NTNU

Innovation and Creativity

System on a chip – Soft IP from the FPGA-vendor or an OpenCore-processor?

Robert Bayona Adam

Master of Science in Electronics

Submission date: February 2007

Supervisor: Bjørn B. Larsen, IET

Norwegian University of Science and Technology

Department of Electronics and Telecommunications

Problem Description

Design reuse is an important part of today's design practice. Reuse may be implemented using Intellectual Property (IP) from a vendor or reusing older internal designs. Effective reuse requires proper documentation and flexibility. The Reuse Methodology Manual [1] gives some recommendatoions in this regard.

Describe how a microprocessor core should be implemented and documented as an IP.
On an FPGA a processor may be implemented using IP from the FPGA-vendor, IP from a third party vendor or free IPs available on the internet.

Identify a free microprocessor core that may be implemented on an FPGA as an alternative to the cores offered by the vendors. Implement the core on FPGAs together with ALTERA NIOS2 and XILINX MicroBlaze. Implement Embedded Linux on the cores, perform a bench mark of the implementations and compare the results.

The comparison of the IPs should also take into account the documentation and ease of use. Also the tools available to the designer should be evaluated. Compare the available modules to the recommendations in [1].

Supervisor: Bjørn B. Larsen, Room B-317, bjorn.b.larsen@iet.ntnu.no

[1] Reuse Methodology Manual, Michael Keating and Pierre Bricaud, Kluwer Academic Publishers, 2002.

Assignment given: 04. September 2006
Supervisor: Bjørn B. Larsen, IET

System on a chip - Soft IP from the FPGA-vendor or an OpenCore-processor?

1. Index

2. Introduction

2.1.Project description

The work will consist of comparing two different processors from two FPGA vendors and an OpenCore-processor.

For this work we are going to use two different boards, the first will be a Cyclone II FPGA Altera Board, in which we will run the Nios II Altera microprocessor and the free processor Leon2.

The second board will be a SUZAKU-S board, in which we will run the Microblaze Xilinx microprocessor and the free processor Leon2.

In all this boards we are going to run two different benchmarks, the Dhrystone and the Whetstone, for compare the different velocities between the free and not free processors.

Also, we are going to take into account the documentation and ease of use of the processors.

2.2.Motivation

Nowadays, there are a lot of different processors, FPGAs, software for the microprocessors, etc… and for this reason, we have thought that is a good idea that somebody studies the differences among different processors, and if is really a good idea to pay for processors or is better to use a free processor.

Also, the ease of use of the processors is something to take care about it. We think that is necessary that somebody makes a serious study about which processor is easier to use, how many different operative systems you need to program and run software in their, and how much difficult can be to use the software needed to program their.

Also the information in internet about this is not much and is all dispersed. To find some interesting information about this subject, you have to search a spend lot of time browsing in the internet webs.

For all these reasons, we have decided to do this project and to make a clear document in which you can find a studied opinion about these processors

3. Benchmarking

In computing, a benchmark is the result of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, by running a number of standard tests and trials against it. The term, benchmark, is also commonly used for specially-designed benchmarking programs themselves. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU, but there are circumstances when the technique is also applicable to software. Software benchmarks are, for example, run against compilers or database management systems.

Benchmarks provide a method of comparing the performance of various subsystems across different chip/system architectures. Benchmarking is helpful in understanding how the database manager responds under varying conditions. You can create scenarios that test deadlock handling, utility performance, different methods of loading data, transaction rate characteristics as more users are added, and even the effect on the application of using a new release of the product.

3.1.Purpose

As computer architecture advanced, it became more and more difficult to compare the performance of various computer systems simply by looking at their specifications. Therefore, tests were developed that could be performed on different systems, allowing the results from these tests to be compared across different architectures. For example, while Intel Pentium 4 processors generally operate at a higher clock frequency than AMD Athlon XP processors, this does not necessarily translate to more computational power. In other words a 'slower' AMD processor, with regards to clock frequency, can perform as well on benchmark tests as an Intel processor operating at a higher frequency.

Benchmarks are designed to mimic a particular type of workload on a component or system. "Synthetic" benchmarks do this by specially-created programs that impose the workload on the component. "Application" benchmarks, instead, run actual real-world programs on the system. Whilst application benchmarks usually give a much better measure of real-world performance on a given system, synthetic benchmarks still have their use for testing out individual components, like a hard disk or networking device.

Benchmarks are particularly important in semiconductor microprocessor design, giving processor architects the ability to measure and make tradeoffs in microarchitectural decisions. For example, if a benchmark extracts the key algorithms of an application, it will contain the performance-sensitive aspects of that application. Running this much smaller "snippet" on a cycle-accurate simulator, can give clues on how to improve performance.

Prior to 2000, computer and microprocessor architects used SPEC to do this, although SPEC's Unix-based benchmarks were quite lengthy and thus unwieldy to use intact.

Computer manufacturers have a long history of trying to set up their systems to give unrealistically high performance on benchmark tests that is not replicated in real usage. For instance, during the 1980s some compilers could detect a specific mathematical operation used in a well-known floating-point benchmark and replace the operation with a mathematically-equivalent operation that was much faster. However, such a transformation was rarely useful outside the benchmark until the mid-1990s, when RISC and VLIW architectures emphasized the importance of compiler technology as it related to performance. Benchmarks are now regularly used by compiler companies to improve not only their own benchmark scores, but real application performance.

Manufacturers commonly report only those benchmarks (or aspects of benchmarks) that show their products in the best light. They also have been known to mis-represent the significance of benchmarks, again to show their products in the best possible light. Taken together, these practices are called bench-marketing.

Users are recommended to take benchmarks, particularly those provided by manufacturers themselves, with ample quantities of salt unless the benchmarks are certified and relate directly to a recognizable application workload. Ideally benchmarks should only substitute for real applications if the application is unavailable, or too difficult or costly to port, to a specific processor or computer system. If performance is really critical, the only benchmark that matters is the actual workload that the system is to be used for. If that is not possible, benchmarks that resemble real workloads as closely as possible should be used, and even then used with skepticism unless independently certified. It is quite possible for system A to outperform system B when running program "furble" on workload X (the workload in the benchmark), and the order to be reversed with the same program on your own workload.

3.2.Challenges

Benchmarking is not easy and often involves several iterative rounds in order to arrive at predictable, useful conclusions. Interpretation of benchmarking data is also extraordinarily difficult. Here is a partial list of common challenges:

Vendors tend to tune their products specifically for industry-standard benchmarks. Use extreme caution in interpreting such results.

- Benchmarks generally do not give any credit for any qualities of service aside from raw performance. Examples of unmeasured qualities of service include security, availability, reliability, execution integrity, serviceability, scalability (especially the ability to quickly and nondisruptively add or reallocate capacity), etc. There are often real trade-offs between and among these qualities of service, and all are important in business computing. TPC

Benchmark specifications partially address these concerns by specifying ACID property tests, database scalability rules, and service level requirements.

- In general, benchmarks do not measure TCO. TPC Benchmark specifications partially address this concern by specifying that a price/performance metric must be reported in addition to a raw performance metric, using a simplified TCO formula.
- Benchmarks seldom measure real world performance of mixed workloads running multiple applications concurrently in a full, multi-department/multi-application business context. For example, IBM's mainframe servers (System z9) excel at mixed workload, but industry-standard benchmarks don't tend to measure the strong I/O and large/fast memory design such servers require. (Most other server architectures dictate fixed function/single purpose deployments, e.g. "database servers" and "Web application servers" and "file servers," and measure only that. The better question is, "What more computing infrastructure would I need to fully support all this extra workload?")
- Vendor benchmarks tend to ignore requirements for development, test, and disaster recovery computing capacity. Vendors only like to report what might be narrowly required for production capacity in order to make their initial acquisition price seem as low as possible.
- Benchmarks are having trouble adapting to widely distributed servers, particularly those with extra sensitivity to network topologies. The emergence of grid computing, in particular, complicates benchmarking since some workloads are "grid friendly," while others are not.
- Users can have very different perceptions of performance than benchmarks may suggest. In particular, users appreciate predictability, servers that always meet or exceed SLAs. Benchmarks tend to emphasize mean scores (IT perspective) rather than low standard deviations (user perspective).
- Many server architectures degrade dramatically at high (near 100%) levels of utilization, "fall off a cliff", and benchmarks should (but often don't) take that factor into account. Vendors, in particular, tend to publish server benchmarks at continuous ~80% utilization, a totally unreal situation, and do not document what happens to the overall system when/if demand spikes beyond that level.

3.3.Types of benchmarks

1. Real program
   - word processing software
   - tool software of CDA
   - user's application software (MIS)
2. Kernel
   - contains key codes
   - normally abstracted from actual program
   - popular kernel: Livermore loop
   - linpack benchmark (contains basic linear algebra subroutine written in FORTRAN language)

   - results are represented in MFLOPS
3. Toy Benchmark/ micro-benchmark
   - user can program it and use it to test computer's basic components
   - automatic detection of computer's hardware parameters like number of registers, cache size, memory latency
4. Synthetic Benchmark
   - Procedure for programming synthetic Bench mark
     - take statistics of all type of operations from plenty of application programs
     - get proportion of each operation
     - write a program based on the proportion above
   - Types of Synthetic Benchmark are:
     - Whetstone
     - Dhrystone
   - Its results are represented in KWIPS (kilo whetstone instructions per second). It is not suitable for measuring pipeline computers.
5. I/O benchmarks
6. Parallel benchmarks: used on machines with multiple processors or systems consisting of multiple machines.

4. Analysis and methods

4.1. Analysis of the processors

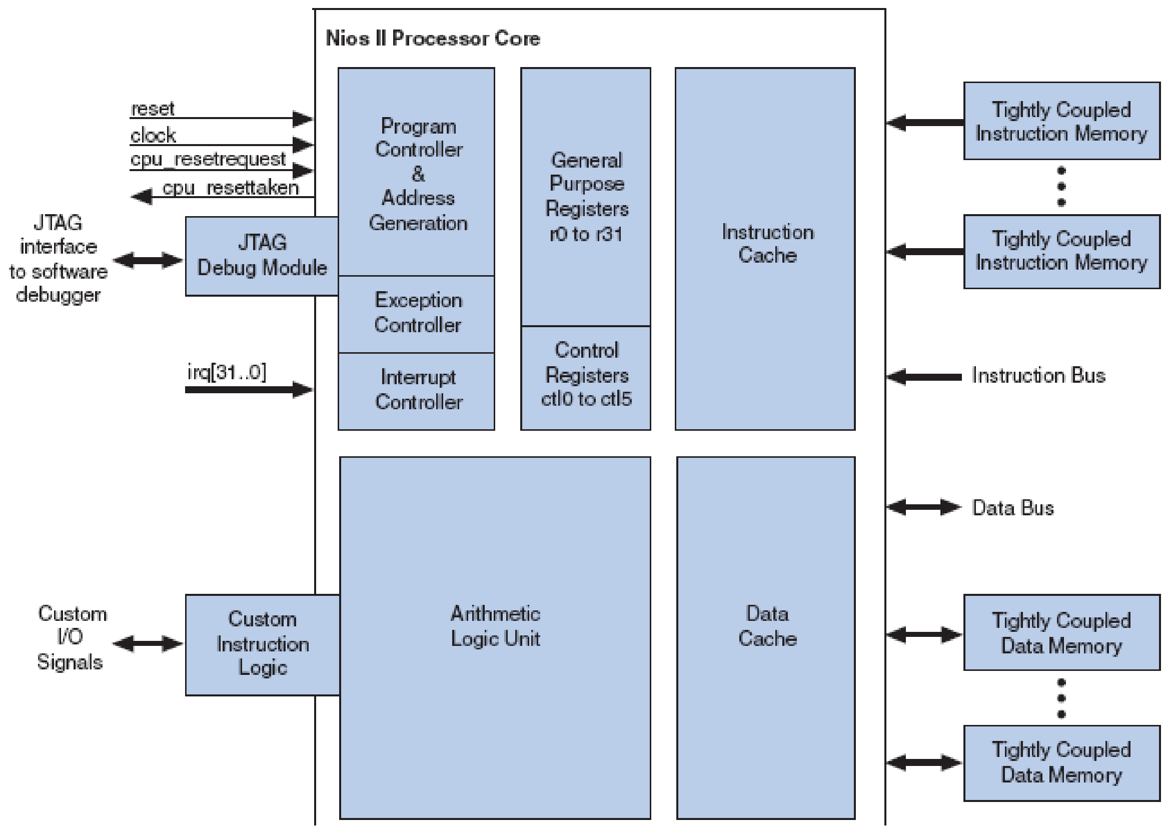
4.1.1. Nios II Altera processor

The Nios II processor is a general-purpose RISC processor core, providing:

- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 external interrupt sources
- Single-instruction 32 × 32 multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Floating-point instructions for single-precision floating-point operations
- Single-instruction barrel shifter
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Hardware-assisted debug module enabling processor start, stop, step and trace under integrated development environment (IDE) control
- Software development environment based on the GNU C/C++ tool chain and Eclipse IDE
- Integration with Altera's SignalTap(r) II logic analyzer, enabling realtime analysis of instructions and data along with other signals in the FPGA design
- Instruction set architecture (ISA) compatible across all Nios II processor systems
- Performance up to 250 DMIPS

A Nios II processor system is equivalent to a microcontroller or “computer on a chip” that includes a CPU and a combination of peripherals and memory on a single chip. The term “Nios II processor system” refers to a Nios II processor core, a set of on-chip peripherals, onchip memory, and interfaces to off-chip memory, all implemented on a single Altera® chip. Like a microcontroller family, all Nios II processor systems use a consistent instruction set and programming model.

Figure 1. Nios II Processor Core Block Diagram


Nios II Processor Core
reset
clock
cpu_resetrequest
cpu_resettaken
JTAG interface to software debugger
JTAG Debug Module
irq[31..0]
Custom I/O Signals
Custom Instruction Logic
Program Controller & Address Generation
Exception Controller
Interrupt Controller
General Purpose Registers r0 to r31
Control Registers ctl0 to ctl5
Instruction Cache
Arithmetic Logic Unit
Data Cache
Tightly Coupled Instruction Memory
Tightly Coupled Instruction Memory
Instruction Bus
Data Bus
Tightly Coupled Data Memory
Tightly Coupled Data Memory

The Nios II architecture describes an instruction set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions. A Nios II processor core is a hardware design that implements the Nios II instruction set and supports the functional units described in this document. The processor core does not include peripherals or the connection logic to the outside world. It includes only the circuits required to implement the Nios II architecture.

4.1.2.Microblaze Xilinx Microprocessor

The MicroBlaze is a soft processor core from Xilinx for use in Xilinx FPGAs. A soft processor is a processor created out of the configurable logic in an FPGA. The MicroBlaze is based on a RISC architecture very similar to the DLS architecture described in a popular computer architecture book by Patterson and Hennessy. It features a 5-stage pipeline, with most instructions completing in a single cycle. Both instruction and data words are 32 bits. The MicroBlaze can reach speeds of up to 150MHZ on the Virtex4 FPGA's family. This processor can connect to the OPB bus for access to a wide range of different modules. It can also communicate via the LMB bus for a fast access to local memory which is normally BRAM that are inside the FPGA.

Many aspects of the MicroBlaze can be configured at compile time owing to the configurable nature of FPGAs. Cache structure, peripherals, and interfaces can be customized to the application. In addition, hardware support for certain operations, such as multiplication, division, and floating-point arithmetic, can be added or removed.

Although lacking a Memory Management Unit, and thus unable to run full Linux, several operating systems have been ported to the MicroBlaze including uClinux and FreeRTOS.

Xilinx includes, as part of its MicroBlaze development package, a MicroBlaze GNU C Compiler (MB-GCC) which allows programmers to use the C programming language to write programs for the architecture.

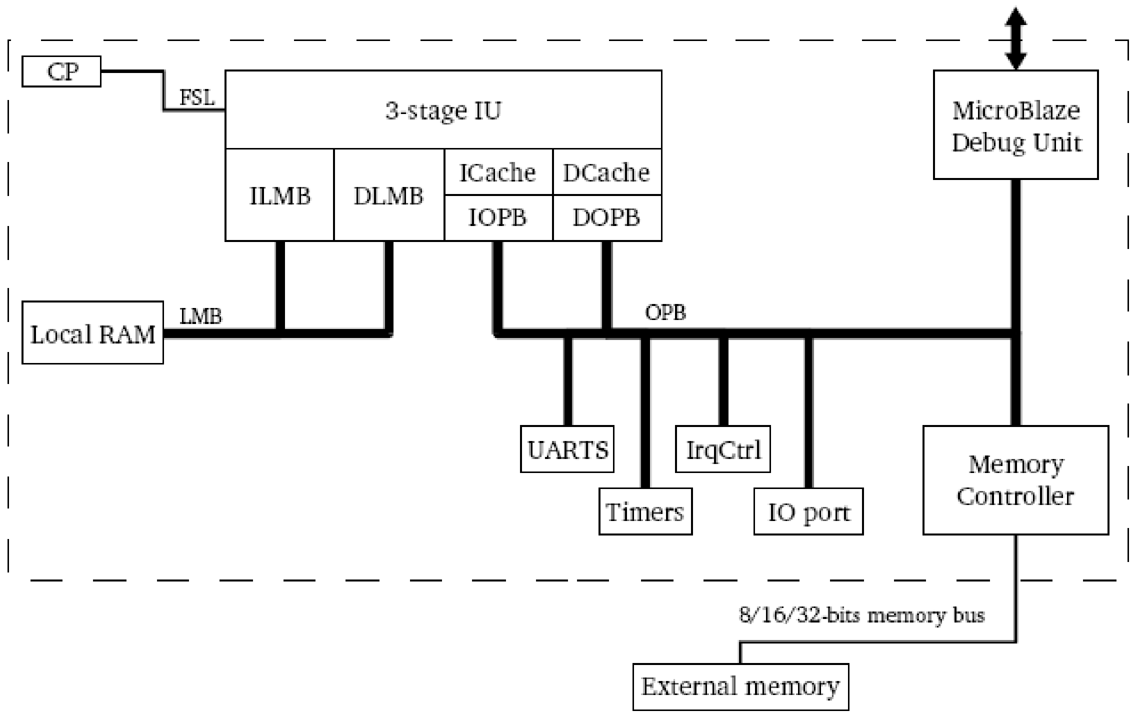


Figure 2: Overview of the MicroBlaze processor architecture.

4.1.3.Leon2 Gaisler Microprocessor

LEON2 is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The model is highly configurable, and particularly suitable for system-on-a-chip (SOC) designs. The full source code is available under the GNU LGPL license, allowing free and unlimited use in both research and commercial applications.

The LEON2 processor has the following features:

- SPARC V8 compliant integer unit with 5-stage pipeline
- Hardware multiply, divide and MAC units
- Interface to the Meiko FPU and custom co-processors
- Separate instruction and data cache (Hardvard architecture)
- Set-associative caches: 1 - 4 sets, 1 - 64 Kbytes/set. Random, LRR or LRU replacement
- Data cache snooping
- AMBA-2.0 AHB and APB on-chip buses
- 8/16/32-bits memory controller for external PROM and SRAM
- 32-bits PC133 SDRAM controller
- On-chip peripherals such as uarts, timers, interrupt controller and 16-bit I/O port
- Advanced on-chip debug support unit and trace buffer
- Power-down mode

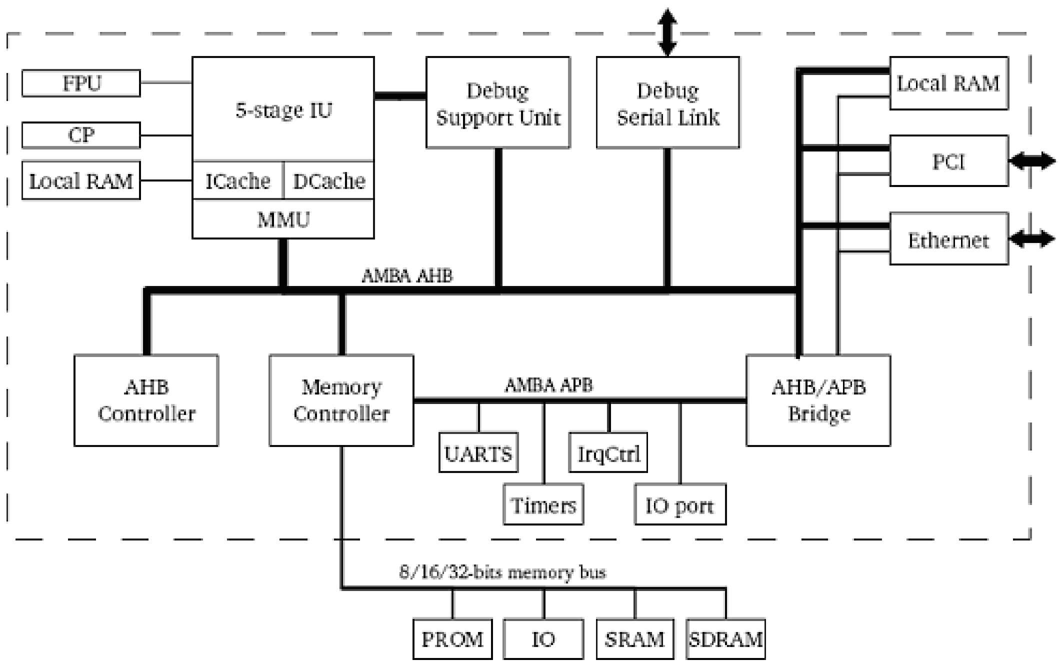


Figure 3: Overview of the LEON2 processor architecture

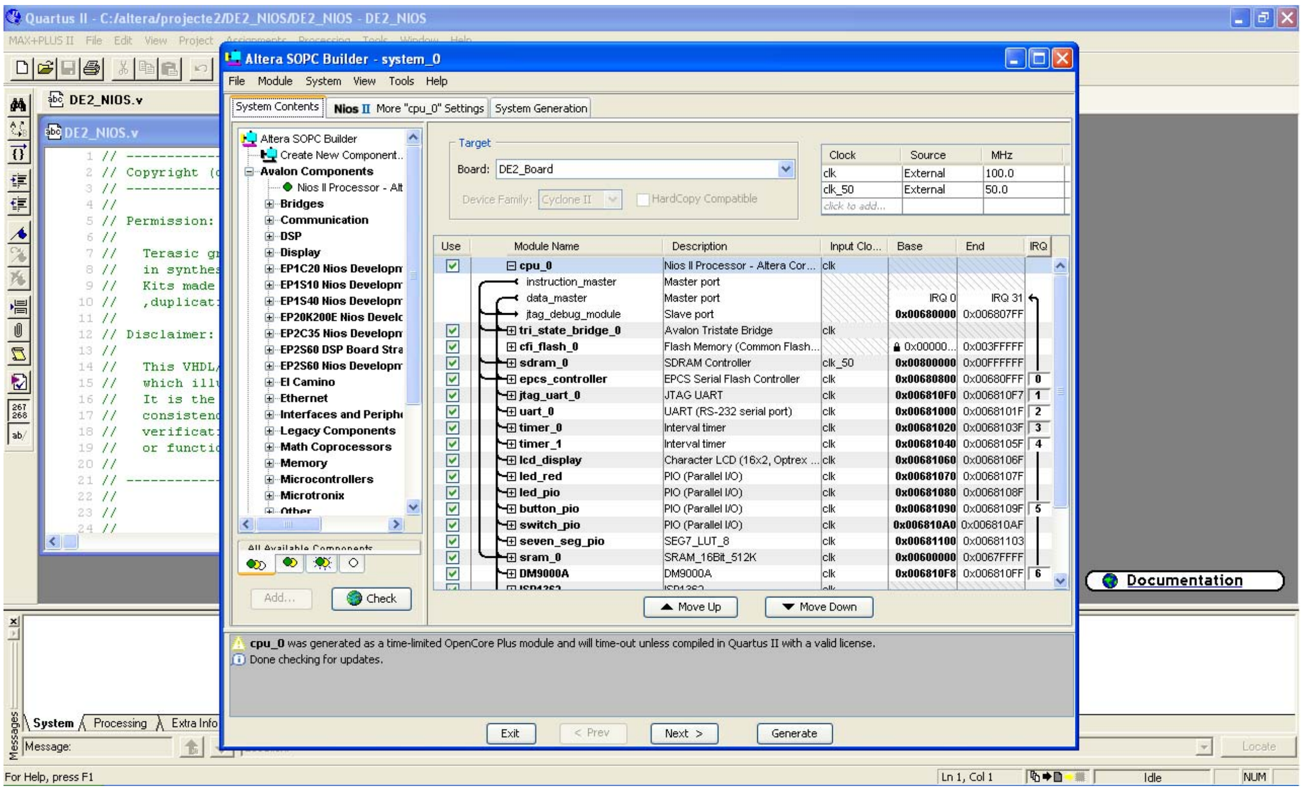
The processor is extensively configurable and can be efficiently implemented on both FPGAs and ASIC technologies. The only technology-specific mega-cells needed are ram cells for caches and register file.

4.2. Development tools

4.2.1. Nios II Altera Microprocessor

For programming, compile and development of the Nios II processor, we need to use the Altera Quartus II software and the Nios Development Kit.

The Altera® Quartus® II design software is a multiplatform design environment. It is a comprehensive environment for system-on-a-programmable-chip (SOPC) design. The Quartus II software includes solutions for all phases of FPGA and CPLD design. We can define with the SOPC builder the characteristics of our Nios II processor. Also with Quartus software we can define the inputs and outputs pins and all the necessary configuration settings for the Nios II microprocessor. Now we can see in the figure 4 an example of SOPC builder tool running on the Altera Quartus II software



Board: DE2_Board
Device Family: Cyclone II

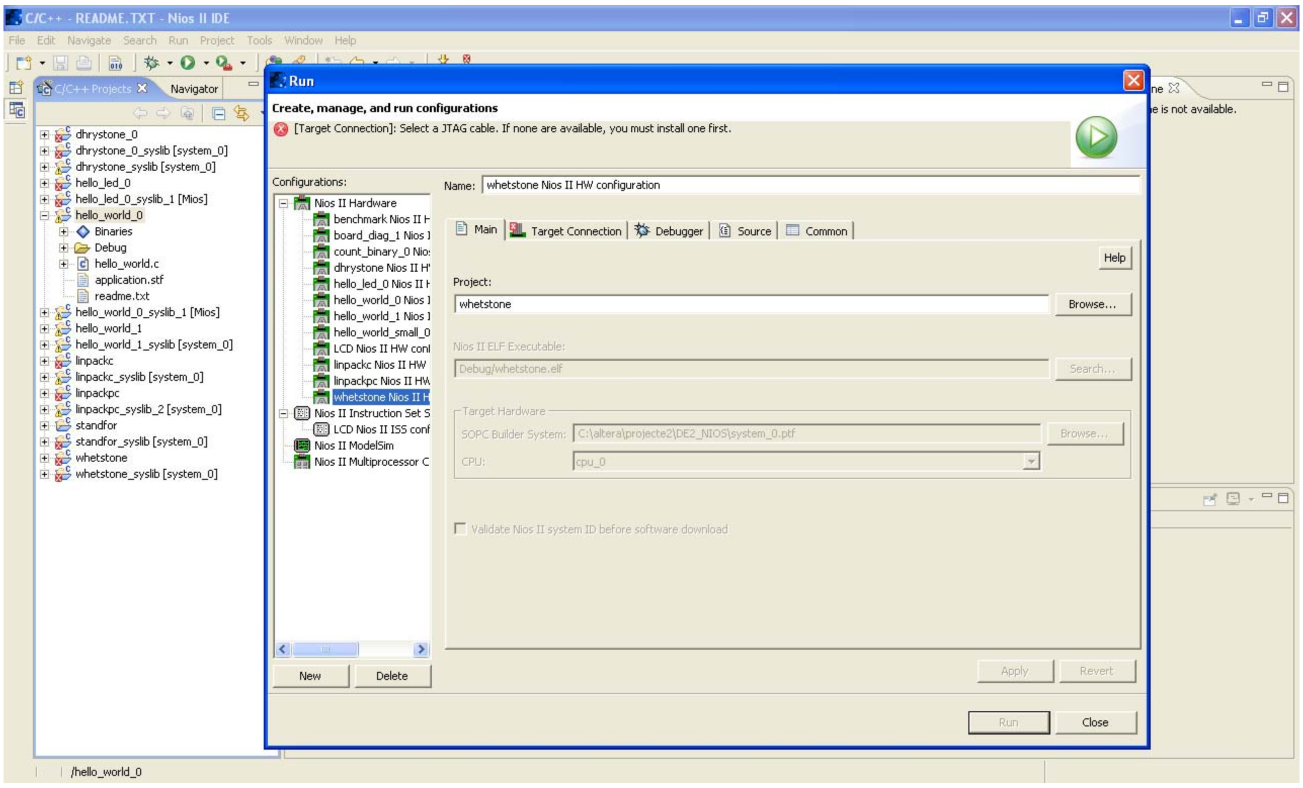
Clock	Source	MHz
clk	External	100.0
clk_50	External	50.0

Use	Module Name	Description	Input Clo...	Base	End	IRQ
✓	cpu_0	Nios II Processor - Altera Cor...	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00680000	0x006807FF	
✓	tri_state_bridge_0	Avalon Tristate Bridge	clk			
✓	cfi_flash_0	Flash Memory (Common Flash...		0x00000...	0x003FFFFF	
✓	sdram_0	SDRAM Controller	clk_50	0x00800000	0x00FFFFFF	
✓	epcs_controller	EPCS Serial Flash Controller	clk	0x00680800	0x00680FFF	0
✓	jtag_uart_0	JTAG UART	clk	0x006810F0	0x006810F7	1
✓	uart_0	UART (RS-232 serial port)	clk	0x00681000	0x0068101F	2
✓	timer_0	Interval timer	clk	0x00681020	0x0068103F	3
✓	timer_1	Interval timer	clk	0x00681040	0x0068105F	4
✓	lcd_display	Character LCD (16x2, Optrex ...	clk	0x00681060	0x0068106F	
✓	led_red	PIO (Parallel I/O)	clk	0x00681070	0x0068107F	
✓	led_pio	PIO (Parallel I/O)	clk	0x00681080	0x0068108F	
✓	button_pio	PIO (Parallel I/O)	clk	0x00681090	0x0068109F	5
✓	switch_pio	PIO (Parallel I/O)	clk	0x006810A0	0x006810AF	
✓	seven_seg_pio	SEG7_LUT_8	clk	0x00681100	0x00681103	
✓	sram_0	SRAM_16Bit_512K	clk	0x00600000	0x0067FFFF	
✓	DM9000A	DM9000A	clk	0x006810F8	0x006810FF	6

cpu_0 was generated as a time-limited OpenCore Plus module and will time-out unless compiled in Quartus II with a valid license.
Done checking for updates.

Figure 4: SOPC running on Quartus II software

The Nios® II Integrated Development Environment (IDE) is the primary software development tool for the Nios II family of embedded processors. All software development tasks can be accomplished within the Nios II IDE, including editing, building, and debugging programs. The Nios II IDE provides a consistent development platform that works for all Nios II processor systems. In the figure 5 we have a capture of the Nios II IDE software.


C/C++ - README.TXT - Nios II IDE

Run

Create, manage, and run configurations

[Target Connection]: Select a JTAG cable. If none are available, you must install one first.

Name: whetstone Nios II HW configuration

Project: whetstone

Nios II ELF Executable: Debug/whetstone.elf

SOPC Builder System: C:\altera\projecte2\DE2_NIOS\system_0.ptf

CPU: cpu_0

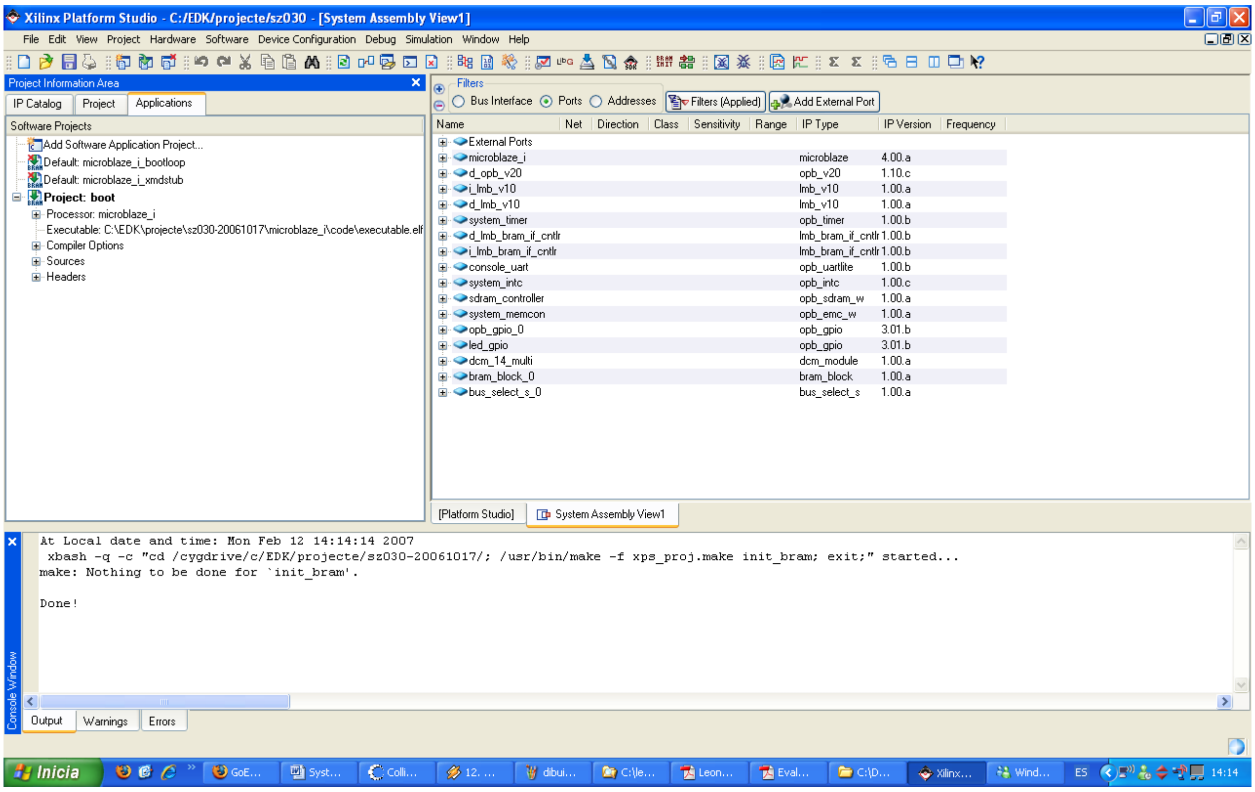
Validate Nios II system ID before software download

Figure 5: Nios II IDE software

With this software, we are ready to work with the Nios II processor.

4.2.2.Microblaze Xilinx Microprocessor

MicroBlaze hardware and software development is done using Xilinx Embedded Development Kit (EDK). EDK is a development environment where the hardware is instantiated as different IP-blocks connected via buses and signals. The software is developed on top of the generated libraries derivated from the hardware design. EDK focuses on system development closely integrated with a microprocessor. Both Xilinx's soft processor MicroBlaze and IBM's PowerPC 405, available as a hard macro in some FPGA circuits, are supported. EDK includes an integrated development environment (IDE) named Xilinx Platform Studio (XPS), which is a graphical GUI on top of the EDK. The figure number 6 shows the Xilinx Platform Studio.



Software Projects

- Add Software Application Project...
- Default: microblaze_i_bootloop
- Default: microblaze_i_xmdstub
- **Project: boot**
  - Processor: microblaze_i
  - Executable: C:\EDK\projecte\sz030-20061017\microblaze_i\code\executable.elf
  - Compiler Options
  - Sources
  - Headers

Name	Net	Direction	Class	Sensitivity	Range	IP Type	IP Version	Frequency
External Ports								
microblaze_i						microblaze	4.00.a	
d_opb_v20						opb_v20	1.10.c	
i_lmb_v10						lmb_v10	1.00.a	
d_lmb_v10						lmb_v10	1.00.a	
system_timer						opb_timer	1.00.b	
d_lmb_bram_if_cntlr						lmb_bram_if_cntlr	1.00.b	
i_lmb_bram_if_cntlr						lmb_bram_if_cntlr	1.00.b	
console_uart						opb_uartlite	1.00.b	
system_intc						opb_intc	1.00.c	
sdram_controller						opb_sdram_w	1.00.a	
system_memcon						opb_emc_w	1.00.a	
opb_gpio_0						opb_gpio	3.01.b	
led_gpio						opb_gpio	3.01.b	
dcm_14_multi						dcm_module	1.00.a	
bram_block_0						bram_block	1.00.a	
bus_select_s_0						bus_select_s	1.00.a	

```
At Local date and time: Mon Feb 12 14:14:14 2007
 xbash -q -c "cd /cygdrive/c/EDK/projecte/sz030-20061017/; /usr/bin/make -f xps_proj.make init_bram; exit;" started...
make: Nothing to be done for `init_bram'.

Done!
```

Figure 6: Xilinx Platform Studio

The design is mainly specified in the Microprocessor Hardware Specification file (MHS) and the Microprocessor Software Specification file (MSS). The MHS file instantiates the different hardware IP-blocks and connects them together. A make script is used to synthesize and route the hardware, compile the software libraries and applications, generate simulation models and bitstreams etc. XPS acts as a graphical front-end for the make scripts when compiling the software and implementing the hardware. The compiler used in the compilation is a modification of the GNU Compiler Collection tools (GCC).

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

4.2.3.Leon2 Gaisler Microprocessor

Altera Board Synthesis

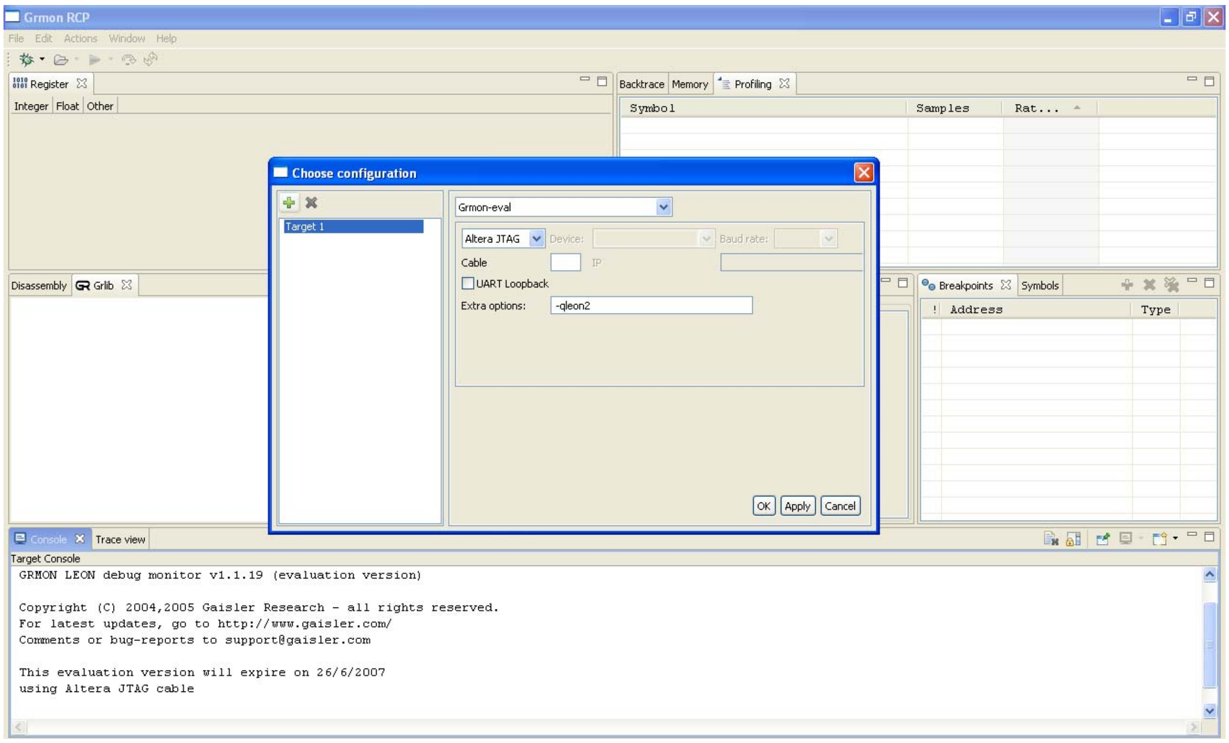
For the Altera board, we can use the Altera® Quartus® II design software

Xilinx Board Synthesis

Synthesis is available through Xilinx XST.

Compiler

There are two cross-compiler toolchains for the LEON2 processor, one for bare-C applications and one for RTEMS applications. Both of them use the GNU compiler toolchain and the GNU debugger. There is a graphical IDE for C/C++ development available as a plugin for Eclipse 3.0. The figure number 7 is the graphical tool for Windows made by Eclipse.



Grmon-eval

Altera JTAG Device: Baud rate:

Cable IP

UART Loopback

Extra options: -qleon2

```
GRMON LEON debug monitor v1.1.19 (evaluation version)

Copyright (C) 2004,2005 Gaisler Research - all rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com

This evaluation version will expire on 26/6/2007
using Altera JTAG cable
```

Figure 7: GRMON graphical tool for Windows

There is a graphical user interface for GRMON, which acts as a front-end for LEON2 debugging. This user interface is also provided as an Eclipse 3.0 plugin. In order to debug a LEON2 microprocessor system, GRMON can be used. GRMON can connect to a LEON2 system implemented on a physical FPGA

4.3.Benchmarks

4.3.1.Pros and cons regarding benchmarking

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

The ideal benchmark is measuring the performance of all the applications the system will ever run, but such a benchmark is difficult to construct. Most benchmarks include fragments from real applications, or algorithms comparable to algorithms in real applications, in an attempt to behave comparable to real applications. It is of great importance to know the differences between the underlying hardware and software when comparing benchmark results from different processors.

For this thesis, we have chosen two general benchmarks like the Dhrystone and the Whetstone because was difficult to know exactly what we have to measure, and which benchmark is really able to make it.

Maybe we could have chosen a more extensive benchmark, like the Stanford, which measures a lot of processor parameters with different tests, or also to create a specific benchmark only for our application to measure really which is the best processor, but maybe it could be made in another thesis, therefore we will include it in the future improvements paragraph.

4.3.2.Dhrystone 2.1

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

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

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

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

4.3.3. Whetstone

The Whetstone benchmark is a benchmark for evaluating the performance of computers. It was first written in Algol 60 in 1972 at the National Physical Laboratory in the United Kingdom and derived from statistics on program behaviour gathered on the KDF9 computer, using a modified version of its Whetstone Algol 60 compiler. The program's behaviour replicated that of a typical KDF9 scientific program and was designed to defeat compiler optimizations that would have adversely affected the accuracy of this model.

4.3.4. Dhrystone vs. Whetstone

The Dhrystone benchmark contains no floating point operations, thus the name is a pun on the then-popular Whetstone benchmark for floating point operations. The output from the benchmark is the number of Dhrystones per second (the number of iterations of the main code loop per second).

Both Whetstone and Dhrystone are synthetic benchmarks, meaning that they are simple programs that are carefully designed to statistically mimic some common set of programs. Whetstone, developed in 1972, originally strove to mimic typical Algol 60 programs based on measurements from 1970, but eventually became most popular in its Fortran version. Whetstone thus reflected the highly numerical orientation of computing in the 1960s.

5. Results

5.1.Benchmarks on Cyclone II Board

This section presents the benchmark results for the Dhrystone 2.1 benchmark, and the Whetstone benchmark on Cyclone II board.

5.1.1.Dhrystone on Cyclone II Board

Dhrystone tries to represent the result more meaningfully than MIPS (million instructions per second), because MIPS cannot be used across different instruction sets (e.g. RISC vs. CISC) for the same computation requirement from users. Thus, the main score is just Dhrystone loops per second. Another common representation of the Dhrystone benchmark is the DMIPS – Dhrystone MIPS- obtained when the Dhrystone score is divided by 1,757 (the number of Dhrystones per second obtained on the VAX 11/780, nominally a 1 MIPS machine). Next, we can see the results of the Dhrystone on the Altera board (figure 8)

	Nios II Altera Processor	**Leon2 Gaisler Processor**
Processor Frequency	50 MHz	50 MHz
Million instructions per second (MIPS)	263,55 MIPS	298,69 MIPS
Dhrystones per second (DMIPS)	150 DMIPS	170 DMIPS
Dhrystones iterations/second/MHz	3 iterations	3,4 iterations

Figure 8: Dhrystone results on the Altera Board

The Dhrystone 2.1 benchmark measures only integer performance and does not stress an 8 Kbytes cache much. The significance of the Dhrystone 2.1 benchmark results should not be taken too seriously since the benchmark is considered unreliable for today's processor architectures as stated in chapter 2.3.2 Dhrystone 2.1

5.1.2.Whetstone on Cyclone II board

The Whetstone benchmark originally measured computing power in units of *kilo-Whetstone Instructions per seconds* (kWIPS). Results for a variety of languages, compilers and system architectures have been obtained and modern workstations typically achieve more than 1 000 000 kWIPS (1 Giga-WIPS)

	Nios II Altera Processor	Leon2 Gaisler Processor
Processor Frequency	50 MHz	50 MHz
Loops	1000	1000
Duration	1430 sec	1236 sec
C converted Double Precision Whetstones	69.9 KIPS	75.6 KIPS

Figure 9: Whetstone results on the Altera Board

5.1.3.Conclusion

Looking the results of the Dhrystone and Whetstone benchmarks executed on the Cyclone II Board, we can arrive to the conclusion that the Leon2 processor is a bit faster than the Nios II processor in both tests.

With the Dhrystone, the Leon2 processor is able to make more Dhrystone iterations per second than the Nios II processor in the same conditions

In the second test, with the Whetstone, also the Leon2 processor is able to finish first the 1000 benchmark loops

With both tests, and like I have said before, now, we know that the Leon2 processor is faster than the Nios II processor.

Also, we have made the test at 50 MHz, but if the processor frequency had been 100 MHz, the processor will continue working, and the results will be practically the double of MIPS in the Dhrystone and the half of the time in the Whetstone, but in comparison, the Leon2 will be faster and more efficient than the Nios II processor.

5.2.Benchmarks on Xilinx Board

This section presents the benchmark results for the Dhrystone 2.1 benchmark, and the Whetstone benchmark on Xilinx board.

5.2.1.Dhrystone on Xilinx Board

	Microblaze Xilinx Processor	Leon2 Gaisler Processor
Processor Frequency	50 MHz	50 MHz
Million instructions per second (MIPS)	70,75 MIPS	81,23 MIPS
Dhrystones per second (DMIPS)	40,26 DMIPS	46,23 DMIPS
Dhrystones iterations/second/MHz	0,80 iterations	0,92 iterations

Figure 10: Dhrystone results on the Xilinx Board

We have spoken before about what means DMIPS and about the Dhrystone results. And now, we can see that the Leon2 is also more efficient than the Microblaze Xilinx processor.

5.2.2.Whetstone on Xilinx board

	Microblaze Xilinx Processor	**Leon2 Gaisler Processor**
Processor Frequency	50 MHz	50 MHz
Loops	1000	1000
Duration	2154 sec	2013 sec
C converted Double Precision Whetstones	60.6 KIPS	73.2 KIPS

Figure 11: Whetstone results on the Xilinx Board

Like in the Dhrystone, in the Whetstone the Leon2 processor is also faster than the Microblaze Xilinx processor.

5.2.3.Conclusion

We can say the same than before about the Nios II processor and the Leon2 processor, but in this case we are speaking about the Microblaze Xilinx processor and the Leon2.

The Leon2 is faster and more effective than the Xilinx processor in both tests at the same frequency

With both tests, and like I have said before, now, we know that the Leon2 processor is faster than the Microblaze processor.

Like we have spoken before, about the processor frequency, in this case, if the processor frequency was been 100 MHz, with the processors will appear the same than in the Nios II board. Always the Leon2 will be more efficient than the Microblaze Xilinx board.

6. Usability

It's time to evaluate how easy the system is to use. Aspects like how easy the available tools are to use, how much documentation there is and how well it is written will be reviewed together.

6.1.Nios II

6.1.1.Tools

The tools evaluated for the Nios II processor are the Altera Quartus II 6.0 Web Studio and Altera Nios II EDS 6.0

With the Altera Quartus II 6.0 Web studio, the user can configure the processor pretty easy with the SOPC builder.

The user only needs to create a new project, and open the Altera SOPC builder. Then he can choose the Nios II processor, the board that is going to use and the user can add or remove the components that he wants with a simple click.

Is an easy tool to configure our processor without to know too much about how to build one and compile.

After configure and compile the processor, only is needed to create a schematic file, or use one created before in the tutorials if the user don't know how to create one, to configure the project outputs and inputs. We have worked some times with Altera Quartus II, and for us are pretty easy to work with this. Then we don't know how problematic could be for a new user to learn to do it, we think that with the manuals it wouldn't be difficult.

Respect the pins, when the inputs and outputs are declared, to assign the correct pins to they is not difficult.

To program the board with the Nios II processor, only is needed to go to the programmer panel and to push start.

About the Nios II EDS 6.0, we never have worked with it before, but the first time that we use it was very easy to understand how it works. Also the software has a lot of C programs to run and test the boards.

The only that we need to start to create a C file are the processor files that have been created by the Quartus software. With these files the user can create the C program to run in it, and with a simple run, the software downloads the program in the board and starts it.

6.1.2.Documentation

We are surprised about the Altera documentation. We guessed that Altera have a lot of documentation and tutorials to work, but only looking for our processor and board, we have found a lot of tutorials, examples and files created no only for the Altera developers, also for a lot of students and people who work in this world.

Also, the Altera Quartus II software has a big database to help the developers to work with it, and solve a lot of problems without look for it in the Altera website or Altera foros.

6.2.Microblaze

6.2.1.Tools

The tools evaluated for the MicroBlaze processor are the ones included in the Embedded Development Kit v8.2, abbreviated EDK. EDK includes the GNU toolchain for compilation and debugging, a graphical user interface named Xilinx Platform Studio (XPS) for developing FPGA systems with a MicroBlaze or PowerPC processor, and additional tools.

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

We never have used the Xilinx software before, and our first impression is that it looks like more professional and complicated than the Altera software. Also, the options are pretty different and it is not easy to start to use this kind of software after to get use to working the Altera software.

A positive aspect of XPS is that it includes a user friendly wizard for building a base system. This wizard works fine for the supported FPGA development boards, but if the user wants to use an unsupported FPGA development board like the GR-PCI-XC2V [PE03], some additional modification has to be done.

The user does not necessary have to use the wizard for building a base system, but this requires deeper knowledge of the hardware being used.

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

The source code for the MicroBlaze processor is proprietary and is not included45 in the EDK. The processor being closed source requires that the documentation follows a high standard. The closed source code makes the simulation and debugging more difficult, because it is difficult to know what is really happening inside the processor.

6.2.2.Documentation

The available documentation for MicroBlaze and for the included tools is very informative. There is also an extensive answer database available at the Xilinx website [XILWEB], where users can submit questions.

On the contrary than the Altera documentation, for us have been more difficult to find how to use the Xilinx hardware and software.

Like we have said before, there is a big database of answers at the Xilinx website but the most part of this place is for experimental users and weird problems. And for us, beginner uses who don't know how to start to program a processor with a not common Xilinx board have been difficult to found easy manuals, tutorials and help.

6.3.Leon2

6.3.1.Tools

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

When we have the VHDL code, we have used the Quartus software to compile and program the Nios II processor and the Xilinx software for the Xilinx board.

To download and run the software in both boards, we can use the GRMON tool, which can works on Linux, Cygwin and also there is a graphical interface for windows made by eclipse.

6.3.2.Documentation

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

6.4.Conclusion

Once time we have worked with all the different software, for us the easiest and best software and documentation is the Altera. Maybe we are influenced by to have worked before with it, but the Xilinx software is harder to know what is happening and what you are doing.

Also, the Leon2 scripts to create the VHDL files are very easy to use but, you have to know what you are doing in every moment, and have a big knowledge of the hardware.

Also, in the Leon2 scripts we can’t change the name of the parameters and other options, and we have to open the VHDL code to configure it.

7. Configurability

7.1.Nios II

In the Nios II processor we can configure absolutely all the parameters that we want with the SOPC builder. And like we have said before, is pretty easy to change these options.

7.2.Microblaze

The amount of configurable parameters for MicroBlaze is less than for the LEON2 and Nios II processors

7.3.Leon2

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

7.4.Conclusion

The graphical Altera tool (SOPC builder) is the easiest way to configure a processor and its parameters, therefore for us; the Altera software is the most configurable and easiest way to configure a processor.

8. Summary

The three processors have been compared on performance, configurability and usability. And now, like we have told before in the conclusions in each paragraph, we are going to write a global summary about this.

- The LEON2 processor shows the best performance both in terms of benchmark results and in terms of performance per clock cycle.
- Nios II is the most configurable processor, while MicroBlaze and Leon2 have less configuration options.
- Also, Altera has the most extensive documentation for the processor. The documentation for Xilinx is extensive too, but is more difficult to find what you are looking for, while the documentation for Leon2 is sufficient. The three processors, Nios II, MicroBlaze and LEON2 have qualified support.
- The three, Nios II, LEON2 and MicroBlaze have satisfying configuration tools, which makes configuration easy. The LEON2 configuration tool has help readily available for all configuration options, the Altera configuration tool, is the easiest to use, while the options in MicroBlaze are well documented but more difficult to access.

9. *Discussion*

9.1.Obstacles

For us, that we never have worked programming processor, have been pretty difficult to start to learn how they work, and how you have to program them.

Also, we have had to learn how to program three different processors, with their three different software and their problems and difficulties.

To start with the Altera has been a good choice, because we knew how to handle its software, and its way to work, but when we have been to work with the Xilinx software it has been difficult. Every thing is different than the Altera, and the way to work, etc…is different.

When we started to work with the Leon2, we had the experience of the two firsts processors, and it has been easier to change the way of thinking and work with it.

We have spoken about the documentation before, but the tutorials and manuals found in different private project webs have been very useful. To choose a thesis about a subject in which there are a lot of people working, interestedly and disinterestedly has been really grateful, and in the private groups of Google and yahoo, where we have written a lot of emails asking for help, a lot of people have answered us, always quickly and with a lot of respect.

9.2.Future improvements

There are several improvements which can be done to achieve more comparable results.

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

Also, a specific benchmark for the board, or what we want to measure, could have been made for improving the comparison among the processors

10.References

[1] Altera Web
http://www.altera.com/

[2] Cyclone II Altera board Web
http://www.altera.com/products/devices/cyclone2/cy2-index.jsp

[3] Altera tutorials Web
http://www.altera.com/literature/lit-cyc2.jsp

[4] Nios II processor Web
http://www.altera.com/literature/lit-nio2.jsp

[5] Xilinx Web
http://www.xilinx.com/

[6] Microblaze Web
http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze

[7] Microblaze tutorials Web
http://www.ii.uam.es/~igonzale/recursos/Tutorial_MicroBlaze_uCLinux_jcra2004.pdf

[8] Suzaku Web
http://www.atmark-techno.com/en/products/suzaku/suzaku-s

[9] Suzaku tools
http://download.atmark-techno.com/

[10] Gaisler Web
www.gaisler.com

[11] Leon2 processor Web
http://www.gaisler.com/cms4_5_3/index.php?option=com_content&task=view&id=12&Itemid=52

[12] Grmon tool
http://www.gaisler.com/cms4_5_3/index.php?option=com_content&task=view&id=39&Itemid=128

[13] Opencores Web
http://www.opencores.org/

[14] Cygwin
http://www.cygwin.com/

11.Attachments

11.1.Leon2 on Altera board Tutorial

Introduction

This tutorial is created to help you design your first embedded system with a Leon SPARC softcore processor. Before you proceed you must have the following software and hardware:

Software:

- Quartus II 3.*
- Cygwin **(installed with devel. tools option)**
- WinRAR
- Synplify PRO 7.2 *[Optional]*

Hardware:

- windows-PC
- Linux-PC
- Altera FPGA development board

Chapter I is a general chapter where you can learn how to define your Leon SPARC processor system.

In chapter II, there are possible 2 tracks to follow:

TRACK 1: completing the design with Quartus II:
Easiest way to set up a Leon SPARC microprocessor system on an FPGA. Synthesis, place & route is
done with Altera Quartus II.

TRACK 2: completing the design with a third party synthesis tool
An alternative way to set up the Leon SPARC microprocessor system on an FPGA. Synthesis is done
in the third party tool (in this case SYNPLIFY PRO), place and route is done in Altera Quartus.

Chapter III describes how to download the embedded system into the FPGA and how to run it.

I Defining the embedded system

I. Download and installation of Leon 2 source files:

A. Download leon2-1.0.15.tar.gz from http://www.gaisler.com

B. Unzip the tar.gz archive to e.g. **c:\leon2** (From now on we will suppose that this is your working directory. If you choose another one, you'll have to replace "c:\leon2" in this tutorial with your own directory.)

C. To get quick access to the Leon configuration utility (which makes use of cygwin), create 2 files in the same directory:

leon.bat
```
@C:\Cygwin\bin\bash.exe --rcfile c:\leon2\leon.bashrc
```

leon.bashrc
```
PATH="/bin:/contrib/bin: /cygdrive/c/leon2/leon"
cd /cygdrive/c/leon2
```

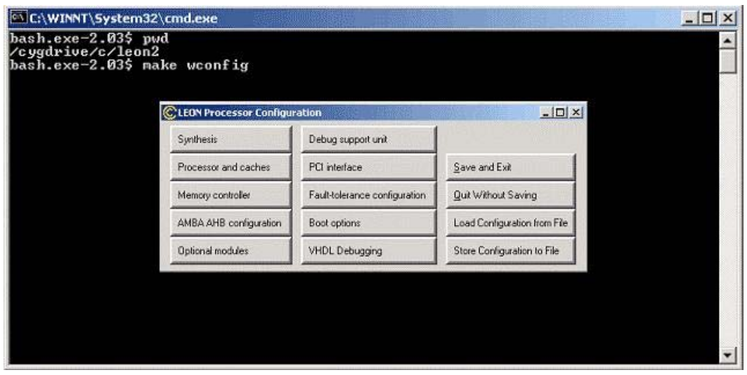
II. Configure the processor

A. Start cygwin by double-clicking on leon.bat

B. Normally you should be in the correct directory (you can check it with the "pwd"-command)

C. Start the Leon configuration utility by typing "**make xconfig**" (figure 1)

Figure 1: start the Leon configuration-utility

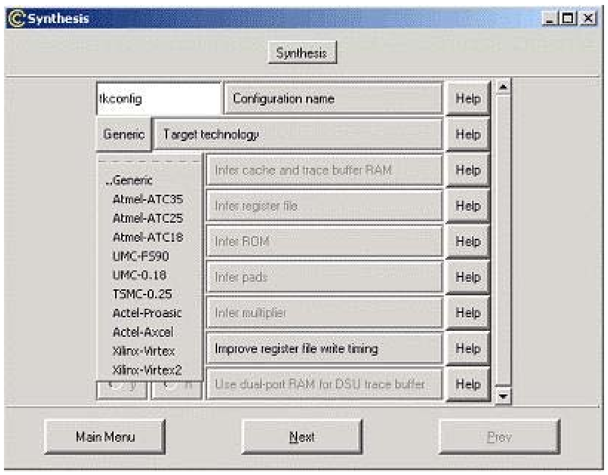


```
bash.exe-2.03$ pwd
/cygdrive/c/leon2
bash.exe-2.03$ make wconfig
```

D. In the Leon configuration utility you can define your own processor system. Below you can find a short description of the necessary configuration you've got to do for building your own embedded system on an FPGA. Most of the settings are correct by default, to make sure however that everything is configured correctly, run trough the following steps.

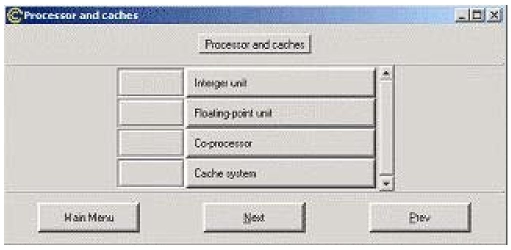
  i. Click on the "synthesis"-option (figure 2), where you'll be able to define your target technology. In your case this will be "generic" for all Altera targets. All other parameters shouldn't be changed unless you know what you're doing.

Figure 2: The synthesis options window



  ii. In the "processor and caches" window (figure 3) you can configure the integer unit and the cache system. For FPGA's this means the following (to avoid possible timing problems):

Figure 3: The processor and caches window

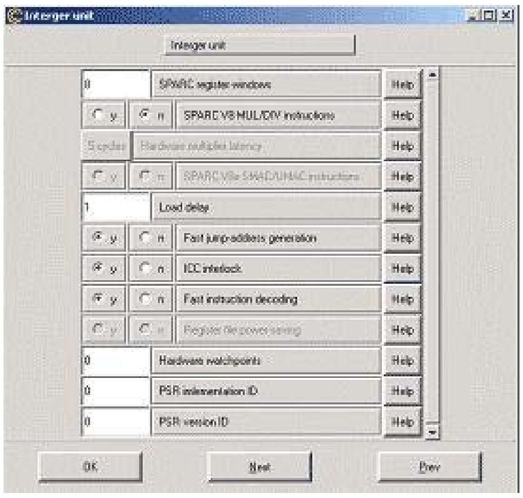


a) Disable the floating point unit and coprocessor unit.

b) Configure the integer unit (figure 4):

- Make sure "fast jum-address" is selected
- Make sure "ICC interlock" is selected
- Make sure "fast instruction decoding" is selected

Figure 4: The integer unit window



Integer unit

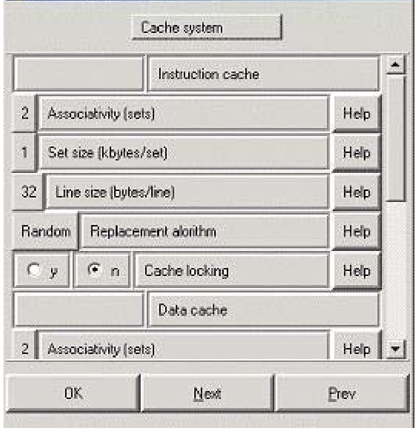
Integer unit

0	SPARC register windows	Help
y [✓ n]	SPARC V8 MUL/DIV instructions	Help
5 cycles	Hardware multiplier latency	Help
y n	SPARC V8e SMAC/UMAC instructions	Help
1	Load delay	Help
[✓ y] n	Fast jump-address generation	Help
[✓ y] n	ICC interlock	Help
[✓ y] n	Fast instruction decoding	Help
y n	Register file power saving	Help
0	Hardware watchpoints	Help
0	PSR implementation ID	Help
0	PSR version ID	Help

c) Configure the data and instruction cache system (figure 5). Depending on the amount of block RAM available, you can use different values. The following are example settings (instruction and data cache have the same configuration):

- Associatively on 2 sets
- Set size on 1kbyte/set
- Line size on 32 bytes/line
- Random replacement algorithm

Figure 5: The cache system window



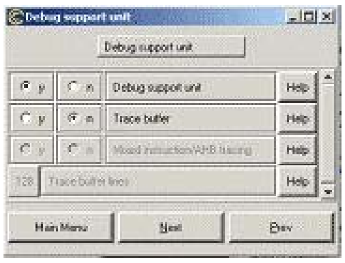
Cache system

Cache system

	Instruction cache	
2	Associativity (sets)	Help
1	Set size (kbytes/set)	Help
32	Line size (bytes/line)	Help
Random	Replacement alorithm	Help
y [✓ n]	Cache locking	Help
	Data cache	
2	Associativity (sets)	Help

iii. Turn on the “debug support unit” in the “debug support unit” window (figure 6)

Figure 6: The debug support unit window



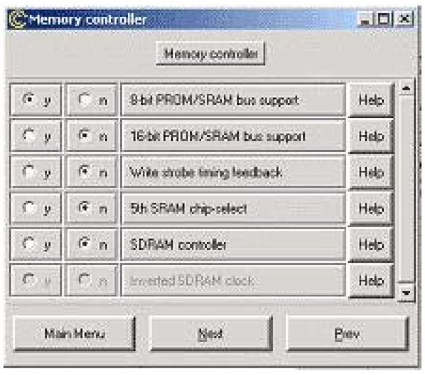
Debug support unit

[✓ y] n Debug support unit
y [✓ n] Trace buffer
y n Mixed instruction/AHB tracing
128 Trace buffer lines

iv. Now you’ve got to configure your “memory controller” (figure 7). Depending on your hardware, use the following settings:

- You’ve got SRAM:
  - Select 8/16 bit prom/scram bus depending on the data bus width to your SRAM/PROM. For 32 bit access select none.
- You’ve got SDRAM
  - Make sure “SDRAM controller” is selected
  - Make sure you selected the "inverted SDRAM clock" option when using an FPGA. Otherwise you could create an unstable system caused by clock-scew
- You want to use on-chip RAM (look at step (v))

Figure 7: The memory controller window

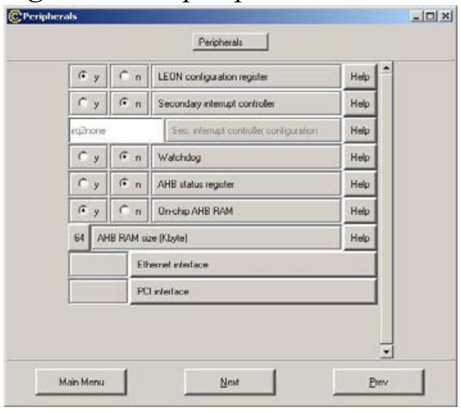


Memory controller

[✓ y] n 8-bit PROM/SRAM bus support
y [✓ n] 16-bit PROM/SRAM bus support
y [✓ n] Write strobe timing feedback
y [✓ n] 5th SRAM chip-select
y [✓ n] SDRAM controller
y n Inverted SDRAM clock

v. In the “peripherals” window (figure 8), make sure that the Leon configuration register is selected. If you need on chip RAM (AHB-RAM), select the On-chip AHB RAM option, and define its size. Take care though that you don't exaggerate its size, as there is a very limited amount of RAM available in the FPGA, and the cache also makes use of it. Disable the Ethernet and PCI interface.

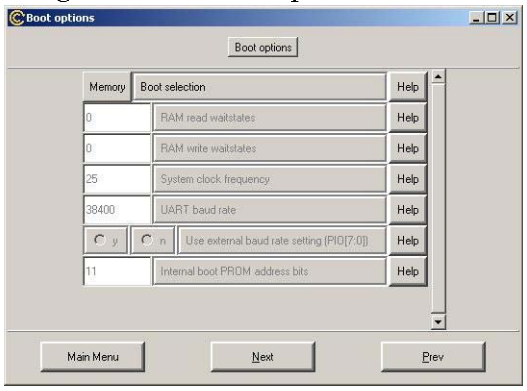
Figure 8: The peripherals window


Peripherals

[✓ y]	n	LEON configuration register	Help
y	[✓ n]	Secondary interrupt controller	Help
irq2none		Sec. interrupt controller configuration	Help
y	[✓ n]	Watchdog	Help
y	[✓ n]	AHB status register	Help
[✓ y]	n	On-chip AHB RAM	Help
64		AHB RAM size (Kbyte)	Help
		Ethernet interface	
		PCI interface	

vi. In the "boot options" window (figure 8) you can select where you want your device to boot from. To avoid synthesis problems, select "memory" as boot location. This way the Leon will try to boot from address location 0x0, probably there will no valid data on address 0x0, but that's no problem as you'll still be able to connect to the debugging unit and download your executable manually.

Figure 9: The boot options window


Boot options

Memory	Boot selection	Help
0	RAM read waitstates	Help
0	RAM write waitstates	Help
25	System clock frequency	Help
38400	UART baud rate	Help
y / n	Use external baud rate setting (PIO[7:0])	Help
11	Internal boot PROM address bits	Help

vii. Don't change anything in the "VHDL-debugging" window.

E. The configuration of the LEON processor system is now complete. In the main window, push on "save and exit". This will save your configuration, after this you'll het the message that you have to type the "make dep" command to generate the proper VHDL configuration files. Type "**make dep**" at the command prompt, and hit the enter button. The VHDL configuration file will now be generated automatically.

III. Adjust the top file to your needs

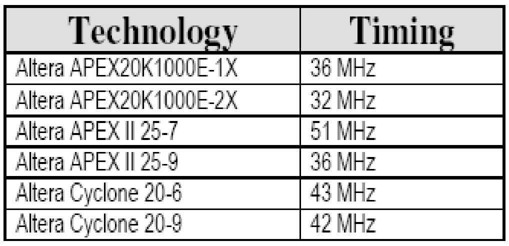
A. Create a backup copy of the file "c:\leon2\leon\leon.vhd".

B. Open "c:\leon2\leon\leon.vhd" and adjust the in & out ports to correspond with your pinning file*:

i. Critical basic signals are:

- resetn: active low reset signal, if possible connect it with a button, otherwise write a little reset-statemachine in the Leon entity that drives the signal low for some clock cycles.
- clk: The clock signal, if needed divide/multiply the incoming clock with your own state machine, or use the Altera PLL's. Make sure however that your target component is able to handle the clock speed you selected. Table 1 gives you an idea of what clock speed is archievable on different Altera hardware components
- dsutx: debugging unit transmit UART port
- dsurx: debugging unit receive UART port
- dsuen: debugging unit enable, this active high input signal enables the debugging unit, a logical '1' has to be provided to activate the debugging unit
- bexcn: memory mapped I/O bus exception, this active low input indicates a memory problem, a logical '1' has to be provided to avoid exceptions
- brdyn: memory mapped I/O bus ready, this active low input indicates that the access to a memory mapped I/O area can be terminated on the next rising clock edge, a logical '1' has to be provided when no memory mapped I/O is used
- dsubre: debugging unit break enable, this active high input will generate break condition and put the processor in debug mode, in this case a logical '1' has to be provided.

Table 1: Leon clockspeed



Technology	Timing
Altera APEX20K1000E-1X	36 MHz
Altera APEX20K1000E-2X	32 MHz
Altera APEX II 25-7	51 MHz
Altera APEX II 25-9	36 MHz
Altera Cyclone 20-6	43 MHz
Altera Cyclone 20-9	42 MHz

ii. Optional signals are:

- errorn: active low output which indicates error condition, connect to a led
- dsuact: active low output that indicates whether the debugging unit is active, connect to a led
- pio[15]: Transmit signal of standard I/O in/out UART
- pio[14]: Receive signal of standard I/O in/out UART

iii. When you use SRAM, following signals have to be connected:

- data: data path
- address: address path
- ramsn: RAM chip select
- ramoen: RAM output enable
- rwen: RAM write enable

iv. When you use SDRAM [PC100/PC133 compatible], following signals have to be connected:

- data: data path
- address: address path ([14:2] is used as address, [16:15] are the bank select signals)
- sdcke: clock enable
- sdcsn: chip select
- sdwen: write enable
- sdrasn: row address strobe
- sdcasn: column address strobe
- sddqm: data I/O mask
- sdclk: SDRAM clock

C. Save all changes to the Leon top file

HINT: * The Leon topfile makes use of "pads" to route its signals to the outside world. Depending on the selected target technologies (Atmel, generic,…) other types of pads are inferred. The 3 types of paths you'll encounter are: [pad = output pin, d/q/en: Leon in/output]

- outpad(d: in st_logic, pad : out std_logic)
- inpad(pad: in st_logic, q : out std_logic)
- iopad(d: in st_logic, en : in std_logic, q : out std_logic, pad : inout std_logic)
- With this knowledge you can easily create extra pads (e.g. and extra addresspad to connect your SRAM and SDRAM simultaneously).

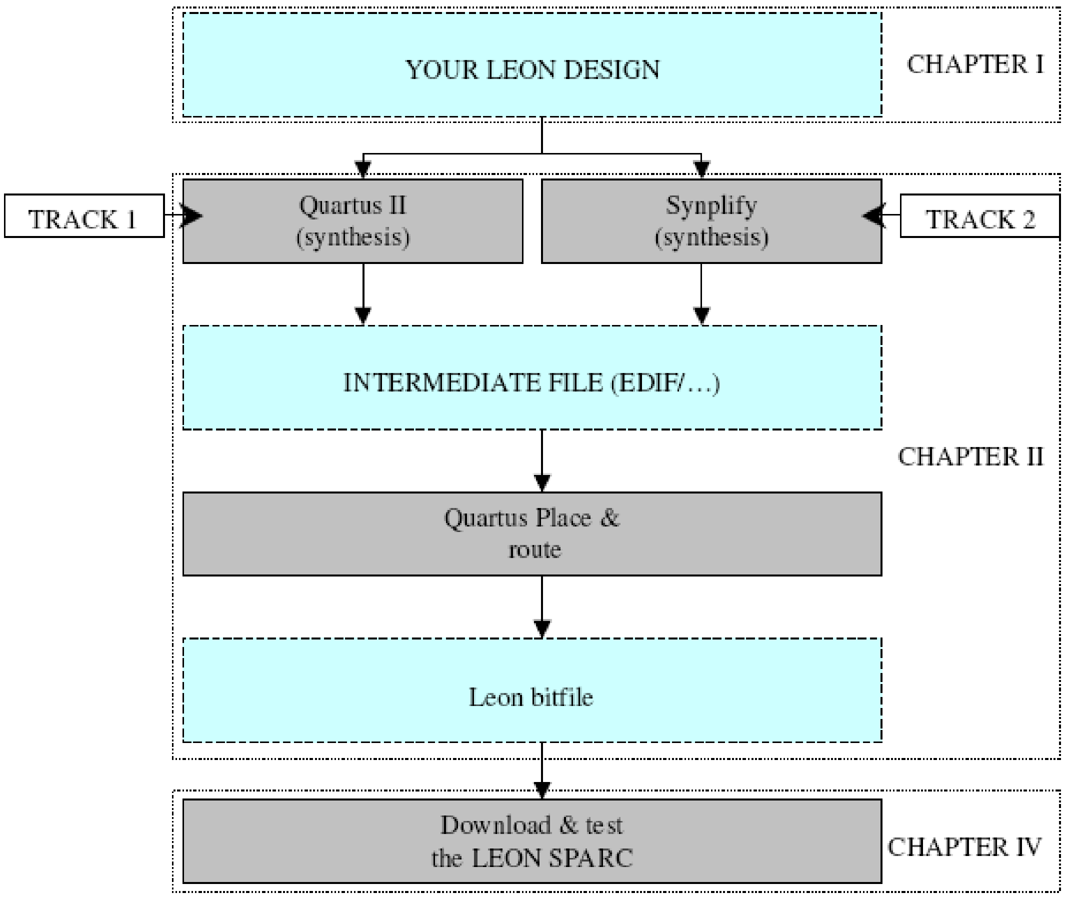
II Synthesis, place & route, generating the bitstream

As mentioned in the introduction, there are 2 possible tracks to complete your design (figure 10):

TRACK 1: completing the design with Altera Quartus:
Easiest way to set up a Leon SPARC microprocessor system on an FPGA. Synthesis, place & route is
done with Altera Quartus.

TRACK 2: completing the design with a third party synthesis tool
An alternative way to set up the Leon SPARC microprocessor system on an FPGA. Synthesis is done
in the third party tool (in this case SYNPLIFY PRO), place and route is done in Altera Quartus.

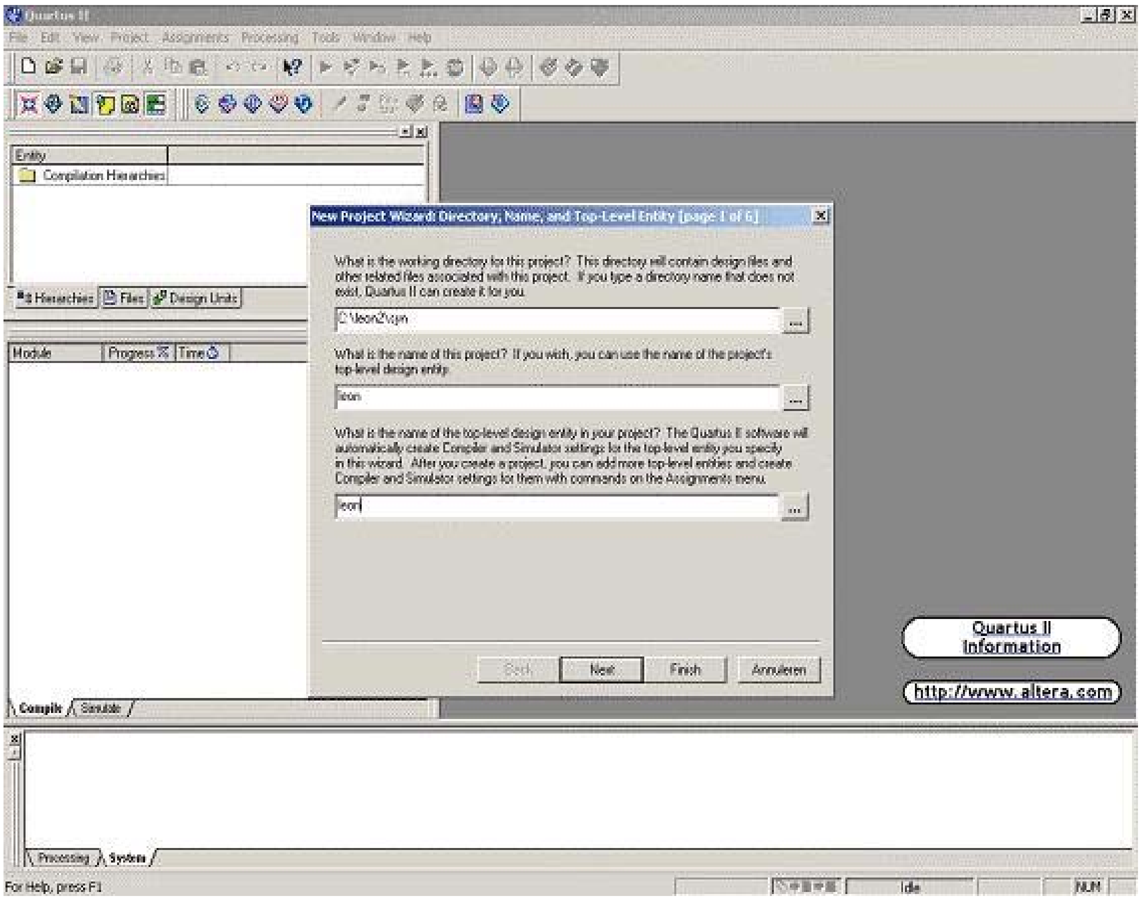
Figure 10: The design flow



TRACK 1 Complete your design with Quartus II

1) Start Quartus II (*Start ‰ programs ‰ Altera ‰ Quartus II 2.2*)
2) Create a new quartus project (*File ‰ new project wizard*) (figure 11)

Figure 11: Create a Quartus II project



New Project Wizard: Directory, Name, and Top-Level Entity [page 1 of 6]

What is the working directory for this project? This directory will contain design files and other related files associated with this project. If you type a directory name that does not exist, Quartus II can create it for you.

C:\leon2\syn

What is the name of this project? If you wish, you can use the name of the project's top-level design entity.

leon

What is the name of the top-level design entity in your project? The Quartus II software will automatically create Compiler and Simulator settings for the top-level entity you specify in this wizard. After you create a project, you can add more top-level entities and create Compiler and Simulator settings for them with commands on the Assignments menu.

leon

Back Next Finish Annuleren

3) Select the directory "c:\leon2\syn\" as working directory, name of the project is "Leon", top level entity is "Leon", click on "next"

4) Add the following vhdl files from the "c:\leon2\leon\" directory [starting with the left column] and
click on next:

amba.vhd	tech_atc18.vhd	dsu_mem.vhd	div.vhd	uart.vhd
target.vhd	tech_atc25.vhd	ahbmst.vhd	meiko.vhd	ahbram.vhd
device.vhd	tech_atc35.vhd	dcom_uart.vhd	fpu_lth.vhd	apbmst.vhd
config.vhd	tech_fs90.vhd	dcom.vhd	fpu_core.vhd	wprot.vhd
sparcv8.vd	tech_umc18.vhd	cachemem.vhd	iu.vhd	ahbstat.vhd
mmuconfig.vhd	tech_virtex.vhd	icache.vhd	proc.vhd	ahbarb.vhd
iface.vhd	tech_virtex2.vhd	dcache.vhd	lconf.vhd	mcore.vhd
macro.vhd	tech_tsmc25.vhd	acache.vhd	irqctrl.vhd	leon.vhd
ambacomp.vhd	tech_proasic.vhd	cache.vhd	sdmctrl.vhd	yourpinningfile.csf
bprom.vhd	tech_axcel.vhd	rstgen.vhd	mctrl.vhd	
multlib.vhd	tech_map.vhd	fpulib.vhd	ioport.vhd	
tech_generic.vhd	dsu.vhd	mul.vhd	timers.vhd	

5) Select “none” as design entry synthesis tool, and click on “next”.

6) Select your component’s target family, and select “yes” to select the component itself directly. Click on “next”

7) Select your target component, click on “next”

8) Press on “finish” to complete the project setup.

9) Start the compilation “processing ‰ start compilation” to synthesize and place & route the design and create the bitstream.

10) In the “message”-window, look for the message indicating the archived clock speed.

TRACK 2 Complete your design using Synplify.

1) Start synplify. (*start* ‰ *programs* ‰ *synplicity* ‰ *synplify pro*)

2) Open the existing Leon project (*File* ‰ *open project* ‰ *existing project*)

3) Browse to “c:\leon2\syn” and open the “Leon” project.

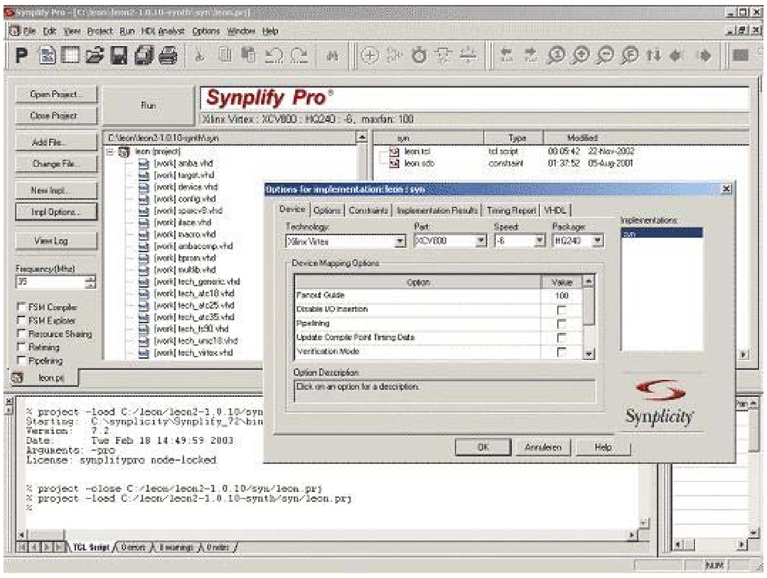
4) Add your own vhdl files to the project by clicking on “add file”. Make sure your hdl files are
synthesized before the “Leon” top entity by optionally altering the synthesize order.

5) Configure the implementation options (*project* ‰ *Implementation Options*) (figure 15)

- *The device tabs:*
  Select your target component (Technology/part/speed/package)
- *The options tab:*
  Not a single option may be selected, as this could result into netlists with a non-working Leon.
- *The constraints tab:*
  Enter your desired clockspeed
- *Implementation results*
  Make sure the resulting format is “vqm”
- *Timing report:*
  These values should be correct
- *Vhdl-tab:*
  Make sure the top level entity is “Leon”

Press on “OK”

Figure 15: Synplify implementation options



Xilinx Virtex : XCV800 : HQ240 : -6, maxfan: 100

Options for implementation: leon : syn

Device | Options | Constraints | Implementation Results | Timing Report | VHDL

Technology: Xilinx Virtex   Part: XCV800   Speed: -6   Package: HQ240

Implementations: syn

Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	☐
Pipelining	☐
Update Compile Point Timing Data	☐
Verification Mode	☐

Option Description: Click on an option for a description.

Frequency (Mhz): 35

☐ FSM Compiler
☐ FSM Explorer
☐ Resource Sharing
☐ Retiming
☐ Pipelining

syn	Type	Modified
leon.tcl	tcl script	00:05:42 22-Nov-2002
leon.sdc	constraint	01:37:52 05-Aug-2001

```
C:\leon\leon2-1.0.10-synth\syn
leon [project]
    [work] amba.vhd
    [work] target.vhd
    [work] device.vhd
    [work] config.vhd
    [work] sparcv8.vhd
    [work] iface.vhd
    [work] macro.vhd
    [work] ambacomp.vhd
    [work] bprom.vhd
    [work] multlib.vhd
    [work] tech_generic.vhd
    [work] tech_atc18.vhd
    [work] tech_atc25.vhd
    [work] tech_atc35.vhd
    [work] tech_fs90.vhd
    [work] tech_umc18.vhd
    [work] tech_virtex.vhd
```

```
% project -load C:/leon/leon2-1.0.10/syn
Starting:  C:\synplicity\Synplify_72\bin
Version:   7.2
Date:      Tue Feb 18 14:49:59 2003
Arguments: -pro
License:   synplifypro node-locked

% project -close C:/leon/leon2-1.0.10/syn/leon.prj
% project -load C:/leon/leon2-1.0.10-synth/syn/leon.prj
%
```

TCL Script / 0 errors / 1 warnings / 0 notes

6) Press on the famous "Run" button to start synthesis, wait until synthesis is complete

7) Now press on the "view log" button, and check whether your desired clock speed has been met. If not, track back and change your clock divider to a lower value.

8) Now you have to place & route your design in Quartus II.

   a. Start Quartus II (*Start* ‰ *programs* ‰ *Altera* ‰ *Quartus II 2.2*)

   b. Create a new quartus project (*File* ‰ *new project wizard*) (figure 11)

   c. Select the directory "c:\leon2\syn\" as working directory, name of the project is "Leon", top level entity is "Leon", click on "next"

   d. Add the "leon.vqm" file generated by synplify

   e. Select "synplify" as design entry synthesis tool, and click on "next".

   f. Select your component's target family, and select "yes" to select the component itself directly. Click on "next"

   g. Select your target component, click on "next"

   h. Press on "finish" to complete the project setup.

9) Start the compilation "processing ‰ start compilation" to synthesize and place & route the design and create the bitstream.

10) In the "message"-window, look for the message indicating the achieved clock speed.

III Download and test the Leon SPARC

1) Open the Quartus programmer (*Tools* ‰ *Quartus*)

a. Make sure you connected your download cable correctly (e.g. byteblaster,…), make sure your board is powered on

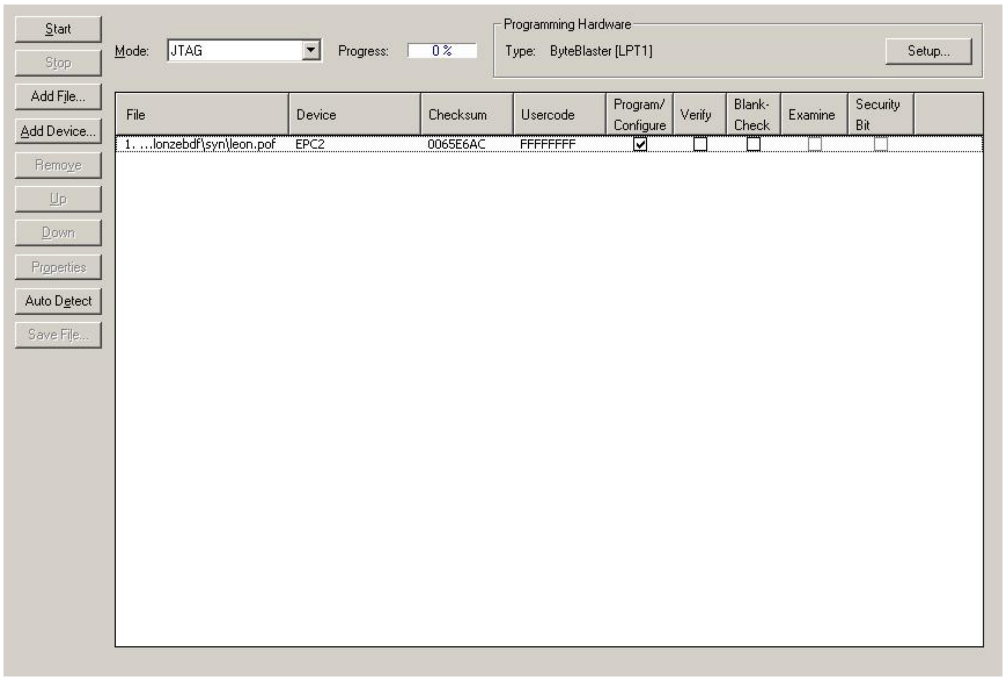
b. Push on the "setup" button to configure your hardware setup.

c. After the hardware configuration, push on the "auto detect" button. This will initialize the device-chain and display it.

d. Remove the device that has to be programmed, and add the generated target file ("leon.sof" or "leon.pof") to the chain by pushing on the button "add file". After the file has been added, make sure the order of the different devices is still correct.

e. Select the program/configure box of your device in the "Program/configure"-column (figure 16)

Figure 16: Quartus programmer



Mode: JTAG Progress: 0 %

Programming Hardware
Type: ByteBlaster [LPT1]

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit
1. …lonzebdf\syn\leon.pof	EPC2	0065E6AC	FFFFFFFF	✓	☐	☐	☐	☐

f. Press on the "start" button (the one in Quartus) to download your design into the device

2) Now you'll have to try to connect to the Leon and run "hello world". (you need linux for that).

a. Download LECCS from http://www.gaisler.org

b. Install the LECCS tools with following commands into the /opt/rtems directory:

- Cd /opt
- gunzip –c leccs-linux.tar.gz | tar xf –
- unzip –c leccs-docs-tools.tar.gz | tar xf –
- unzip –c leccs-docs-rtems.tar.gz | tar xf –
- add /opt/rtems/bin to your search path

c. Now compile the test programs that are available within the installed package.

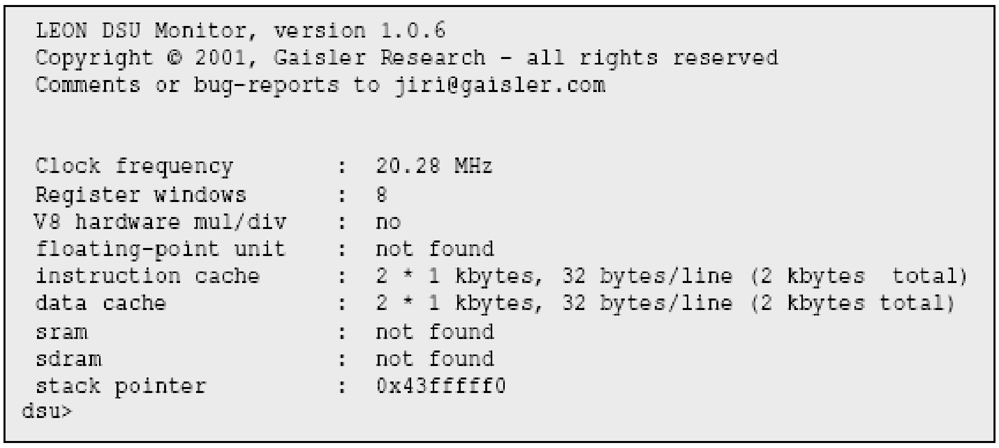
- cd /opt/rtems/src/examples/samples
- sparc-rtems-gcc –msoft-float –Ttext=XXXXXXXXXX hello.c –o hello (with -Ttext=0x40000000 for S(D)RAM or -Ttext=0x60000000 for AHB-RAM)

d. Connect the DSU-UART cable to you linux-system

e. Connect to the target board:

- cd /opt/rtems/src/examples/samples
- ./dsumon –i –u –uart /dev/ttyS0
  (or ttyS1 if you use COM1)

f. If all goes well, you get the following message (with your correct clock speed and other *settings):*



```
 LEON DSU Monitor, version 1.0.6
 Copyright © 2001, Gaisler Research - all rights reserved
 Comments or bug-reports to jiri@gaisler.com


 Clock frequency        :  20.28 MHz
 Register windows       :  8
 V8 hardware mul/div    :  no
 floating-point unit    :  not found
 instruction cache      :  2 * 1 kbytes, 32 bytes/line (2 kbytes  total)
 data cache             :  2 * 1 kbytes, 32 bytes/line (2 kbytes total)
 sram                   :  not found
 sdram                  :  not found
 stack pointer          :  0x43fffff0
dsu>
```

g. Connection with the LEON has been succesfull!

h. Use the following commands to load the “hello world” program into the Leon

- lo hello
- run 0xXXXXXXXX (with XXXXXXXX = 40000000 for S(D)RAM, or 60000000 for
  AHB-RAM)

i. The message “Hello world” should appear on your screen!

11.2.Leon2 on Xilinx board Tutorial

Introduction

This tutorial is created to help you design your first embedded system with a Leon SPARC softcore processor. Before you proceed you must have the following software and hardware:

Software:

- Xilinx ISE 5.x
- Cygwin, not Xygwin **(installed with devel. tools option)**
- WinRAR
- Synplify PRO 7.2 *[Optional]*

Hardware:

- windows-PC
- Linux-PC
- Xilinx FPGA development board

Chapter I is a general chapter where you can learn how to define your Leon SPARC processor system.

In chapter II, there are possible 2 tracks to follow:

TRACK 1: completing the design with Xilinx ISE:

Easiest way to set up a Leon SPARC microprocessor system on an FPGA. Synthesis, place & route are done with Xilinx XST.

TRACK 2: completing the design with a third party synthesis tool

An alternative way to set up the Leon SPARC microprocessor system on an FPGA. Synthesis is done in a third party tool (in this case SYNPLIFY PRO), place and route is done in Xilinx ISE

Chapter III describes how to download the embedded system into the FPGA and how to run it.

Defining the embedded system

1) download and installation of Leon 2 source files:

   a) download leon2-1.0.15.tar.gz from http://www.gaisler.com

   b) unzip the tar.gz archive to e.g. **c:\leon2** (From now on we will suppose that this is your working directory. If you choose another one, you'll have to replace "c:\leon2" in this tutorial with your own directory.)

   c) To get quick access to the Leon configuration utility (which makes use of cygwin), create 2 files in *the same directory:*

leon.bat

```
@C:\Cygwin\bin\bash.exe --rcfile c:\leon2\leon.bashrc
```

leon.bashrc

```
PATH="/bin:/contrib/bin: /cygdrive/c/leon2/leon"
cd /cygdrive/c/leon2
```

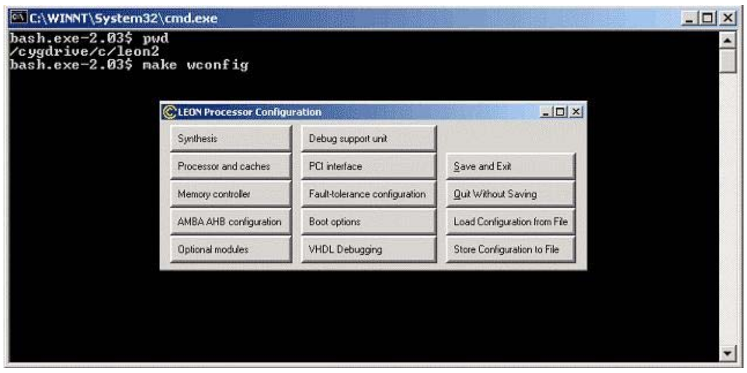
2) configure the processor

   a) start cygwin by double-clicking on leon.bat

   b) normally you should be in the correct directory (you can check it with the "pwd"-command)

   c) start the Leon configuration utility by typing "**make xconfig**" (figure 1)

Figure 1: start the Leon configuration-utility



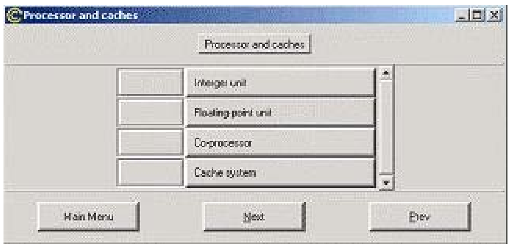
```
C:\WINNT\System32\cmd.exe
bash.exe-2.03$ pwd
/cygdrive/c/leon2
bash.exe-2.03$ make wconfig
```

d) In the Leon configuration utility you can define your own processor system. Below you can find a short description of the necessary configuration you've got to do for building your own embedded system on an FPGA. Most of the settings are correct by default, to make sure however that everything is configured correctly, run trough the following steps.

i) Click on the "synthesis"-option (figure 2), where you'll be able to define your target technology. In your case this will be "Xilinx Virtex (2)" for "Virtex (2)" targets, or "generic" for spartan targets. All other parameters shouldn't be changed unless you know what you're doing.

ii) In the "processor and caches" window (figure 3) you can configure the integer unit and the cache system. For FPGA's this means the following (to avoid possible timing problems):

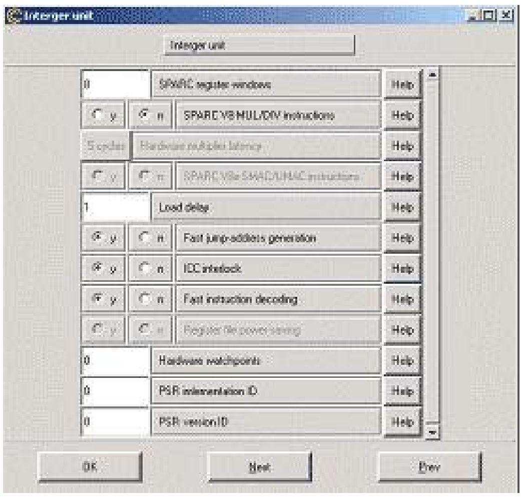
Figure 3: The processor and caches window



(1) Disable the floating point unit and coprocessor unit.

(2) Configure the integer unit (figure 4):
- Make sure "fast jump-address" is selected
- Make sure "ICC interlock" is selected
- *Make sure "fast instruction decoding" is selected*

Figure 4: The integer unit window

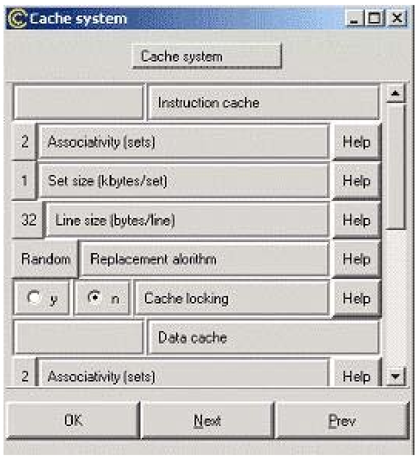


0	SPARC register windows	Help
y [✓ n]	SPARC V8 MUL/DIV instructions	Help
5 cycles	Hardware multiplier latency	Help
y n	SPARC V8e SMAC/UMAC instructions	Help
1	Load delay	Help
[✓ y] n	Fast jump-address generation	Help
[✓ y] n	ICC interlock	Help
[✓ y] n	Fast instruction decoding	Help
y n	Register file power saving	Help
0	Hardware watchpoints	Help
0	PSR implementation ID	Help
0	PSR version ID	Help

(3) Configure the data and instruction cache system (figure 5).
Depending on the amount of block RAM available, you can use different values. The following are example settings for instruction and data cache:

- Associativity on 2 sets
- Set size on 1 kbyte/set
- Line size on 32 bytes/line
- *Random replacement algorithm*

Figure 5: The cache system window


Cache system

Instruction cache

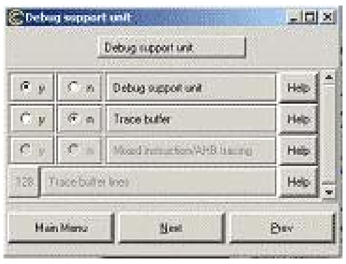
2	Associativity (sets)	Help
1	Set size (kbytes/set)	Help
32	Line size (bytes/line)	Help
Random	Replacement alorithm	Help
y [✓ n]	Cache locking	Help

Data cache

2	Associativity (sets)	Help

iii) Turn on the “debug support unit” in the “debug support unit” window (figure 6)

Figure 6: The debug support unit window


Debug support unit

[✓ y] n	Debug support unit	Help
y [✓ n]	Trace buffer	Help
y n	Mixed instruction/AHB tracing	Help
128	Trace buffer lines	Help

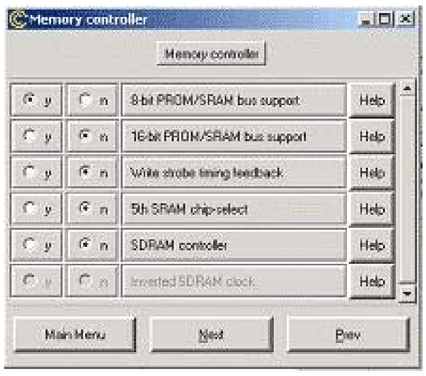
iv) Now you’ve got to configure your “memory controller” (figure 7). Depending on your
hardware, use the following settings:

- You’ve got SRAM:
  - Select 8/16 bit prom/sram bus depending on the data bus width to your SRAM/PROM. For 32 bit access select none.
- You’ve got SDRAM
  - Make sure “SDRAM controller” is selected
  - Make sure you selected the “inverted SDRAM clock”

option when using an FPGA. Otherwise you could create an unstable system caused by clock-skew

- You want to use on-chip RAM (look at step (v))

Figure 7: The memory controller window



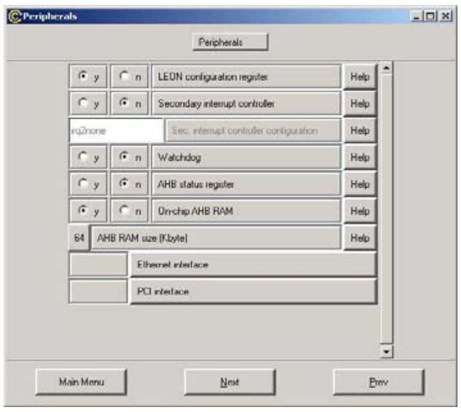
Memory controller

[✓ y]	n	8-bit PROM/SRAM bus support	Help
y	[✓ n]	16-bit PROM/SRAM bus support	Help
y	[✓ n]	Write strobe timing feedback	Help
y	[✓ n]	5th SRAM chip-select	Help
y	[✓ n]	SDRAM controller	Help
y	n	Inverted SDRAM clock	Help

v) In the "peripherals" window (figure 8), make sure that the Leon configuration register is selected. If you need on chip RAM (AHB-RAM), select the On-chip AHB RAM option, and define its size.

<u>Hint:</u> Take care though that you don't exaggerate its size, as there is a very limited amount of RAM available in the FPGA, and the cache also makes use of it. Disable the Ethernet and PCI interface.

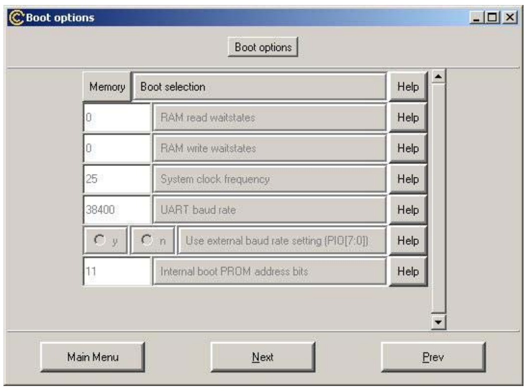
Figure 8: The peripherals window



[✓ y]	n	LEON configuration register	Help
y	[✓ n]	Secondary interrupt controller	Help
irq2none		Sec. interrupt controller configuration	Help
y	[✓ n]	Watchdog	Help
y	[✓ n]	AHB status register	Help
[✓ y]	n	On-chip AHB RAM	Help
64		AHB RAM size (Kbyte)	Help
		Ethernet interface	
		PCI interface	

vi) In the "boot options" window (figure 8) you can select where you want your device to boot from. To avoid synthesis problems, select "memory" as boot location. This way the Leon will try to boot from address location 0x0, probably there will no valid data on address 0x0, but that's no problem as you'll still be able to connect to the debugging unit and download your *executable manually.*

Figure 9: The boot options window



0	RAM read waitstates
0	RAM write waitstates
25	System clock frequency
38400	UART baud rate
○ y ○ n	Use external baud rate setting (PIO[7:0])
11	Internal boot PROM address bits

vii) Don't change anything in the "VHDL-debugging" window.

e) The configuration of the LEON processor system is now complete. In the main window, push on "save and exit". This will save your configuration, after this you'll get the message that you have to type the "make dep" command to generate the proper VHDL configuration files. At the prompt type "**make dep**", and hit the enter button. The VHDL configuration file will now be generated automatically.

3) Adjust the top file to your needs

a) Create a backup copy of the file "c:\leon2\leon\leon.vhd".

b) Open "c:\leon2\leon\leon.vhd" and adjust the in & out ports to correspond with your pinning file*:

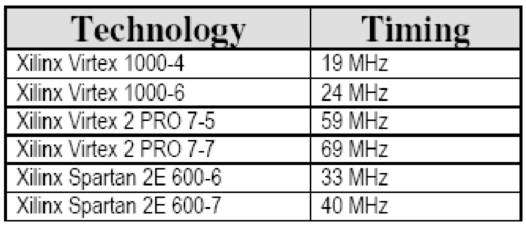
i) Critical basic signals are:

- resetn: active low reset signal, if possible connect it with a button, otherwise write a little reset-state machine in the Leon entity that drives the signal low for some clock cycles.
- clk: The clock signal, if needed divide/multiply the incoming clock with your own state machine, or use the Virtex DLL's. Make sure however that your target component is able to handle the clock speed you selected. Table 1 gives you an idea of what clock speed is achievable on different Xilinx hardware devices
- dsutx: debugging unit transmit UART port
- dsurx: debugging unit receive UART port
- dsuen: debugging unit enable, this active high input signal enables the debugging unit, a logical '1' has to be provided to activate the debugging unit
- bexcn: memory mapped I/O bus exception, this active low input indicates a memory problem, a logical '1' has to be provided to avoid

exceptions

- brdyn: memory mapped I/O bus ready, this active low input indicates that the access to a memory mapped I/O area can be terminated on the next rising clock edge, a logical '1' has to be provided when no memory mapped I/O is used
- dsubre: debugging unit break enable, this active high input will generate break condition and put the processor in debug mode, in this case a logical '1' has to be provided.

Table 1: Leon clock speed



Technology	Timing
Xilinx Virtex 1000-4	19 MHz
Xilinx Virtex 1000-6	24 MHz
Xilinx Virtex 2 PRO 7-5	59 MHz
Xilinx Virtex 2 PRO 7-7	69 MHz
Xilinx Spartan 2E 600-6	33 MHz
Xilinx Spartan 2E 600-7	40 MHz

ii) Optional signals are:

- errorn: active low output which indicates error condition, connect to a led
- dsuact: active low output that indicates whether the debugging unit is active, connect to a led
- pio[15]: Transmit signal of standard I/O in/out UART
- pio[14]: Receive signal of standard I/O in/out UART

iii) When you use SRAM, following signals have to be connected:

- data: data path
- address: address path
- ramsn: RAM chip select
- ramoen: RAM output enable
- rwen: RAM write enable

iv) When you use SDRAM [PC100/PC133 compatible], following signals have to be connected:

- data: data path
- address: address path ([14:2] is used as address, [16:15] are the bank select signals)
- sdcke: clock enable
- sdcsn: chip select
- sdwen: write enable
- sdrasn: row address strobe
- sdcasn: column address strobe

- sddqm: data I/O mask
- sdclk: SDRAM clock

c) Save all changes to the Leon top file

HINT: * The Leon topfile makes use of “pads” to route its signals to the outside world. Depending on the selected target technologies (Atmel, Xilinx,…) other types of pads are inferred. The 3 types of paths you’ll encounter are: [pad = output pin, d/q/en: Leon in/output]

- outpad(d: in st_logic, pad : out std_logic)
- inpad(pad: in st_logic, q : out std_logic)
- iopad(d: in st_logic, en : in std_logic, q : out std_logic, pad : inout std_logic)

With this knowledge you can easily create extra pads (e.g. an extra address pad to connect your SRAM and SDRAM simultaneously).

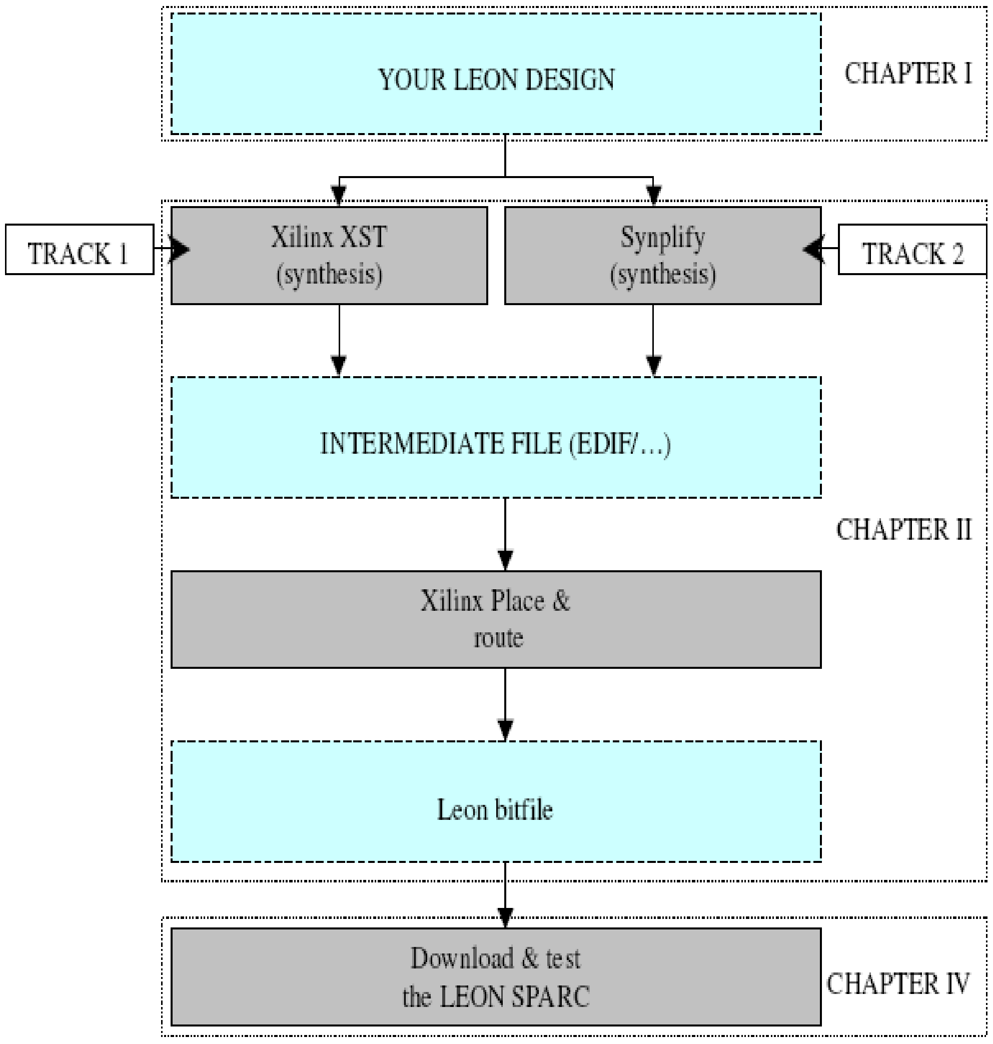
II Synthesis, place & route, generating the bitstream

As mentioned in the introduction, there are 2 possible tracks to complete your design (figure 10):

TRACK 1: completing the design with Xilinx ISE:
Easiest way to set up a Leon SPARC microprocessor system on an FPGA. Synthesis, place & route is done with Xilinx XST.

TRACK 2: completing the design with a third party synthesis tool
An alternative way to set up the Leon SPARC microprocessor system on an FPGA. Synthesis is done in a third party tool (in this case SYNPLIFY PRO), place and route is done in Xilinx ISE

Figure 10: The design flow



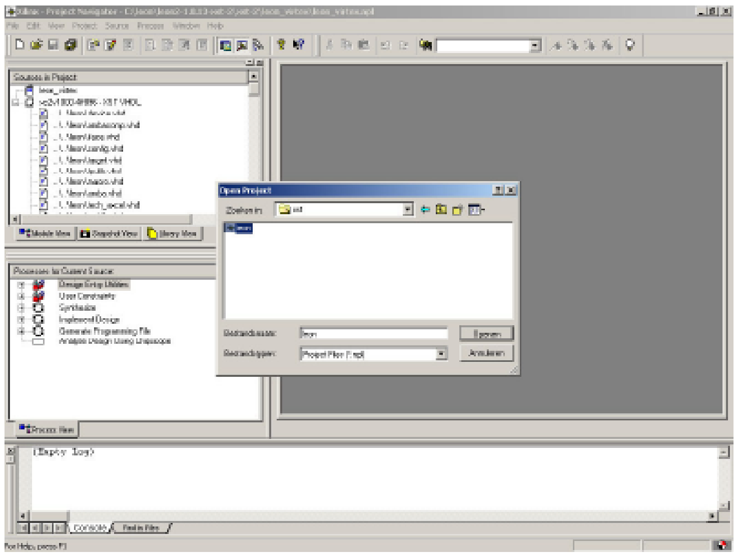
TRACK 1 Complete your design with Xilinx XST

1) Start Xilinx project navigator (*Start* ‰ *programs* ‰ *Xilinx ISE 5* ‰ *Project navigator*)

2) Open the standard leon ISE project (File ‰ open project)

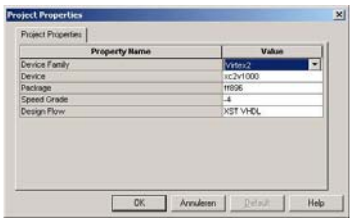
3) Browse to “c:\leon2\syn\xst” (figure 11) and click on “Open”

Figure 11: Open the Leon SPARC ISE project



4) Configure your target device by double-clicking on the currently selected hardware target in the "project workspace" (figure 12)

Figure 12: Configuring the target device

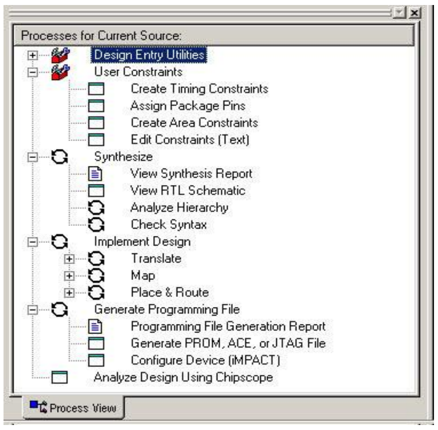

Project Properties

Property Name	Value
Device Family	Virtex2
Device	xc2v1000
Package	ff896
Speed Grade	-4
Design Flow	XST VHDL

5) Add your vhdl files that are needed to complete the design (e.g. clock divider,…) by right clicking on your selected hardware target and then clicking on "add source". Also add your pinning file (*.UCF) and assign it to the leon top entity.

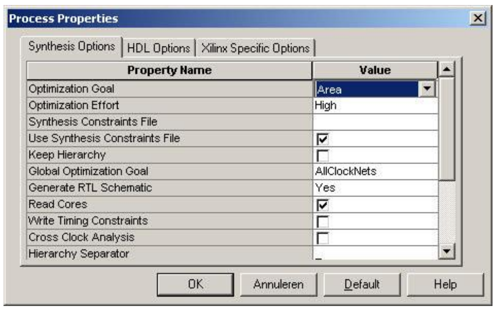
6) Select the "leon" top entity in the "project workspace". This will show the available you the available processes for this file (figure 12)

Figure 13: The process workspace



7) Right click on "synthesize", set optimization goal on "Area" or "speed" depending on your own expectations, and set the optimization effort to "high", push on "OK". (figure 14)

Figure 14: Process properties



Property Name	Value
Optimization Goal	Area
Optimization Effort	High
Synthesis Constraints File	
Use Synthesis Constraints File	☑
Keep Hierarchy	☐
Global Optimization Goal	AllClockNets
Generate RTL Schematic	Yes
Read Cores	☑
Write Timing Constraints	☐
Cross Clock Analysis	☐
Hierarchy Separator	_

8) Double-click on "synthesize" in the "process workspace" and your design will be synthesized. If there are any errors, you'll have to trace back to figure out what is going on.

9) Double-click on "Implement design" in the "process workspace" and your design will be placed & routed on your target component. If there are any errors, they will probably be generated because of errors in your pinning file (*.UCF).

HINT: If you have unmatched LOC constraints in your pinning file, do not remove them as you might need them later. Instead use the following trick to avoid getting into trouble later:
1. Open the preferences window (edit ‰ preferences), select the "processes" tab, and select the "advanced" property display level.
2. Right click on "Implement design", select "properties", and make sure the "allow unmatched LOC constraints" value is selected, after this, push on "OK".

10) Open the "Place & Route Report" by double-clicking on it in the "process

workspace”, and check whether your design fits into your component.

11) Open the “Text-based Post Place & Route Static Timing Report” by double-clicking on it in the “process workspace”, and check whether you meet your desired clock speed. If you don’t meet it, trace back and alter your design.

12) Double-click on “Generate Programming file” in “the process workspace” and the bitstream will be generated. Normally you will not experience any errors here

TRACK 2 Complete your design using Synplify.

1) Start synplify. (*start* ‰ *programs* ‰ *synplicity* ‰ *synplify pro*)

2) Open the existing leon project (*File* ‰ *open project* ‰ *existing project*)

3) Browse to “c:\leon2\syn” and open the “leon” project.

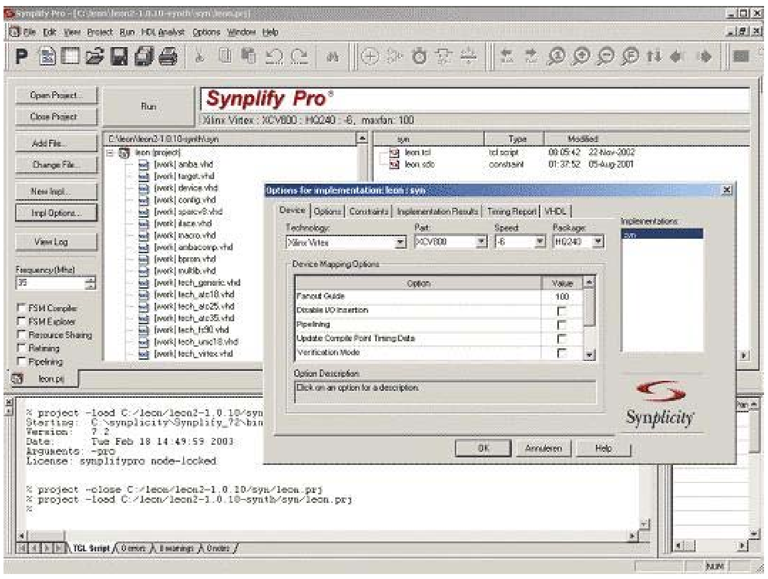
4) Add your own vhdl files to the project by clicking on “add file”. Make sure your hdl files are synthesized before the leon top entity by optionally altering the synthesize order.

5) Configure the implementation options (*project* ‰ *Implementation Options*) (figure 15)

- *The device tab:*
  Select your target component (Technology/part/speed/package)
- *The options tab:*
  Not a single option may be selected, as this could result into netlists with a non-working leon.
- *The constraints tab:*
  Enter your desired clock speed
- *Implementation results*
  Make sure the resulting format is “edif”
- *Timing report:*
  These values should be correct
- *Vhdl-tab:*
  Make sure the top level entity is “leon”

Press on “OK”

Figure 15: Synplify implementation options



Synplify Pro - [C:\leon\leon2-1.0.10-synth\syn\leon.prj]

Xilinx Virtex : XCV800 : HQ240 : -6, maxfan: 100

Options for implementation: leon : syn

Technology: Xilinx Virtex | Part: XCV800 | Speed: -6 | Package: HQ240

Option	Value
Fanout Guide	100
Disable I/O Insertion	☐
Pipelining	☐
Update Compile Point Timing Data	☐
Verification Mode	☐

Option Description: Click on an option for a description.

```
% project -load C:/leon/leon2-1.0.10/syn
Starting:  C:\synplicity\Synplify_72\bin
Version:   7.2
Date:      Tue Feb 18 14:49:59 2003
Arguments: -pro
License:  synplifypro node-locked

% project -close C:/leon/leon2-1.0.10/syn/leon.prj
% project -load C:/leon/leon2-1.0.10-synth/syn/leon.prj
%
```

6) Press on the famous “Run” button to start synthesis, wait until synthesis is complete

7) Press on the “view log” button, and check whether your desired clock speed has been met. If not, track back and change your clock divider to a lower value.

8) Now you have to place & route your design in Xilinx ISE.

a. Start Xilinx ISE (*Start ‰ programs ‰ Xilinx ISE 5 ‰ Project navigator*)

b. Start a new project (*File ‰ new project*)

c. Select a project name, e.g. “leon” and a working directory (create a new one). Make sure the correct target device is selected, as design flow choose “EDIF”.

d. Add the “leon.edf” file generated by synplify by right clicking on your target device in the “project workspace”, selecting “add source”, and adding the EDIF file to the project

e. Add your pinning file (*.UCF) the same way

f. Select the “leon” entity in the “project workspace”.

g. Double-click on “Implement design” in the “process workspace” and your design will be placed & routed on your target component. If there are any errors, they will probably be generated because of errors in your pinning file (*.UCF).

HINT: If you have unmatched LOC constraints in your pinning file, do not remove them as you might need them later. Instead use the following trick to avoid getting into trouble later:

- Open the preferences window (edit ‰ preferences), select the "processes" tab, and select the "advanced" property display level.
- Right click on "Implement design", select "properties", and make sure the "allow unmatched LOC constraints" value is selected, after this, push on "OK".

h. Open the "Place & Route Report" by double-clicking on it in the "process workspace", and check whether your design fits into your component.

i. Open the "Text-based Post Place & Route Static Timing Report" by double-clicking on it in the "process workspace", and check whether you meet your desired clock speed. If you don't meet it, trace back and alter your design.

9) Double-click on "Generate Programming file" in the "process workspace" and the bitstream will be generated. Normally you will not experience any errors here.

III Download and test the Leon SPARC

1) Download your design with Xilinx iMPACT

a. Make sure you connected your download cable correctly (e.g. parallel cable IV,…), make sure your board is powered on

b. Start iMPACT *(Start ‰ programs ‰ Xilinx ISE 5 ‰ Accessories ‰ iMPACT)*

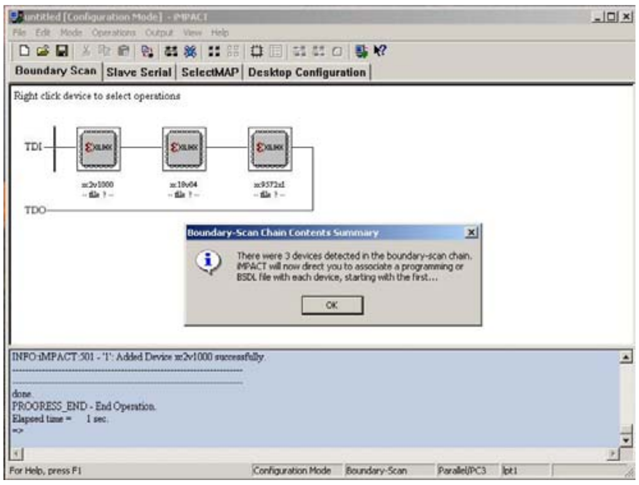
c. As operation mode select "configure devices", press on "next"

d. Configure the device via "Boundary-Scan Mode", press on "next"

e. Select "automatically connect to cable and identify Boundary-Scan chain", press on "next"

f. Xilinx will report which hardware has been found on the JTAG chain (figure 16)

Figure 16: iMPACT



Right click device to select operations

There were 3 devices detected in the boundary-scan chain. iMPACT will now direct you to associate a programming or BSDL file with each device, starting with the first...

```
INFO:iMPACT:501 - '1': Added Device xc2v1000 successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
done.
PROGRESS_END - End Operation.
Elapsed time =      1 sec.
=>
```

g. Now Xilinx asks for the configuration files for each device. Select "Leon." bit for the correct device and click op "open", press "cancel" for the other devices.

h. Right click on the target component and select "program…", in the next screen press on "OK", and your target device will be programmed.

2) Now you'll have to try to connect to the leon and run "hello world". (you need linux for that).

a. Download LECCS from http://www.gaisler.org

b. Install the LECCS tools with following commands into the /opt/rtems directory:

- Cd /opt
- gunzip –c leccs-linux.tar.gz | tar xf –
- unzip –c leccs-docs-tools.tar.gz | tar xf –
- unzip –c leccs-docs-rtems.tar.gz | tar xf –
- add /opt/rtems/bin to your search path

c. Now compile the test programs that are available within the installed package.

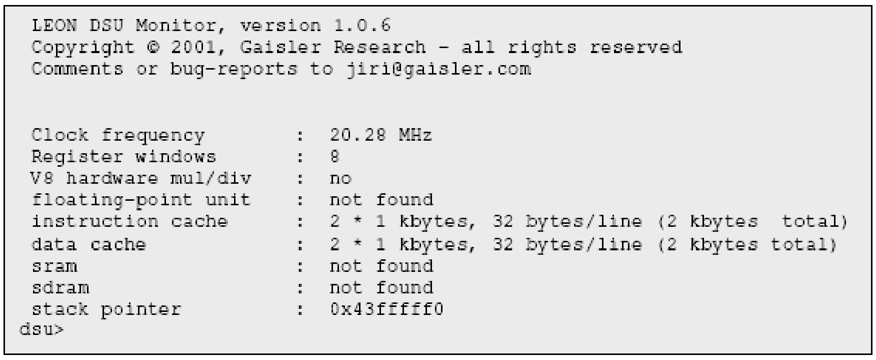
- cd /opt/rtems/src/examples/samples
- sparc-rtems-gcc –msoft-float –Ttext=XXXXXXXXXX hello.c –o hello (with -Ttext=0x40000000 for S(D)RAM or -Ttext=0x60000000 for AHB-RAM)

d. Connect the DSU-UART cable to you linux-system

e. Connect to the target board:

- cd /opt/rtems/src/examples/samples
- ./dsumon –i –u –uart /dev/ttyS0 (or ttyS1 if you use COM1)

f. If all goes well, you get the following message (with your correct clock speed and other settings):



```
 LEON DSU Monitor, version 1.0.6
 Copyright © 2001, Gaisler Research - all rights reserved
 Comments or bug-reports to jiri@gaisler.com


 Clock frequency        :  20.28 MHz
 Register windows       :  8
 V8 hardware mul/div    :  no
 floating-point unit    :  not found
 instruction cache      :  2 * 1 kbytes, 32 bytes/line (2 kbytes  total)
 data cache             :  2 * 1 kbytes, 32 bytes/line (2 kbytes total)
 sram                   :  not found
 sdram                  :  not found
 stack pointer          :  0x43fffff0
dsu>
```

g. Connection with the LEON has been successful!

h. Use the following commands to load the "hello world" program into the leon

- lo hello
- run 0xXXXXXXXXX (with XXXXXXXX = 40000000 for S(D)RAM, or 60000000 for AHB-RAM)

i. The message "Hello world" should appear on your screen!