Hard and Soft-Core Implementation of Embedded Control Application Using RTOS

Ahmed Karim Ben Salem, Slim Ben Othman, Slim Ben Saoud
LECAP-EPT- INSAT - B.P. 676, 1080 Tunis Cedex, Tunisia
ahmed.bensalem@ept.rnu.tn, slim.benothman@ept.rnu.tn, slimbensaoud@fulbrightweb.org

***Abstract*- The performances of System on Chip (SoC) and the Field Programmable Gate Array (FPGA) particularly, increase continually. Thus, they can be an interesting solution for embedding many applications and especially complex electric control, where the demands of Real Time (RT) platforms increase every year. In this paper, there is presented different design for the implementation on embedded processor cores and performance analysis of such embedded systems. Moreover, the following pages outline the new SoC design techniques; investigate the gain with using embedded RTOS-kernel for higher RT performance. Experimental results carried on a prototyping platform are given to illustrate successfully that the digital control implementation on hard-core processor associated with an embedded RTOS provides interesting control performance.**

I. INTRODUCTION

A System on Chip (SoC) can provide a single-chip solution, lower power usage, better performance, more efficient use of board real estate, and simpler integration. Consequently, the SoC has huge advantages compared to multichip solutions.

Besides, today SoC designers face important product development decisions in choosing intellectual property (IP) cores for SoC. this can impact system performance and quality [1]. Indeed, determining which core is most appropriate for a given SoC requires careful consideration. Decisions must be made about the type of core (soft vs. hard), the architecture to integrate this core, including hardware, software, and the relationships between the two [2].

In general, hardware architecture problems can range from special-purpose hardware units as created by hardware/software codesign to micro-architectures for processors or multiprocessors. On the other hand, software architectures determine how we can take advantage of parallelism and non-determinism to improve performance and lower cost.

This paper discusses some of these areas and offers guidance on how best to evaluate the features of IP cores. Our application to these areas is an ongoing work aiming to develop Real Time (RT) control drive algorithms for embedded SoC. Some basic ideas about new trends on SoC, the gain with integrating embedded RT Operating System (RTOS) kernel for task management, are also described in this paper.

II. ENHANCING RT PERFORMANCE WITH RTOS

A. RTOS Overview

Many embedded systems must perform several tasks simultaneously. RTOS allow a single CPU to juggle multiple processes. Unlike general-purpose operating systems (OS), RTOS must adhere to strict timing requirements. The main task of an RTOS is to manage the resources of the processor with its general purpose I/O device such that a particular operation executes in precisely the same amount of time every time it occurs [3].

There are many aspects of OS that we should be concerned about, such as context-switch time, service call performance, scheduling algorithms and tools integrations. However, the most important characteristic, what makes an OS a RTOS, is its ability to service interrupts load quickly [4]. A failure to meet a response time requirement can be catastrophic.

B. RTOS Benefits For SoC

Traditional software systems employed a looping construct for the main part of the application and used interrupts to handle time-critical events. These are so-called "foreground/background" systems, where the interrupts run in the "foreground" (because they take priority over everything else) and the main loop runs in the "background" when no interrupts are active [5]. As applications grow in size and complexity, they are requiring a very fast execution to answer strict RT constraints. Hence, the approach loses its appeal because it becomes increasingly difficult to characterize the interaction between the foreground and background.

An alternative method for structuring these applications is to use an RTOS that manages overall program execution according to a set of clearly defined rules. With these rules in place, the application's performance can be enhanced, regardless of its size and complexity [6]. Furthermore, using new SoC devices, computing-intensive tasks would be interpreted by software scheduled by an RTOS kernel. So that, the application can be divided into smaller manageable tasks that share the same processor. The kernel scheduling policy could accelerate these embedded software applications.

III. SOC ADVANTAGES

A. New SoC Platforms

Microprocessors and Digital Signal Processors (DSP) based solutions are generally used for the implementation of several

978-1-4244-1666-0/08/$25.00 ©2008 IEEE

systems. Nevertheless, hardware solutions such as FPGAs can also be considered as an appropriate solution to enable high performance for novel complex applications [7]. In fact, FPGA based solutions enables to implement real time applications because of their capability to realize very fast computation of digital algorithms. Furthermore, as DSP, FPGAs are very low cost components [1].

New high-performance SoC are typically composed of both hardware and software components which interact in order to perform the given task. They have more than one processor and memory, all on the same chip. Dealing with the global on chip memory allocation/de-allocation in a dynamic yet deterministic way is an important issue for the new complex systems [1].

Moreover, new FPGA families are considered as new trends in SoC, since they have integrated embedded processors, they provide flexible and fast interconnect options for interfacing peripherals and creating multi-processors system.

The FPGA embedded processors (hard or soft-core) have an arithmetic logic unit particularly dedicated to the real-time computation. Therefore, the extremely fast computation capability of these FPGAs allows a few microseconds real-time computation of control algorithms in spite of their complexities [1].

A key aspect of any software-to-hardware design flow is the use of parallelism to increase performance. C applications can be accelerated, using FPGAs, by adapting the approach of leaving the task of low-level optimization to automated compiler tools, while at the same time providing software programmers an appropriate and easy-to-use programming model that allows higher level parallelism to be expressed [8].

B. New Software-to-Hardware Tools

Today, the power of FPGAs can be made available to embedded system designers and software programmers through the use of software to hardware tools. These tools serve two basic needs: At the front end, software to hardware compiler tools accept high-level descriptions written in a language familiar to embedded software programmers. At the back end, existing synthesis and place and route technologies are combined with system-level platform building tools, allowing designers to develop and target complete systems on programmable logic to specific development boards [8].

C. Soft vs Hard-Core FPGA Processor

1. Hard-Core Processor

A hard-core processor is implemented in the FPGA at the transistor level so it has dedicated silicon on the FPGA. This allows it to operate with a core frequency and have a rating similar to that of a discrete microprocessor. Unfortunately, this hard implementation does not provide the ability to adjust the core for the application, nor does it allows for the flexibility of adding a hard-processor to an existing design or an additional hard-processor for more processing capabilities.

In addition, only specific FPGAs will have the option of having a hard-core. Therefore, the choice of vendors and FPGAs are limited [2]. Such hard-core is the case for the ARM922T inside the Altera Excalibur family and the PowerPC 405 inside the Xilinx Virtex-II Pro, Virtex-4/5 families.

2. Soft-Core Processor

A soft-core processor is built using the FPGA's general-purpose logic. It is a microprocessor fully described in software, usually in a VHDL (Hardware Description Language), and capable to be synthesized in programmable hardware solution [2]. Because of this implementation, the processor will not operate at the speeds or have the performance of a hard-core.

However, in many embedded applications, the high performance achieved by the hard-core processor is not required, and performance can be traded for expanded functionality and flexibility. Indeed, using a soft-core processor alleviates many of the issues encountered due to changing requirements and enables further customization of the system to be performed. Hence, reconfigurable SoC (RSoC) became a reality with soft-core processor [9] since this type of processor have provided a substantial amount of flexibility through the configurable nature of the FPGA.

Furthermore, a soft-core processor solution will prevent designers from being confined to a specific set of peripherals that may no longer fit the application as requirements change or new features are desired. Therefore, soft-core processors may be appropriate for a simple system, where the only functionalities are the manipulation of GPIO (General Purpose Input/Output). They may also fit a complex system, where a multiprocessors architecture is required [9].

IV. XILINX FPGA EMBEDDED PROCESSORS

A. PowerPC405 Architecture

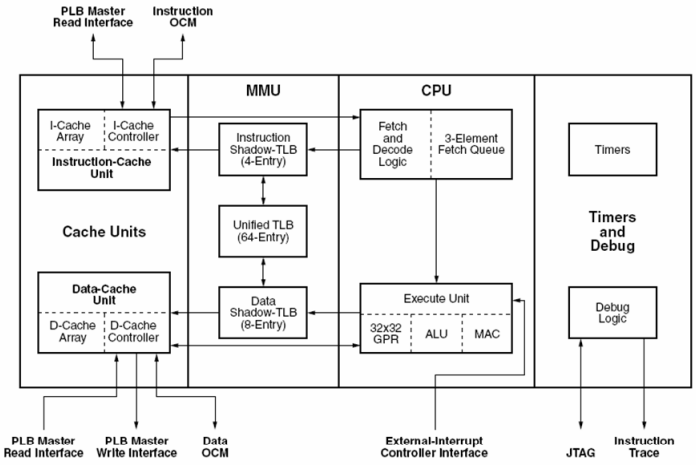


Fig. 1. PPC405 Organization [10].

The PowerPC 405 hard-core processor is available in some Xilinx Virtex families. It has a 64-bit architecture with a 32-bit subset. The various features of the PowerPC architecture are defined at three levels. This layering provides flexibility by allowing degrees of software compatibility across a wide range of implementations [10]. Fig. 1 shows the internal PowerPC405 organisation.

B. MicroBlaze Architecture

The Microblaze soft-core processor is available for both Spartan and Virtex FPGA families. It is a 3-stage pipeline 32-bit RISC Harvard architecture with 32 general purpose registers, Logic Unit (LU), and a rich instruction set optimized for embedded applications [11].

The Microblaze can be customized with different peripherals and memory configurations. The 2 Local Memory Busses (LMB) are used to connect the instruction and data memories. Additionally, the On-Chip Peripheral Bus (OPB) is used to alleviate systems performance bottlenecks and is designed to support low performance/speed peripherals such as UART, GPIO, USB, external bus controllers... [12].

The Microblaze core block diagram is illustrated in Fig. 2.

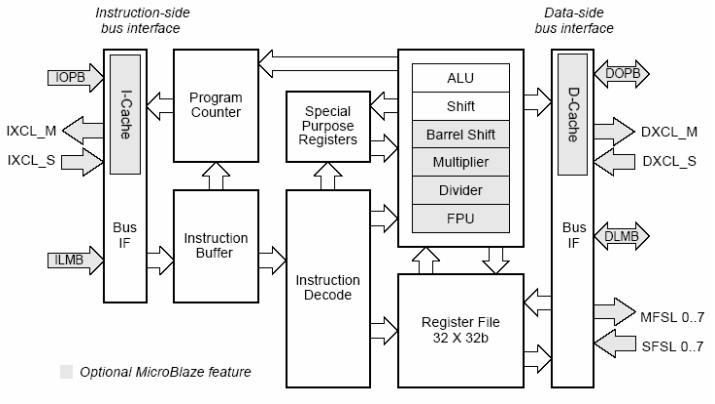


Fig. 2. MicroBlaze Core Block Diagram [11].

The MicroBlaze core has been developed to support a high degree of user configurability. This allows tailoring of the processor to meet specific cost/performance requirements.

C. MicroBlaze / PowerPC405 Operations Treatment Comparison

TABLE I. OPERATIONS TREATMENT COMPARISON [12].

Embedded processor	Integer Arithmetic	Floating Point Arithmetic
MicroBlaze	- Addition and subtraction operations are provided in hardware (Hw) - Multiplication or divide are done in software (Sw) (in the C library) and can be done in Hw - Double precision multiplication, division and mod functions are carried out by Sw libraries	All floating point arithmetic are implemented using Sw functions
PowerPC405	All integer arithmetic are implemented using Hw functions	

Tab. I describes an arithmetic computing comparison between hard and soft-core processors and present the different possible optimizations of processor's parameterizable features for MicroBlaze integer arithmetic.

V. FPGA CORES BENCHMARKS

A. Hard/Soft-Core Bus Benchmark

The industry standard benchmark for FPGA embedded processors is Dhrystone MIPs (DMIPs). Both Altera and Xilinx quote DMIPs for most, if not all, of the available embedded processors.

The achieved DMIPs reported by the manufacturers are based on several things that maximize the benchmark results. Some of these factors include the following:

- Fastest available device family and fastest speed grade in that device family,
- Executing from fastest, lowest latency memory, typically on-chip,
- Optimization of processors parameterizable features [5].

Another benchmark is the Stanford which has as purpose to compute 10 algorithms and reports the score of architecture in each computation field. Then, it reports a weighted average of these scores in two cases: NFPC (No Floating Point Composite) and FPC. Unlike Dhrystone benchmark, scores reported by Stanford represent a time in number of clock cycles.

Using the two previous benchmark for the Xilinx Microblaze and PowerPC405 processors of the VirtexII-Pro device family, with a fixed frequency of 100Mhz, we have obtained results of Tab. II.

TABLE II. DMIPS EXPERIMENT RESULTS

Processor	Memory Controller Bus	Cache	Dhrystone	Stanford	
			DMIPS	NFPC	FPC
PowerPC405	PLB	NO	22.9	1.36e+07	1.24e+08
MicroBlaze	OPB	NO	15.7	1.97e+07	3.63e+08
MicroBlaze	LMB	-	91.4	3.47e+06	6.21e+07

PLB: Processor Local Bus

Tab. II shows that for the Microblaze benchmark, DMIPS depend on the controller used to connect memory. We have more speed performance with LMB controller than the OPB one. Indeed, the program (code, data, stack etc) can be located in both LMB-SRAM and external OPB-SRAM. But, there is a trade-off: LMB is faster but it has lower capacity [11].

Furthermore, the LMB interface include automatically a cache and the Microblaze can be configured to cache instructions or data over either the OPB interface or the dedicated Xilinx CacheLink (XCL) interface [11]. So, if we would compare the performances of the PowerPC hard-core with the Microblaze soft-core, we should consider the case of OPB memory controller for the Microblaze versus PLB controller for PowerPC. Consequently, considering the results of Tab. II, we conclude that the PowerPC is more speed-performant than Microblaze.

B. Benchmark Using Memory Cache

To enhance the performance of the two previous cores, a cache memory can be enabled or used.

The PowerPC in Xilinx FPGAs has instruction and data cache built into the silicon of the hard processor. Enabling the cache is almost always a performance advantage for the PowerPC.

The MicroBlaze cache architecture is different than the PowerPC because the cache memory is not dedicated silicon. The instruction and data cache controllers are selectable parameters in the MicroBlaze configuration. When these controllers are included, the cache memory is built from BRAM. Therefore, enabling the cache consumes BRAM that could have otherwise been used for local memory. Compared to local memory, cache consumes more BRAM than local memory for the same storage size because the cache architecture requires address line tag storage. Additionally, enabling the cache consumes general-purpose logic to build the cache controllers [5].

We note from Tab. III, that enabling the cache for PowerPC processor gives more performance compared to the Microblaze performance using LMB controller.

TABLE III. COMPARING THE PROCESSORS' USING CACHE PERFORMANCE

Processor	Memory Controller Bus	Cache	Dhrystone	Stanford	
			DMIPS	NFPC	FPC
PowerPC405	PLB	YES	114	2.61e+06	2.37e+07
MicroBlaze	LMB	YES	91.4	3.47e+06	6.21e+07

C. Benchmark of Different Memory Usage

FPGA embedded processors provide access to fast local memory, as well as an interface to slower secondary memory. The way this memory is used has a significant affect on performance [5].

As shown in Fig. 2, the Microblaze processor has up to three interfaces for memory accesses: LMB, OPB, and XCL. It uses speculative accesses to reduce latency over slower memory interfaces. This means that the processor will initiate each memory access on all available interfaces. When the correct interface has been resolved (i.e. matched against the interface address map) in the subsequent cycle, the other accesses are aborted [11].

TABLE IV. COMPARING DIFFERENT MEMORY USAGE

Processor	Controller & Memory	DMIPS
MicroBlaze	OPB SDRAM (external)	4.57
MicroBlaze	OPB BRAM (internal)	15.7
MicroBlaze	LMB BRAM (internal)	91.4

The benchmark results of Tab. IV affirm clearly that the fastest possible memory option is the use of a local memory over an LMB controller.

VI. APPLICATION TO MOTOR CONTROL DRIVE

As an application to test the performance of the different previous FPGA architectures, we have chosen an electrical control consisting of the simple case of DC motor drive. The electrical machine control loop is performed as shown in Fig. 3. It consists of two parts: the process to control (motor and inverter) and the control unit (PI current and speed controllers).

The control unit is implemented on an embedded processor core. In order to validate this unit without using a real motor platform, an embedded emulation concept of the process has been used [13]. So, all the digital control loop components of Fig 3 have been implemented on a SoC.

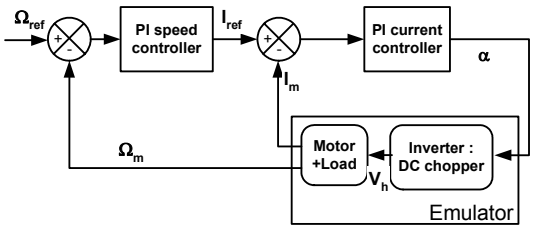


Fig. 3. Control loop diagram for motor drive.

The aim of this ongoing work is to develop a performant RT digital control system for SoC. Indeed, we have previously succeeded to implement a new embedded digital control approach on an FPGA processor core. This approach uses µC/OS-II RTOS that schedules different control tasks handling time-critical events (PI current, PI speed and emulator). However, we have discovered several execution time latencies. Therefore, we propose in this paper to take advantages of different FPGA embedded architectures to improve control performance.

A. µC/OS-II Presentation

µC/OS-II [14] is a completely portable, ROMable, scalable, RT multitasking kernel. It is portable since it has been written in ANSI C and contains a small portion of assembly language code to adapt it to different processor architectures. µC/OS-II has been ported to different processors architectures [15], among them, the PowerPC 405 and Microblaze.

In order to achieve timeliness, priority scheduling is supported. Furthermore, preemption is supported in order to perform a time-critical function. The kernel provides a number of system services such as semaphores, mailboxes, queues, memory management, time management…

µC/OS-II is a small RT kernel with a memory footprint of about 20KB. It is a good candidate for application specific RTOS configuration.

The kernel code is organized into three segments [14]:

- Application Specific Code (in ANSI C)
- Processor-Independent Code (in ANSI C)
- Processor-Specific Code (in ANSI C and assembler)

For our implementation, we have integrated the previous open source segments and we have adapted them to our FPGA platform setting.

B. Platform: ML-310 Board

The used development platform consists of Xilinx ML310 boards illustrated by Fig. 4. It provides a Multi-Processor-System-on-Chip (MPSoC) FPGA. Its Xilinx Virtex-II Pro XC2VP30 FPGA combines more than 30,000 Configurable Logic Blocks (CLBs) and two PowerPC405 hard-cores on a single chip. The large amount of peripherals offers a variety of different interfaces [16].

The Xilinx EDK environment provides the tools and libraries to integrate the IBM PowerPC 405 cores on chip, soft MicroBlaze cores, IBM CoreConnect buses and customizable peripherals to design multiprocessor micro-architectures [17].

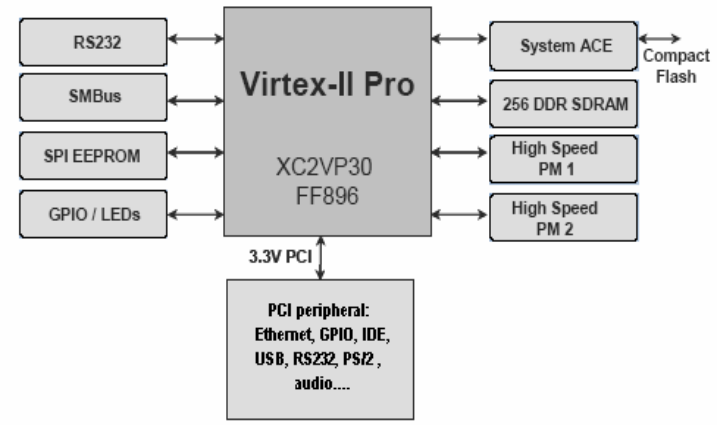


Fig. 4. Xilinx ML310 development platform.

C. *Implementation Architecture Based On The PowerPC405*

Fig. 5 describes in detail our FPGA design block diagram and the different components used in our design. As shown in Fig. 5, the embedded processor used is the PowerPC405 (PPC405). This hard core processor is running at 100 MHz.

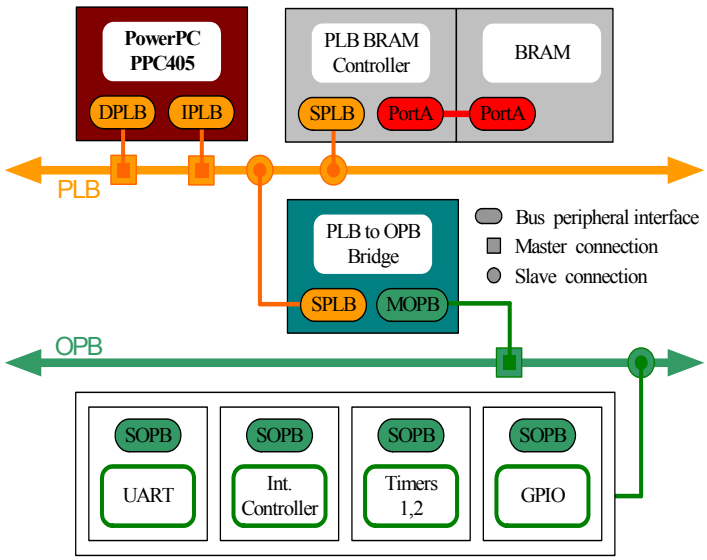


Fig. 5. PPC 405 based design block diagram.

D. *Implementation Architecture Based On The Microblaze*

The secondly design block diagram is implemented in the same Virtex-II Pro FPGA device on the ML310 board. It consists, as shown in Fig. 6, of the same previous peripherals but the processor used is the Microblaze (Mblaze), soft-core processor, running at 100 MHz.

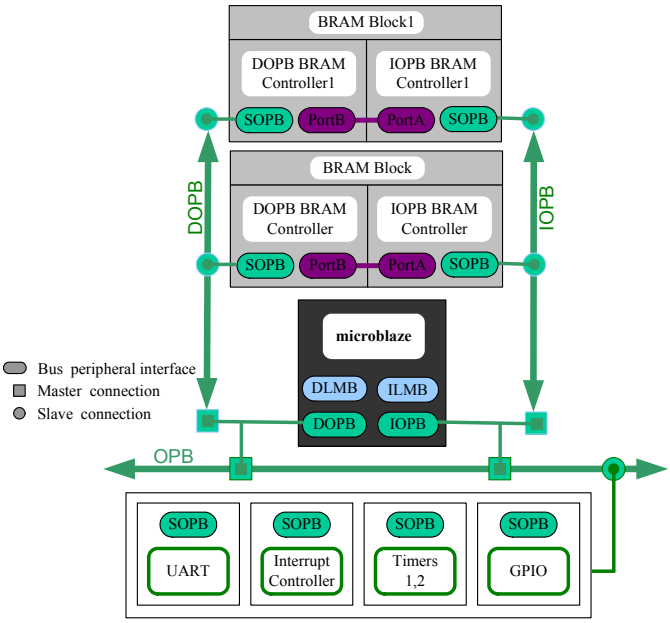


Fig. 6. MicroBlaze based design block diagram.

VII. IMPLEMENTATION RESULTS

As every RT kernel, our µCOS-II kernel has a heart beat, which is configured with an interval timer using a RT interrupt clock as a hardware peripheral device. The timer permits task control on a timed basis using the "OStimedly ( )" routine.

The µCOS-II kernel efficiently manages multiple tasks using a priority-ordered list of pending timer events. Our software code for control consists of 3 tasks, deduced from the control loop shown in Fig. 3.

Respecting physical and digital time constraints presented in Tab. V, the first task, having the high priority, should be the emulator. The goal of this task is to track the real functioning of the set Inverter/DC Motor/Sensors. It needs the pulse-width **alpha** for the DC chopper as input and delivers the current **Im** and the speed **Wm** of the motor. The medium priority task is the PI current controller. It needs the reference current **Iref**, the feedback **Im** and generates **alpha**. Finally, the low priority task is a PI speed controller, using as inputs the constant reference speed **Wref**, the feedback **Wm** and delivers the **Iref** reference current. The last task could run every 70 times of the first two tasks.

To validate the different control loop software components, we have used the hardware UART peripheral to send values of the different control variables, **Im**, **Wm**, **Iref** and **alpha,** to the PC at different time intervals in order to analyse them.

After plotting the feedback speed parameter **Wm** generated by the two architectures, we have obtained the two curves shown in the Fig. 7. The last figure illustrates clearly the difference between the values generated by the soft-core processor and the hard-core one. In addition, the graph shows clearly that for the two cases, speed reached exactly the fixed chosen reference speed 100 rad/s.

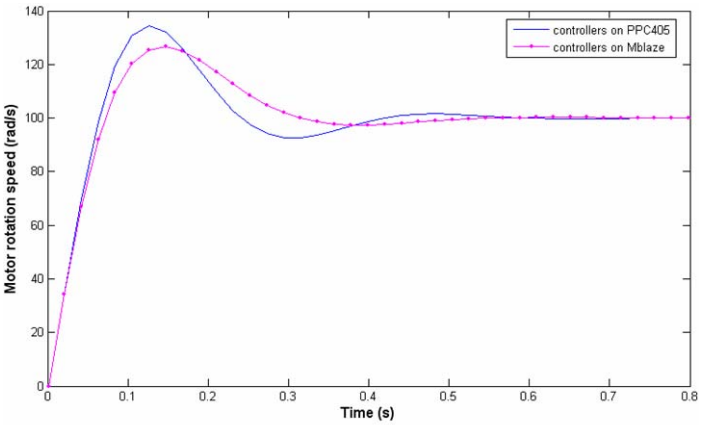
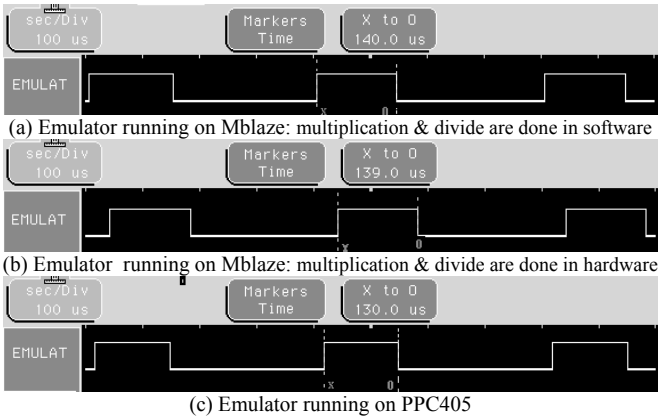


Fig. 7. Comparison between digital control implementation on soft and hard-cores.

TABLE V. TASKS ALLOCATED TIME

Task	Task Delay Time	Allocated Time for
Emulator	300us	PI Current & Speed
PI Current controller	300us	Emulator & PI Speed
PI Speed controller	20ms	Emulator & PI Current

To measure the time that elapsed during the executing course of each task, we have connected an external logic analyser to our target platform using the GPIO pins.



(a) Emulator running on Mblaze: multiplication & divide are done in software

(b) Emulator running on Mblaze: multiplication & divide are done in hardware

(c) Emulator running on PPC405

Fig. 8. Emulator task execution on different processors

Fig. 8 illustrates a performance test concerning the use of soft/hard arithmetic operations for treatment acceleration. We note from the generated waveforms, showing the execution time of emulator task running alone, that the emulator runs on a short time period on the PPC405 core.

TABLE VI. TASKS ALLOCATED TIME

Task	Execution Time	
	On PPC405	On Mblaze
Emulator	138us	153us
PI Current controller	114us	130us
PI Speed controller	66us	80us

In addition, if we run all the software code scheduled by µcOS-II, Tab. VI confirms and illustrates clearly that we obtain more performance using PPC405 hard-core processor for implementation than Mblaze soft-core one.

CONCLUSION AND FUTURE WORK

This paper has detailed different approaches used to optimize the implementation of an FPGA embedded processor system. We have shown that performance can be affected by the use of hard or soft-core, the use of different memory strategies and peripheral controllers or bus.

In addition to the previous hardware architecture optimizations, the embedded designer can perform many software optimizations concerning the code manipulation and compiler options.

The implementation results of the used application (motor control systems with multiple concurrent tasks) illustrate the efficiency of the µCOS-II kernel executing on a hard-core processor.

As suggestions to improve our application design implementation, we propose to use a configuration based on heterogeneous MultiProcessor SoC (MPSoC) with a custom software RTOS. This allows us to move the emulator functionality on another processor in order to reduce the context switch latencies and to allow it to work on real-time conditions at optimal time period.

REFERENCES

[1] E. Monmasson, M. N. Cirstea, "FPGA Design Methodology for Industrial Control Systems—A Review", IEEE Trans on Industrial Electronics, Vol. 54, N° 4, pp. 1824-1842, August 2007.
[2] "Choosing an Intellectual Property Core", MIPS Technologies, Inc, June 2002.
[3] R. Jastrzębski, O. Laakkonen, K. Rauma, J. Luukko, H. Sarén, and O. Pyrhönen, "Real-time Emulation of Induction Motor in FPGA using Floating Point Representation", on Proc 443, Applied Simulation and Modelling, Finland 2004.
[4] J. Regehr, U. Duongsaa, "Eliminating Interrupt Overload in Embedded Systems", CiteSeer.IST, May 2004.
[5] B. H. Fletcher, "FPGA Embedded Processors, Revealing True System Performance", Embedded Systems Conference, San Francisco, USA, March 2005.
[6] D. Kleidermacher, "Minimizing Interrupt response Time", Design strategies and methodologies, Vol 4, N°1, 2005.
[7] E. Monmasson and Y. A. Chapuis, "Contributions of FPGA's to the control of electrical systems—A review," IEEE Ind. Electron. Soc. Newslett., vol. 49, no. 4, pp. 8–15, Dec 2002.
[8] Xilinx embedded magazine, N°2: "Embedded solutions for programmable logic designs: unlock the power of Xilinx programmability", September 2005.
[9] H. Calderon, C. Elena, and S. Vassiliadis, "Soft core processors and embedded processing: A survey and analysis," in Proc. Safe ProRisc Workshop, 2005.
[10] PowerPC Processor Reference Guide, UG011(v1.2), Xilinx Inc, January 2007.
[11] MicroBlaze Processor reference Guide, Embedded Development Kit, EDK 7.1i, UG081 (v5.1) April 2005.
[12] EDK OS and Libraries Document Collection, Xilinx, Inc., July 2005.
[13] S. Ben Saoud, D. D. Gajski, "Co-design of Emulators for Power electric Processes Using SpecC Methodology", UC Irvine, Technical Report ICS-TR-01-46, July 2001.
[14] Jean J. Labrosse, MicroC/OS-II: The Real-Time Kernel. CMP Books, 2002.
[15] G.Schirner, G. Sachdeva, A. Gerstlauer, R. Domer, "Modeling, Simulation and Synthesis in an Embedded Software Design Flow for an ARM Processor", Technical Report CECS-06-06, May 2006.
[16] Virtex-II Platform FPGA User Guide, version 4.0, Xilinx, Inc., 2005.
[17] Embedded System, Tools Reference Manual, Embedded Development Kit, EDK 8.1i, UG111 (v5.0) October 2005.