FPGA technology for multi-axis control systems

Armando Astarloa *, Jesús Lázaro, Unai Bidarte, Jaime Jiménez, Aitzol Zuloaga

University of the Basque Country, Department of Electronics and Telecommunications, Faculty of Engineering, Urquijo s/n, E-48013 Bilbao, Spain

Abstract

The research presented in this article applies the newest Field-Programmable-Gate-Arrays to implement motor controller devices in accordance with the actual core-based design. The flexibility of the System-on-a-Programmable-Chips in motor multi-axis control systems enables the processing of the most intensive computation operations by hardware (PID IP cores) and the trajectory computation by software in the same device. In those systems, the trajectory generation software may run in powerful microprocessors embedded in the FPGA. In this paper, we present a high-performance PID IP core controller described in VHDL, the design flow that has been followed in its design and how the simulation and the PID constants tuning has been approached. The reusability of this module is demonstrated with the design of a 4 axis SoPC controller. Additionally, an experimental self-reconfigurable SoPC design using Run-Time-Reconfiguration is presented. In this case, the control IP core can be replaced dynamically by another module with another with different features.

1. Introduction

The main drawbacks of the traditional ASIC implementation are the lack of flexibility and the high development costs. Field-Programmable-Gate-Arrays (FPGAs) are hardware in-system programmable devices whose function is not fixed. Specifically, the main advantages of algorithm implementations using FPGAs are: cost efficiency, high data throughput, architecture efficiency and ability to modify and update the algorithm even dynamically.

Moreover, nowadays the Field-Programmable-Gate-Arrays are big enough to fit a whole digital system in a single device. Those System-on-a-Chips (SoCs) are designed using the core-based approach [1], interconnecting pre-designed hardware modules (IP cores) using standard on-chip buses. This design flow is valid for ASIC and FPGA design. Since the number and diversity of the available IP cores for FPGAs has increased greatly, the industry is adopting the core-based design methodology massively using reconfigurable devices which leads to the appearance of the System-on-Programmable-Chip (SoPC) platforms [2]. Apart from the fact that the FPGAs do not incur in non-recurring engineering charges due to their reconfigurable nature, one major benefit of these is the ability to be reconfigured during the execution of the application, even partially. This feature called Run-Time Reconfiguration (RTR) must be able to be integrated into core-based SoPC design flow showing its benefits when applied individually to each core. Some of the benefits of core customization, such as size, power and complexity reduction have already been analyzed by the Department of Electronic and Electrical Engineering of the University of Strathclyde (Glasgow). They focused their research on the application of dynamic reconfiguration to programmable, multifunction cores (PMCs) [3]. In this category one can find circuits such as UARTs, PCI, CRT and USB controllers. As an example of the introduction of RTR into the cores, they obtain area reductions of more than 21% and a simultaneous increment of 14% in maximum operating speed for the UART case. When the RTR is applied to SoPC core based designs, the specific name used to identify these systems is Configurable-System-on-a-Programmable-Chip (CSoPC) designs [4–7].

Although motor control applications are one of the most recent targeted fields covered by System-on-Programmable-Chips [8], closed-loop control algorithms have been studied and implemented previously in FPGAs. Samet et al. implement three different PID architectures (parallel, serial and mixed) in a FPGA [9]. Chen et al. present in [10] a full wheelchair controller implemented on a FPGA using a parallel PID design. Zhao et al. analyze the area, speed and power consumption trade-off between different FPGA PID implementations for small-scale robots [11].

The SoPCs used to implement motor control systems are very flexible in different ways: the number and type of IP cores and processors, bus architectures, hardware and software co-processing, etc. This flexibility allows a multi-axis control system integrating in a single chip, not only the control IP cores, but also the remaining modules of the digital system.

The research work presented in this article covers the three main advances in this field: IP core, SoPC and CSoPC. In the first section a novel FPGA optimized and scalable PID IP core is presented together with the design flow that has been followed in...
its design, and how simulation and PID constants tuning have been approached. This IP core implementation results are presented and compared to other modules reported in the literature. In the second section, the reusability and scalability of the PID IP is analyzed embedding four PID IPs in conjunction with other 12 IPs to obtain a multi-axis SoPC controller. The next step, covered by the third section, is the CSoPC design where the basic infrastructure to allow dynamic core interchanging using partial reconfiguration will be presented.

2. PID IP core implementation

2.1. Hardware description

This IP core is responsible for controlling a DC motor position and speed set in internal registers, using the data provided by a motor encoder as well as generating the PWM output. To do so, the system is composed of four main blocks (see Fig. 1).

1. A quadrature decoder that takes the phase signals from an encoder attached to the motor and gives the current angular speed and position.
2. A PID controller that performs the control algorithm.
3. A PWM modulator that takes the output of the PID and controls a motor drive using a PWM signal.
4. A generic Wishbone wrapper that interfaces the core with a standard bus.

This basic core can be replicated as many times as needed by the application, each of them controlling a separate motor as will be seen in Section 3. Since all the cores are connected to the same on-chip bus, any microprocessor embedded into the FPGA and attached to the on-chip bus can control the PID cores. This microprocessor may be responsible for generating the curves and constants for the different PID controllers.

2.1.1. Quadrature decoder

The first block in the system is the quadrature decoder. This block receives the two phase signals from the encoder attached to the motor and transforms them into the current angular speed and position. The main elements are two counters (one up, one down). Each of them counts the number of edges present in the two phase signals, one of them when the motor turns clockwise and the other one when it turns counterclockwise. Although traditionally this has been done using a CPLD in asynchronous mode [12], using the phase signals as clocks, this implementation is going to be synthesized in a FPGA and is designed to work with several other cores and therefore a synchronous approach has been used [13].

2.1.2. PID block

The PID block follows the classical structure [14]. It contains two saturation blocks, one for the integral part and the other for the overall sum (see Fig. 2). The controller has a pipeline structure of three stages, in other words, it needs three clock cycles to perform all the operations. In order to improve the area and speed, hardware multipliers have been used [15]. These multipliers are included in the Spartan 3 family of Xilinx and subsequent FPGAs. These multipliers have 18 bit input data bus and are signed. This leads to optimum implementation when the fixed point implementation uses less than 18 bit in the two's complement.

The module is fully configurable. The width of the input/output data and constants can be changed. The sizes are independent, which means that different configurations of target width (input), PWM bits number (output) and proportional constant can be used. The multiplicative coefficients are inputs to the systems, not constants. This means that they can be changed while the system is active, as in a software version. Output saturation values are also input to the system allowing them to be changed while working, in a way similar to that of the constants. Any of the inputs can be converted into constant. To do so the input is fixed and the synthesis software is in charge of simplifying the design reducing the area used by the module and increasing the speed.

2.1.3. PWM modulator block

The PWM modulator admits a two's complement input and transforms it into a PWM signal. The module has two outputs, one the modulated PWM and the other one the sign of the
modulation. These two outputs permit direct connection with common Full-Bridge Motor Drivers [16]. The PWM module also generates the enable signal for the control loop. This signal makes the PID controller begin a new cycle and calculate a new PWM input value. The PWM has a saturating block; this saturation value is symmetrical for positive and negative values and can be configured. The ratio between PWM cycles and PID cycles and the number of bits of the PWM input can also be configured. The PWM block also has an on/off input, allowing the disconnection of the modulator and the brake of the motor.

2.1.4. Wishbone wrapper block
The Wishbone generic wrapper is responsible for converting the generic PID core into a Wishbone compatible one [17] in order to facilitate its reusability. The wrapper contains registers for every data input to the generic PID core that is intended to be controlled through the bus, for example target speed and the various constants. The wrapper takes all the registered outputs that are going to be read from the bus, for example the current speed and position. The wrapper includes a Finite State Machine to allow other cores to read the outputs from the PID core and to write to the registers that control the PID core.

2.2. Simulink-Modelsim simulation
A key problem in hardware implementation is simulation. Simulation of complex systems can be very difficult and time consuming. This is greatly increased when interaction with the outside world is included. Communication ports, data input and output systems, etc. can be difficult to model and simulate. Furthermore, a continuous analog electronic system, such as a DC motor controlled by a discrete digital system, is one of those elements that greatly increase the complexity of the simulation task.

On the other hand, the simulation of DC motors, or other complex systems, can be done in Simulink [18]. In this point, Xilinx software System Generator [19] is of great use. This software allows the simulation of VHDL modeled hardware within Simulink modeled systems.

The circuits explained in the prior section have been simulated both separately and within the whole circuit. This simulation scheme allows finding the optimal configuration of bus widths and fixed point signals. This allows the circuit tuning and optimization within a Simulink environment, without any knowledge of the physical implementation in VHDL.

2.2.1. Simulation framework
The simulation of the VHDL core is performed around the Xilinx System Generator. The System Generator is a collection of Simulink blocks that permit interaction between hardware and modeled systems. The toolboxes include a series of hardware blocks, such as multiplexers, logic gates, adders, etc. that can be used to build a system. Another interesting capability of System Generator is that it can include blocks described in VHDL. There is a series of models that allow the translation of signals from the modeling system into the VHDL model and vice versa.

This is the key capability used in the present article in order to simulate hardware modules (written in VHDL) with Simulink/Modelsim models. This has allowed the validation of VHDL described cores using Simulink modeled motors and electrical devices.

The VHDL simulation is done using Modelsim [20] from Mentor Graphics. This hardware simulator admits the interaction with outside programs through FLI (Foreign Language Interface). FLI routines are C programming language functions that provide procedural access to information within Modelsim. A user written application can use these functions to traverse the hierarchy of a HDL design, get information about and set the values of VHDL objects in the design, get information about a simulation, and control (to some extent) a simulation run.

The simulation framework consists of a Matlab/Simulink instance with a special toolbox (the Xilinx blockset toolbox that uses the System Generator) and a VHDL simulator (Modelsim). Data from the electrical model is fed to the VHDL simulator through the FLI. A VHDL simulation cycle is run to obtain the new outputs of the core. These results are taken from Modelsim using the FLI and are converted into the electrical domain. Once in Simulink, these outputs are fed to the system and a new simulation cycle begins. It should be noted that both programs must run simultaneously.

2.2.2. Quadrature decoder simulation
The DC motor simulation module in Simulink, gives the instantaneous angular speed. Although this data is valid for the simulation of the rest of the modules, it cannot be used directly to simulate the quadrature encoder. First of all, the speed must be translated into a series of pulses before it enters the quadrature decoder. In this example, the angular speed is fixed at 50 rad/s. The Quadrature decoder has two outputs, angular speed and position. These outputs are scaled depending on the update rate. In the example the update rate is 0.25 s, since one pulse is given in each phase per turn (non scaled encoder), the speed is given in Hertz and the position in quarters of turn (90°).

Fig. 3 shows Modelsim input and output. The two phase signals arrive and the circuit updates the speed and position every quarter of a second. In this example the speed is 8 Hz (50 rad/9m = 7.96 Hz) positive.

2.2.3. PID block simulation
The simulation of this block allows the selection of the constants that are needed for the proper functioning of the controller. In this step the optimal proportional, integral and differential constants are selected, as well as the size of the fixed point notation of the inputs and outputs. The basic system overview is depicted in Fig. 4. The main electrical elements are the PWM generator, the power bridge and the DC motor. In the example several ramps are tested by the use of a switch. A Simulink modeled PID circuit is also shown to compare the results of the VHDL implementation.
with those of the Simulink PID controller in order to compare the impact of the fixed point implementation.

The result of the simulation can be seen in Fig. 5 while the simulated system can be seen in Fig. 6. It shows the initial convergence as well as another variation of speed at 1.5 s due to a change in the mechanical torque. Approximately at 2.5 s the constant speed is changed to a ramp (see Fig. 5).

2.2.4. PID IP core simulation

The final step, prior to physical implementation, is the simulation of the whole IP core system. The impact of the added non ideal response of the different systems can be seen. One can also evaluate the impact of the global clock frequency in the response of the PWM as well as the interaction between PID controller and PWM.

With this global simulation one can see the impact of the fixed point approximations and see how they propagate through the circuit. In this way, the designer can select the correct size of the buses in accordance with the accuracy and minimal area.

Other point to take into account is the update rate of speed. The PID cannot correct the PWM output while the speed has not been updated. In a similar manner, speed cannot be controlled within the quadrature encoder accuracy. Other critical points are the PWM to PID cycles ratio. If this ratio is too big, and the clock frequency is low, convergence problems may arise. These issues are of no great concern in the hardware implementation since the clock frequency can be really high (see Section 2.3) and the commercial encoders output gives several hundreds pulses per turn, allowing high accuracy in speed.

2.3. PID IP core implementation results

One of the main goals of the hardware cores is their reusability [21,22]. This module has been described using a Hardware Description Language, VHDL specifically, and a full synchronous digital design scheme in order to facilitate the migration from one FPGA platform to another. However, each FPGA vendor is embedding hard modules, like multipliers and memory, spread on the FPGA general purpose resource matrix. Thus, different fine-grain FPGA architectures [23] are emerging. Depending on the HDL synthesizer and the circuit description, those hard modules are mapped and used optimizing the FPGA resources utilization and improving the circuit performance.

This core has been described in a way that the Place and Route tools easily identify the multiplication operations and map them on embedded multipliers if they available in the target FPGA.
In order to evaluate the implementation results of the developed module, it has been implemented in a Spartan 3 low cost FPGA which has 18 × 18 embedded hardware multipliers.

Table 1 summarizes the implementation results of 8 FPGA PID circuits. Zhao parallel [11], Zhao serial [11], Chen [10], Samet serial [9], Samet parallel [9] and Samet mixed [9] are different FPGA PID implementations (see Section 1). PID and WBS_PID are two implementations of the module presented in this research. The first one is a simple example of this PID controller in standalone mode, in other words, without any Wishbone wrapper or embedded microprocessor (see Section 2). The second one, the WBS_PID, a Wishbone interface has been added to the module in order to obtain a full Mix&Run core that fits with the modern platform based design approach [1].

Taking into account that FPGA fine-grain architectures are slightly different, ‘logic cell’ unit has been selected for this comparison. A basic logic cell has a Look-Up-Table for combinational processing and a Flip-Flop for sequential processing and for data storage. When different FPGA architectures are involved in the comparison, the use of ‘equivalent gates’ unit is not recommended because the formula used by each vendor is different [24].

In order to compare these implementations, the following ratio \( \rho \) has been calculated for each approach and summarized in Table 1:

\[
\rho = \frac{\text{Data Throughput}}{\text{Logic Cell}} = \frac{\text{Maximum speed (MHz)} \cdot \text{Bits number}}{\text{Clock cycles} \cdot \text{Logic Cells}}
\]

where Maximum speed is the running frequency of the core, Bits number is the internal precision of the mathematical operations, Clock cycles is the number of clock edges to wait for a new PWM data and Logic Cells is the area occupied by the core.

In accordance with this ratio, the proposed PID IP core offers the highest ratio value (1.85). This PID IP core is a full pipelined 16 bits design that almost doubles the maximum running speed of the fastest one, thanks to the proposed architecture. This takes advantage of the new FPGAs logic cells plus the embedded multipliers utilization. These multipliers help reducing the number of general resources spent and, at the same time, improve the speed performance. Moreover, the internal pre-scalers size has been decreased.

![Fig. 5. Scope output of the angular speed of the motor and desired. There are not visible difference between them.](image)

![Fig. 6. Simulink model for the simulation of the whole system (PID IP core and DC motor).](image)
using a DCM, a Digital Clock Management block [25]. This DCM enables the generation of secondary clocks related to a single input. The high speed clock spread over the dedicated FPGA clock matrix, has been transformed into another two clocks, one for the generation of the update rate of speed and the other for the main application. The update rate is very slow compared to the input clock, so it has been divided to use a smaller counter.

The last column in the table summarizes the implementation results for the Wishbone compatible PID core version. As shown, although in terms of resource utilization the addition of the Wishbone interface is significant, for the SoPC design the utilization of standard interfaces is essential. However, this approach ratio (0.94) is high enough to be the most suitable solution for SoPC integration.

### 3. SoPC multi-axis controller

The next step is the embedding of the core inside a complex system, such as a SoPC. This design will implement several basic PID IP cores interconnected using a standard bus to provide a complete multi-axis controller.

Fig. 7 shows the block diagram of the 4 axis controller SoPC. It has been implemented on a X3S1000 FPGA using Xilinx Platform Studio 8.1 software. The system modules are:

- The Microblaze processor [26]: This is the 32 bit soft processor promoted by Xilinx for the platform-based designs built with its Embedded Development Kit-Xilinx Platform Studio tool. Microblaze is highly configurable. It has local buses (Data Local Memory Bus -DLMB- and Instruction Local Memory Bus -ILMB-) and peripheral buses (Data On-Chip Peripheral Bus -DOPB- and Instruction On-Chip Peripheral Bus -IOPB-). The WBS PIDs are attached to the DOPB bus. The OPB buses are compliant with a reduced version of the IBM Coreconnect specification [27]. This processor runs the software that controls the trajectory and synchronizes the operation of the WBS PID cores. The Xilinx Platform Studio software allows a seamless integration of hardware and software. It uses GNU tools and the software can be described in C or C++. Moreover, from the high level design tool the integration of an Operating System and a subset of Libraries is managed. The tool supports the integration of VxWorks, Linux and a specific Xilinx Kernel as the Operating System of the SoPC. Taking into account that not only high level languages for the trajectory description are supported but Real-Time Operating Systems as well, the selection of microprocessors embedded into FPGAs offers a powerful and easy framework for complex trajectory definitions.

- Memory controllers and memory blocks for the LMB buses.
- Apart from the FPGA internal RAM (block RAMs) the system memory is extended with an external 64 Mbit low cost SDRAM. This dynamic memory is mapped in the OPB bus through the SDRAM controller IP core.
- Two high-speed UART IP core. One UART is used to redirect the stdout messages to an external host. The other UART is used for debugging and upgrading purposes.
- An Ethernet 10/100 M IP core. This module is configured to support full DMA transfers. It uses the external SDRAM to store the

---

**Table 1**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>General purpose resources</td>
<td>1230</td>
<td>912</td>
<td>774</td>
<td>352</td>
<td>1024</td>
<td>550</td>
<td>432</td>
<td>848</td>
</tr>
<tr>
<td>Hardware Multipliers</td>
<td>24</td>
<td>24</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>Bits number</td>
<td>22.58</td>
<td>20.42</td>
<td>30</td>
<td>4.76</td>
<td>8.33</td>
<td>8.69</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Maximum clock speed (MHz)</td>
<td>1</td>
<td>4</td>
<td>0.31</td>
<td>0.005</td>
<td>0.097</td>
<td>0.03</td>
<td>1.85</td>
<td>0.94</td>
</tr>
</tbody>
</table>

*a* PID core without Wishbone IF.

*b* PID core with a Wishbone IF.

*c* Spartan II Logic Cells.

*d* Altera Logic Cells.

*e* XC4000 Logic Cells.
Ethernet frames. This widely extended communication interface, even in industrial networks, can be used to communicate with a remote host, to receive new configurations and for debugging and monitoring purposes.

- A cryptography peripheral to perform 128 AES encryption and decryption by hardware of the Ethernet frames. The aim of this crypto-core is to allow the SoPC to establishment secure sessions at the link layer between endpoints in Ethernet networks [28,29]. This approach, that secures directly the Ethernet packets suits with the requirements of the Industrial Ethernet Networks.
- Four WBS PID cores, mapped in the system memory. Each one is responsible for controlling one DC motor. The cores are attached to the OPB bus, so the Microblaze is able to write and read the target and position registers of all the WBS PID modules. To attach the Wishbone compliant WBS PIDs to the OPB Bus, a simple wrapper that adapts the Wishbone signals to the Coreconnect specification has been included. Fig. 8 shows the detailed attachment of the WBS PID core to the OPB bus through the OPB2WB wrapper.
- To complete the digital system, a timer and an interrupt controller IP cores are included. Also, to control the clock division and synchronization a DCM module is instantiated. This module uses the Digital Clock Manager [30] primitive included in the new Xilinx FPGA.

Table 2 summarizes the implementation results of the whole system in a medium capacity Spartan-3 device (XS31000-5). This implementation uses about the 70% of the FPGA general purpose logic (slices). Upgrading the CSoPC to a 7 axis control system, all the multipliers are used and the 72% of the XS31000 slices are occupied, but the maximum running speed falls to 48 MHz.

Each WBS PID requires 3 embedded multipliers and the Microblaze processor 3 more. With the four WBS PIDs of this design, 15 of the 24 available multipliers are placed. So using this FPGA, the system can easily be upgraded to control a 7 motor system.

Although the maximum global clock frequency obtained for this implementation is about 50 Mhz, this device has 3 Digital Clock Managers not mapped in this design. They can be used to drive different global clock frequencies to any IP core.

To implement the whole control system a 6 layer board (see Fig. 9) has been designed. It includes the main following elements: a XS31000 FPGA, an Ethernet physical layer controller, a 64 Mbit SDRAM and a 16 Mbit parallel FLASH ROM used to store the bitstream and the Microblaze software.

4. CSoPC controller

In the last few years, the Run-Time Reconfiguration [31,32] has been a very active research field for many research groups [33–35]. The interest for this mode of operation has increased greatly because nowadays, the FPGA capacity makes the ‘Virtual Hardware’ concept possible [36]. Moreover, the System-on-Chip design combined with partial reconfigurable device gives rise to self-reconfigurable systems (Configurable-SoPC). For those self-reconfigurable systems, the decisions of when and with which content a given core is reconfigured are taken inside the device by the implemented application. For the FPGA vendors, the RTR is still a research field. The FPGA technology has limitations for this operation mode [37], and the design tools are not stable. However, the addition of RTR support in the newest FPGA design tools, like in Xilinx PlanAhead [38], promises a speedy incorporation of this feature in commercial designs.

But in order to apply partial reconfiguration or self-reconfiguration to SoPC designs, the fact that the internal architecture of the FPGA admits this mode of operation it is not enough. The design that runs in the FPGA, the application, must be able to control the RTR. In this way, the module that is being reconfigured can be disconnected from the on-chip bus and its I/O pins logic level can be controlled. In this field of research, RTR and self-reconfiguration control systems, there are many approaches focused on different applications. Horta et al. [39] describe how communication circuits are implemented as Dynamic Hardware Plugins, reconfigured with data sent over the network. In this case, the reconfiguration controller is implemented outside the main FPGA. Fong et al. [40] propose a Framework for FPGA field updates embedding a reconfiguration controller with cryptographic capabilities and a media interface through the bitstream which is transmitted to the FPGA. The reconfiguration is performed using the ICAP [41] internal reconfiguration interface of the Virtex-II devices. There is no communication between the controller and the static or dynamic section of the design. Danne et al. [42] present a technique to implement multi-controller systems using partial reconfigurable FPGAs. They use an external configuration manager which receives a reconfiguration request from an internal supervisor. The FPGA is divided into two reconfigurable sections, one of them being updated when the reconfiguration is performed. With reference to the application of self-configuration to SoPC designs, the work of Blodget et al. [4] must be highlighted. They present a Self-Reconfiguring Platform (SRP) for Xilinx Virtex-II and Virtex-II Pro. The SRP has a reconfiguration controller built with a soft Microprocessor core (Microblaze) on the Virtex-II or a hard Microprocessor core (PowerPC) on the Virtex-II Pro. The internal reconfiguration interface ICAP is wrapped to fulfill the on-chip bus specification.
building a Reconfiguration Peripheral core. The SRP is completed with a reconfiguration cache built around an embedded memory block (BlockRAM). The communication between the different cores is performed over the CoreConnect® Open Peripheral Bus [27].

In this field of research, self-reconfiguration control systems, the approach of the Applied Electronics Research Team (APERT) of the University of the Basque Country [43] is called Tornado [44]. This control system defines an infrastructure of signals, protocols and logic that are defined to apply safe partial RTR to SoPC core based designs.

Fig. 10 represents a simplified architecture of a SoPC that includes n Tornado Compatible (TC-Cores), which are cores that admit controlled reconfiguration, and z IP-Cores, which have not Tornado reconfiguration control. All of them use a standard interface to be linked with the on-chip bus. The bus topology is only constrained by the bus specification used, having selected for the representation a Shared Bus topology.

A dynamic reconfigurable system requires an extra computation to set the context for each reconfigurable module and apply it. This processing is called metacomputation [45]. For example, in pattern matching applications, the metacomputation would include for each new pattern: The computation necessary to identify or receive the new pattern match, the generation of a new bitstream for the new circuit adapted to the new pattern, and the load of this bitstream into the FPGA configuration memory. The Tornado approach follows the natural architecture of the core based designs, where each core is in charge of doing independent tasks. These cores are able to write a configuration word, that includes information of which module and with which context want to be reconfigured, to the reconfiguration controller (Tornado Advanced Controller -TAC-) through the standard on-chip bus (see Fig. 10).

The reconfiguration controller manages the requests and the application of the partial reconfiguration bitstreams using the signal handshake managed by the Tornado InterFace (TIF). This

Fig. 9. Board prototype for multi-axis SoPC control systems.

Fig. 10. Tornado interfaces for reconfiguration control.
interface is Master for the TAC and Slave for the TC-Cores. The reconfiguration requests to the controller can come from either TC-Cores or in general from IP-Cores, including the hard or soft powerful microprocessors that may be embedded into the platform. These reconfiguration requests are written to the TAC through the on-chip bus. Distributing the metacomputation between different cores the complexity of the reconfiguration controller is reduced.

When the controller has to apply a requested reconfiguration to a target TC-Core, it sets the \texttt{REQ} \texttt{ACK} signal (single for each TC-Core). If the reconfiguration is enabled inside the TC-Core, the TC-Core asserts the \texttt{REQ} \texttt{ACK} signal. Temporally, the TC-Core can reject the reconfiguration (for example, if it is busy in a critical task). In this case, the TAC will retry the reconfiguration request again after having tried the remaining ones stacked in the reconfiguration stack. During the reconfiguration the embedded processor, if it is present into the TC-Core, is frozen. That is, it does not attend its interfaces and the internal modules are stopped (the Program Counter and the access to the Program Memory are locked). The control options for tiny processor that could be embedded into the TC-Cores are specified for each one using two reconfiguration directives included into the assembler. The Program Counter Reset (PCR) and Stack Pointer Reset (SPR) signals define the state of the software execution after the reconfiguration. If asserted, the Program Counter and the Stack Pointer are reset after the partial reconfiguration process.

If the changes the design using partial reconfiguration involve only small modifications (intra-task reconfiguration), and do not imply the FPGA routing changes then, to control the status of the internal logic of the core that will suffer the reconfiguration, is enough with the handshake defined in the TIF. However, when a whole module want to be replaced, this is, the inter-task reconfiguration, the specific characteristics of the reconfigurable technology that will be used must be taken into account. For Xilinx devices, when this reconfiguration modality is used, partial bitstreams involve the logic and routing information of all the involved vertical FPGA configuration frames. Thus, the routing between the static part and dynamic part must be ensured using a special pre-routed circuits (Bus-Macro) and the “Module Based Design Flow” must be followed. Tornado faces inter-task reconfiguration with a specific Bus-Macro designed to fulfill both Tornado control protocol and Xilinx routing specific requirements.

Fig. 11a shows the Tornado compatible Bus-Macro block diagram. This pre-routed and relocatable module has two on-chip Wishbone interfaces. The slave one, at the left, links the Bus-Macro with the CSoPC static part on-chip bus. The master Wishbone interface, on the right, is used to connect dynamically interchangeable Wishbone compatible IP-Cores, wrapping them. The Bus-Macro has a Tornado slave interface at the left that manages the reconfiguration handshake with the reconfiguration controller implemented in the static section of the design. Inside the Bus-Macro a small Finite State Machine made with two slices is stored. This FSM is in charge of controlling Tornado slave interface handshake signals. To complete the Bus-Macro, some pre-routed signals have been included to connect I/O ports located in the dynamic sections to signals generated in the static one. In Fig. 11b the pre-routed Bus-Macro module is represented. To fix the defined routes, tri-state buffers have been used following the Xilinx directives given for Spartan and Virtex devices.

In order to apply the ‘Virtual Hardware’ concept to the presented motor control SoPC, Tornado infrastructure has been included in the design. The system needs intra-task reconfiguration because a replacement of a whole IP-Core is required. Fig. 12 depicts the resulting CSoPC. The dynamic section is restricted to the right side of the design separated from the left side by a vertical boundary. This boundary also matches with the separation made in the FPGA resource matrix (logic and routing resources). Thus, in the right section an IP core may be loaded or replaced by dynamically loading a partial bitstream in the SRAM FPGA configuration memory. Although SRAM FPGA configuration memory write access is usually accessible, each FPGA vendor sets some specific rules that must be followed when partial reconfiguration is going to be accomplished. For example, Xilinx sets these main rules for dynamic partial reconfiguration [37]:

1. The dynamic reconfigurable section must be bound to a restricted area. Both in logic resources and routing resources.
2. An especial design flow must be followed.
3. A pre-mapped and pre-routed module must be instantiated in the HDL design to ensure the proper connection between the static and dynamic sections. Because of the routes chosen by the Place and Route tools cannot be constrained, for a given dynamic core the connection routes between its on-chip bus interface and the on-chip bus would be different. The relocatable pre-routed modules used to ensure a proper connection are called Bus-Macro [46,32,37].
4. The application must provide a proper handshake mechanism to ensure that the partial reconfiguration is made at
the right time and safely; in other words, disconnecting the
dynamic part from the on-chip bus and controlling the
dynamic core I/O ports.

To implement this CSoPC in a partially reconfigurable FPGA,
these rules have been accomplished with Tornado in this way:

- ISE Modular Design flow [37] has been followed to ensure that
  the rules 1 and 2 are fulfilled. However, the static part has been
  built using Xilinx Platform Studio 8.1. Although this tool does
  not admit Modular Design flow, the static part can be exported
to ISE tool as one module and follow the Modular Design flow.

- To ensure the link between the static and dynamic sections, a
  Tornado Bus Macro [47] is set between both areas. The routing
  links are made using internal FPGA tri-state buffers. It wraps
  the dynamic IP core enclosing the on-chip bus interface (Wish-
  bone). This Bus-macro provides also a small logic to arrange
  with the reconfiguration controller when the dynamic core
  replacement can be archived.

- The control handshake, rule 4, is carried out by the reconfigura-
tion controller IP core (Tornado Advanced Controller -TAC-,
  Fig. 12). This controller is in charge of receiving the Configura-
tion Request Words written by any module attached to the
  on-chip bus with writing capabilities. Each word has the number
  of the dynamic IP core that want to be loaded. The reconfigura-
tion controller stacks the reconfiguration request and agrees
  with the Tornado Bus Macro through the master Tornado Inter-
  face in the reconfiguration controller and the slave Tornado Inter-
  face in the Bus-Macro. The Tornado Advanced Controller
  writes the partial bitstreams through the internal Virtex-II
  reconfiguration port, called ICAP [48]. Thus, no external access
  is needed. However, the partial bitstreams size can be quite
  big, in those cases an off-chip storage is required. The reconfig-
  uration controller has a master Wishbone interface through it
  can read both internally stored bitstreams and externally stored
  ones accessing to the proper IP core interfaces (FLASH IF, SDRAM
  IF, etc.).

The physical implementation of this CSoPC has been achieved
using a X2VP100-6ff1696 Virtex-II Pro device. This FPGA has
an older FPGA architecture than the Spartan-3 one. However,
compared to the Spartan-3 architecture, the Virtex-II Pro architecture
has tri-state buffers, which Tornado Bus Macro needs, and the ICAP
module that enables internal access to the SRAM configuration
memory.

In the dynamic section, any IP core with a slave Wishbone inter-
face that links with the Tornado Bus Macro, and fits in the se-
cion is a valid candidate to be loaded dynamically. In this CSoPC,
three dynamic modules are interchanged. The first one is the
WBS PID presented in this paper, the second one is the AES IP core
module (Wishbone version) and the third one is an intelligent
ADC converter module [49,35]. The dynamic area is big enough
to fit the largest core, the AES IP core. This area is about the 20%
of the FPGA matrix and is configured with a 200 Kbyte partial
bitstream.

The addition of the infrastructure to control the RTR has a cost
in terms of FPGA resource utilization and time penalty. FPGA
resources are consumed by the Tornado Bus Macro and by the
Tornado Advanced Controller. The Bus-Macro needs two tri-state
buffer for each signal, taking into account that the bidirectional
ones must be split in two. It also includes a minimum logic (2
flip-flops and one Look-Up-Table) to manage the reconfiguration
control handshake with the Tornado Advanced Controller.
This reconfiguration controller is implemented using 143 Virtex Slices
and 1 Block RAM, less than the 1% of the x2vpX2VP100 FPGA
resources. Thus, the FPGA resource overhead is not significant for
the new high capacity FPGAs.

However, the time penalty must be taken into account. The pro-
posed CSoPC is focused on inter-task [50] dynamic reconfiguration.
In other words, a whole IP core is replaced. In this case, the partial
bitstream may be quite large, what in addition to the internal
reconfiguration control handshake may arise in tens of millisec-
onds global reconfiguration time. Depending on the application,
this delay can be acceptable: the static section is always running
and the dynamic section is under control during the reconfigura-
tion (I/O pins and on-chip bus connection).

5. Conclusions

In this paper we have presented a VHDL described PID IP core.
This PID IP core is capable of achieving great speed due to the parallel
nature of FPGA designs. This core contains all the elements
needed to control a DC motor, from the decoder to the PWM mod-
ulator. The core is fully flexible in terms of size of operands, it also
allows ‘hot’ change of the PID constants and limits. The PID has
been provided with a standard interface that facilitates its integra-
tion in SoPC designs. The powerful 32 bits processors of the new
FPGAs, in conjunction with these PID IPs, can compute complex
trajectories. The design has been done to optimally fit the internal
architecture of Xilinx devices, using embedded hardware resources
such as multipliers to enhance the performance. The constant tuning problem has been solved using a simulation framework that allows the interconnection of electrical equipment modeled in Simulink and circuits described in VHDL. In this way the overall circuit, not only the digital part, can be validated.

In order to prove the reusability and modularity of the proposed architecture, we have integrated four PID IPs with a 32 bit processor and many IPs in a single low-cost FPGA device. This SoPC is a full 4 axis controller, easily scalable to other configurations.

The ‘Virtual Hardware’ concept using dynamic partial reconfiguration has been introduced in motor control system presenting a CSoPC controller implementation. It is provided with a proper control infrastructure to manage IP cores dynamically with involves a new concept in motor controller design.

References