EPOS-Plasma SoC: A High Functionality, Low Overhead SW/HW Platform for Deeply Embedded Systems

Tiago de Albuquerque Reis and Antonio Augusto Frohlich
Laboratory for Software/Hardware Integration
Federal University of Santa Catarina
88040-900 Florianopolis – SC – Brazil
{reis, guto}@lisha.ufsc.br

Abstract

A system-on-a-chip should be as small and cheap as possible. To achieve this goal, this work presents a SoC that unites the advantages of an application-focused portable API with operating system functionalities with the simplicity of the Plasma processor. Therefore, a small system, in terms of software and hardware, was obtained while maintaining the same functionalities and abstraction level when compared to similar systems.

1 Introduction

Systems-on-Chips (SoC) promotes software and hardware integration to create the final product that is generally focused on a specific application. The increasing complexity of such systems makes the search for design strategies that minimize the responsibility and increase the productivity of designers become an important topic on the field. In this context, Platform-based Design (PBD) shows it's importance proposing platforms as component libraries and it's integration rules [12]. These platforms allows the designer to focus on the application by abstracting the system's details.

Defining methodologies and strategies to generate application-specific systems are a challenge to PBD [8]. A way of obtaining this is through the utilization of API-based development frameworks, with operating system functionalities to allow the designer to focus on the application and abstract the hardware layer.

If such API is also portable, non-recurring engineering (NRE) costs and time-to-market are reduced. NRE costs are reduced because the same application can be used with different system architectures, allowing to choose the most appropriate, while time-to-market can be improved by the provided hardware and software abstractions.

This paper proposes that an application-focused portable API can adapt to different hardware architectures while keeping the same abstraction level to the application. Therefore, processors with different functionalities can be used in the same way, in other words, hardware with less resources are better used. This generates an economy in hardware production costs or FPGA reconfigurable area.

2 EPOS API

EPOS is a framework for building architecture-independent dedicated application support systems [14]. The generated system is composed only by the application and the necessary software and hardware support, which is achieved by generating the system after a domain engineering step. It differs from HALs by better exploring the architectural variability of embedded systems through an extensive domain analysis. This domain engineering consists of a systematic development of a domain model and its implementation [6]. A domain model is a representation of common and variant aspects of a representative number of systems into a domain [1].

This support system comes from the variations and similarities identified on embedded operating systems domain: scheduling politics, synchronization, timing, memory management, interrupt handling and I/O support [6]. Making such characteristics configurable, a certain level of portability is achieved, thus allowing a better adaptation to different hardwares and applications [7]. Therefore, EPOS API tries to be as complete as possible, considering the OS domain, giving these functionalities even for very small embedded computers.

Hardware Mediators abstracts all hardware architecture dependent units [9]. Such artifacts exports the necessary functionalities to the higher level System Abstractions through a software/hardware interface. These abstractions represents the standard operating systems services previously cited (Figure 1).

The execution context, that depends on the CPU registers, and the stack manipulation, that depends on the application binary interface (ABI), are defined by the CPU architecture. To treat such architectural dependencies, the process management is done by a CPU mediator (Figure 2). This mediator abstracts the providing architec-

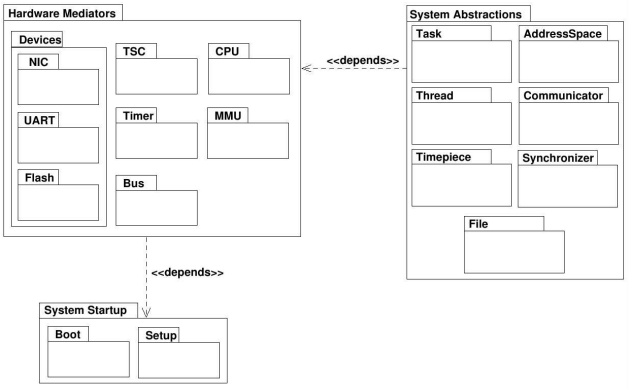

Figure 1. Components overview

ture, for example, the abstraction Context defines the data that characterizes the execution flow and that needs to be stored for context changes.

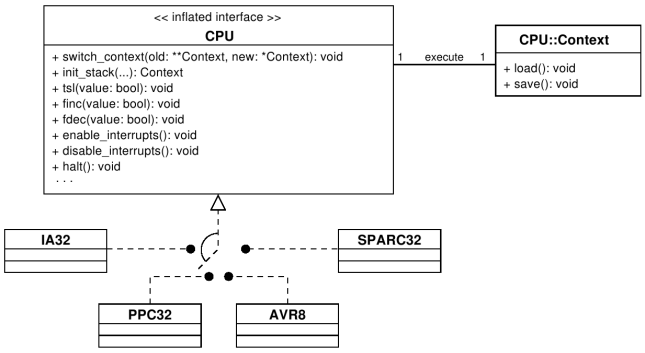

Figure 2. CPU mediator

Process in EPOS are managed by the *Task* and *Thread* abstractions, which represents activities and the entities that perform them, respectively. The *Thread* abstraction implements the standard aperiodic functionalities found in the literature [13]. This abstraction may also be used to define periodic tasks through the *PeriodicThread* specialization which uses an Alarm to execute the thread each period and a Timer to measure the time passing. To manage the task scheduling, the *Scheduler* and *SchedulingCriteria* abstractions are provided. The task scheduling are implemented in two different classes to separate the scheduling politics (criteria) from the mechanism (queue implementation) (Figure 3).

To guarantee data consistency on EPOS' shared process environment, the *Synchronizer* family is used (Figure 4). It is composed of three members: *Mutex*, for mutual exclusion using lock and unlock operations; *Semaphore*, which implements a semaphore with P and V operations; and *Condition*, a condition variable that allows a thread to wait for a predicate in a shared variable to become true.

Also important in multiprocess systems, the notion of time passing is done by the *Timepiece* family. There is an abstraction to generate events (function calls or thread wake-up) called *Alarm*. Time measurements are done

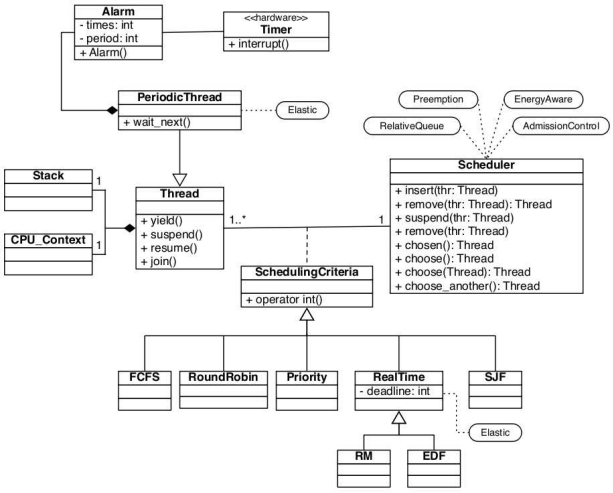

Figure 3. Thread scheduling infrastructure

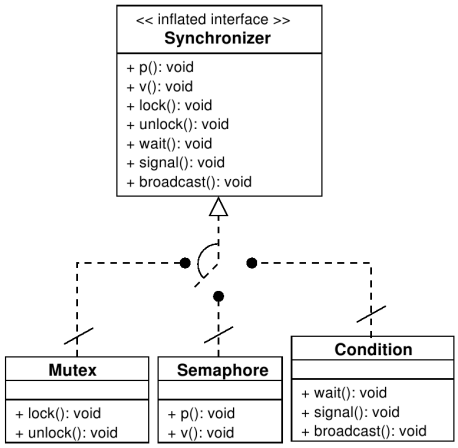

Figure 4. Synchronizer family

through the *Chronometer* abstraction and, for systems with real-time clocks, the *Clock* abstraction stores the current time.

The memory management tasks like address space protection, address translation and memory allocation are done by the MMU family of mediators, which interfaces with the hardware MMU. Systems without MMU considers that both physical and logical addresses are the same, keeping the same interface. The address space is represented by the *Address_Space*, a container to the memory's physical regions called *Segments* (Figure 5).

The external communication of embedded systems is made by sensors and actuators that interacts with the environment where it is located. Such devices may have several access interfaces and each one of them needs a mediator to interface with EPOS.

Beside these basic functionalities, some efforts extended EPOS by adding filesystem support, wireless sensor networks support, DSP API and power management support.

As EPOS was conceived in a storageless environment, the growing number of embedded systems with storage

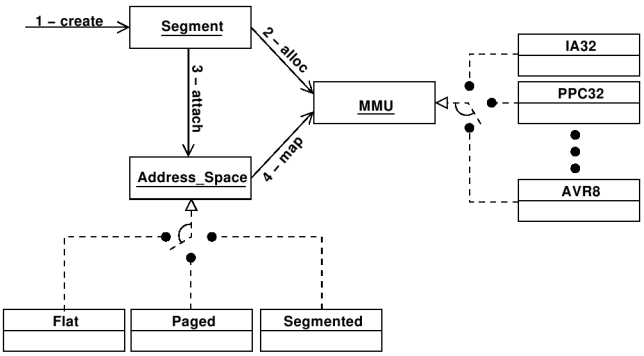

Figure 5. Memory management

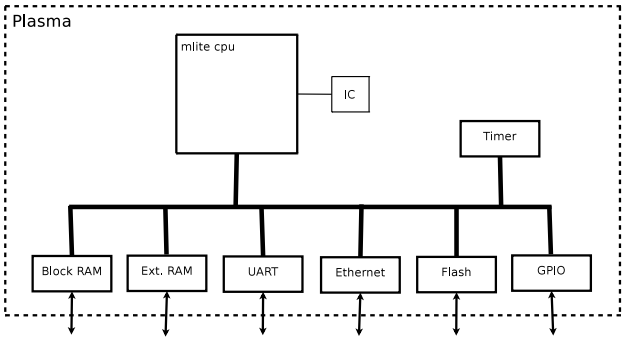

Figure 6. Plasma block diagram

support, such as Flash memories, created the necessity of being able to manage files and organize them in directories. A filesystem support was proposed [5] using a set of families of abstractions to implement the functionalities of the storage hardware as well as volume, allocation and meta-data management.

In his work, Wiedenhoft proposes an approach to explore energy as QoS parameter to battery-powered embedded systems. This work aims at guaranteeing the application to finish and preserve deadlines of real-time tasks without running out of batteries [16].

Motivated by the lack of systems that correctly deals with sensor networks application requirements or presents prohibitive overhead [14] an operating system support, based on EPOS, was proposed to provide applications with hardware support, configurable communication, power management and data acquisition.

As the number of multimedia applications in embedded systems grows, addressing application portability is an important issue since most applications are developed targeting a specific platform. Since tailoring this kind of application for a specific hardware is important for performance, an API is proposed to abstract hardware details. The application gets tied to the API but porting it to different DSPs becomes a simple operation. This API is composed of common DPS functionalities like AAC encoding, FIR filtering or matrix operations and the portability is achieved by hardware mediators, like EPOS itself.

3 Plasma

Plasma is a 32-bit RISC softcore processor that supports all MIPS I instructions, except the patented unaligned load and store *opcodes* [11]. It's currently on the third version and is considered stable.

This MIPS machine can be used to execute code generated by the GNU C Compiler that doesn't generate the unsupported instructions by default. The VHDL code implements either a two or three-stage pipeline, interrupt controller, hardware multiplier, timer and UART, SRAM, DDR SDRAM, Ethernet and Flash controllers running at 25 MHz (Figure 6).

This processor was chosen because of its simplicity and small size thus indicated for embedded systems, besides being open source. This facilitates code changes like inserting and removing functionalities and/or adjusting the processor to different FPGA models.

There are several softcore processor currently available, among them LEON and OpenRISC 1200 are very popular in the literature.

LEON2 is a 32-bit SPARC V8 machine with Harvard architecture (different caches for instructions and data) [3]. It is currently discontinued in favor of its successor: LEON3. It's modular with several IPs that can be inserted or removed from the SoC and a coprocessor interface to ease the communication. LEON's main features are hardware multiplier and divider, interrupt controller, two 24-bit timers, watchdog, 16-bit GPIO, Ethernet MAC, PCI interface and Power Down mode. It also has controllers for PROM, SRAM, SDRAM and UART (Figure 7). On our tests it ran at 54 MHz.

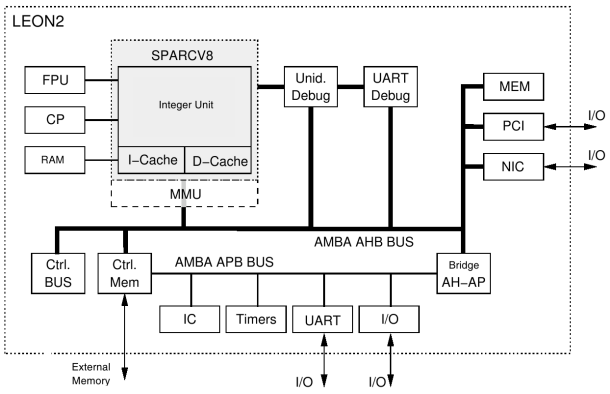

Figure 7. LEON2 block diagram

OpenRISC 1200 is an open source implementation of OpenRISC 1000 family of processors. It's a 32-bit RISC machine based on MIPS with Harvard architecture. OpenRISC 1200 features includes a 5-stage pipeline, virtual memory support (MMU), debug unit, high resolution timer, programmable interrupt controller and power management support [2] (Figure 8).

To make a quantitative comparison between them, their number of look-up tables (LUT) and the size of the same software generated for them was measured. The results can be seen on Tables 1 and 2. The LEON2 and Open-

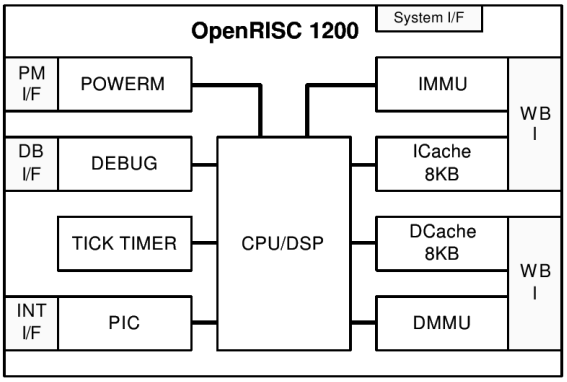

Figure 8. OpenRISC 1200 block diagram [2]

RISC 1200 values were taken from the literature [10] [15].

Table 1 - Size comparison of Plasma, LEON2 and OpenRISC 1200 (LUTs)

	Full	Custom
Plasma	3756	3625
Leon2	14582	6792
OR1200	5286	-

Table 1 shows two sizes for each processor, the total number of LUTs with all functionalities enabled and the number of LUTs using only the necessary part of the processors to execute a producer/consumer application on EPOS. The greater number of features of LEON2 reflects on its size, almost four times bigger than Plasma. OpenRISC 1200 is sightly bigger than Plasma, also justified by the number of features.

Table 2 - Size of EPOS on Plasma and LEON2 (bytes)

	.text	.data	.bss	total
Plasma	16692	68	221	16981
Leon2	8988	28	8400	17416

Table 2 presents the size of EPOS with a producer/consumer application that uses the UART as a shared bounded buffer, for each architecture, except for OpenRISC 1200 which is not currently supported by EPOS. Even with a large different between the processor sizes, the resulting code is almost the same, proving that code size isn't a penalty for a smaller processor size.

4 Mapping EPOS into the Hardware

The first configured entity was the CPU mediator, setted to Sparc V8 and MIPS for LEON2 and Plasma, respectively. Since Plasma has no hardware MMU, both system were set to use a *flat* address space.

The producer/consumer experimental application uses the UART to send and receive data so, to provide the programmer with a high-level communication system, the *Serial_Communicator* abstraction was selected to be instantiated. An interrupt mechanism was used to signalize the threads when new data in the UART buffer is available. To achieve this, both Interrupt Controller (IC) mediator and IP were selected. EPOS implements architecture independent ISRs by using the *handle_wrapper* template-function. This function encapsulates architecture-specific operations, like context saving and restoring.

To complete the support for this application, the Thread and Synchronizer (represented by the Semaphore member) System Abstractions were also selected. As the thread scheduler requires a timer, a hardware timer must be present and its mediator instantiated.

With the System Abstractions chosen and the depending Hardware Mediators configured the systems were generated, obtaining the results previously presented and shown on Figures 9 and 10.

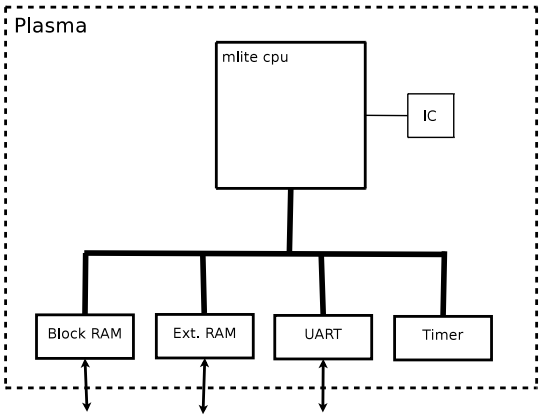

Figure 9. Custom Plasma

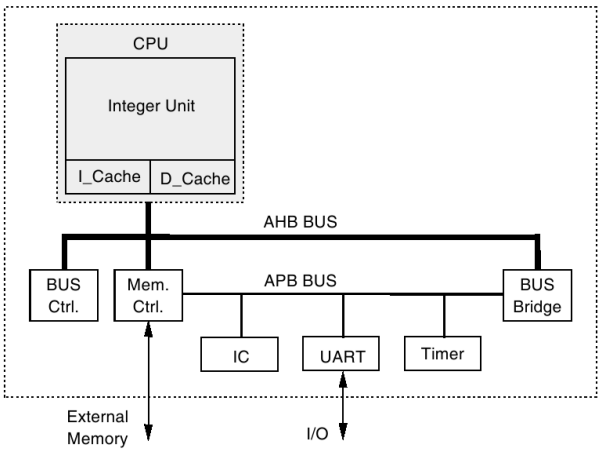

Figure 10. Custom LEON2

5 Case Study

An example of use of such SoC is to do video encoding. By engaging an video encoding hardware module to Plasma, this module can be controlled by software through an application running on EPOS. To achieve that, an encoder proposed by Husemann [4] was used.

In this VHDL implementation, two complementary techniques were used to increase the Motion Estimation (ME) algorithm performance: 4:1 sub-sampling and truncation of the two least significant bits of each sample. The results shown a small loss of quality, less than 0.25 dB of noise, but with a performance increase when compared to other architectures, needing only 130 clock cycles at 1.55 MHz to encode H.264 CIF resolution at 3 frames per second. The chip area needed for this implementation is considered competitive by the authors: 71800 gates, about 860 LUTs.

With a small change to Plasma one can add memory mapped registers to communicate with external modules. This way, the application can control the module's execution through control registers and read and write data through I/O registers.

With the addition of this module to the SoC, the total system size becomes 4485 LUTs (about 377000 gates), 14,6% of a Xilinx Virtex 4 SX35 FPGA configurable area.

Considering the system's functionality, less than 15% of an medium size FPGA can be considered small, thus suitable for embedded systems. As the Plasma processor is used only as a bridge between the software and the encoder module, it still can be used to perform other operations, not limiting the SoC to video encoding.

6 Conclusion

Using EPOS with Plasma and LEON2, two softcore processors with a great difference in functionality number, can be done with the same API which doesn't need to change its size to adapt. This is possible through AOSD, by knowing what the application needs, it uses the hardware resources intelligently. This way, a SoC made with a smaller (and cheaper) hardware may reach the same functional level of larger ones.

This kind of solution is important to embedded systems due to its commercial feasibility is directly connected to the production costs. This work approaches this issue in two ways: decreasing the project's development costs by the abstraction provided by EPOS and decreasing the production costs by using a small and simple hardware that, together with EPOS, owes nothing in terms of functionality.

References

[1] K. Czarnecki, U. Eisenecker, and P. Steyaert. Beyond objects: Generative programming. In *ECOOP'97 Workshop on Aspect-Oriented Programming*.

[2] M. Erlandsson. Openrisc 1000: Openrisc 1200. `http://www.opencores.org/?do=project&who=or1k&page=openrisc%201200`, 2009.

[3] Gaisler Research. *LEON2 XST User's Manual*. 2005.

[4] R. Husemann and V. Roesler. A new approach for high performance motion estimation algorithm implemented in hardware. *Symposium on Integrated Circuits and System Design*, 2007.

[5] H. Marcondes. Um Sistema de Arquivos para o EPOS. Master's thesis, Universidade Federal de Santa Catarina, 2004.

[6] H. Marcondes, A. Junior, L. Wanner, R. Cancian, D. Santos, and A. Frhlich. EPOS: Um Sistema Operacional Portvel para Sistemas Profundamente Embarcados. *Workshop de Sistemas Operacionais*, 2006.

[7] H. Marcondes, A. Junior, L. Wanner, and A. Frohlich. Operating Systems Portability: 8 bits and beyond. *11th IEEE ETFA*, pages 124–130, 2006.

[8] F. Polpeta. Uma Estratgia para a Gerao de Sistemas Embutidos baseada na Metodologia Projeto de Sistemas Orientados Aplicao. Master's thesis, Universidade Federal de Santa Catarina, 2006.

[9] F. Polpeta and A. Frohlich. Hardware mediators: A portability artifact for component-based systems. *Lecture notes in computer science*, pages 271–280, 2004.

[10] F. Polpeta and A. Frohlich. On The Automatic Generation of SoC-based Embedded Systems. *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*, 2005.

[11] S. Rhoads. Plasma - most mips i opcodes. `http://www.opencores.org/projects.cgi/web/mips`, 2008.

[12] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, 2002.

[13] A. Tanenbaum. Modern operating systems. *Upper Saddle River, NJ*, 7458.

[14] L. Wanner and A. Frhlich. Operating System Support for Wireless Sensor Networks. *Journal of Computer Science*, 2008.

[15] J. Whitham and N. Audsley. MCGREP-A Predictable Architecture for Embedded Real-time Systems. In *Proc. RTSS*, pages 13–24, 2006.

[16] G. Wiedenhoft, L. Wanner, G. Gracioli, and A. Frhlich. Power Management in the EPOS System. *SIGOPS Operating System Review*, 2008.