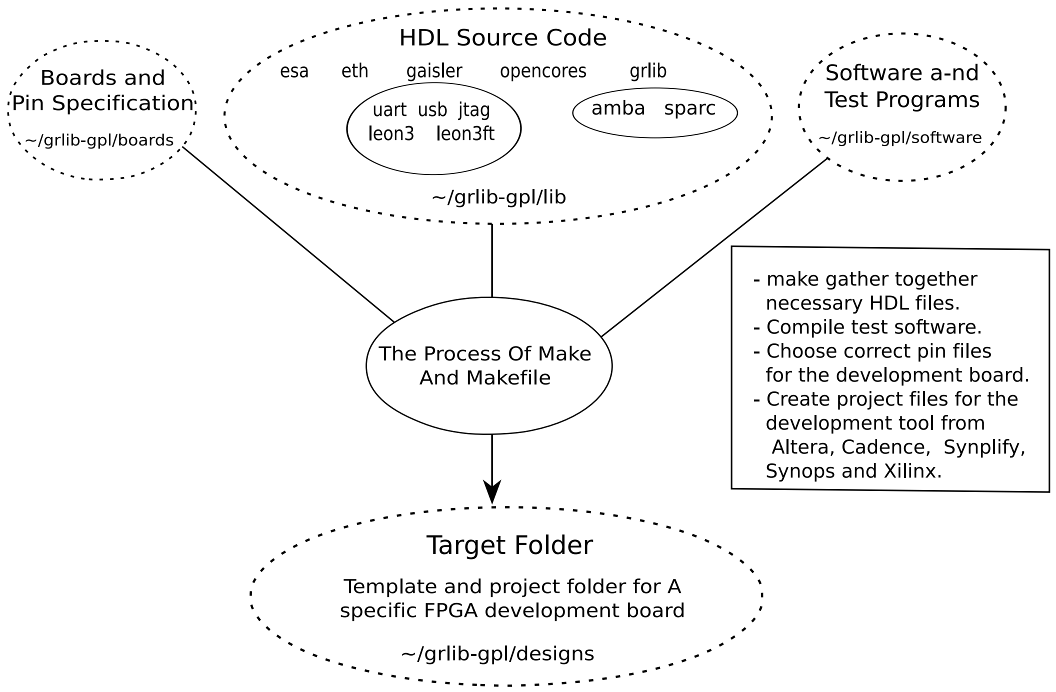

HDL Source Code
esa
eth
gaisler
opencores
grlib
uart usb jtag
leon3 leon3ft
amba sparc
~/grlib-gpl/lib
Boards and Pin Specification
~/grlib-gpl/boards
Software a-nd Test Programs
~/grlib-gpl/software
The Process Of Make And Makefile
- make gather together necessary HDL files.
- Compile test software.
- Choose correct pin files for the development board.
- Create project files for the development tool from Altera, Cadence, Synplify, Synops and Xilinx.
Target Folder
Template and project folder for A specific FPGA development board
~/grlib-gpl/designs

Figure 3.4. An outline of some of the library in GRLIB and the process of make and Makefiles.

The GRLIB uses the AMBA bus as a backbone and then attaches different IP components to the bus through well defined interfaces. The AMBA bus architecture is complemented by sideband signals, that assist in the process of automatically adding IP blocks. The IP components in GRLIB are written in HDL and use generic data types. The generic data types allow IP blocks to be resizable and help them fit into the processor design. The LEON3 processor design is configured with these generic data types, which are collected in one file. This feature of automatically adding IP components from the GRLIB library is called "plug and play" [13, p. 6].

The process to construct a LEON3 processor design with the GRLIB depends heavily on the UNIX concepts "make" and "Makefiles". As can be seen in Figure 3.4, the command "make" collects files and information from different folders in a target template folder, which is specific for the desired development board. Simultaneously the HDL code is configured with the generic data types, according to the user configuration. The target template folder will contain a complete design with the source code for the LEON3 processor. The "make" command can also create project files for different development tools. GRLIB support development tools from Altera, Actel, Cadence, GHDL, Lattice, Mentor, Synopsys, Synplify and Xilinx. Support for other tools can easily be added to the GRLIB [13, p. 5].

3.4 The operating system, RTEMS

The operating system RTEMS was initially developed in the late 80's by the U.S. Army Missile Command. The RTEMS was from the beginning a research project, in the military, about Ada in operating systems but in 1994, RTEMS was transfered to the OAR Corporation for further development as an open-source project. The GNU toolset supports multiple program languages and it was used to develop RTEMS in an open-source environment. The GNU toolset is commonly used for development of applications and is a common environment for open source projects. Originally RTEMS was developed in Ada but it was later transfered to C. Therefore, applications in RTEMS can equally be developed in C or Ada. The acronym for this operating system has always been RTEMS but the name itself has changed several times over the years. From the beginning it was "Real-Time Executive for Missile Systems", then it became "Real-Time Executive for Military Systems" and it finally changed to "Real-Time Executive for Multiprocessor Systems" [18].

The RTEMS follows most of the Portable Operating System Interface for uniX (POSIX) standard. The POSIX is the standard for an application-interface to the UNIX operating system. The only part of the standard that the RTEMS does not follow is the shared memory specifications for a multi-processor system. In addition to POSIX, RTEMS also includes the Industrial The Real-time Operation Nucleus (ITRON) standard. The ITRON is an industrial standard for small scale real-time operating embedded systems.

The RTEMS supports many different processors architectures. For exam-

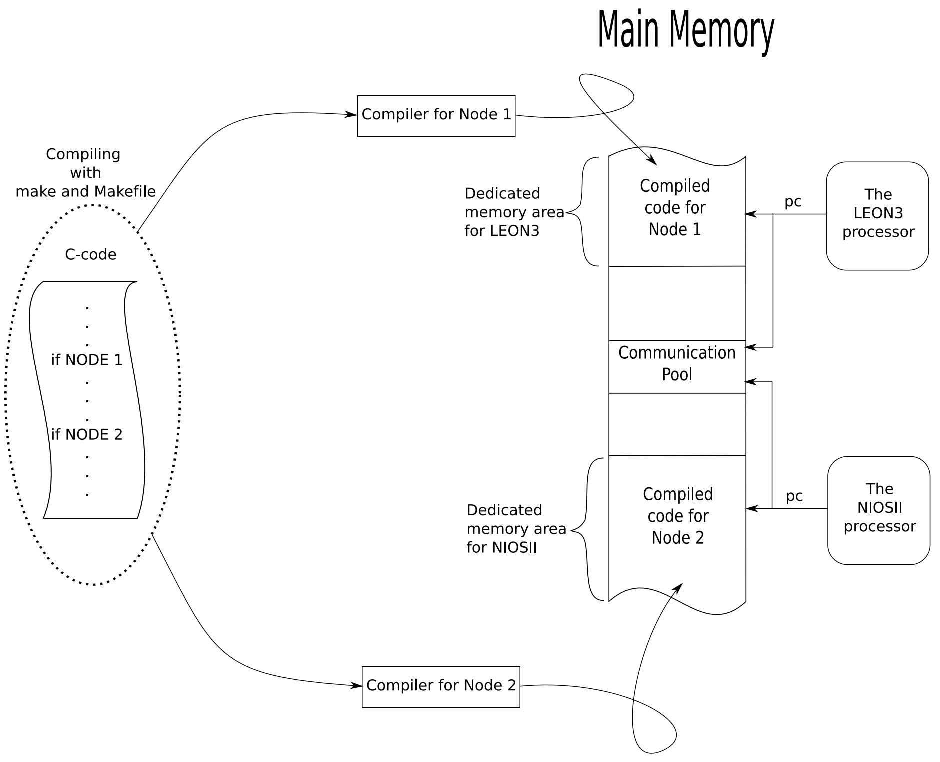


Figure 3.5. An outline of RTEMS with the LEON3 and the NIOSII in memory.

ple, Intel i386, ARM, PowerPC, OpenCores OR32, Unix and SPARC. Its architecture is divided into a general part and a processor specific part. The general part of the source code is hardware independent and the same for all processors. The other part of the system is the Board Support Package (BSP). The BSP contains all the hardware configurations that are required for individual processor's architectures. The process to develop RTEMS is an on-going project and many people contribute with new source codes. These new codes originate from their need to solve problems or develop new features. Therefore, different BSPs have been developed to different degrees. Some features can exist in one BSP and not in others. This study uses the SPARC BSP for the LEON3 processor. The BSP for the LEON3 processor is well developed but there are still many things to improve. The RTEMS source code for LEON3 is regularly updated and patched by Aeroflex Gaisler AB, among others.

The RTEMS supports both homogeneous and heterogeneous multiprocessor environments. This study utilizes four LEON3 processors, which gives a homogeneous multiprocessor system. However, it is possible to use the

RTEMS in a mixed environment with different processors. As can be seen in Figure 3.5, it would be possible to use RTEMS with a design that has both a LEON3 processor and a NIOSII processor. Figure 3.5 also illustrates how applications can be compiled and stored separately in the memory at different places. Generally, processor utilize only their own dedicated memory space and all communication are managed by a communication pool in the main memory. The LEON3 processor and the NIOSII processor should in this example communicate with each other through messages that are sent to the communication pool in the main memory. The LEON3 processor uses big endian and the NIOSII processor little endian but RTEMS solves this issue with its shared memory driver. A good way to summarize RTEMS ability as a multiprocessing system is to say that the RTEMS acts as a single processor operating system with support to communicate with other RTEMS operating systems.

The RTEMS is an operating system that has full multitasking capabilities, not to be confused with multiprocessing. The RTEMS uses the concepts Tasks, Timeslices and priority levels to manage multitasking and real time scheduling. An application is divided into different Tasks that are then handled by the RTEMS scheduling manager. A Task executes during the time for a Timeslice and then the RTEMS Task manager makes a context switch. As can be seen in Figure 3.6, the length of a Timeslice is determined by the number of Ticks and the "time for a Tick" by a real-time definition in microseconds. The length for a TimeSlice is the same for all Tasks in the entire Node [18, p. 66]. The concept Node is a RTEMS definition there Node1 = processor 0, Node2 = processor 1 etc.

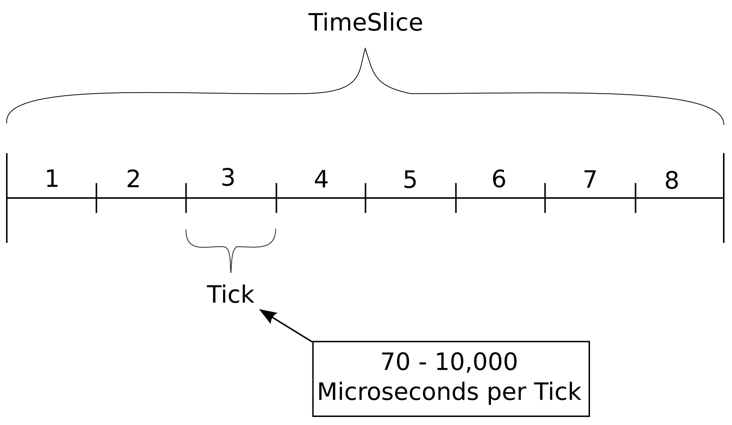


Figure 3.6. An outline of the concept Timeslice in the RTEMS. The length of TimeSlice is the same for all Task in the entire Node.

Figure 3.7 illustrates one type of configuration for Task managing in the RTEMS. It shows how different Tasks are configured to different priority levels. All Tasks are given a priority level from 1 to 255, where Tasks on level one have highest priority. For Tasks on the same priority level, RTEMS uses Round Robin scheduling within their group. However, Task managing in the RTEMS is flexible and can be configured to suit many different applications. As an example, each task can be individually configured to be in preemption mode. A Task in preemption mode can be interrupted if a Task with higher priority is ready to be executed before its Timeslice is finished. The communication and synchronization between tasks are done with messages and signals. The Figure 3.7 also illustrates that a Task can be in one of five different states: Initiating, Ready, Executing, Blocked and Deleted.

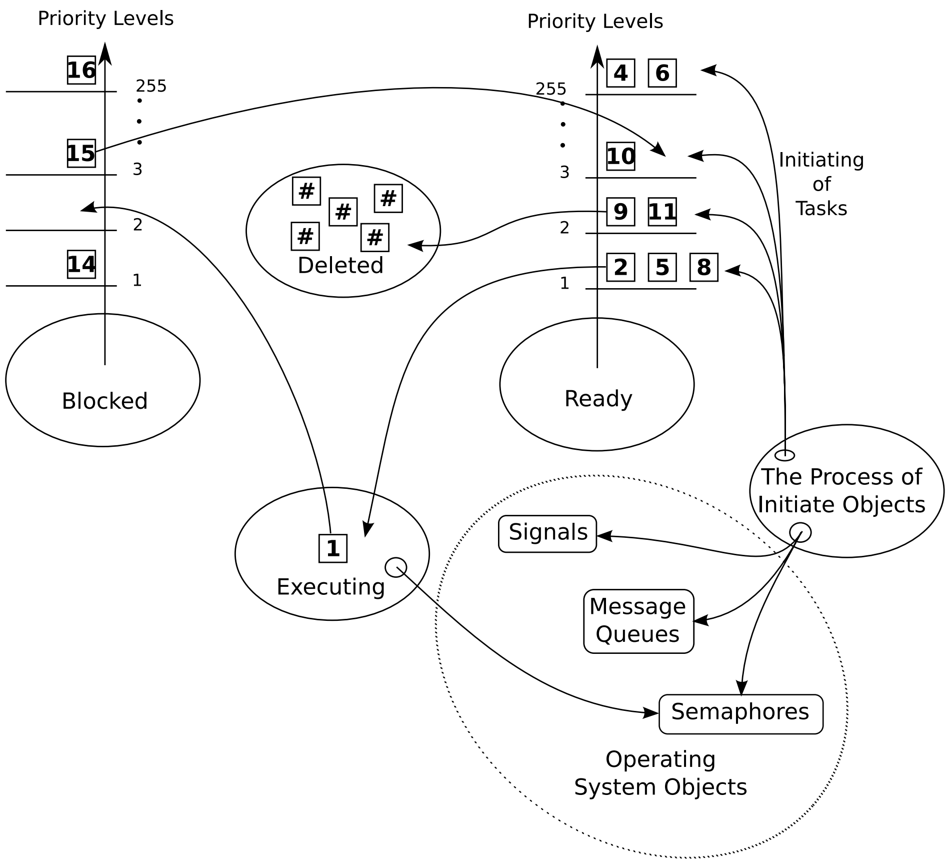


Figure 3.7. An outline of how the RTEMS schedule Tasks.

Furthermore, the RTEMS utilizes both event-driven and priority-based preemptive scheduling to schedule tasks. The priority inheritance avoids priority inversion. This feature will allow low priority Tasks to be completed before the resource is given to a task with higher priority. Furthermore, the RTEMS has the ability to read and write to external storing devices of many different standards, for example FAT32, FAT16 and FAT12. For integration to the Internet the RTEMS has been ported to the TCP/IP stack from FreeBSD and the most common client and server services are available [2].

For communication between different Nodes, the RTEMS utilizes messages passing to a main memory that is shared by all processors. For example, as can be seen in Figure 3.4, Node1 places a message in the communication pool and Node2 will be notified and collect the message. For a Node to get access

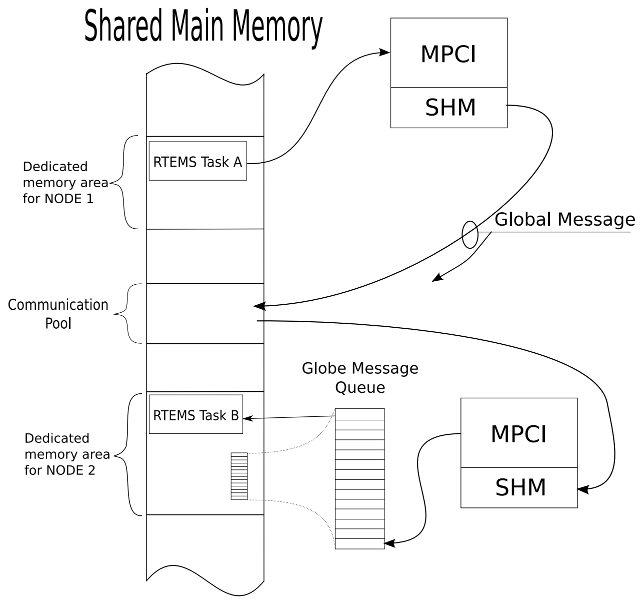


Figure 3.8. An outline of messages passing between different Nodes in RTEMS.

to other Node's system the resources such semaphores, message queues and Tasks, the resources must be declared as global objects. For example, if a Task is declared global, then it is possible for other Nodes to stop or start that Task. As also can be seen in Figure 3.4, the RTEMS sends global messages with the MultiProcessor Communication Interface (MPCI) and SHared Memoy driver (SHM) to the communication pool.

3.5 License for the LEON3 processor and the RTEMS

As can be seen in Figure 3.9, both the RTEMS and the LEON3 processor are within the General Public License (GPL). They are free to download, copy and distribute with the GPL license for further development. However, whenever selling or distributing a product developed originally from a source code within a GPL license, the improved source code ought to be within GPL license [1]. The advantage with the GPL license is that the overhead cost for a project is very low and it does not depend on a single company to supply the source code. The drawback is that open-source code is more difficult to use and therefore the user has to pay for the know-how, through competent staff or some kind of service support.

The IP library GRLIB from Aeroflex Gaisler is mainly within GPL license apart from some components such as the floatingpoint unit. However, Aeroflex Gaisler also offers a commercial license, in case someone wants to conceal their design. Development tools from Aeroflex Gaisler are available for download and evaluation, but are not within GPL license.

The RTEMS source code is principally within GPL license but parts like the TCP/IP stack are licensed to Digital Equipment Corporation1, RPC/XDR is licensed to Sun Microsystem and the Webserver is licensed to Go Ahead Software.

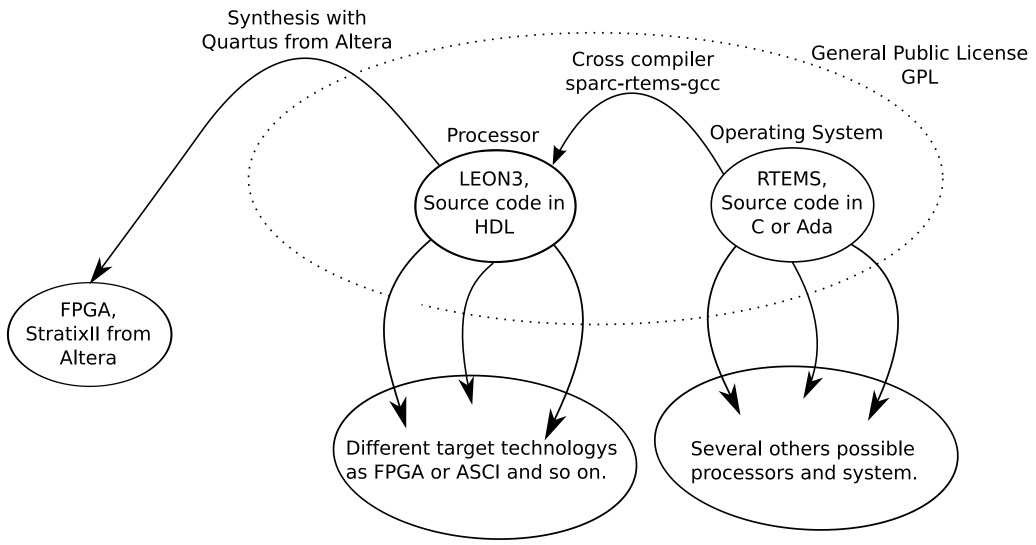


Figure 3.9. A schematic outline of the system and which parts are within GPL.

Chapter 4

Implementation of the LEON3 and the RTEMS

The difficulty of implementation lies within managing many documents and files at the same time. Figure 4.1 shows that the system configuration used in this study is a chain of four different building blocks. These blocks are of different abstraction levels and are merged together to a complete multi-processor system. At the bottom level is the development board from Altera with an FPGA circuit. Next comes the hardware level, which is described in HDL and transfered as a netlist to the Altera's development board. In the top levels are the operating system RTEMS with some test applications. For each

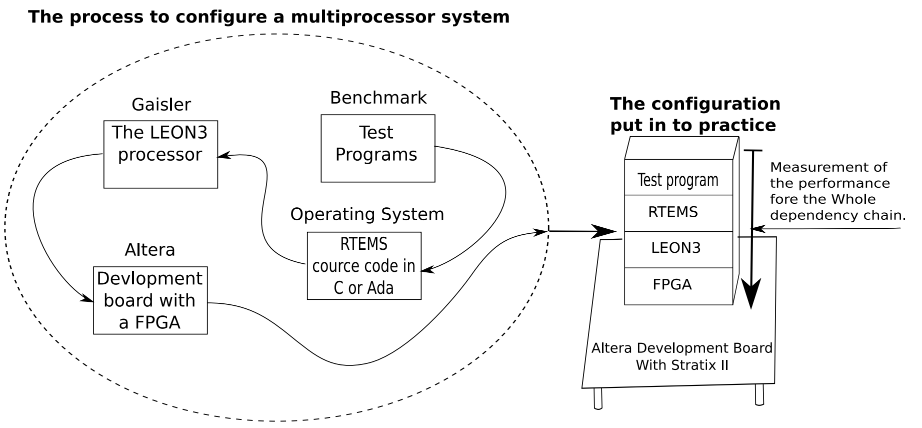

The process to configure a multiprocessor system
Gaisler
The LEON3 processor
Benchmark
Test Programs
Operating System
RTEMS cource code in C or Ada
Altera
Devlopment board with a FPGA
The configuration put in to practice
Test program
RTEMS
LEON3
FPGA
Measurement of the performance fore the Whole dependency chain.
Altera Development Board With Stratix II

Figure 4.1. The process to configure a multiprocessor system.

part of these blocks there are lots of documents, code files and development programs. The most important documents and files are reviewed in Appendix A.

This study utilized a development board from Altera with a Stratix II FPGA, and therefore it was natural to use the program QuartusII from Altera for the synthesis, “place and route” and to download the netlist to the board. The development environment for the host computer is preferable Linux since the software from Gaisler and the GNU tool chain are totally compatible here. Development in a Windows environment with Cygwin should also work but can be unstable and time consuming. Implementation of the system can be summarized in these steps:

1. Download the HDL source code, the Gaisler Reusable LIBrary (GRLIB), for the LEON3 processor from Aeroflex Gaisler AB.

2. Select correct template folder in GRLIB and configure the config.h file with a text editor or through the graphical interface Xconfig.

3. Use the “make and Makefile” to automatically gather, configure and compile the necessary files for the selected FPGA template folder.

4. Download the netlist for the LEON3 multiprocessor system to the FPGA through the Altera USB blaster.

5. Compile the application program with the RTEMS or the SPARC compiler.

6. Download and start the program with Gaisler’s target interface program GRMON.

The Aeroflex Gaisler web page is generally the best download source for files and documentation. This is special true for the RTEMS since the versions from Gaisler have some special patches for the LEON3 processor.

4.1 Altera’s StratixII development board and LEON3

The connection between the LEON3 processor and the development board is its template folder in the GRLIB. The process to install the LEON3 processor is very easy and fast if there exists a template for the specific FPGA in the GRLIB. If the FPGA number can be found in the file “grlib/boards/specific-folder/Makefile.inc”, confirms that the template folder belongs together with the FPGA. The GRLIB contain 46 templates for different development boards. However, if there does not exist any template in GRLIB for the proposed FPGA, then the LEON3 processor has to be ported to the FPGA.

The development board used in this study was a NIOSII development board from Altera with a StratixII FPGA. As it can be seen in Figure 4.2, the board contains a setup of different types of memories, SDRAM, SRAM and a flash memory. Note also the circuit U3 in Figure 4.2, this circuit is responsible

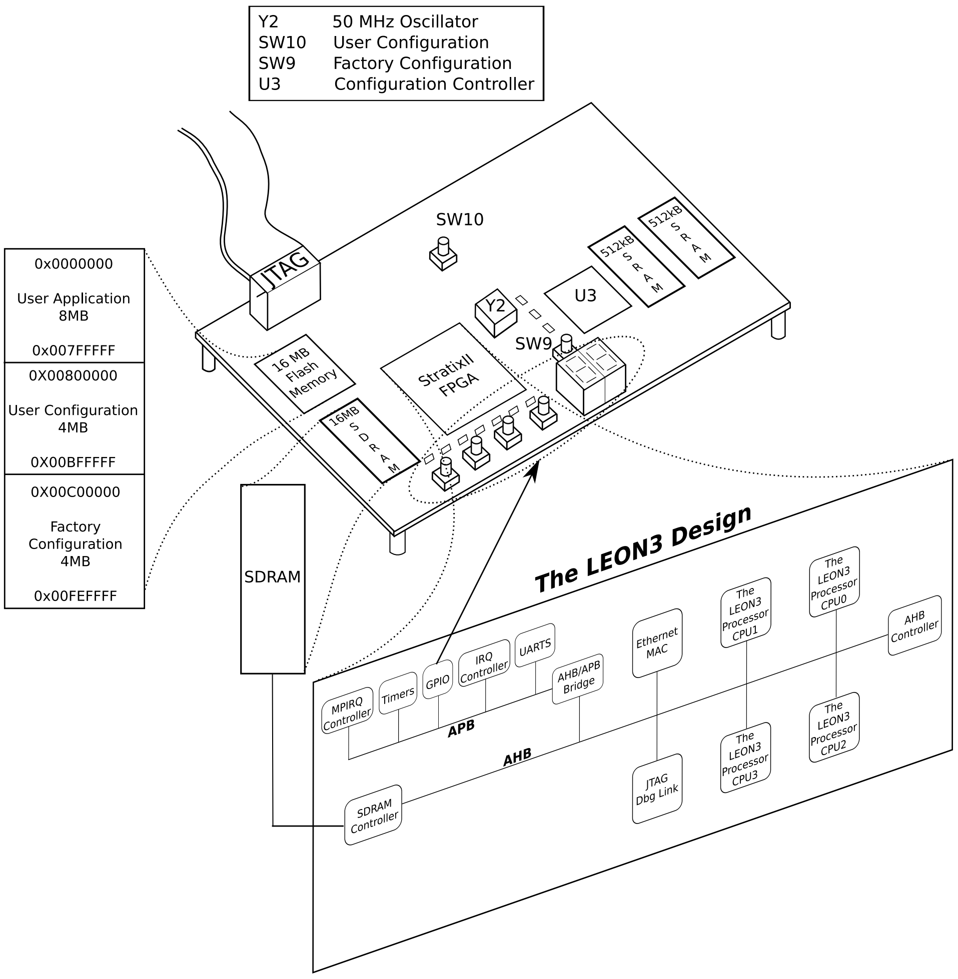


Figure 4.2. An outline of Altera's StratixII development board with the LEON3 processor.

for the process of programming the FPGA from the flash memory. The circuit is connected to the FPGA so that the design itself can trigger a reinstallation from the flash memory. Figure 4.2 also illustrates that the flash memory contains two areas. The memory area, from address 0x00C00000 contains the default factory configuration with a netlist of the NIOSII processor. The design with the NIOSII processor contains features that make it possible to download an user design to the flash memory [5]. In this study it was done through the NIOSII shell with the two command:

1. sof2flash -offset=0x00800000 -input=leon3mp.sof -output=leon3mp.flash

2. nios2-flash-programmer -base=0x00000000 -program leon3mp.flash

With a push of the button SW9, the circuit U3 installs the netlist that is present at address 0x00C00000, and with the button SW10 the netlist that is present at address 0x00800000 is installed into the FPGA. Furthermore, when the board is started the netlist from address 0x00800000 is installed. Hence, it is very convenient to download the netlist of the LEON3 processor to the flash memory at address 0x00800000. With this configuration for the development board, either the LEON3 processor can be installed with a push of button SW9 or the NIOSII processor with button SW10.

The Figure 4.2 also illustrates eight light diodes, D0-D7, and four push buttons, SW0-SW3. In the LEON3 design these are connected to the General Purpose Input Output (GPIO) ports at the address 0x80000500. However, all addresses in the LEON3 design are placed relative to a base address which depend on the HDL codes generic data types and the specific development board. The GRMON is the best source to see how addresses are set in a specific LEON3 design. An alternative way to get the information about the system and its address spaces is to read the HDL code in the template folder.

4.2 Configuration, synthesis and downloading LEON3

The process to configure, synthesize and download the LEON3 processor to the FPGA is easy to perform. Only keep in mind that the installation process totally depends on the “Makefile” concept. For example, this study uses the folder “designs/leon3-altera-ep2s60-sdr”. In the folder’s name it is clearly shown that it is for the ep2s60 FPGA with SDR RAM from Altera. The installation is done through the Terminal in the template folder with three small commands [13, p. 32].

1. “make xconfig”, Configures the design.

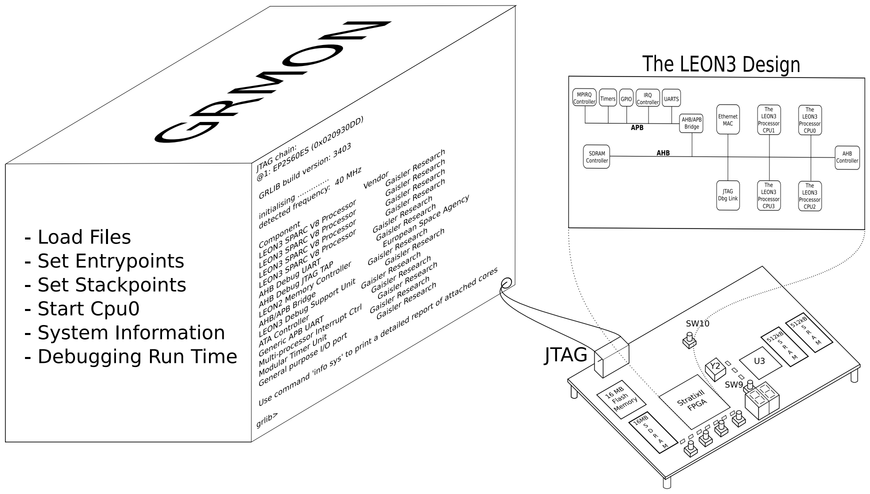
2. “make quartus“ or “make quartus-launch“, synthesizes the design.

3. “make quartus-prog-fpga”, downloads the netlist of the design to the FPGA.

The command “make xconfig” starts the graphical interface Xconfig. The Xconfig changes the design parameters in the file config.hdl. However, it can also be done with a text editor. The Command “make quartus“ synthesizes the design in batch mode, and “make quartus-launch“ launches QuartusII so that the synthesis can be started manually.

4.3 The LEON3 processor in a multiprocessor system

The LEON3 processor system is connected to the development computer through the JTAG debug link. The development computer controls the JTAG debug link through the program GRMON. The GRMON is a general debug monitor and links to the LEON3 processor system through Altera’s USB Blaster. Figure 4.3 illustrates that the GRMON can download program files to the development board, start the processor, debug the system at run time and show additional information about the system. Configuration of the GRMON



```
JTAG chain:
@1: EP2S60ES (0x020930DD)

GRLIB build version: 3403

initialising ............
detected frequency:  40 MHz

Component                         Vendor
LEON3 SPARC V8 Processor          Gaisler Research
LEON3 SPARC V8 Processor          Gaisler Research
LEON3 SPARC V8 Processor          Gaisler Research
LEON3 SPARC V8 Processor          Gaisler Research
AHB Debug UART                    Gaisler Research
AHB Debug JTAG TAP                Gaisler Research
LEON2 Memory Controller           European Space Agency
AHB/APB Bridge                    Gaisler Research
LEON3 Debug Support Unit          Gaisler Research
ATA Controller                    Gaisler Research
Generic APB UART                  Gaisler Research
Multi-processor Interrupt Ctrl    Gaisler Research
Modular Timer Unit                Gaisler Research
General purpose I/O port          Gaisler Research

Use command 'info sys' to print a detailed report of attached cores

grlib>
```

Figure 4.3. An outline of the GRMON connected to Altera’s development board whose FPGA contains a netlist of a multiprocessor system with LEON3 processors.

is done with different flags during the start-up of the GRMON. This study initiated the GRMON with the command “grmon-eval -altjtag -u -nosram”. The “-altjtag” flag is used for the connection to Altera’s “USB Blaster rev.B”. The “-u” flag is used to put UART1 in a loop-back mode to the JTAG. Any output to the UART1 for the monitor, loops back through the JTAG to the GRMON terminal window. The “-nosram” flag will exclude the one MB SRAM and put

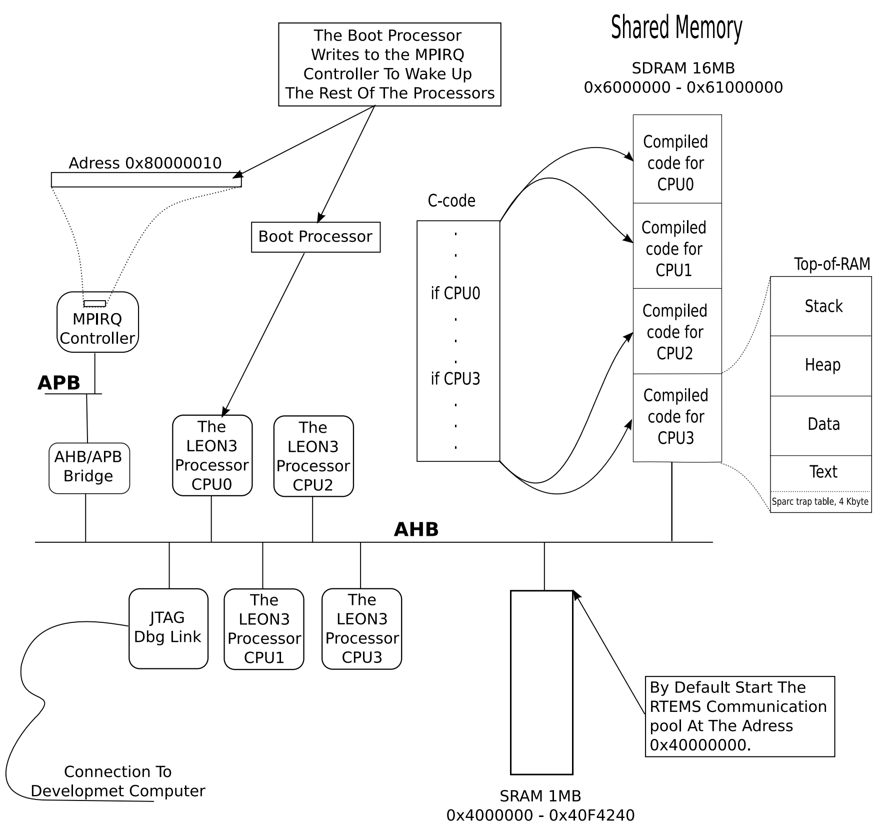


Figure 4.4. An outline of the LEON3 processor in a multicore environment.

the 16 MB SDRAM at the address 0x40000000, which is normally the default download and start address for the GRMON. Programs are downloaded to the board with the command “load file”and then the command “run” in the GRMON is used. However, for a multiprocessor system there are several more steps to go through, which are listed below:

1. Configure the address for the text field compile time with the flag “-Ttext=0x60100000”.
2. Download the files to the development board with the “load file” command in GRMON.
3. Separately configure each processor’s entry point and stack address. In GRMON one uses the commands “cpu act 0”, “ep 0x60400000” and “stack 0x607fff00”.
4. Start the boot processor with the “run” command in GRMON. The boot processor will then initiate the rest of the system.

Fortunately, there exists a “batch file” command in the GRMON, which makes the operation simpler. An example of such a batch file can be seen in Appendix C.1. Figure 4.4 illustrates how a multiprocessor system configures. The batch file compiles the application from the same source code, downloads the files to the memory, configures processors and initiates CPU0. Then the boot processor writes to Status register in the IRQMP to start up the rest of the system [9, p. 526]. Figure 4.4 illustrates how one can put the source code for different processors in the same file. In a multiprocessor system this gives a good overview of the application. The code for different processors is separated with ordinary C-code or hash code. Figure 4.5 and Appendix C.1 illustrate these two styles. The differences between these two styles are

Hash Code

```
#if NODE_NUMBER == 1
        //Code for Processor 1
#endif
```

Ordinary C-code

```
if( NODE_NUMBER == 1 )
{
        //Code for Processor 1
}
```

Figure 4.5. An example of Hash code versus C-code.

that the precompiler uses hash code to configure the application compile time, while the configuration of the application is static. The C-code on the other hand is dynamic and the application can be reconfigured at run time.

4.4 The union of the LEON3 and the RTEMS

The RTEMS and the LEON3 processor work well together and a first application “rtems-hello.c”, is easy to compile, download and run with the GRMON. By default runs and compiles the RTEMS applications to minimum space, and extra features can be added to RTEMS with macros in the C-code. However, some parts of the RTEMS are enabled or disabled for the operating system when it builds from the source code. For example the POSIX, networking, board support pack and multiprocessing need to be configured when the RTEMS is built from the source code. The process to build the RTEMS for the LEON3 processor can be done in two ways, completely from basic with a new tool chain or with the tool chain within the RTEMS supplied by Aeroflex Gaisler. The process to build and compile the RTEMS from basic needs a new GNU tool chain to be constructed from the beginning. With this new tool chain the RTEMS can be recompiled. The tools needed for the process are listed below:

- Autoconf, producing configure scripts.
- Automake, producing portable makefiles.
- GNU m4, General purpose macro processor.
- gcc-core and gcc-g++, GNU Compiler Collection.
- Binutils, binary untiles.
- Newlib, collection of standard C library for embedded systems.

The above tools must be installed with patches in the right order and right versions. The process to build the GNU tool chain can be time consuming and troublesome. A second and easier way is to use the existing GNU tool chain in the RTEMS that is supplied from Aeroflex Gaisler. The only requirement is to link or make a copy of the RTEMS source code in the "rtems/src" folder. The RTEMS is then recompiled from the terminal in the "rtems/src" folder with the command "make all". For example, in this study multiprocessing was enabled through recompiling RTEMS with the tool chain supplied from Aeroflex Gaisler.

Chapter 5

Measurement of performance

The performance can be measured in many different aspects. As an example, calculation of scientific problems or data management can give completely different results. Other things that also affect performance are the type of system and how the hardware is designed, etc. The performance of this system depends on the whole chain of building blocks, as can be seen in Figure 4.1. Each part can independently be improved and contribute to a better performance. This study uses two different tests to illustrate a multiprocessor system performance:

1. The benchmark Dhrystone. This benchmark is very common, and in some sense, it will be possible to estimate the level of the system in comparison to other people's work.

2. The second test sequence is for monitoring the performance change in a complete multiprocessor system with the operating system RTEMS.

This chapter shows tables containing summary of the test results. For more detailed information see Appendix D.

5.1 The Dhrystone, a common benchmark

The Dhrystone is a common benchmark and is used for measurement of processor performance. The Dhrystone was developed in the middle of the 1980s and originally written in Ada, but later transfered to C. Dhrystone was constructed to be an all-round testbench for general applications. It uses arithmetic integer operations and no floating point calculations. The Dhrystone calculates mean values of Dhrystone iterations per second, [DMIPS], and the result is often normalized with the processor frequency, [DMIPS/MHz].

$$Performance = \frac{Iterations}{Time} \tag{5.1}$$

However, the benchmark Dhrystone is sensitive depending how it is used. For example, with some optimization of the benchmark's C library, Dhrystone can indicate a three times better performance. Another drawback is that Dhrystone spends a lot of time in string library which is not representative for embedded systems. The advantage with the Dhrystone is that it is easy to apply and therefore it is used in many other studies of computer systems [22].

Table 5.1. A summary of the result from measurement with Dhrystone v.1 on four different hardware configurations. All values in the table are in the unit [DMIPS/MHz].

	One processor	Two processors	Three processors	Four processors
	8 kB Data and 8 kB Instruction cache and Round Robin			
Sum	1'037	1'847	2'201	2'346

	One processor	Two processors	Three processors	Four processors
	8 kB Data and 8 kB Instruction cache and Fixed Priority			
Sum	1'037	1'870	2'203	2'503

	One processor	Two processors	Three processors	Four processors
	0 kB Data and 0 kB Instruction cache and Round Robin			
Sum	239	252	259	259

	One processor	Two processors	Three processors	Four processors
	0 kB Data and 0 kB Instruction cache and Fixed Priority			
Sum	239	256	244	-

This study used four different hardware configurations. Generally, the systems are very much alike, and only two different parameters in the hardware were changed, the cache and the algorithm that gives masters access to the bus. The algorithm on the AHB controller alternated between Round Robin and Fixed Priority. The cache changed between no cache and 8kB, both for Instruction and Data cache. The combination of these parameters led to four different hardware configurations. During these tests, Dhrystone was compiled with leon3-elf-gcc compiler and the flags "O2" and "msoft-float". These flags tells the compiler to use maximum optimization and soft floating point emulation when required. Dhrystone ran 400'000 iterations and 16 times to calculate the mean and standard deviation values. Table 5.1 shows the sum of performance for the systems.

5.2 Performance with RTEMS and the LEON3 processor

The measurement with the RTEMS and the LEON3 processor was done in four steps.

1. Reference measurement without RTEMS.
2. Measurement with RTEMS, Ticks per TimeSlice versus microseconds per Tick.
3. Measurement with RTEMS, the Timeslice versus the number of Tasks.
4. Measurement with RTEMS in a multiprocessor environment, with one Task and a fixed length for the Timeslice.

Since it was problem to implement the benchmark Dhrystone in RTEMS uses a simple Bubble sort algorithm as workload at the performance tests with RTEMS. However, the approach for the measurement was the same as that uses by the benchmark Dhrystone. The measurement was done in iterations per second for sorting an array. Equation 5.2 shows how the number of iterations was normalized with the measured time.

$$Performance = \frac{IterationsOfBubbleSort}{Time} \tag{5.2}$$

The first reference measurement was done without the RTEMS and with the "sparc-elf-gcc" compiler. These measurements were used as a reference to the measurement with the RTEMS. The flags that were used for the compiler were the "O2" and the "msoft-float" flag. The time measurement was done with the system clock where the scale value was set to 40 clock cycles for one tick. The test was performed 16 times to calculate the mean and the standard deviation values. Table 5.2 shows a summary of the results from the measurement with two different hardware configurations without cache. The values in Table 5.2 give a reference to what could be expected as best performance in the tests sequence with the RTEMS.

Table 5.2. A summary of performance measurement with a simple Bubble sort algorithm. Two different hardware configurations used. All values in the table are in the unit iteration/second.

	One processor	Two processors	Three processors	Four processors
	0 kB Data and 0 kB Instruction cache and Round Robin			
Sum	5’978	6’332	6’412	6’417

	One processor	Two processors	Three processors	Four processors
	0 kB Data and 0 kB Instruction cache and Fixed Priority			
Sum	5’968	6’302	6’273	-

The concept of Tasks is a key concept in the RTEMS for managing multitasking. Therefore it is important to look into how basic features of Tasks affect performance. The second measure sequence compare the number of Ticks for one Timeslice versus the time for one Tick. The measurement was done with one processor and one Task and the number of Ticks. The length of one Tick was kept as a variable parameter. Table 5.3 shows a summary of the values from the measurement. In Table 5.3 one sees that the parameter

Table 5.3. A summary of the performance measurement with the RTEMS between numbers of ticks for one timeslice versus microseconds for one tick. The measurement was done with one processor, Round Robin algorithm for the AMBA bus controller and without cache. The performance was measured with iterations per second of a simple bubblesort algorithm. The ratio is the relationship between the measured value and the highest value in the table.

	Microseconds Per Tick				
	70	100	500	900	1100
TimeSlice	Performance, Iterations Per Second				
1 Tick	220.66	1’940.86	5’162.94	5’520.9	5’602.1
Ratio	3.9%	34.3%	91.3%	97.6%	99.0%
4 Ticks	906.80	2’425.22	5’259.52	5’520.90	5’602.13
Ratio	16.0%	42.9%	93.0%	98.5%	99.8%
10 Ticks	1’044.20	2’521.07	5’279.1	5’582.48	5’652.9
Ratio	18.5%	44.6%	93.3%	98.7%	99.9%
16 Ticks	1’079.05	2’546.11	5’284.02	5’588.4	5’657.3
Ratio	19.1%	45.0%	93.4%	98.8%	100.0%

“Number of Ticks per Timeslice” affects the performance only at low values. Therefore, at the next step keeps that parameter fixed 10 Ticks per Timeslice while the parameter number of Tasks is the variable parameters. In the third

Table 5.4. A summary of the performance measurement with RTEMS between microseconds for one tick versus number of tasks. The measurement was done with one processor, Round Robin algorithm for the AMBA bus controller and without cache. The performance was measured with iterations per second of a simple bubble sort algorithm. The ratio is the relationship between the measured value and the highest value in the table.

	Microseconds Per Tick				
	70	100	500	900	1100
Number Of Tasks	Performance: Iterations Per Second				
One Task	1’124	2’587	5’313	5’508	5’685
Ratio	19.8%	45.5%	93.4%	96.8%	100.0%
Two Tasks	287	1’8670	5’154	5’530	5’608
Ratio	5.0%	32.8%	90.6%	97.2%	98.6%
Four Tasks	308	1’884	5’163	5’541	5’609
Ratio	5.4%	33.1%	90.8%	97.4%	98.6%
Six Tasks	185	1’776	5’146	5’535	5’608
Ratio	3.2%	31.2%	90.5%	97.3%	98.6%
Eight Tasks	252	1’843	5’153	5’546	5’608
Ratio	4.4%	32.4%	90.6%	97.5%	98.6%

test changed the “microseconds for a Tick” and the number of Tasks are made as variable parameters. Table 5.4 shows a summary from the measurement.

The last measurement was done with varying quantities of processors. The time for one Tick and the number of ticks were set high, 10’000 and 10, to avoid affecting the measurement. In Table 5.5, a summary of the values from the measurement is shown.

Table 5.5. A summary of the result from measurement with the RTEMS in a multiprocessor system. All values in the table are in the unit [I/S].

0 kB Data and 0 kB Instruction cache and Round Robin

	One processor	Two processors	Three processors	Four processors
Sum	5’980	6’358	6’416	6’416

Chapter 6

Results and performance analysis

A key concept when analyzing the performance of multicore systems, is the system's ability to be parallelized. A equation that estimates the ability to parallelize is the Amdahl's Law. This study fits Amdahl's equation, through an estimate of F in the equation, to the results from the measurements. F is the portion of the system that not can be parallelized, see chapter 2. With different values of F, the systems enhancement is discussed.

The performance measurement in this study was done with four different multiprocessor systems. Generally, the systems were very much alike, only two parameters in the hardware were changed, the cache and the algorithm that gives bus masters access to the bus. These parameters were changed within the limits of Gaisler Reusable LIBrary (GRLIB). The algorithm on the AHB controller alternates between Round Robin and Fixed Priority. The configurations for the caches were, without Data and Instruction cache, and with 8 kB Data and 8kB Instruction cache. The combination of these two parameters resulted in four different multicore systems. Listed below are the most important hardware system characteristics that are good to keep in mind when going through the results:

- Reduced Instruction Set Computer (RISC) processors need a new instruction every clock cycle.
- The size of Data and Instruction cache in the processor.
- The single bus architecture, that is a bottleneck.
- The bus algorithm for the AHB controller that gives masters access to the bus, the Round Robin or the Fixed Priority algorithm.

This study also included some performance measurement with the operating system RTEMS. An important concept to remember when looking at the figures is the time for a context switch and the resolution of the time in the operating system.

6.1 The result from the QuartusII synthesis

The FPGA in this study was the StratixII EP2S60 from Altera and therefore the synthesis tool Quartus 8.1 from Altera is used. The StratixII is a FPGA with an architecture of different kinds of logical elements, such as Random Access Memory (RAM) blocks, DSP blocks and Adaptive LookUp Table (ALUT). Hence, the StratixII is a very complex design and to get an uncomplicated overview of the results from the synthesis, this study compares only used ALUTs, pins and Memory Bits.

As can be seen in Table 6.1, the synthesis of designs with the QuartusII resulted in four different netlists of different sizes. This analysis uses area optimazion to reduce the size of netlists. An attempt to synthesize a design with five processors cores was also done but the resulting netlist became too big to fit into the StratixII EP2S60. The limit was therefore set at four processor cores. The amount of used space in the synthesis, (Table 6.1), can

Table 6.1. Result from synthesis with the QuartusII 8.1.

	Designs				StratixII
	RR8C	RR0C	FP8C	FP0C	Available
Sum of ALUT:s	25,137	23,022	25,167	23,001	48,352
% Total	52%	48%	52%	48%	100%
ALUT:s for CPU0	5,078	4,561	5,118	4,569	48,352
% Total	11%	9%	11%	9%	100%
Number of Pins	222	222	222	222	493
% Total	45%	45%	45%	45%	100%
Memory Bits	777,992	149,377	745,344	149,376	2,544,192
% Total	30%	6%	29%	6%	100%

be compared with guidelines given in the manual from Gaisler Research [9, p. 8]. According to the manual for Altera Stratix devices, the ALUTs for a CPU core should be roughly the same as for Virtex2 [9, p. 7]. The given value in the manual for a LEON3 processor with 8 kB Data and Instruction cache is 4300 LookUp Tables (LUTs). With the tools used in this work, the processor core LEON3 with 8 kB Data and Instruction cache was synthesized to 6495 ALUTs. According to the manuals, the available ALUTs in StratixII EP2S60 are 48352, and an equivalent LUT-based architecture is 60440 LUTs [4, p. 1-2]. Hence, the rate between an ALUT and a LUT is 1.25. The comparison between the given number, 4300 LUTs, and this study result, 6348 LUTs, shows a big difference that can not be explained by different synthesis tools or level of optimization. This study does not go any further in the analysis of the synthesis.

6.2 Measurement with the benchmark Dhrystone

During the test with the Dhrystone benchmark, the workload was the same for all processors and the benchmark itself was not parallelized. All LEON3 processors worked totally independently of each other and there was no intercommunication between them. The only limit for the processors to reach full execution speed was the single bus architecture and the shared main memory, these two formed a bottleneck and reduced the performance. The sum of performance is an addition of each processor's result and reflects the system's overall performance.

6.2.1 With cache and the Round Robin

As can be seen in Figure 6.1, the overall performance for the system increased when processors are added to the system. The comparison between Amdahl's

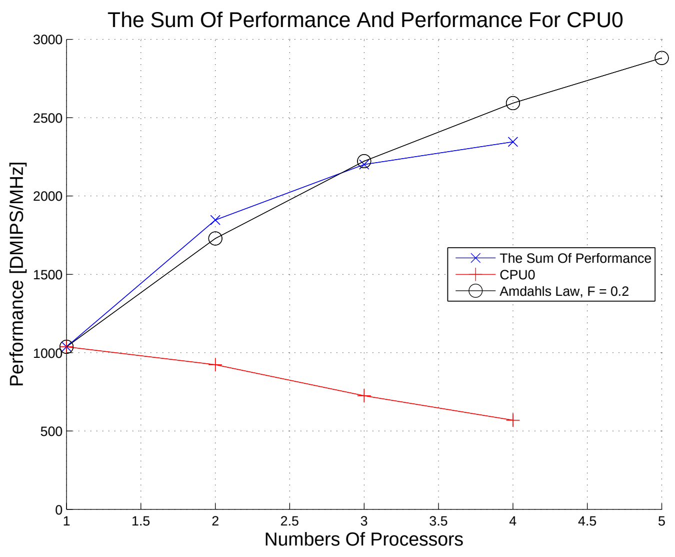


Figure 6.1. Result from measurement with the benchmark Dhrystone, C Version 2.1, in a multiprocessor system with four processors. The hardware configurations were, **with 8kB Data and 8kB Instruction cache and Round Robin algorithm** for the AHB controller.

Law, when F is set to 0.2, and the sum of performance shows that with two processors the parallelization of the system is 20% above Amdahl's law and with four processors it is 20% below. The difference from Amdahl's Law stems from the fact that Amdahl's equation does not take into account different bottlenecks and limitations in the system. In this case, it is the single bus architecture that is the bottleneck. The 8 kB Data and Instruction cache give an increasing capacity to the system's ability to be Parallelized, but the capacity for communication via the bus has a fixed limit. Therefore, the overall performance will increase a lot for the first processors but pass to a flat line at higher numbers of processors. The individual processors performance is equal for all processors and therefore only CPU0 performance is depicted in Figure 6.1.

6.2.2 Without cache and the Round Robin

As can be seen in the Figure 6.2, the overall performance for the system increases only slightly when processors are added to the system. The difference in the overall performance between the use of one or two processors is less than 20%. The Figure 6.2 shows that the F in Amdahl's Law has been set

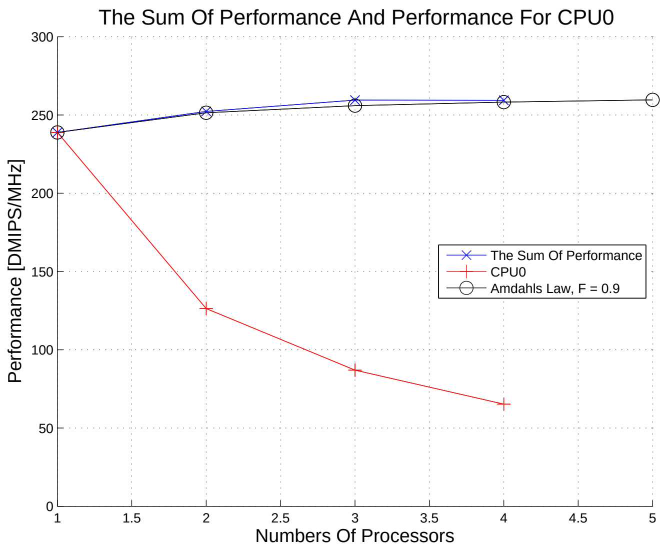


Figure 6.2. Result from measurement with the benchmark Dhrystone, C Version 2.1, in a multiprocessor system with four processors. The hardware configurations were, **without Data and Instruction cache and Round Robin algorithm** for the AHB controller.

at 0.9. The curve for the Amdahl's fits well to the curve for the overall performance. The parameter F shows that only 10% of the system has the ability to be parallelized. The weak performance increase is due to the single bus architecture and the use of RISC processors that do not have data or instruction cache. The bus will be occupied all the time when one processor uses the bus, the others will stand by and wait.

6.2.3 With cache and Fixed Priority

Figure 6.3 illustrates how the overall performance increases significantly when more processor cores are added to the system. Amdahl's Law fits well to the systems performance, where F is set to 0.2, which indicate that the system is enhanced approximately 80%. All four processor's performance is shown in Figure 6.3. The effect of the Fixed Priority algorithm on the AHB controller can be seen clearly at the use of four processor cores. As can be seen in the figure 6.3, the individual performance results at four processors are clearly separated but well grouped together.

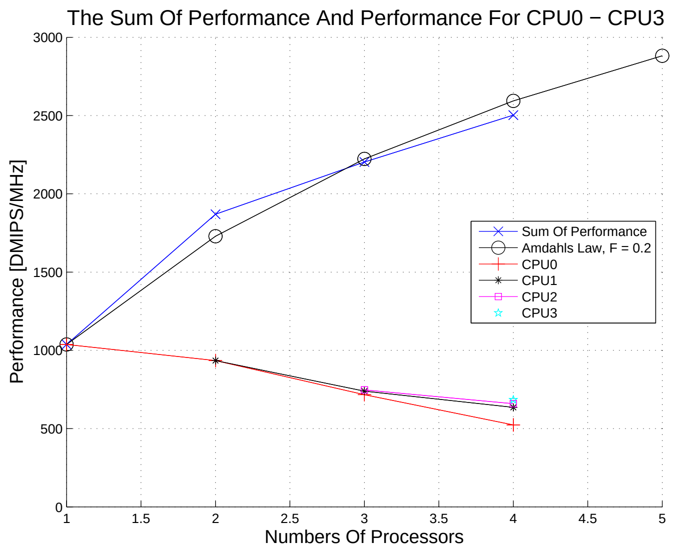


Figure 6.3. Result from measurement with the benchmark Dhrystone, C Version 2.1, in a multiprocessor system with four processors. The hardware configurations were, **with 8kB Data and 8kB Instruction cache and Fixed Priority algorithm** for the AHB controller.

6.2.4 Without cache and the Fixed Priority

Figure 6.4 shows that the overall performance for the system does not increase much when more processors are added to the system. The Amdahl's Law fits well to the curve for the overall performance. In this case F is set to 0.9. In other words, 90% of the system is not parallelized. When the second processor is added the performance increases a little, but when the third processor is added the figure indicate that the performance will start to diminish instead of increasing. With three processors in the system, Figure 6.4 shows a significant discrepancy of performance between different processors. For example, CPU0 contributes with 11%, CPU1 41% and CPU2 with 48%. Furthermore, with four processors the system did not work at all. All these signs together indicate that the boot processor, CPU0, in a system with four processors is given very little access to the bus. The boot processor has the lowest identification number and therefore the lowest priority to access the bus [15, p. 42]. Therefore the bus controller will terminate the boot processors data burst and allow other processors with higher priority to access the bus [8, p. 63]. The others processors, CPU1, CPU2 and CPU3, will take over the bus completely and let CPU0 starve, the boot processor.

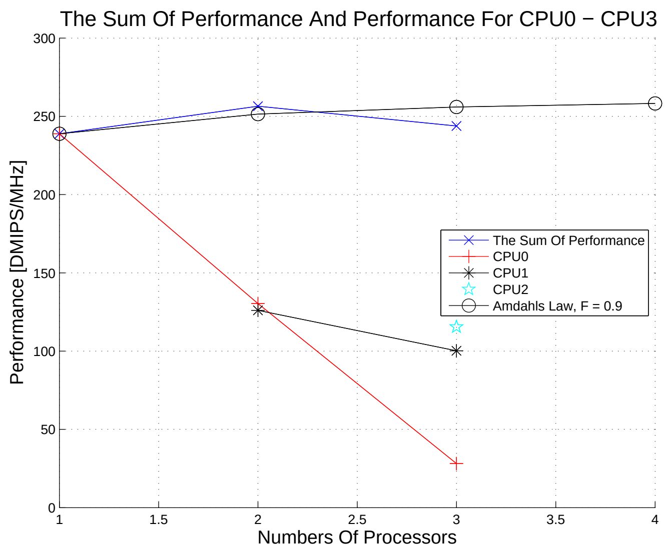


Figure 6.4. Result from measurement with the benchmark Dhrystone, C Version 2.1, in a multiprocessor system with four processors. The hardware configurations were, **without Data and Instruction cache and Fixed Priority algorithm** for the AHB controller.

6.3 Performance measurement with the RTEMS

This part of the study investigates how the performance changes with the three different parameters below.

1. Ticks per Timeslice.
2. Microseconds for one Tick.
3. Number of Tasks in the system.

During the tests, a Bubble sort algorithm was used as workload for the system. The workload was the same for all tasks. The Figures in this chapter show overall performance for the system.

6.3.1 The reference measurement

The reference measurements is illustrated in Figure 6.5. The Figure show the same trend as the measurement with the benchmark Dhrystone. Amdahl's law indicates that the enhancement of the overall performance for the system is about 10%.

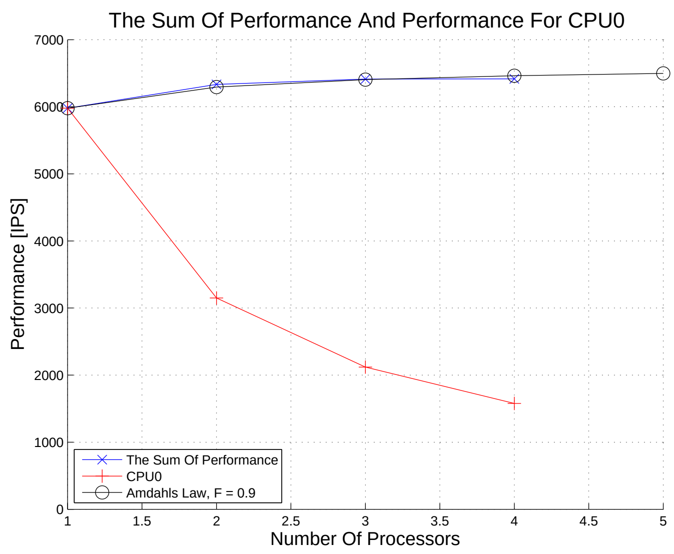


Figure 6.5. Result from performance measurement with simple bubble sort algorithm. The hardware configurations were, with 0 kB Data and 0 kB Instruction cache and **Round Robin** algorithm for processors to get access to the bus.

6.3.2 Ticks per Timeslice versus Microseconds for one Tick

This test sequence measured Ticks per Timeslice versus microseconds for one Tick. The measurement resulted in 42 values which are illustrated in Figure 6.6. As can be seen in Figure 6.6, values below 300 microseconds for one Tick give a low performance. Furthermore, a small number of Ticks per Timeslice also affects the performance. However, Figure 6.6 shows that the system's performance is most sensitive to low values of microseconds per Tick.

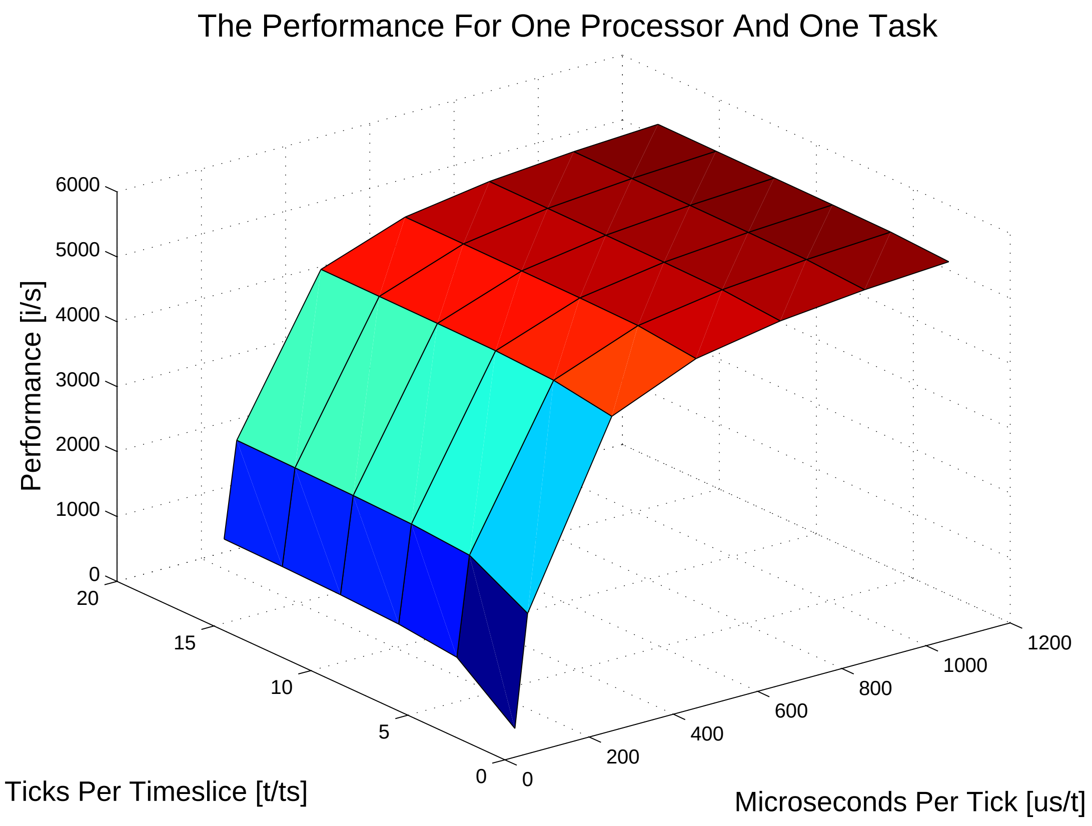


Figure 6.6. The figure illustrates how the performance for a system with the RTEMS is related to the time for a Tick and the number of Ticks for a Timeslice. The hardware configurations were with 0 kB Data and 0 kB Instruction cache and Round Robin algorithm for processors to get access to the bus.

6.3.3 Microseconds for one Tick versus number of Tasks

The measurement with microseconds for one Tick versus number of Tasks resulted in 47 values representing the overall performance of the system. These 47 values are shown in Figure 6.7, and as it can be seen, it is still the time for a Tick that is the key issue for the overall performance in the system.

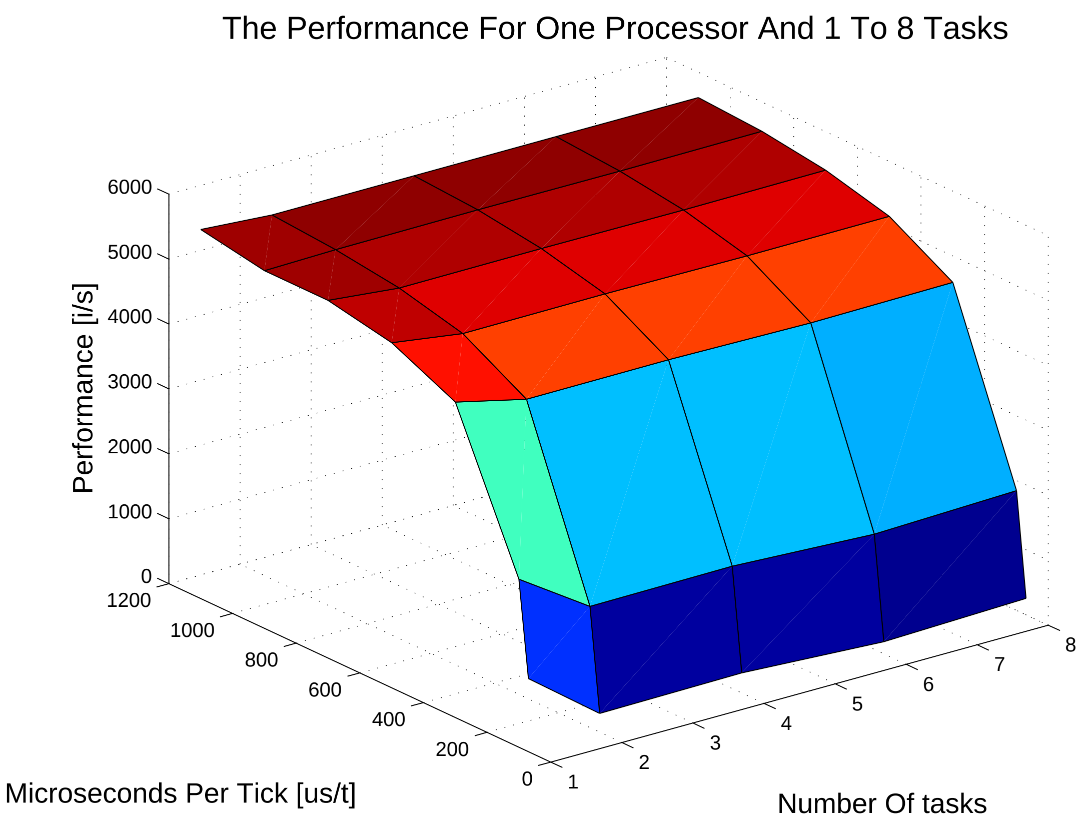


Figure 6.7. The figure illustrates how the performance for a system with the RTEMS is related to the time for a Tick and the number of Tasks in the system. The hardware configurations were with 0 kB Data and 0 kB Instruction cache and Round Robin algorithm for processors to get access to the bus.

6.3.4 The RTEMS with multiple cores

The last test with RTEMS was in a multicore environment. The parameters above were set so that they would not interfer with the measurement. The time for one tick was set to 10,000 microseconds and the number of Ticks for one Timeslice to 10. Furthermore, only one task was used during the test. As can be seen in Figure 6.8, the overall performance for the multisystem with RTEMS is about the same as without the RTEMS as it is shown in Figure 6.5.

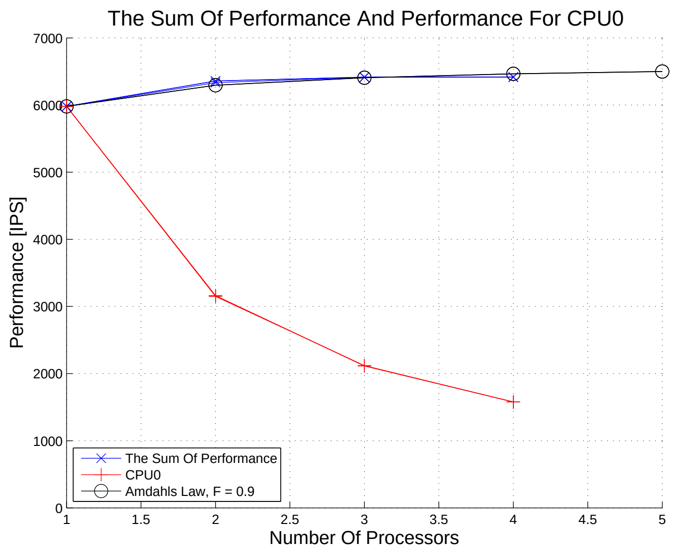


Figure 6.8. The figure illustrate the performance for a system with the RTEMS. In RTEMS used one Task and 10000 microseconds and 10 ticks for a timeslice. The hardware configurations were, with 0 kB Data and 0 kB Instruction cache and Round Robin algorithm for processors for the bus controller.

6.4 Discussion of measurement results

A comparison of the test results between the four hardware configurations shows that the system with cache and Fixed Priority has 7% better overall performance than the system with Round Robin algorithm for the AMBA bus controller. Without cache the system with Round Robin is 6% faster than the system with Fixed Priority for the bus controller. A drawback in the measurements with the Dhrystone benchmark was that the performance for one processor with 8 kB Data and Instruction cache only indicated about 1000 [DMIPS/MHz]. According to the guidelines, the value should be around 1500 DMIPS/MHz [9, p. 534]. However, Westlund did performance tests with the LEON2 and NIOSII processor, using both a small and a big design for comparison. His results indicate, "586 < LEON2 > 1495" [DMIPS/MHz] and "599 > NIOSII < 1529" [DMIPS/MHz] [21]. The results from this study are in this range. Two factors can explain the difference between the result in this study and Westlund's. First, Westlund's used a two-way-associative cache while in this study a one-way-associative cache was used. Second, this study used snooping and 8kB cache because it was stated in the user's manual as a reference [9, p. 534]. Further research after the experiment found a recommendation to use the same cache size as the page size in the Memory Management Unit (MMU) and enable fast snooping. Therefore, for better performance, the system should be configured with 4 kB cache, two or four associativity and fast snooping. Furthermore, if the Dhrystone compiles with the flag -mv8, which generates multiplication and division instructions, then the performance would increase considerably for the Dhrystone benchmark.

As can be seen in Figure 6.4, the system stopped working with four processors without cache and Fixed Priority algorithm for the AMBA bus controller. A possible solution could be to change the AHB bus transfer from Incremental bursts to Fixed-length burst mode [9, p. 14]. This change to Fixed-length has to be done in the HDL code and is not a parameter in the config.vhd file.

The performance measurement with the RTEMS and several Tasks shows that it is the parameter "time for a Tick" that most affect the performance. The reason that "time for a Tick" affects the performance is that the RTEMS makes an interrupt for each Tick. For a system of the same type as covered in this study, values below 900 microseconds per Tick would require special considerations. The overhead for the interrupt makes a significant difference for the performance when the interrupt occurs more often than once per 12'000 clock cycles, in a system without cache. The difference in performance between a system with two Tasks and one Task is also worth noting, see table D.9. The results illustrate the difference between a system with or without a context switch and how the time resolution of the system affects the overall performance.

The last measurement with the RTEMS in a multiprocessor environment shows that when the parameter "time for a Tick" was set sufficiently high the performance became about the same as without the RTEMS.

Chapter 7

Conclusions

This study has shown how to implement a multicore system with the LEON3 processor and what performance could be expected from it with and without the operating system RTEMS. The first test with the benchmark Dhrystone without the RTEMS showed that a multicore system with four processors without cache gives almost the same performance as a system with one processor. From that we conclude: the cache for the processor affects the performance on two different levels. First, on an individual level the cache gives a higher performance for each processor. Second, the cache also affect the performance on system level. The processor cache will reduce the traffic on the AHB bus and give a higher overall performance for the whole system.

The test results also show that there is a clear difference between the performance for hardware configurations with Round Robin and Fixed Priority algorithm for the AHB controller. The choice betwen these algorithms for the AHB controller is therefore important to consider during development of this type of system.

The performance measurement with the RTEMS showed that it is the parameter “time for a Tick” that affects the performance negatively if it is set too low. For example, the measurement showed a performance loss of about 20% when the “time for a Tick” in RTEMS was 12’000 clock cycles. The number of Tasks, in the range from one to eight, did not affect the overall performance much in comparison to the resolution of the time in RTEMS.

The figures in the performance analysis shows the Amdahl’s Law put into effect. The discrepancy between measured values and the Amdahl’s Law clearly show that the Amdahl’s Law does not take into account different bottle necks in the system, such as single bus architecture and a shared main memory. However, Amdahl’s Law is a good starting point for disscussion about enhancement of the performance in a multiprocessor system.

The process to configure the system set-up showed that the GRLIB’s plug and play functionality to configure the LEON3 processor worked flawless during this study. The operating system RTEMS was difficult to recompile from the basic and this study used the ready tool-chains from Aeroflex Gaisler AB. The RTEMS still has several bugs and is at present a development project.

7.1 Further work

Many existing systems are of the same type as those used in this study. Therefore, continue research on this type of system using performance measurement, development of new tools and theoretical models are very important.
Some suggestions are listed below.

7.1.1 Performance counter

A useful feature for performance research in the future would be to develop and implement a performance counter in the LEON3s hardware, similar to what can be found in Altera's NIOSII processor.

7.1.2 Performance for global tasks in the RTEMS

It would be interesting to measure the time it takes to start and stop global Tasks in RTEMS with LEON3 processors. This project would require that the multiprocessor support in the RTEMS works correctly with its message pool.

7.1.3 Performance test for the MMU and cache snooping

During the experiment for this study, a clear difference in performance was noted to declare an array global or local. Furthermore, when different processors wrote to the same array, the performance was higher than when they wrote to separate arrays. The performance results should be the opposite when cache snooping was enabled, see chapter 3.2.2. It would be interesting to know how the MMU and cache snooping affect the performance.

Bibliography

[1] Gnu general public license version 2. Free Software Fermentas Inc, 1991. `http://www.gnu.org/licenses/old-licenses/gpl-2.0.html` Last checked: 20090520.

[2] Rtems webpage. On-Line Applications Research Corporation, 2008. `www.rtems.com` Last checked: 20090714.

[3] Altera Corporation. *Nios Development Board Reference Manual, Stratix II Edition*, july 2005 edition, 2004.

[4] Altera Corporation. *Stratix II Device Handbook, Volyme 1*, 2005.

[5] Altera Corporation. *Nios II Flash Programmer User Guide*, version 1.6, may 2008 edition, 2007.

[6] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. Association for Computing Machinery, 1967.

[7] Isabelle Puaut Antoine Colin. Worst-case execution time analysis of the rtems real-time operating system. 2001.

[8] ARM Limited. *AMBA Specification*, version 2.0 edition, 1999.

[9] Edvin Catovic, Jiri Gaisler, Marko Isomäki, Kristoffer Glembo, and Sandi Habinc. *GRLIB IP Core User's Manual*, version 1.0.20, february 2009 edition, 2008.

[10] Michael J. Flynn and Patrick Hung. Microprocessor design issues: Thoughts on the road ahead. *Micro, IEEE*, 2005.

[11] Jiri Gaisler. *GRMONE Users Manual*. AEROFLEX GAISLER AB, version 1.1.45, march 2009 edition, 2004.

[12] Jiri Gaisler. *BCC-Bare-C Cross-Compiler User's Manual*, version 1.0.29, february 2007 edition, 2006.

[13] Sandi Habinc, Jiri Gaisler, and Edvin Catovic. *GRLIB IP Library Users Manual*, version 1.0.19, february 2008 edition, 2008.

[14] Gaisler Jiri. *RCC Users Manual*, version 1.1.1, february 2008 edition, 2008.

[15] A. Sai Pramod Kumar. A prototype and validation platform for leon based multiprocessor socs. Master's thesis, Indian Institute of Technology Delhi, 2002.

[16] John L.Hennessy and David A. Patterson. *Computer Architecture*. Morgan Kaufmann Publishers, 2003.

[17] Jian Li and Jose FMart nez. Power-performance implications of thread-level parallelism on chip multiprocessors. In *Performance Analysis of Systems and Software*.

[18] On-Line Applications Research Corporation. *RTEMS C User's Guide*, edition 4.6.5, for rtems 4.6.5 edition, 2003.

[19] Luis Moura E Silva and Rajkumar Buyyaz. *High performance cluster computing, vol. 2 : programming and applications*. Prentice Hall, NJ, USA, 1999.

[20] SPARC International, Inc. *The SPARC Architecture Manual*, version 8 edition, 2008.

[21] Klas Westlund. Comparation of synthesizable processor cores. Master's thesis, Chalmers University of Technology, 2002.

[22] Richard York. Benchmarking in context: Dhrystone. ARM, 2002. `http://www.arm.com/pdfs/Dhrystone.pdf` Last checked: 20090513.

Appendix A

Overview of information sources

The review of documents and files below aims to avoid the confusion in the process of implementing the LEON3 processor and the operating system RTEMS on an Altera development board with the Stratix II FPGA.

A.1 Documents for a kick start to implementation

- **Grlib IP Library User's Manual** [**13**]. The Grlib IP User's Manual document gives a fast overview and introduction to GRLIB. Note the commands on page 32 "make xconfig" to configure the system, "make quartus" to synthesize the design and "make quartus-prog-fpga" to program the design, they are the most important things to start with in this document.

- **NiosII Development Board Reference Manual, Stratix II Edition** [**3**]. The NIOSII board manual contains good hands on knowledge about the about Altera's NIOS II development board.

- **GRMON User's Manual** [**11**]. The GRMON manual is an introduction to the interface between the development computer and the target board.

- **BCC-Bare-C Cross-Compiler User's Manual** [**12**]. The BCC manual helps out with compiling and running the first "Hello World" program with the LEON3 processor. There are some examples for multiprocessing with the LEON3 processor in the software folder and there is also a collection of examples that are possible to download from www.gaisler.com.

- **RCC User's Manual** [14]. The RTEMS Cross Compiler (RCC) manual is a poor introduction to RTEMS and multiprocessing but the best available. In addition, take a look at the examples in the sample folder from the RTEMS that is available on www.gaisler.com.

A.2 Documents and code files for a deeper understanding

- **GRLIB IP Core User's Manual** [9]. The IP Core manual describes how IP cores in GRLIB work in detail. It is invaluable when developing systems with GRLIB. The document describes the configuration of all IP cores and the names of theirs properties, these names relate to the HDL code for the design. Take especially a look into the config.vhd file that contains some of the IP cores names.
- **The AMBA Specification** [8]. A good complement to the GRLIB IP Core User's Manual.
- **The SPARC Architecture Manual** [20]. A good complement to the GRLIB IP Core User's Manual.
- **The NiosII Flash Programmer User Guide** [5]. The Flash manual guide to how to program the flash memory on the development board with the LEON3 processor design. It is very convenient that the LEON3 processor design always is on the development board.
- **The RTEMS C User's Guide** [18]. Of all documentation about RTEMS, this is the one that is really helpful even if it is more of a specification of development of RTEMS than for developing applications. For real knowledge and understanding of RTEMS is the best source the RTEMS source code.

Another very good source for problem solving and detailed information is also Yahoo's group “LEON SPARC”. The “LEON SPARC” group is a living community and there it is possible to ask questions and take part in discussions.

Appendix B

System specifications

B.1 Software Versions

Provider	Software	Version
Aeroflex Gaisler AB	Grmon	1.1.35 (evaluation version)
Aeroflex Gaisler AB	Grlib	1.0.20-b3403 (GPL)
Aeroflex Gaisler AB	SPARC GCC compiler	3.4.4 (GPL)
Aeroflex Gaisler AB	SPARC RTEMS compiler	4.10 (GPL)
OAR Corporation	RTEMS source code	4.10 (GPL)
Altera	Quartus II	8.1
Linux	Ubuntu	8.10 (Intrepid Ibex)

B.2 Hardware configurations

Table B.1. List of shortenings for the hardware configuration. table

RR8C	8 kB Data and 8 kB Instruction cache and Round Robin
RR0C	0 kB Data and 0 kB Instruction cache and Round Robin
FP8C	8 kB Data and 8 kB Instruction cache and Fixed Priority
FP0C	0 kB Data and 0 kB Instruction cache and Fixed Priority

Table B.2. Hardware configurations for different tests.

Hardware Configuration	RR8C	RR0C	FP8C	FP0C
Clock Frequency	40MHz	40MHz	40MHz	40MHz
Number of Processors	4	4	4	4
Floating-point Unit	No	No	No	No
Pipline Stages	7	7	7	7
Instructions Cache	8kB	-	8kB	-
Associativity	1	-	1	-
Line Size	32	-	32	-
Data Cache	8kB	-	8kB	-
Associativity	1	-	1	-
Line Size	32	-	32	-
AHB Snooping	Yes	-	Yes	-
Fast Snooping	No	-	No	-
Separate Snoop Tags	No	-	No	-
MMU	Yes	Yes	Yes	Yes
MMU Type	Split	Split	Split	Split
Replacement Scheme	LRU	LRU	LRU	LRU
AHB Controller	RR	RR	FP	FP
SDRAM Controller	Yes	Yes	Yes	Yes
Number of Timers	4	4	4	4

Appendix C

Program example for a multiprocessor environment

C.1 Batch file example

Below is an example of a batch file, that runs from GRMON. The example compiles the example.c file to four executable files, put them into the memory at different places, configures all processors and then starts processor 0.

```
# Batch file for a LEON3 multiprocessor system monitored with GRMON.
# A 16 MB SDR RAM memory with base at 0x6000000 gives a address space  0x6000000 - 0x61000000.

shell sparc-elf-gcc -msoft-float -O2  -Ttext=0x60000000 example.c  -DNODE_NUMBER=1 -o node1.exe
shell sparc-elf-gcc -msoft-float -O2  -Ttext=0x60020000 example.c  -DNODE_NUMBER=2 -o node2.exe
shell sparc-elf-gcc -msoft-float -O2  -Ttext=0x60040000 example.c  -DNODE_NUMBER=3 -o node3.exe
shell sparc-elf-gcc -msoft-float -O2  -Ttext=0x60060000 example.c  -DNODE_NUMBER=4 -o node4.exe

load node1.exe
load node2.exe
load node3.exe
load node4.exe

cpu ena 0
cpu act 0
ep 0x60000000
stack 0x60010000
cpu act 1
cpu act 1
ep 0x60020000
stack 0x60030000
cpu ena 2
cpu act 2
ep 0x60040000
stack 0x60050000
cpu ena 3
cpu act 3
ep 0x60060000
stack 0x60070000
```

```
cpu act 0
run
```

The command that executes in the GRMON should look like “grlib> batch name-of-batch-file”. Note also, all the files should be in the same folder that the GRMON is started in. The concept Node is a RTEMS definition there Node1 = CPU0, Node2 = CPU1 etc.

C.2 C program example for a multiprocessor system

Below is an example of a C program that shows how to program a multiprocessor system with four processors. It is convenient and it gives a good overview of the application to write the C code for all processors in the same source file. The key issue here is to illustrate the use of ordinary C and hash code to configure applications on different processors. The hash code is used by the precompiler, compile-time, and is static, i.e. it can not be configured after compiling. The C code on the other hand can be configured during run-time and applications can dynamically be configured on different processors.

```
#include <stdio.h>
main(){
     #if NODE_NUMBER == 1
     // To start all others processors.
     // Writes to the status register.
     int*  status = (int *) 0x80000210;;
     *status = 0x3800000e;
     resultat = (int *) 0x60080000;
     #endif
     #if NODE_NUMBER == 2
          resultat = (int *) 0x60080004;
     #endif
     #if NODE_NUMBER == 3
     resultat = (int *)0x60080008;
     #endif
     #if NODE_NUMBER == 4
          resultat = (int *) 0x6008000c;
     #endif
     *resultat = NODE_NUMBER;
     if(NODE_NUMBER == 1){
          int i = 0;
          for(i =0; i < 4;i++)
               printf("Hello from Node %u \n", *(resultat + i));
     }
#if NODE_NUMBER != 1
     while(true){};
#endif
     return(0);
}
```

Appendix D

Test results

D.1 Test results from the benchmark Dhrystone, C v. 2.1

The result in the tables shows the performance from four different multiprocessor systems. The measurement with Dhrystone C v. 2.1, was done with 400'000 iterations and 16 times. From these measurements mean values and confidence intervals, u, were calculated. The Ratio is the quota between the performance from individual processors and the sum of performance. The probability that mean values are in the confidence interval is 68.3%.

Table D.1. Result from measurement with Dhrystone v 2.1. The hardware configuration was 8 kB Data and 8 kB Instruction cache, Round Robin algorithm on the AHBs arbiter. All results are in the unit [DMIPS/MHz].

	One processor	Two processors	Three processors	Four processors
CPU0	1'037.2	923.4	725.2	569.3
$\pm\mu$	0.2	0.2	0.4	0.3
Ratio	100.00%	50.00%	33.00%	24.25%
CPU1	-	923.4	737.8	592.1
$\pm\mu$	-	0.2	0.5	0.3
Ratio	-	50.00%	33.50%	25.25%
CPU2	-	-	737.8	592.1
$\pm\mu$	-	-	0.5	0.2
Ratio	-	-	33.50%	25.25%
CPU3	-	-	-	592.1
$\pm\mu$	-	-	-	0.2
Ratio	-	-	-	25.25%
Sum	1'037	1'847	2'201	2'346

Table D.2. Result from measurement with Dhrystone v 2.1. The hardware configuration was 0 kB Data and 0 kB Instruction cache, Round Robin algorithm on the AHBs arbiter. All results are given in the unit [DMIPS/MHz].

	One processor	Two processors	Three processors	Four processors
CPU0	238.865	126.300	86.964	65.2400
$\pm\mu$	0.008	0.003	0.002	0.0007
Ratio	100.00%	49.20%	33.25%	25.00%
CPU1	-	130.79	86.625	64.707
$\pm\mu$	-	0.003	0.003	0.001
Ratio	-	50.80%	33.45%	25.00%
CPU2	-	-	86.251	64.707
$\pm\mu$	-	-	0.003	0.001
Ratio	-	-	33.30%	25.00%
CPU3	-	-	-	64.707
$\pm\mu$	-	-	-	0.001
Ratio	-	-	-	25.00%
Sum	239	252	259	259

Table D.3. Result from measurement with Dhrystone v 2.1. The hardware configuration was 8 kB Data and 8 kB Instruction cache, Fixed Priority algorithm on the AHBs arbiter. All results are given in the unit [DMIPS/MHz].

	One processor	Two processors	Three processors	Four processors
CPU0	1’037.3	934.8	716.9	523.7
$\pm\mu$	0.2	0.1	0.3	0.5
Ratio	100.00%	50.00%	32.54%	20.92%
CPU1	-	934.8	739.2	635.43
$\pm\mu$	-	0.1	0.2	0.08
Ratio	-	50.00%	33.55%	25.39%
CPU2	-	-	747.1	659.34
$\pm\mu$	-	-	0.4	0.07
Ratio	-	-	33.91%	26.34%
CPU3	-	-	-	684.20
$\pm\mu$	-	-	-	0.05
Ratio	-	-	-	27.35%
Sum	1’037	1’870	2’203	2’503

Table D.4. Result from measurement with Dhrystone v 2.1. The hardware configuration was 0 kB Data and 0 kB Instruction cache, Fixed Priority algorithm on the AHBs arbiter. All results are given in the unit [DMIPS/MHz].

	One processor	Two processors	Three processors	Four processors
CPU0	238.865	130.37	28	-
$\pm\mu$	0.008	0.03	1	-
Ratio	100.00%	50.93%	11.50%	-
CPU1	-	126.16	100.195	-
$\pm\mu$	-	0.02	0.008	-
Ratio	-	49.07%	41.19%	-
CPU2	-	-	115.438	-
$\pm\mu$	-	-	0.003	-
Ratio	-	-	47.31%	-
Sum	239	256	244	-

D.2 Performance measurement with a Bubble sort algorithm

Table D.5. Result from measurement with the RTEMS, The hardware configuration was 0 kB Data and 0 kB Instruction cache, Round Robin algorithm on the AHBs arbiter. All results are in the unit [I/S].

	One processor	Two processors	Three processors	Four processors
CPU0	5'978.2	3'150.4	2'118.7	1'577.9
$\pm\mu$	0.02	0.3	0.6	0.2
Ratio	100.00%	49.75%	33.05%	24.64%
CPU1	-	3'182.2	2'146.754	1'612.8792
$\pm\mu$	-	0.1	0.006	0.0004
Ratio	-	50.25%	33.47%	25.14%
CPU2	-	-	2'146.71	1'612.8791
$\pm\mu$	-	-	0.03	0.0004
Ratio	-	-	33.48%	25.14%
CPU3	-	-	-	1'612.8791
$\pm\mu$	-	-	-	0.0004
Ratio	-	-	-	25.14%
Sum	5'978	6'332	6'412	6'417

Table D.6. Result from measurement with the RTEMS. The hardware configuration was 0 kB Data and 0 kB Instruction cache, Fixed Priority algorithm on the AHBs controller. All results are in the unit [I/S].

	One processor	Two processors	Three processors	Four processors
CPU0	5'968.5	3'151.46	985.07	-
$\pm\mu$	0.4	0.03	0.07	-
Ratio	100.00%	50.00%	15.70 %	-
CPU1	-	3'151.49	2'413.56	-
$\pm\mu$	-	0.03	0.04	-
Ratio	-	50.00%	38.47%	-
CPU2	-	-	2'874.89	-
$\pm\mu$	-	-	0.03	-
Ratio	-	-	45.83 %	-
Sum	5'968	6'302	6'273	-

Table D.7. Performance measurement with RTEMS between numbers of ticks for one timeslice versus microseconds for one tick. The measurement was done with one processor, Round Robin algorithm for the AMBA bus controller and without cache. The performance was measured with iterations per second of a simple bubblesort algorithm.

	Microseconds Per Tick						
	70	100	300	500	700	900	1100
TimeSlice	Performance, Iterations Per Second						
1 Tick	220.66	1’940.86	4’626.10	5’162.94	5’393.1	5’520.9	5’602.1
Ratio	3.9%	34.3%	81.8%	91.3%	95.3%	97.6%	99.0%
$\pm\mu$	0.01	0.01	0.05	0.07	0.1	0.1	0.1
4 Ticks	906.80	2’425.22	4’769.2	5’259.52	5’462.20	5’520.90	5’602.13
Ratio	16.0%	42.9%	84.3%	93.0%	96.6%	98.5%	99.8%
$\pm\mu$	0.02	0.03	0.07	0.1	0.07	0.09	0.03
7 Ticks	1’004.71	2’493.37	4’810.67	5’273.8	5’472.25	5’574.71	5’646.4
Ratio	17.8%	44.1%	85.0%	93.2%	96.7%	98.7%	99.9%
$\pm\mu$	0.02	0.03	0.07	0.1	0.08	0.03	0.1
10 Ticks	1’044.20	2’521.07	4’819.78	5’279.1	5’476.03	5’582.48	5’652.9
Ratio	18.5%	44.6%	85.2%	93.3%	96.8%	98.7%	99.9%
$\pm\mu$	0.02	0.02	0.07	0.1	0.09	0.05	0.1
13 Ticks	1’065.77	2’535.95	4’824.98	5’282.2	5’478.9	5’587.07	5’656.5
Ratio	18.8%	44.8%	85.3%	93.4%	96.8%	98.8%	99.9%
$\pm\mu$	0.01	0.03	0.07	0.1	0.1	0.09	0.1
16 Ticks	1’079.05	2’546.11	4’827.97	5’284.02	5’479.66	5’588.4	5’657.3
Ratio	19.1%	45.0%	85.3%	93.4%	96.9%	98.8%	100.0%
$\pm\mu$	0.02	0.03	0.07	0.03	0.09	0.1	0.1

Table D.8. Result from measurement with the RTEMS in a multiprocessor environment. The hardware configuration was 0 kB Data and 0 kB Instruction cache, Round Robin algorithm on the AHBs arbiter. All results are in the unit [I/S].

	One processor	Two processors	Three processors	Four processors
CPU0	5’980.15	3’161.5	2’114.9	1’577.0
$\pm\mu$	0.03	0.2	0.2	0.3
Ratio	100.00%	49.72%	32.96%	25.58%
CPU1	-	3’196.94	2’150.7238	1’613.1735
$\pm\mu$	-	0.08	0.0007	0.0005
Ratio	-	50.28%	33.52%	25.14%
CPU2	-	-	2’150.7485	1’613.1548
$\pm\mu$	-	-	0.0007	0.0005
Ratio	-	-	33.52%	25.14%
CPU3	-	-	-	1’613.1364
$\pm\mu$	-	-	-	0.0005
Ratio	-	-	-	25.14%
Sum	5’980	6’358	6’416	6’416

Table D.9. Performance measurement with RTEMS between between microseconds for one tick versus number of tasks. The measurement was done with one processor, Round Robin algorithm for the AMBA bus controller and without cache. The performance was measured with iterations per second of a simple bubblesort algorithm.

	Microseconds Per Tick						
	70	100	300	500	700	900	1100
Tasks	Performance: Iterations Per Second						
Task 1	1’124.677	2’587.65	4’859.60	5’313.98	5’508.37	5’508.37	5’685.19
$\pm\mu$	0.007	0.03	0.03	0.04	0.05	0.05	0.05
Sum	1’124	2’587	4’860	5’313	5’508	5’508	5’685
Ratio	19.8%	45.5%	85.4%	93.4%	96.8%	96.8%	100.0%
Task 1	143.518	933.69	2’300.1	2’577.161	2’696	2’7610	2’803
$\pm\mu$	0.004	0.03	0.4	0.005	1	2	1
Task 2	143.484	933.805	2’300.91	2’576.96	2’700.46	2’769.642	2’804
$\pm\mu$	0.004	0.004	0.06	0.01	0.06	0.006	0.1
Sum	287	1’8670	4’601	5’154	5’397	5’530	5’608
Ratio	5.0%	32.8%	80.9%	90.6%	94.9%	97.2%	98.6%
Task 1	77.133	471.25	1’152.4	1’292.8	1’350.7	1’381.2	1’401.9
$\pm\mu$	0.002	0.01	0.3	0.5	0.7	1	0.8
Task 2	77.126	471.3533	1’151.8693	1’293.7457	1’351.5	1’386.950	1’402.5
$\pm\mu$	0.001	0.0008	0.3	0.0009	0.1	0.001	0.1
Task 3	77.084	471.1408	1’150.84	1’288.51	1’351.41	1’386.7422	1’402.37
$\pm\mu$	0.004	0.0008	0.03	0.08	0.07	0.0007	0.09
Task 4	77.067	471.00	1’150.82630	1’288.46	1’351.387	1’386.653	1’402.32
$\pm\mu$	0.004	0.01	0.002	0.04	0.02	0.002	0.04
Sum	308	1’884	4’606	5’163	5’405	5’541	5’609
Ratio	5.4%	33.1%	81.0%	90.8%	95.0%	97.4%	98.6%
Task 1	30.98	296.351	762.8	857	897	920	935.042
$\pm\mu$	0.01	0.001	0.4	2	2	4	0.005
Task 2	30.946	295.94	761.0	858.4	900.816	924.604	934.70
$\pm\mu$	0.006	0.06	0.2	0.3	0.001	0.002	0.4
Task 3	30.965	296.22	762.8	858.536	900.957	924.755	934.9
$\pm\mu$	0.009	0.03	0.2	0.2	0.002	0.002	0.3
Task 4	30.926	295.4	760.9	854.7	894.8	916.6	934.4
$\pm\mu$	0.007	0.2	0.3	0.8	0.2	0.2	0.2
Task 5	30.968	296.24	762.82	858.54	900.955	924.755	934.9
$\pm\mu$	0.005	0.03	0.06	0.08	0.003	0.004	0.1
Task 6	30.939	295.94	761.0	858.424	900.815	924.604	934.758
$\pm\mu$	0.009	0.09	0.4	0.009	0.002	0.002	0.031
Sum	185	1’776	4’571	5’146	5’396	5’535	5’608
Ratio	3.2%	31.2%	80.4%	90.5%	94.9%	97.3%	98.6%
Task 1	31.595	230.34	573.9216	644.2267	674.1	690.5	699.5
$\pm\mu$	0.002	0.05	0.0003	0.0005	0.6	0.8	0.8
Task 2	31.491	230.14	573.70	643.97	675.8349	693.60	701.04
$\pm\mu$	0.005	0.03	0.05	0.07	0.0003	0.07	0.08
Task 3	31.627	230.6229	575.3782	644.24	676.11	693.91	701.35
$\pm\mu$	0.002	0.0005	0.0007	0.06	0.0003	0.06	0.07
Task 4	31.570	230.5968	573.84	644.14	675.9924	693.790	701.23
$\pm\mu$	0.002	0.0004	0.03	0.04	0.0001	0.045	0.06
Task 5	31.629	230.6193	575.3787	644.264	676.11	693.93	701.37
$\pm\mu$	0.002	0.0006	0.0008	0.04	0.0003	0.04	0.05
Task 6	31.501	230.12	573.74	644.01	675.8346	693.65	701.010
$\pm\mu$	0.003	0.03	0.02	0.03	0.0002	0.03	0.034
Task 7	31.596	230.6140	575.3336	644.2284	676.05	693.88	701.336
$\pm\mu$	0.003	0.0006	0.0003	0.02	0.0002	0.01	0.02
Task 8	31.499	230.14	573.753	644.032	675.8352	693.672	701.11
$\pm\mu$	0.003	0.02	0.005	0.006	0.0008	0.002	0.01
Sum	252	1’843	4’595	5’153	5’405	5’546	5’608
Ratio	4.4%	32.4%	80.8%	90.6%	95.0%	97.5%	98.6%