Multi-Core Processor System on A Chip Using Leon3 Processor

Dipesh D. Somani

Research Scholar, Department of Electronics Engineering

Shri Ramdeobaba College of Engineering and Management, Nagpur

Abstract - With the development of IC design and architecture, embedded system ranges from a single microprocessor to a complex multi-core processor with an embedded operating system (OS). A multi-core processor is a single integrated circuit in which two or more processors have been attached for enhanced performance and more efficient simultaneous processing of multiple tasks. This paper presents a dual-core embedded system designed with LEON3 open source processors. Dual-core processor has been developed with centralized shared memory for process sharing and shared bus. The system design is implemented using Cyclone II FPGA. The dual core system is tested with matrix multiplication using multi-threading using pthreads under the embedded operating system (eCos) to enhance the performance of the system in terms of its speed. Dual core processor system designed using LEON3 processor accelerates the matrix multiplication by 1.81 times as compared to single core processor system.

Keywords— LEON3 Multi-Core Processor, pthreads, Multithreading, Embedded Configurable Operating System (eCos), Cyclone II FPGA.

I. INTRODUCTION

With the growing needs for advanced functionalities, communication speed and performance requirements in modern embedded systems, it is now necessary to integrate multi-core processors in the system, preferably on a single chip. Multi-core processor systems consist of two or more independent cores. A processing part of a processor system is called as Core. Multi-core processors have the potential to run applications more efficiently than single-core processors. Multi-core processor based systems are complex because they contain multiple or share resources and execute multiple tasks, which interact with each other in complex ways. Deploying an SMP system with shared memory is the ability to use multi-core processors simultaneously to execute different tasks and the shared resources based system uses same resources between multiple tasks executing simultaneously.

Therefore, the systems require an operating system to manage the resources and tasks [1]. The multi-core architectures have the potential to satisfy the timing performance when the special coding techniques are incorporated like multithreading using pthreads.

To design a multi-core embedded system Leon3 soft core processor is used. The LEON3 is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The model is highly configurable which is particularly suitable for MP-SOC designs [2]. For case study matrix multiplication using multi-threading is used. Multi-threading is supported by an operating system eCos. The eCos is a real-time operating system with SPARC port supports LEON3 and its standard on-chip peripherals. The highly configurable nature of eCos allows the operating system to be customized to precise application requirements for delivering the best possible run-time performance and an optimized hardware resource footprint. Multi-core embedded system is designed using LEON3 soft core processor and it is implemented on Altera Cyclone II FPGA emulation board. An application of matrix multiplication using pthreads for implementing multi-threading is executed on designed embedded system through eCos and got an acceleration of 1.81 times in multithreaded matrix multiplication over single core processor system.

II. LEON3 PROCESSOR

The LEON3 SPARC V8 processor architecturehas been designed to fit into a large variety of applications requirement. It can also be combined with the IEEE-STD-754 compliant Floating Point Unit (GRFPU Lite). The architecture is centered on the AMBA Advanced High-speed Bus (AHB), to which the LEON3 core and other high-bandwidth units are connected, while low-bandwidth units are connected to the AMBA Advanced Peripheral Bus (APB) which is accessed through an AHB to APB Bridge. The architecture is shown in figure 1 [3].

The LEON3 core has integer unit implements the full SPARC V8 standards, including hardware multiply and divides instructions. The numeral of register windows is configurable within the limit of the SPARC standard (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture). Cache system consisting of a separate instruction and data cache, and each can be configured with 1-4 sets, 1–256 Kbyte/set, 16 or 32 bytes

per line.The data cache performs bus-snooping on the AHB bus. The LEON3 integer unit has interfacesfor a floating-point unit (FPU) and a custom co-processor. Two FPU controllers, one availablefor the high-performance GRFPU (available from Gaisler Research) and one for the Meiko FPU core (available from Sun Microsystems). The floating-point processor unit and co-processor unit execute inparallel with the integer unit, which

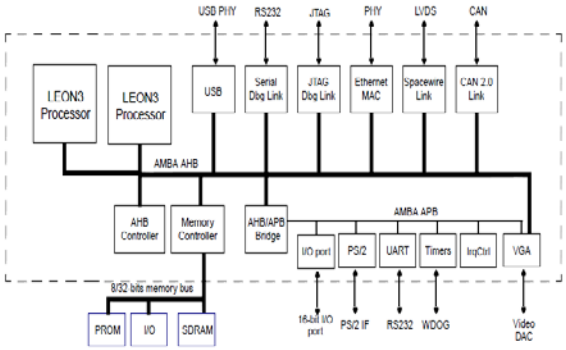

Fig. 2.1. Dual Core Leon3 Processor [3]

does not block the operation unless a data or resource dependency exists. A SPARC V8 Reference Memory Management Unit (SRMMU) can be enabled (optional). The SRMMU implements the full SPARC V8 MMU specification and has mapping between multiple 32-bit virtual address spaces and 36-bit physical memory. The LEON3 pipeline includes functionality to provide non-intrusive debugging on target hardware. To aid software debugging, any or all four watch-point registers can be enabled. Each register can cause a breakpoint setup on an arbitrary instruction or data address range. When the (optional) debug support unit is attached, the watch-pointregisters can be used to enter debug mode. A debug support interface gives full access to all processor registers and caches. An internal trace buffer can monitor and store executed instructions, which can be read out laterover the debug interface. LEON3 also supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupts. The interrupt interface delivers functionality to generate and acknowledge interrupts. The cache system implements an AMBA AHB master to load and store data to and from the caches. The interface is compliant with the AMBA-2.0 standard. Incremental bursts are generated to optimize the data transferduring line refill. LEON3 processor core implements a power-down mode. It will halt the pipeline and caches until the next interrupt. A suitable clock-enable signal is produced by the processor to implement clock-gating.It is designed to be use in multi-core processor systems. Each processor core has a unique index to allow processor core enumeration. The write-through caches and snooping mechanism confirms memory coherency in shared-memory systems [4][5].

III. REAL TIME OPERATING SYSTEM

kEmbedded Configurable Operating System (eCos) is a real-time operating system deeply designed for embedded applications. It is also an open source and royalty-free operating system. eCos is a highly configurable operating system whichcan be configured to application specific requirementsand bringing the best possible run-time performance, and an optimized hardware resource footprint. A booming net community has grown up around the operating system ensuring on-going technical innovation and wide platform support. Figure shows the layered architecture of eCos.

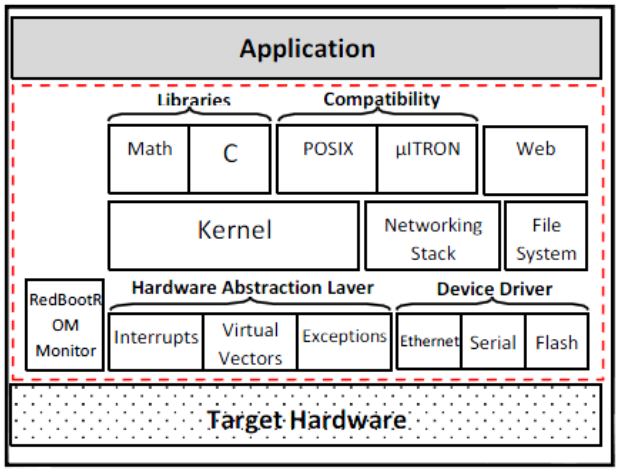

Fig. 3.1. Overview of eCos Architecture.[6]

The main components in eCos architecture are the Hardware Abstraction Layer (HAL) and eCos Kernel. The purpose of eCos HAL is to allow the application to be independent of target hardware. eCos can manipulate the hardware layer using the HAL API. This HAL is also used by others upper OS layer which make porting eCos to a new hardware target a simple task consisting of developing the HAL of the new target. eCos kernel is the core of eCos system, it includes the most part of modern operating system components: scheduling, synchronization, interrupt, exception handling, counters, clocks, alarms, timers, etc. It is written in C++ language allowing application written in this language to interface directly to the kernel resources. The eCoskernel also has supports to interface standard library like µITRON and POSIX compatibility layers. POSIX (Portable Operating System Interface) threads, usually referred to as Pthreads, are a POSIX standard for threads. POSIX Threads extensions (IEEE Std 1003.1c-1995), defines an API (Application Programming Interface) for creating and manipulating threads. [6][7]

IV. POSIX THREADS

The Pthreads standard provides not only an uniform base for multi-core processor shared-memory applications but

also a real-time system environments and a cheap model for multi-threaded programs. The notion of threads can be used to express parallelism within applications at the level of programming languages. An implementation of Pthreads can be carried out as:

- A kernel implementation, where all functionality is part of the operating system kernel;
- A library implementation, where all functionality is part of the user program and can be linked in; or
- A mixture of the above.

A kernel implementation simplifies control over thread operations and signal handling but adds the overhead of entering and leaving the kernel at each call. A library implementation can be more efficient since it does not have to enter the operating system kernel but it complicates signal handling and some thread operations, and it also has to deal with two different scheduler, one for processes (kernel level) and one for threads (library level).

A. Pthreads Standard

The Pthreads standard specifies various servicesthat can be provided to support multi-threaded applications.Most of the interface specifications leavemany details to the implementation. For example,support for certain functions and the detection ofsome errors is optional. Therefore, Pthreads compliantimplementations may vary considerably.pthreads implementation supports the following functionality:

- *thread management:* initializing, creating, joining, exiting and destroying threads;
- *synchronization:*mutual exclusion, condition variables;
- *thread-specific data*;
- *thread priority scheduling:* priority management, preemptive priority scheduling;
- *signals:* signal handlers, asynchronous wait, masking of signals, long jumps;
- *Cancellation:* cleanup handlers, different interruptibility states.

The support is currently being extended to include process control.

B. Design and Implementation

The design of Pthreads has been stronglyinfluenced by constraints of the Pthreads standard and to some extend by the use of eCoson SPARC architecture. The interface consists of a C library with linkableentry points and can optionally be compiled togenerate a language-independent interface. An interface allows programs to use Pthreads services. In case of the programming language C the library routines of Pthreads are immediately available. Any other programming language needs a language interface to the Pthreads library to pass parameters correctly, perform type conversion and other language or compiler-dependent adjustments.

The Pthreads library contains a set of routines whose interface and functionality are defined by the Pthreads standard. Pthreads routines partially execute asauser mode and within critical sections it operates in the Pthreads kernel mode to guarantee mutual exclusion between multiple threads.

In order to introduce an application on top of a multiprocessor architecture, the code needs to be parallelized in several threads. Then the entry of the flow is a parallel application composed of several concurrent POSIX threads. A POSIX thread is created by a call to:

intpthread_create (pthread_t *thread, pthread_attr_t *attr, void *(* start_function) (void *arg), void *arg)

This executes the thread whose behaviour is the start_function called with arg as argument. The attr structure contains thread attributes, such as stack size, stack address and scheduling policies. These attributes are particularly useful when dealing with embedded systems or SoCs, in which the memory map is not standardized. The value returned in the thread pointer is a unique identifier for the thread. For more details, we refer the reader to [8].

The application threads communicate using different communication primitives. Generally it distinguish two types of parallel programming models suited to multiprocessor architectures: shared memory model likeOpenMP and message passing model like MPI. In our case study, we used the shared memory model at the implementation level, under a Symmetric MultiProcessor (SMP) kernel.

V. SYSTEM IMPLEMENTATION

A. Configuration of Leon3 Processor

The highly configurable nature of LEON3 processor allows the model to be customized for a certain application or target technology. A graphical configuration tool based on the Linux kernel tkconfig scripts is used to configure the model. Issue the command ‘make xconfig’ in a bash shell ofthe directory ‘designs/LEON3-altera-de2ep2C35’ will launch the xconfig GUI tool as shown in Figure.

It is used to modify the LEON3 template design. When the configuration is saved and ‘xconfig’ is exited, that updated

the config.vhd automatically with the selected configuration.

B. Compilation and Synthesis of Leon3 Processor

On Windows systems, the LEON VHDL model and test bench wascompiled by running in transcript of 'modelsim' using the 'vcom' and 'vlog' compilers. Once the LEON3 model has been compiled, it is simulated by "vsim" to verify the behavior of the model. The simulationruns a test bench that stimulates the main components of LEON3. The simulation is halted by putting integer unit in error mode. The test program executed by the test bench consists of two parts, a simple prom boot loader (prom.S) and the test program itself (systest.c). Both parts can be re-compiled using the 'make soft' command in the bash shell (LINUX)[10]. This requires that the BCC tool-chain is installed on the host computer. The simulation is terminated by generating a VHDL failure. It is the only way of stopping the simulation from inside the model. An error message is then printed and should beignored. The template design can be tested and synthesized with various synthesis tools like Xilinx, Quartus, Simplify, Precision or ISE/XST. After configuration is over the Leonmodel is synthesized for Altera Cyclone II using Quartus II software [11].

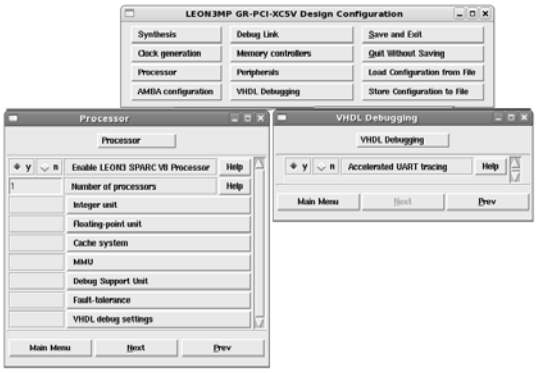

Fig. 5.1. Configuration GUI for Leon3 processor[9]

C. Installation and Configuration of Real Time Operating System (eCos)

All the required tools to begin developing application using Leon3 multicore processor is supported by RTOS such as eCos. The process contains three major steps - eCosinstallation and configuration phases, theapplication compilation and their respective execution environment. The eCos RTOS installation and configuration can be divided into four steps. In a Linux host to produce Leon3 executable Leon3 cross compiler'sparc-elf-gcc' must have installed bydecompressingit in the "/opt" directory and installed using "export PATH=/opt/sparc-elf-3.4.4/bin:$PATH" command in bash shell[12]. Then install the eCos source code and supporting configuration tools bydecompressingit in a chosen directory that can be used in the configuration phase. This configuration tools are available in different versions;here Linux version is used that does not need additional library. eCos is one of the most architecture free RTOS. The choices of a specific target, wide range of hardware platform and software configurations is done by running the eCos configuration tool GUI and select the target platform "LEON3 processor", 'net package' and predefined setting that customized later by selecting specific networking stack, debugging interface or any other specific software component[13]. Finally save the configuration file and starting the building process using build item in the configuration tool GUI. After building eCos library giving a simple command will compile the application by mentioning the location of the eCos library.

D. Multithreaded Matrix Multiplication

For the case study a simple matrix multiplication using multiple threads using POSIX thread standard is used. The exact operation is explained by figure shown below.

Operation of an application is divided into three steps – Initialization, multiplication and printing results. At first initialization of variable, thread space creation and populating matrix memory space with random values is done. Then threads are created in which matrix multiplication is done.

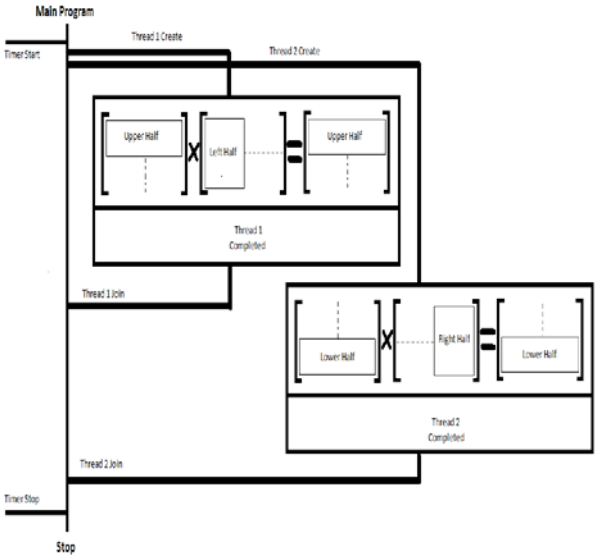

Fig. 5.2. Multithreaded matrix multiplication.

The multiplication is so divided that no overlapping or double multiplication happens. Above figure illustrate how matrix multiplication is divided into two parts for each thread. Returning of these threads to main program confirms the completion of threads operation. And finally noted time at the start and end of threads creation and completion is converted into seconds and`displayed.

E. Debugging

GRMON is a debug monitor for the Leon3 DSU and for SOC designs based on the GRLIB IP library. It provides a non-intrusive debug environment on real target hardware. With the help of this tool LEON3 applications was downloaded and executed [15].

VI. RESULT

Matrix multiplication using multithreading is run on dual core Leon3 embedded system design on Cyclone II FPGA with eCos. The results obtained are shown in tabular form below.

Table 1 shows the execution time of an application obtained using matrix of dimension 49x49withdifferent no. of threads on single as well as dual core embedded system,while Table 2 shows the execution time of an application obtained using matrix of dimension 99x99 with different no. of threads on single as well as dual core embedded system. Timing shows are in seconds.

TABLE 1. Execution Time For 49x49 Matrix Multiplication

Matrix Dimension = 49x49		
No. Of Threads	**No. Of Cores**	
	Single Core	**Dual Core**
1	0.13	0.13
2	0.12	0.08
3	0.13	0.09
4	0.12	0.08

TABLE 2. Execution Time For 99x99 Matrix Multiplication

Matrix Dimension = 99x99		
No Of Threads	**No. Of Cores**	
	Single Core	**Dual Core**
1	1.11	1.13
2	1.1	0.63
3	1.1	0.63
4	1.04	0.6

VII. CONCLUSION

In the project, a multiprocessor system using Leon3 soft core processors on Altera Cyclone II FPGA emulation board was designed and results are obtained. It is concluded from the results that the execution time to multiply the matrix, on dual core system is less than single core system, when multiple threads are used. It is also seen that when no. of threads exceeds the no. of core in the system then there is minimal enhancement in the execution time.

VIII. ACKNOWLEDGEMENT

We thank Gaisler Research, for providing us all the obligatory tools, due to which we were capable to obtain the quality results.

REFERENCE

[1] NabilLitayem, BochraJaafar, SlimbenSaoud, “Embedded Microprocessor Performance Evaluation Case Study ofthe Leon3 processor”, Journal of Engineering Science and Technology Vol. 7, No. 5 (2012) 574 - 588.
[2] P.Huerta, J.Castillo, J.I.Martinez, V.Lopez, “A MicroBlaze Based Multiprocessor SoC”,
[3] Jiri Gaisler. The LEON-3 Processor User’s Manual, Version 1.0.20, February 2009 http://www.gaisler.com,
[4] Gaisler Research, “GRLIB IP Library User’s Manual”, Version 1.1.0 B4113 January 2012.
[5] GaislerResearch, “GRLIB IP Core User’s Manual”,Version 1.1.0 - B4113, January 2012.
[6] Red Hat Inc., “eCos Reference Manual”, Version 2.0, September 2000,
[7] Red Hat Inc., “eCos User Guide”, 2003,
[8] Andrew S. Tanenbaum. Distributed Operating Systems, chapter 6.3, pages 315.333. Prentice Hall, 1995
[9] AeroflexGaisler, LEON/GRLIB Configuration and Development Guide, December 2012
[10] AeroflexGaisler, LEON3 LEON3-FT CompanionCore Data Sheet,
[11] Altera Corporation, “Cyclone II FPGA Starter Development Kit User Guide”, Version 1.0.0 October 2006.
[12] Jiri Gaisler, “BCC - Bare-C Cross-Compiler User’s Manual”, Version 1.0.41, July 2012.
[13] Massa, A.J. (2003). Embedded software development with eCos. Prentice Hall.
[14] Frank vahid,TonyGivargis, “Embeded System Design”, John Wiley & Sons Inc.,2002 edition.
[15] Gaisler Research, “GRMON User’s Manual”, Version 1.1.52, January 2012.
[16] A. Patel, C. Madill, M. Saldana, C. Comis, R. Pomes, and P. Chow, “A Scalable FPGA- based Multiprocessor”, in 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 24-26 April 2006, pp.111-120.
[17] John Fruehe, “ Planning Considerations for Multi core Processor Technology”, Dell Power solutions, May 2005,
[18] BenaoumeurSenouci, AimenBouchhima, Frédéric Rousseau, FrédéricPetrot, Ahmed Jerraya, “Fast Prototyping of POSIX based applications on a Multiprocessor SoC Architecture: “Hardware-dependent Software oriented approach””, Seventeenth IEEE International Workshop on Rapid System Prototyping 14-16 June 2006, pp. 69-75,
[19] Frank Mueller, “A Library Implementation of POSIX Threads under UNIX” 1993 Winter USENIX – January 25-29, 1993 – San Diego, CA,