Towards a Field Configurable non-homogeneous Multiprocessors Architecture

Guillermo A. JAQUENOD
Facultad de Ingeniería, Universidad del Centro de la Provincia de Buenos Aires.
La Plata, 1900– ARGENTINA.
<chipi@netverk.com.ar>

Horacio A. VILLAGARCÍA
Comisión de Investigaciones Científicas de la Pcia. de Buenos Aires.
Facultad de Informática, Universidad Nacional de La Plata.
La Plata, 1900 – ARGENTINA.
<hvw@info.unlp.edu.ar>

Marisa R. DE GIUSTI
Comisión de Investigaciones Científicas de la Pcia. de Buenos Aires.
Facultad de Informática, Universidad Nacional de La Plata.
La Plata, 1900– ARGENTINA.
<marisadg@volta.ing.unlp.edu.ar>

ABSTRACT

Standard microprocessors are generally designed to deal efficiently with different types of tasks; their general purpose architecture can lead to misuse of resources, creating a large gap between the computational efficiency of microprocessors and custom silicon.

The ever increasing complexity of Field Programmable Logic devices is driving the industry to look for innovative System on a Chip solutions; using programmable logic, the whole design can be tuned to the application requirements.

In this paper, under the acronym MPOC (Multiprocessors On a Chip) we propose some applicable ideas on multiprocessing embedded configurable architectures, targeting System on a Programmable Chip (SOPC) cost-effective designs. Using heterogeneous medium or low performance soft-core processors instead of a single high performance processor, and some standardized communication schemes to link these multiple processors, the "best" core can be chosen for each subtask using a computational efficiency criteria, and therefore improving silicon usage.

System-level design [1] [2] is also considered: models of tasks and links, parameterized soft-core processors, and the use of a standard HDL for system description can lead to automatic generation of the final design.

Keywords: embedded multiprocessing, system-on-chip, distributed heterogeneous embedded system, programmable logic, IP cores, system-level design.

1. INTRODUCTION

Microprocessors are the dominant devices used in general-purpose computations. Since a standard processor is designed to solve efficiently tasks of different types, resources may be misused or not used at all in a specific application, this lead to a large gap between the computational efficiency of micro-processors (software) and custom silicon (hardware).

If a given application is completely known and it will not change, an ASIC (Application Specific Integrated Circuit) may be designed; however, if the application is not completely known or it can change, a software programmable unit has been usually selected.

As the complexity of embedded systems grows, the industry is being forced to look for SOC solutions (SOC: System On a Chip), and different alternatives for hardware-software codesign must be considered [3][4][5]. Recently, the availability of high-density Field Programmable Logic devices has opened new ways to explore, adding embedded processors to hardware blocks as processing engines [6][7][8].

Almost every modern product behavior can be defined by multi-task processes, and some different solutions must be considered to synthesize it:

- Hardware-only: this solution is well suited for very high speed applications, with simple control decisions (e.g., a router).
- Mono-processor multi task: using a single embedded processor and an OS kernel, multiple tasks may be served in parallel using a time-slice scheme. If real-time constraints must be satisfied, a RTOS (Real Time Operating System) is needed to provide an adequate service latency.
- Multi-Homogeneous processors: if highly parallel tasks must be solved (e.g., image processing), many processors of the same type running under a SIMD multiprocessing scheme may be the optimal solution;. since processors must exchange information, different buses have been proposed according to the coupling degree of processes.
- Multi Non Homogeneous processors: if a process is described by multiple, different tasks, different specialized processors can be used, choosing the best core for each subtask (or group of similar subtasks) with a computational efficiency criteria.

2. MULTI-TASK CHALLENGES

Any multitask application possesses some challenges related to how processes run, how they interchange information, and how they keep synchronized.

Communication and synchronization between processes: When two or more tasks are running concurrently, some methods must be provided for communication and synchronization between processes:

- In single-processor architectures, the use of semaphores and shared memory overcomes the problem; in this case the only new requirement is the availability of indivisible TEST&SET instructions.
- In multi-processor systems, a multimaster bus is often used to access semaphores and shared memory, adding new levels of complexity (bus arbitration, lock-prevent, access latency, etc.); for SOC systems [9] it is usual to find high performance buses like ARM's AMBA, IBM's CoreConnect, Silicore's Wishbone, or proprietary solutions. As an alternative, peer to peer links can be used to communicate processes, changing from bus to mesh topology. An innovative approach to interprocessor communication for tightly coupled tasks were the Transputers' Tlinks [17], where no differences were made –from a software point of view- if two tasks exchanged data in the same or in different processors.

Task scheduling: When one CPU must serve multiple tasks, a task scheduler is required to determine which task will run and how much time, and what other task will be activated after. In almost all cases this schedule is managed by a software OS, being an interesting exception the hardware task scheduler built within each transputer.

3. THE SOPC SOLUTION

With Field Programmable Logic (FPL) devices surpassing one million gates, SOPC integrated solutions (SOPC: System on a Programmable Chip) are a real option for embedded multitask solutions.

Some companies are offering soft-core solutions for proprietary processors (NIOS [13]) or standard (8051 and others [12]); for intensive throughput applications hard-core solutions are also offered, with pre-burned high performance processors (like MIPS32_4Kc or ARM922 [14]) embedded together with large blocks of memory and programmable logic. In any case, no resources are built for hardware multitask scheduling.

An FPL core is a flexible logic fabric that can be customized to implement any digital circuit after fabrication. FPL has led to a new design paradigm, adding great flexibility to the design process:

- Processor selection: when using soft-core solutions, the processor can be selected according to the application needs.
- Processor resources: the designer can define what memory and peripherals will be embedded. This feature provides efficient silicon usage, and the possibility of adding standard or special-purpose peripherals as required.
- IP (Intellectual Property): the reuse of existing validated modules, that can be regarded as library blocks with a given implementation, can dramatically reduce the design cycle.
- Reconfigurability: using programmable logic, a SOPC can be developed before some specifications have been defined, or accommodated to last minute changes. In some cases, reconfigurability can be used for a post-market upgrade or customization feature.

It is of main importance to understand that any SOPC design involves a complex set of hardware-software codesign decisions, with product cost and flexibility in mind, and real-time constraints to be met. Each SOPC defines a specific design scenario: it is obvious that decisions, target costs, complexity and timing constraints for a 622 Mbps ATM bridge are absolutely different to those for an high-end automotive computer, and therefore it is clear that the design criteria will be different.

4. THE MPOC NON-HOMOGENEOUS MULTIPROCESSORS ON-A-CHIP PROPOSAL

The MPOC (***MultiProcessors On a Chip***) proposal is oriented to cost-effective applications: the key idea is that the use of multiple non homogeneous and simple processors [10][11] can exploit the strengths of different architectures for different tasks more effectively than a single high performance CPU. This proposal is not oriented to dynamically reconfigured devices, neither to high-performance mainstream computing applications with complex features such as cache memories, dynamic memory allocation, garbage collection or similar matters.

From a "software" point of view, the MPOC approach looks for a structured and a simple way to build multitask applications, with abstraction of the hardware resources involved in the solution.

From a "hardware" point of view, the MPOC approach looks for a structured way to embed heterogeneous medium or low performance soft-core processors, linking them through the use of standardized communication schemes. The selection criteria for the different processing units involves many aspects, according to the application (general purpose or specialized), the hardware required to manage the multi-process environment, and the memory and I/O needs [18]. The ability of choosing in each case a processor just sized to the requirement can reduce the fitting challenges, minimize routing delays and fan-out, and maximize overall performance in speed and/or cost.

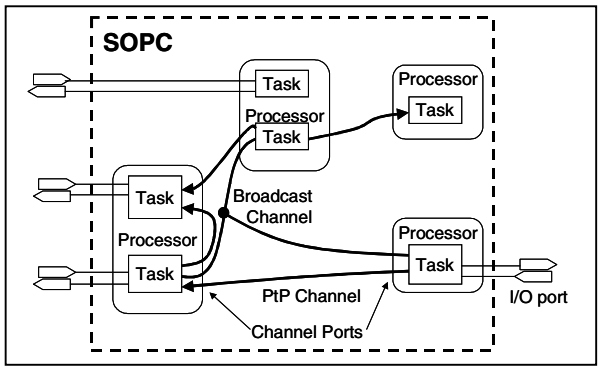
That means, the "best" core can be chosen for each subtask (or group of similar subtasks) using a computational efficiency criteria, and therefore improving silicon usage. Using field programmable logic, software and hardware compilation can be done together, making feasible the hardware/software codesign paradigm.

5. MPOC DESCRIPTION

A SOPC multiprocessor can be described as a hierarchical structure of processors, tasks, channels and ports. Through the use of a standard HDL for system-level description, automatic generation of the final design can be achieved.

A SOPC: is built using a set of processors, I/O ports and channels. From an HDL point of view, the SOPC description will include a complete enumeration of I/O ports, multiple instances of different processors and channels, and the

interconnects between processors, or between processors and I/O ports.



Using VHDL hardware description language, a SOPC entity could be defined as:

```
LIBRARY mpoc; USE mpoc.sopc.all;
ENTITY this_sopc IS
 GENERIC ( pin_mapping: string :=
  "IN_a:23:HiZ,..,IN_k:31:PUP,"&
  "OUT_a:44:PP:fast,..,OUT_b:35:OD:slow,");
 PORT (
 -- global signals
  clk,clr: IN  STD_LOGIC;
 -- external I/O
  ...: OUT STD_LOGIC;
  ...: IN  STD_LOGIC;
END ENTITY this_sopc;

ARCHITECTURE x OF this_sopc IS
 SIGNAL link_1: LPP8S1;
 ...
 SIGNAL link_j: ...;

BEGIN
  processor_1: ENTITY work.mcu_1 PORT MAP (..);
  ...
  processor_n: ENTITY work.mcu_n PORT MAP (..);
  channel_1: ENTITY mpoc.chpp8s1 PORT MAP (..);
  ...
  channel_m: ENTITY ... PORT MAP (..);
END ARCHITECTURE x;
```

Some points must be noted in this top-level description:

- A package ***sopc*** describes non-VHDL data types (such as links) and objects (different types of channels)
- A generic constant string `pin_mapping` is used to define some architectural synthesis attributes of all I/O ports (like pin number, slew rate, pull-ups, etc.). This constant can be inherited from lower hierarchy objects and is used to generate the constraints file required by the HDL compiler.
- The entity's `PORT` field includes the I/O ports used by all internal processors
- The `ARCHITECTURE` local signals instantiate the interprocessor channels, using data types (`LPP8S1..`) defined in the ***sopc*** package
- No specific hardware is described; only processors and channels are instantiated and connected. Note that channels are standard MPOC objects, therefore their description comes from the ***mpoc*** library; instead, embedded processors are special cases of parameterized processors, and they are taken from the ***work*** directory.

A processor: has a processor type and a processor name. Each processor entity is created in the ***work*** directory, as a special instance of a "standard" parameterized processor, with specific parameters values (ROM memory size, RAM memory size, name and type of I/O ports, name and type of channels, etc.), and some optional settings (like DEBUG_STATUS).

```
LIBRARY mpoc; USE mpoc.sopc.all;
ENTITY mcu_n IS
 GENERIC (
  pin_mapping: string :=
  "IN_a:23:HiZ,.., OUT_b:35:OD:slow,"
  code_generator: string := "mcu_n.cmd");
 PORT (
 -- global signals
  clk,clr: IN  STD_LOGIC;
 -- external I/O
  ...: OUT STD_LOGIC;
  ...: IN  STD_LOGIC;
 -- external channels
  ...: OUT ...
  ...: IN  ...
END ENTITY mcu_n;

ARCHITECTURE x OF mcu_n IS
 SIGNAL ilink_1,..,ilinkj: LPP8S1;
BEGIN
 this_cpu : ENTITY mpoc.mc6805
    GENERIC MAP (..=>..,..=>..)
    PORT MAP (..=>..,..=>..);
  ...
END ARCHITECTURE x;
```

The processor description has also some points to be noted:

- The `GENERIC` field only contains the `pin_mapping` constant with the architectural synthesis attributes of I/O ports used by this processor.
- This field also defines a generic string constant `code_generator` to point to a script file ("`mcu_n.cmd`" in this example) needed to transform the source code of the tasks assigned to this processor (written in C, C++, Assembler or other languages) in a synthesizable format. As an example, "MIF" format when using ALTERA FLEX10K devices.
- The entity's `PORT` field now includes not only the I/O ports used by this processor, but also the channel ports required to communicate with other processors.
- If this processor supports multitasking, and the internal tasks must communicate between them, some local signals are instantiated in the `ARCHITECTURE` field for intertask channels.
- An specific instance "`this_cpu`" of a standard parameterized processor (`mpoc.mc6805` in the example) is created, where parameters values are given.

A Task: has a name, a description of the channel ports it uses, an I/O ports enumeration, and a source code; it is assigned to a specific processor, and a specific command file is used to transform the source code description to a synthesizable format.

A task is mainly a "software object", although some pre-compile information (such as I/O ports and channel ports) and some post-compiled information (code size, RAM size, executable code) is used to parameterize the VHDL description of the associated processor.

To enable the use of different C/Assembler compilers and linkers but still forcing coherence, a good solution is to include pre-compile information in the source code as synthesis

directives or attributes formatted as comments, using a ruled syntax (e.g., strings such as ***/*synthesis <directive>*/***); this method to embed directives or attributes is commonly used by EDA design tools [16]. Using these directives, some hardware information (name, type and address of I/O and channel ports, pin numbers of I/O ports) can be automatically exported from the source code to the processors VHDL instances.

A Channel: is usually a hardware virtual object, since it only describes pass-through connections between "talkers" and "listeners"; it is only in the case of multi-master channels, when wired-OR lines are required to resolve access rights (I^2C or CAN), where a channel may consume some hardware resources.

In all cases, however, a channel is used as a formalism to force type coherence in its assigned channel ports. Channels are defined within the ***mpoc*** library, being included as components in the ***sopc*** package.

Every channel instance has:

- A channel name.
- A channel type, associated to a physical link (Peer to Peer, Broadcast Single Master or Broadcast Multimaster, or others) and to a logical protocol related to the handshake method, access arbitration, etc.; each channel type is described as a separate component in the ***sopc*** package.

This is a key point of the MPOC proposal: the use of "standard" channel types having a syntax defined in the ***sopc*** package; if only standardized communication schemes are used, both for inter and intra-processor messages, each processor interface can be designed with abstraction of the partner's type, ensuring connectivity and avoiding a multiplicity of protocols; this feature can also open the door to a multi-vendor portfolio of IP solutions.

The following is a VHDL definition of a "peer to peer" serial port that transports 8-bit packets:

```
LIBRARY mpoc; USE mpoc.sopc.all;
ENTITY chpp8s1 IS
 PORT(
  din: IN LPP8S1;   rxrdy: IN STD_LOGIC;
  dout: OUT LPP8S1; chrdy: OUT STD_LOGIC);
END ENTITY chpp8s1;

ARCHITECTURE x OF chpp8s1 IS
BEGIN
  dout <= din; chrdy <= rxrdy;
END ARCHITECTURE x;
END;
```

This description, included in the ***mpoc*** library, shows how the right type of data is required to define the pass-through connections.

A Channel port: has a fixed relationship with a given channel, a task and a processor.

An I/O port: has an I/O pin number, and is described within a task. As detailed before, synthesis attributes such as "I/O port" and "I/O pin number" are the best way to export these attributes to the HDL description of the processor where this port is connected.

6. THE SOPC PACKAGE

The ***mpoc*** library and the ***sopc*** package are of main importance for the MPOC proposal. As shown in the listing, the ***sopc*** package describes the new data types associated to channels, and the name and type of every channels port.

```
library IEEE; use IEEE.std_logic_1164.all;
package sopc is
 type LPP8S1 is STD_LOGIC;
 type LPP8P  is STD_LOGIC_VECTOR (0 to 7);
  ...
 component chpp8s1
  port (din: IN LPP8S1; rxrdy: IN STD_LOGIC;
        dout: OUT LPP8S1; chrdy: OUT STD_LOGIC);
 end component;
 component chpp8p
  port (din: IN LPP8P;...
        dout: OUT LPP8P;...);
 end component;
 component ...
  port (...);
 end component;
end package;
```

7. THE MPOC BUILDER

The name MPOC BUILDER identifies a Graphic User Interface, still under development, that will be used for MPOC design.

The definition of a MPOC design involves three main steps:

- **Instantiation of processors:** a processor can be created, modified or deleted. To create a new processor it is required to give it a name, and to select its type from the list of configured processors. During this step, or later, the designer can add/modify/remove links to existing channels, and define or clear the assigned task.
- **Channels definition:** a channel can be created, modified or deleted. To create a new channel it is required to give it a name, and to select its type from the list of configured channels. At this moment, or later, the designer can link the channel ports to specific ports of existing processors.
- **Task definition:** a task can be added or removed. When a task is added, the name of the source code and the name of the command file used to process it are both required.

Once an MPOC is defined, a "MAKE" utility can be invoked to process the tasks' source code, to create the processors instances, to build the top VHDL files, and to run the synthesis tool. This utility reads synthesis directives/attributes from the source code, and upgrades an internal data-base, used later to create the processors instances.

Additionally, the MPOC GUI has auxiliary functions to add or remove new types of processors and channels from the list of available objects; this maintenance process upgrades the ***sopc*** package file and adds these new processors to the ***mpoc*** library. Also, it should be configured to define the hardware and the place&root synthesis tools needed to generate the final hardware.

8. SYNCHRONIZATION

Tasks synchronize themselves by messages sent and acknowledged through channels. Although the MPOC proposal defines standard channels from the channel side point of view,

each processor can manage internally the transmission/reception of messages in a different way.

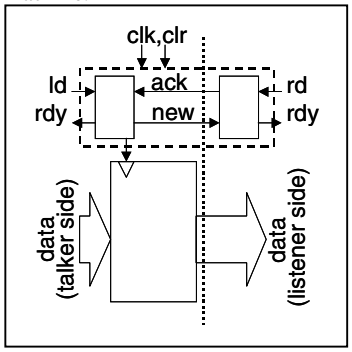
If the processor has an internal RDY feature (analog to the RDY line used to access slow memories), transmission/reception of messages can behave as blocking statements. A transmitter sending a message halts the task until the message is acknowledged; similarly, a receiver request for data halts the task until the message arrives.

If no blocking is desired, interrupts generation or polling can be used for synchronization.

9. EXAMPLE OF CHANNEL PORTS

Proposed channel interfaces are serial monobyte/monoword, parallel links, and shared memories.

A parallel port: The most simple interprocessor port is a blocking parallel port. The talker side uses registers to latch the data and a state machine for synchronization; the listener side only has another small state machine.



- In the talker side the ***rdy*** line is deasserted and ***new*** is asserted – halting the processor- when new data is written (***ld*** asserted); when the listener reads the data, ***ack*** is activated, the listener restarts (***rdy***='1') and ***new*** is cleared.
- In the listener side, if there is no new data when the listener tries to read, the listener processor is halted until data becomes available.

A VHDL93 declaration of both ports could be:

```
ENTITY tx_ptp_8p IS
PORT (
-- global signals
  clk,clr: IN  STD_LOGIC;
-- public signals (channel side)
  txpdat:  OUT LPP8P;
  new :   OUT STD_LOGIC; -- new data flag
  ack:    IN  STD_LOGIC; -- ack from listener
-- private signals (processor side)
  ld :      IN  STD_LOGIC;
  rdy :     OUT STD_LOGIC; -- sync output
  data:     IN  STD_LOGIC_VECTOR (0 TO 7));
END ENTITY tx_ptp_8p;

ENTITY rx_ptp_8p IS
PORT (
-- global signals
  clk,clr: IN  STD_LOGIC;
-- public signals (channel side)
  rxpdat:  IN LPP8P;
  new : IN STD_LOGIC; -- new data flag
  ack:  OUT STD_LOGIC; -- ack to talker
-- private signals (processor side)
  ld :      IN  STD_LOGIC;
  rdy :     OUT STD_LOGIC; -- sync output
  data:    OUT STD_LOGIC_VECTOR (0 TO 7));
END ENTITY rx_ptp_8p;
```

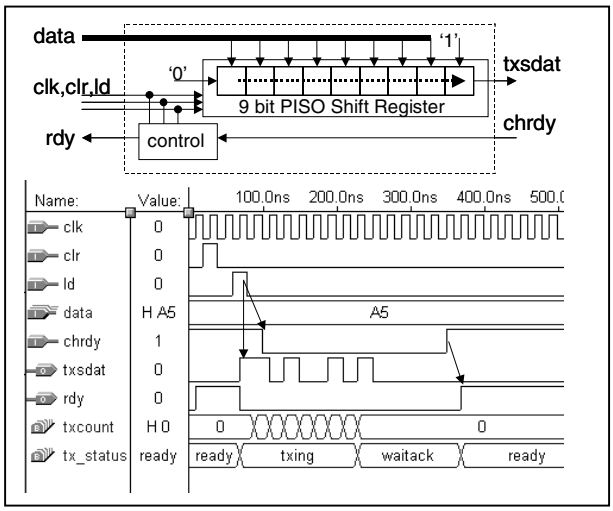
A serial talker: uses 2 lines, one for transmitting serial data and the other for acknowledgment purposes. Each time the talker transmits a byte, a start bit (logic '1') is output on the channel, and then 8 bits of data. For synchronization, after transmission is started, the remote listener deasserts ***chrdy*** and the local processor is halted (***rdy***=0) until the listener reads the data (asserting ***chrdy*** to '1').

A serial channel has the following benefits:

- It is simple, taking only few resources (e.g., less than 30 macrocells when using a FLEX10K device).
- It uses very low connectivity resources within the FPL device.

The block-level architecture, a functional simulation, and a VHDL93 definition of a transmitter port (for peer to peer, 8 bits data, serial single-bit channel), could be as follows:

```
ENTITY tx_ptp_8s1 IS
PORT (
-- global signals
  clk,clr: IN  STD_LOGIC;
-- public signals (channel side)
  txsdat:  OUT STD_LOGIC; -- serial output
  chrdy:   IN  STD_LOGIC;    -- ack input
-- private signals (processor side)
  ld :      IN  STD_LOGIC;
  data:     IN  STD_LOGIC_VECTOR (0 TO 7);
  rdy :     OUT STD_LOGIC); -- sync output
END ENTITY tx_ptp_8s1;
```



In this serial port, the data frame start bit behaves as the transmitter handshake signal (equivalent to ***new*** in the parallel port); in this instance ***chrdy*** is the receiver's response (equivalent to ***ack*** in the parallel port).

Other interfaces: many other interfaces between processors may be proposed, as a function of each FPL architecture. As an example, shared memories can be easily implemented when using ALTERA FLEX10KE devices, where dual port memory blocks (EABs) are available as a standard feature.

10. ADDING MULTITASK CAPABILITIES TO EMBEDDED PROCESSORS

Multitask processors are not common as IP solutions in the programmable logic market. Multitasking management involves context-switching, sleeping/waking up of processes, hardware

schedulers, and channel interfaces, and is costly from the point of view of hardware resources and processing overhead.

However, if the number of tasks is reduced, and they are completely known before the hardware is synthesized, these problems can be enormously simplified:

- Instead of using the stack to save internal registers, a hardware stack based on a circular queue can be added to each register, with as many positions as tasks are defined for this processor.
- Three different events can put a task in the "sleeping" state: a timeout generated by the hardware scheduler, a blocking write on a channel that waits for acknowledge, and a blocking read of a channel that has no new data.
- Since data and code size of each task are known and fixed, a simple adder (or two, for a Harvard architecture) and one (or two) constant generators can be used to transform tasks' addresses in real memory addresses, adding fixed offsets.

If a "round-robin" priority scheme is enough, the task scheduler needed to manage a multitask system is a simple state machine plus a "slice" timer, and it requires very low hardware resources. The inputs to this state machine are the RDY signals coming from the channel ports, and its outputs control the offset adders and the registers stacking queues.

11. CONCLUSIONS

If an application is known before the design cycle is started, an optimized product can be created by tuning the hardware and the software to the application requirements. It has been shown that, as a main difference with standard multitask systems, software is defined before the hardware synthesis process is started.

This new hardware/software codesign paradigm, with intensive use of IP pretested solutions (both for hardware and software), and the fast hardware design cycles offered by Field Programmable Logic, will ensure a very reduced TTM (time to market). But, the most important conclusion is that by defining a common public design environment and some "standard" channels, a new and wide market is open, making feasible the fast integration of IP solutions coming from different partners.

This is a starting proposal, and many decisions are still taken by the designer. A broad research field is opened to look for automatic optimization algorithms (such as Search Explore Refine [15]), and performance & resources evaluation. Another field to explore is how a common interface to OnChip Emulation circuits could be used for multiprocessors debugging.

12. REFERENCES

[1]. J.Plantin, E.Stoy, "Aspects on System-Level Design", Proc. of the 7th. Intl. Workshop on Hardware/Software Codesign. Rome, Italy, May 1999, pp.209-210.

[2]. S.Y.Liao, "Towards a New Standard for System-Level Design", Proc. of the 8th. Intl. Workshop on Hardware/Software Codesign. San Diego, USA, May 2000, pp.2-6.

[3]. P.Coste et al, "Multilanguage Design of Heterogeneous Systems", Proc. of the 7th. Intl. Workshop on Hardware/Software Codesign. Rome, Italy, May 1999, pp.54-58. .

[4]. H.Oh, S.Ha, "A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling", Proc. of the 7th. Intl. Workshop on Hardware/Software Codesign. Rome, Italy, May 1999, pp.183-187.

[5]. R.S.Janka, L.M.Wills, "A Novel Codesign Methodology for Real-Time Embedded COTS Multiprocessor-Based Signal Processing Systems" , Proc. of the 8th. Intl. Workshop on Hardware/Software Codesign. San Diego, USA, May 2000, pp.157-161.

[6]. M.Meerwein et al, "Linking Codesign and Reuse in Embedded Systems Design", Proc. of the 8th. Intl. Workshop on Hardware/Software Codesign. San Diego, USA, May 2000, pp.93-97.

[7]. S.J.E.Wilton, R.Saleh, "Programmable Logic IP Cores in SOC Design: Opportunities and Challenges", IEEE Custom Integrated Circuits Conference, May 2001.To be published.

[8]. P.Chou et al, "ipChinook: An Integrated IP-based Design Framework for Distributed Embedded Systems", DAC-99, http://www.cs.washington.edu/research/chinook/COHPB98.html.

[9]. R.Usselmann, "Open Cores SoC Bus Review", Rev.1.0, http://www.opencores.org, Jan. 2001.

[10]. G.Jaquenod, M.De Giusti: "Diseño de microcontroladores empotrados mediante procesamiento serial: análisis usando FLEX10K para sintetizar un microcontrolador tipo COP8Sax". VII Workshop IBERCHIP, Montevideo, Uruguay, March 22, 2001.

[11]. G.Jaquenod.: "Diseño de un microcontrolador MC6805 usando lógica programable FLEX de ALTERA". VI Workshop IBERCHIP, Sao Paulo, Brasil, March 2000

[12]. ALTERA Corp., "Intellectual Property Catalog". San José, CA, USA, 1999.

[13]. ALTERA Corp., "NIOS Soft core Embedded Processor Data Sheet. Version 1". San José, CA, USA, 2000.

[14]. ALTERA Corp., "ARM-based Embedded Processor Device Overrview. Version 1.1", "MIPS-based Embedded Processor Device Overrview. Version 1.1". San José, CA, USA, 2000.

[15]. R. Dömer et al, "Specification and Design of Embedded Systems", it+ti Magazine N° 3, Oldenbourg Verlag, Munich, Germany, June 1998.

[16]. "Synplicity Synthesis Reference Manual". Synplicity Inc., Sunnyvale, CA. USA, May 2000.

[17]. INMOS Ltd., "Transputer Reference Manual". Prentice Hall, ISBN 0-13-929001-X, UK, 1988.

[18]. H.A.Villagarcía, O.Bria, "Diseño de bloques IP: Programabilidad y Re-utilización". WICC2001, San Luis, Argentina, May 2001. To be published.