American J. of Engineering and Applied Sciences 5 (1): 15-24, 2012
ISSN 1941-7020

Dynamic Partial Reconfiguration Contribution on System on Programmable Chip Architecture for Motor Drive Implementation

Hedi Abdelkrim, Slim Ben Othman,
Ahmed Karim Ben Salem and Slim Ben Saoud
LSA Laboratory, Department of Electrical Engineering,
National Institute of Applied Sciences and Technology,
B.P. 676, 1080 Tunis Cedex, Tunisia

Abstract: Problem statement: Nowadays, Reconfigurable System on Chip (RSoC) shows great potential in many high performance applications that benefit from Hardware customization. **Approach:** In this study, we present a design approach of FPGA based Controller for electromechanical system. In this way, we present solutions obtained by Hardware/Software Code sign methodology targeted for the implementation of a motor control drive system using Multiprocessor SoC (MPSoC) architecture. In order to enhance flexibility and performance of the considered system, we design different modules of HW current controller of electronic motor. A Dynamic Partial Reconfiguration (DPR) mechanism allowing switching on the fly between those modules is described. **Results:** Test and validation are done to validate the approach adopted. Experimental results confirmed the efficiency of the approach and allow us to determine more recommendations that should be considered while designing a RSoC control drive system. **Conclusion/Recommendations:** DPR enable flexible control system hardware design. This concept allows switching between different low order controls.

Key words: Motor drive, dynamic partial reonfiguration, MPSoC, FPGA, RSoC

INTRODUCTION

Nowadays, due to technology advances and also with the increasingly requirements from applications supported by contemporary embedded systems, they became more and more complex, which have a direct impact on the design of such systems.

Besides low cost and tight time to market, other constraints, like for instance the limited amount of memory available, low power consumption requirement (Paiz and Porrmann, 2007) make the design of such a system a challenge.

Current embedded systems integrate more and more functionalities and require, therefore, higher computation performance. With technological advances most of system components can be incorporated into a single chip leading us to a so called System-on-Chip (SoC) (Jerrraya, 2004).

System on Programmable Chip (SoPC) such those based Field Programmable Gate Arrays (FPGA) solutions are replacing traditional digital technologies such as general purpose processors, DSP or ASIC, thanks to their better cost/performance ratio or due to shorter time-to-market.

When implementing digital controllers using SoPC, there are many advantages, offered by their parallelism especially regarding the required computation time. Implementing a controller on FPGA can lead to a faster response in motion drive applications. However, the performance and efficiency of such FPGA technology strongly depends on the design methodology of the controller (Salem *et al*., 2008).

Hardware (HW) solutions suffer from the lack of flexibility, but their reliability and performance make them more interesting. The reconfigurability, particularly the dynamic partial reconfiguration has made hardware solutions more flexible. In fact, it has given the FPGA the capability to modify its internal structure on the fly, while it is turning on.

Designers were often forced to begin architecture design and system implementation before the specification of a product is fully completed. In such an environment, late changes in the design cycles are mandatory. Reconfigurable HW promises to best solve these problems and challenges.

Corresponding Author: Hedi Abdelkrim, LSA Laboratory, Department of Electrical Engineering,
National Institute of Applied Sciences and Technology, B.P. 676, 1080 Tunis Cedex, Tunisia

In fact, to overcome the constraints due the static system realization where the reconfiguration of the FPGA does not change during system running, leading to high resources cost, dynamic and partial HW reconfiguration can be used, allowing loading only the suitable controller for the current situation of the system.

This study deals with the design of a Runtime Reconfigurable DC-Motor Controller. A Multiprocessor System on Chip (MPSoC) architecture is used to validate the design. In fact, this platform includes a processor executing the model of the motor and a second one for adapting the controller's HW/SW wrapping.

The next section gives an overview of the reconfigurability concept with some design recommendations and different reconfigurability strategies. Then we present the partitioning model for motor control application process and give a generic multiprocessor architecture supporting a mixed HW/SW code sign of controller devices. After, the design flow of the Reconfigurable Module (RM) and its implementation steps are given. After that, the obtained results are presented and a future work discussion is done. Finally, conclusions with perspectives are given.

It is important to precise that the DC motor application case is used only to validate the approach proposed.

Reconfigurability general overview:

Background and related works: The introduction of FPGA, made the digital systems design flow in continuous changing (Brown and Rose, 1996; Hauck, 1998). In fact, the Dynamic Partial Reconfiguration (DPR) has allowed the appearance of a new paradigm: HW as flexible as programming (Abdelkrim and Saoud 2008). This DPR computing consists of an execution of various algorithms on the same HW structure device, by several reconfigurations of the HW chip array in a limited time period and with a defined partitioning and scheduling mechanism. DPR offers important benefits for the designs implementation. Several architectures have been designed and have validated the DPR computing concept for the real time processing (Wirthlin and Hutchings, 1995; Vuillemin *et al*., 1996; Kastrup *et al*., 1999; Goldstein *et al*., 2000; Kessal *et al*., 2000; Donlin *et al*., 2005). However, optimal partitioning of an algorithm is not defined yet. Exploiting the Run-Time Reconfiguration (RTR) mechanism is a domain in which many works has left. The works in the domain of temporal partitioning and logic synthesis exploiting the dynamic reconfiguration generally focus on the application development approach (Zhang and Ng, 2000). Thus, firstly we observe that the efficiency obtained is not always optimum with respect to the available spatiotemporal resources. Secondly, the choice of the number of partitions is never specified. Thirdly, this can be improved by a judicious temporal partitioning (Tanougast and Weber, 2001).

The reconfigurable architectures can easily be used in power electronic domain particularly in fault diagnosis and their compensation. In fact, the reconfiguration of the converter topology and the control algorithm has great real time constraints. Besides, the FPGAs flexibility is very interesting when we have to commutate quickly from an algorithm to another (Imecs *et al*., 2000; Sklyarov, 2000).

Figure 1 gives a Multiprocessor design example of an RSoC for a motor control drive. Different versions of a reconfigurable HW module controller are designed and stored in a memory. This memory can be internal (DDR, BRAM) or external (Compact Flash CF, EEPROM).

Some FPGAs allow performing DPR, where a reduced bit stream reconfigures only a given subset of internal components (Mermoud *et al*., 2005). Xilinx has proposed two PR flows: module based and difference based Xilinx, 2004. For the first case, the designer has to edit low-level modifications manually; a PR bit streams related to these modifications is generated as described in Xilinx, 2004. For the second case: the Module Based flow, the whole design is partitioned on different modules. Each module is treated separately in different phases. For earlier ISE versions communications between modules was censured by hard Bus Macros. Actually, with new FPGA chips and ISE versions, Bus Macros are no longer used.

Reconfiguration design recommendations: When designing a RSoC some recommendations needs to be respected. In fact, the Partial Reconfiguration (PR) system design involves an FPGA area partitioning: static and dynamic region. As mentioned before the design of the different modules is done independently and as a final step merging the different design (Hagemeyer *et al*., 2007).

HDL design rules:

Top level design module: All the lower level modules instantiated in the top level module must be treated as black boxes. The top-level must contain only instantiations of: Clock primitive (DCMs and BUFGs), Input Output signals, base design, PR modules, bus macros and signal declarations.

The Bus Macro ensures all the crossing signals (communication one) between PR modules.

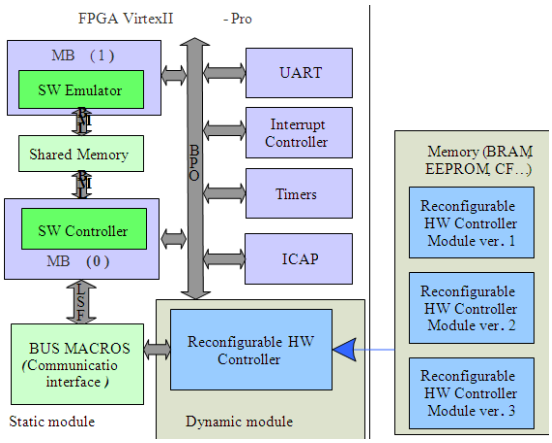

Fig. 1: Example of a RSoC

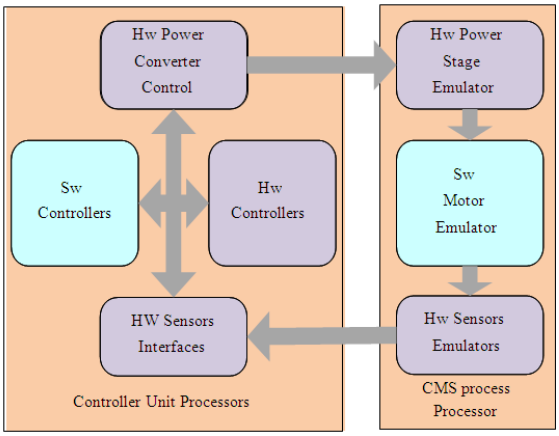

Fig. 2: Generic partitioning model of motor drive

Lower level modules (or base design modules): These modules cannot contain any clock or reset-related primitives (for example, BUFG, DCM…), Xilinx 2006. While Xilinx Embedded Development Kit (EDK) places the DCM primitives in a lower level modules, the designers has to take care of this.

PR modules: In order to be linked correctly to the Top level instantiations, all PR Modules for a given PR region must have the same module and file names, all these developed modules should also have the same pins placement and names.

Reconfiguration strategies: There are several ways to reconfigure a device, which will determine its complexity and its autonomy.

The first possibility is to reconfigure a device externally through JTAG cable, based on an external decision and data source.

The second one is to let the device itself taking the decision when to improve the performance by modifying partially a portion of the target device. This operation is done through the Internal Configuration Port Access (ICAP). Self-reconfiguration can be applied by simply giving a partial bit stream to the ICAP interface. Blodget *et al.* (2003) demonstrates the benefits of dynamic self-reconfiguration through ICAP.

The third and not the last possibility is a hybrid solution could be considered, it consists of a combination of the above.

Drive control structures:

Digital controllers devices: A motor drive system is composed of (1) the process to control including the converter circuit, the electric motor and load components and the different used sensors and (2) the digital control unit which is composed of the digital controller based on a specific algorithm and the necessary interfaces for sensors outputs acquiring and control signals generation (Othman *et al.*, 2010a).

The control unit is modeled according to the applied CMS (Converter/Motor/Sensors) process and to its target input. Generally, this unit is shaped by several overlapping control loops. For example, the position control unit for DC machines can be composed by three functional control blocks: the current loop inside the speed loop inside the position loop.

In this work, we seek solutions, which enable us to implement on the same SoC the Converter Motor Sensor (CMS) emulator and different control unit. The use of MPSoC architecture allows the emulator to work on RT conditions at optimal processor execution time without affecting control unit interrupts working at higher time. Figure 2 presents a generic approach of partitioning model for motor drive design.

In order to validate our approach, we consider the case study of DC process speed controller. We assume that work on simple process templates allows us to validate our approach. However, it can be easily generalized to any other process control.

Motor control specifications: The control application design was built using Xilinx EDK Base System Builder toolchain. Both the reconfigurable interconnections associated to Configurable Logic Blocks (CLBs) (HW) and the programmable embedded processor cores (SW) were used in varying combinations so that application can be rapidly tested for performance, by selectively partitioning the design

into portions suitable for the HW or for SW resources on the FPGA.

The process (in Fig. 3) is based on two nested closed loops. The inner loop is in charge of the current control while the external loop manages the motor speed. To achieve powerful processing capabilities in the control unit, we have considered a specific HW accelerator approach for the controller processor by implementing the current controller in a HW part and controlling it by a SW part. The CMS emulator is used in order to simplify the validation of the concept.

In addition, motor emulation (Saoud and Hapiot, 2000) is an interesting approach to complete the validation of new digital control unit and to perform the diagnosis tasks. The objective of this approach is to design an electronic system, which can reproduce the physical system functioning in RT and with high precision (Salem *et al*., 2010). This system, called emulator, will be used for the new control device validation with the opportunity of extensive testing, before it is switched for use with the physical process in real conditions.

The proposed architecture allows integrating a full control system in a single chip, avoiding external components and additionally reducing cost and complexity. Additional application-specific components such as Pulse Width Modulation (PWM), encoder can be added as custom HW IP without major adapting.

MPSoc architecture: The architectural model is designed for FPGA MPSoC architecture in the Xilinx ML310 board Xilinx 2005. The architecture consists of two soft MicroBlaze (MB) processors and a set of modules interconnected with buses: a first processor for the control unit and a second one for the emulator unit. MB processor use Local Memory Bus (LMB) as the local memory interface. Slow peripherals such as timer, UART and interrupt controller are available with the On-chip Peripheral Bus (OPB) main system bus interface. The shared dual port memory is supported to passing information between processing subsystems. It is the easy and efficient way to make a communication channel between processors. Bus interface logic is needed to bridge the gap between the I/O ports on the peripheral and the processor connection. In MB systems, there are three commonly available HW/SW interfaces: direct connection busses (FSL), processor local busses (LMB) and general-purpose system busses (OPB).

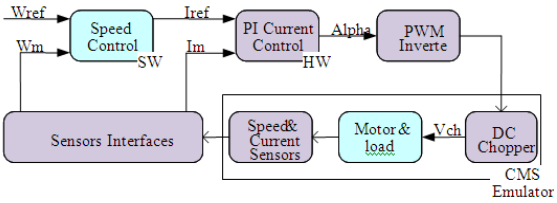

Fig. 3: General diagram of motor control drive

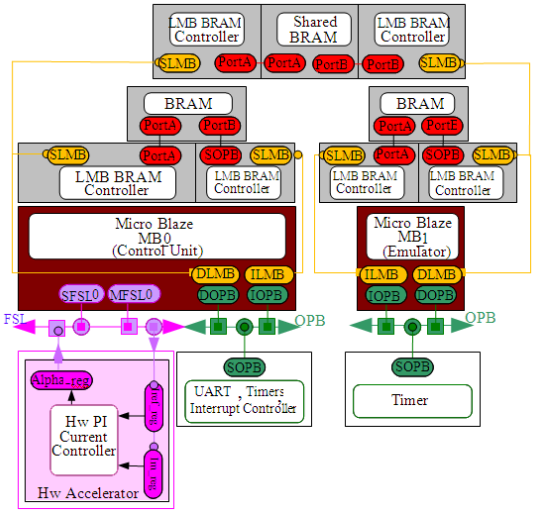

Fig. 4: MPSoC architectural model

The FSL bus is the most used one in order to connect HW accelerators (Salem and Othman, 2010). A HW FSL interface is developed in order to link MB1 to the RMs (controller) through bus macro.

Figure 4 defines our multiprocessor system topology. Each processor connects two independent data memory blocks to its Instruction LMB (ILMB) and Data LMB (DLMB) bus. However, memory controller peripheral is used to interface Block RAM (BRAM) memory to its bus. Each port of the shared BRAM is connected to the respective LMB bus of two different processors and therefore constitutes a communication channel. Processors can then, access dual port memory via a normal memory access.

On-chip Peripheral Bus (OPB) is used to connect larger external memory for the Microblaze. But it presents less performance than implementations using Local Memory Bus (LMB) interface. LMB is designed to allow fast memory access. Thus, Microblaze can be configured to cache instructions or data only over the OPB interface to enhance system performance (Gambier, 2008).

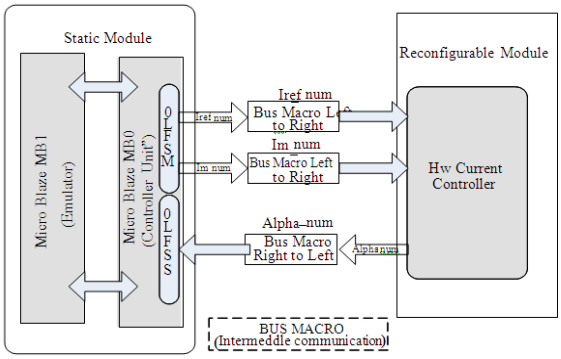

Fig. 5: Communication between modules through Bus Macro

Table 1: HW PI Controller Implementation Results (Virtex2 Pro XC2VP30)

Number of External IOBs 50 (8%)
Number of SLICEs 1193 (8%)
Minimum period (ns) 7.386

For the designed architectures, the OPB is used to connect slow peripherals that are the following:

- RS232 serial channel, connected to an UART peripheral, used for communication between an external user interface and the platform
- Interrupt controller peripheral, used to manage multiple interrupts
- Timer peripheral, used for emulator scheduling.
- A second timer, used for controllers' synchronization

There are 2 types of the considered DC-motor controller P and PI. Each controller has two inputs (reference current Iref and measured current Im) and one output (calculated pulse width alpha). While we're implementing the HW controller as RM, we'll ensure that its communication with any other module is done through bus macro as depicted on Fig. 5.

Managing HW controller:

HW controller FSL interface: The Current HW controller is implemented as a FSL master/ slave peripheral module. When data is written to the FSL, the FSL-S-exist is asserted to indicate that valid data indeed exists within the FSL. Peripheral reads data input, asserts the FSL-S-read signal and updates content of the corresponding register. After reading the Im and Iref data from the master component, the current HW controller launches the computing process. When alpha value is ready, peripheral writes out the content of alpha-reg into the FSL-M-data while asserting the FSL-M-write signal.

HW controller coprocessor design: To achieve powerful processing capabilities in the control unit, the fast loop controller algorithms will be implemented in the FPGA fabric in HW, working in a parallel fashion. For that reason, we have considered a specific HW accelerator approach for the control unit by implementing the current controller in a HW part and controlling it by a SW part. The FSL based solutions were supported because they are the best access alternative to HW accelerated cores on MB. The FSL connections are necessary to integrate the current control unit on MB. In this design, two FSL connections were implemented to provide bi-directional communication to write Iref and Im to the FSL peripheral and to read alpha from this last.

HW controller data path: Because the HW current controller is designed for integer form, the I/O Iref_num, Im_num and alpha_num are the numerical conversion of corresponding variables and are accessible from 16 bits registers by using an adapted scaling factor 2n.

The HW controller is designed using a FSMD architecture model (Gajski, 2000) that combines a control unit with a datapath (Othman *et al*., 2010b). The designed FSMD architecture for this PI Controller is optimized for surface while keeping a minimal latency (Table 1).

System design steps:

Design-flow for reconfigurable controllers: The Hw controller were realized individually using VHDL, then they were tested and verified using Xilinx ISE tools by means of HW-in-the-loop simulations (Schulz *et al*., 2007). The static and the dynamic parts of the system have to be implemented separately (Fig. 4). In a first step, an initial bit stream including the static system components is generated. The partial bit streams that are needed to configure the target FPGA during run-time have to be generated separately.

In practice, the first step is to synthesize a "skeleton" of the top-level design by leaving all components as black boxes. This is done by opening up the ISE project that was created during the export process and removing all of the underlying sources from the project. Next, synthesize the top-level file and copy the synthesis output (the top.ngc file) to imp/top-initial. Then, we can run the modular flow. As Xilinx recommends in his application note Xilinx, 2004, the modular flow is broken into 3 different phases. As

given in (Abdelkrim and Saoud, 2008), we give below a brief description.

Initial budgeting: In this first phase, we make the floor planning of the modules areas and the IOB's. Then, the bus macros must also be LOCed manually. So at the end we obtain a UCF file which will be used for the top-level and all the modules during the active module phase. The physical location of the hard macro channels (used to route signals between modules) is indicated on the UCF file, as an example:

INST "macro_1" LOC = "SLICE_X34Y40";

Active module implementation: We have at this point a synthesized, floorplanned and constrained design. So now it is possible to implement each static or dynamically RM. Each module is treated separately, in a different directory, but in conformance with the global UCF file and the top-level netlist. In this file, the RMs are distinguished from the static one by inserting the lines below in the UCF file.

INST "reconfig_module" AREA_GROUP = "AG_reconfig_module";
AREA_GROUP "AG_reconfig_module" MODE = RECONFIG;

The design is implemented as usual with the Xilinx tools, using NGDBUILD, MAP, PAR and finally BITGEN.

The output is a placed and routed design (in a NCD file) and a partial bitstream (BIT file). For an easier management of each module design, we can publish them using the tool PIMCREATE.

First, we treat the static modules. Then, we use some of the files obtained with the static phase to pursue the implementation of the RMs. In fact, as it is shown in Fig. 6, two particular files are used as inputs for "ngdbuild" instruction: static.used and the UCF file. These files represent respectively the routed nets for the first and the physical constraints and pins assignation for the second.

When finishing the ngdbuild, mapping, place and route steps, the final phase is launched so that we obtain one BIT file for each module.

Final assembly: The final assembly phase is the process of combining each of the individual modules back into a complete FPGA design. The placement and routing achieved during the active implementation phase for each module will be preserved, thereby, maintaining the performance of each module. So we can perform a unique assembled design and then changing the partially RMs with their bit stream. We implement the assembled design using the same flow as in the active module phase.

For every controller and for all possible controller positions an implementation has to be generated, from which the controller configuration is extracted, resulting in a partial configuration (Schulz *et al*., 2007).

Software consideration: The system considered is partitioned on two Micro Blaze (MB) processors, the emulator on MB1 and the controller on MB0. Particularly, the controllers developed are driven by SW part by using low level SW drive of FSL.

The FSL ports on MB are accessed via simple get and put assembly instructions (available in blocking and non-blocking varieties). C macro routines were used for solving the FSL HW controller drivers (Othman *et al*., 2010a). This last one is consistent with the sequence in which data are read in and written out the HW peripheral. Both operands (Im_num and Iref_num) are written to the FSL FPU with non-blocking calls, followed by a blocking read that stalls MB until the HW controller returns a result. Since MB is an in-order execution machine, this penalty cannot be overcome.

As given in Fig. 6, the software (SW) code is merged in the final step. In fact, the local memory of each processor will be loaded with .elf file which contain the executable SW.

Switching schemes: In this work, we try to implement different types of HW current controller and so it is possible, by switching on the fly from one controller to another and by analysing performances criteria, to improve the system performance and responses. When supervisor detects that a new controller structure is needed, he lunches reconfiguration procedure. There are different schemes of reconfiguring a FPGA. In fact, it can be done through fixed schedule scheme or event-driven one. In the fixed schedule scheme, parts of a design or independent designs are loaded on the FPGA sequentially (Schulz *et al*., 2007). The time and order in which the sub-modules are loaded is known during the design phase. In the event-driven scheme the reconfiguration time and order are unknown during the design phase. The realization of a RTR architecture becomes more complex depending on the kind of reconfiguration (full or partial) and the reconfiguration scheme (fixed schedule or event-driven).

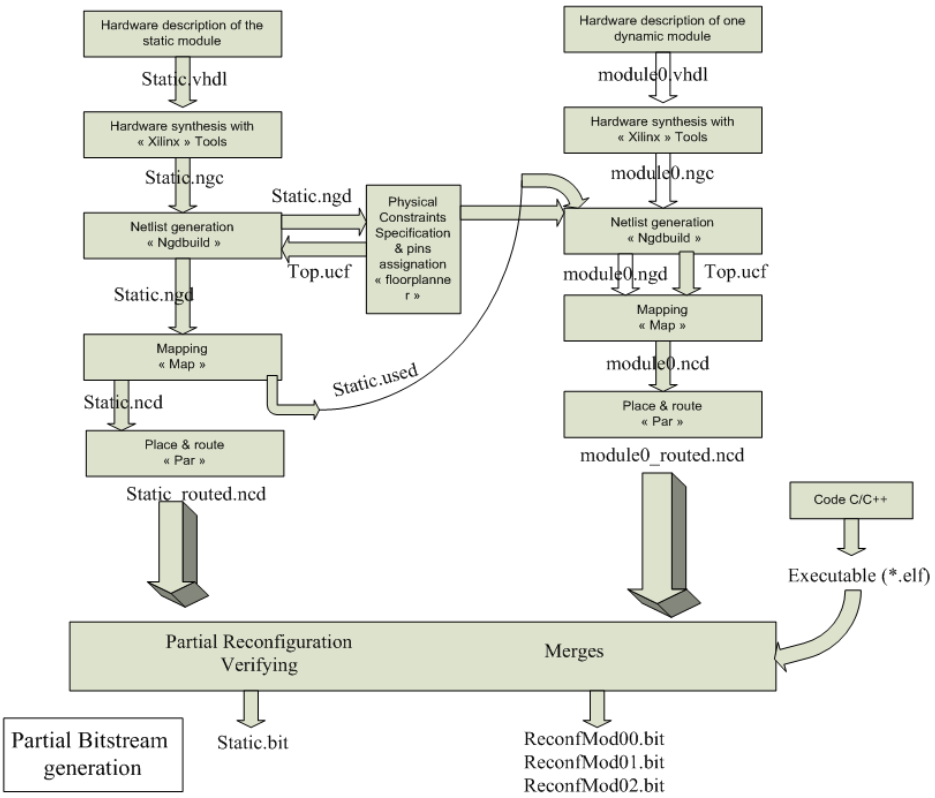

Fig. 6: Conception flow of a partially reconfigurable SoC

The run-time partial reconfiguration in event-driven mode is the most complex, because special architectural concepts and design-flows are needed to realize its implementation (Hagemeyer *et al*., 2007). In our case, a fixed schedule is used.

RESULTS

In order to validate the developed structures, we tested them under different conditions. Table 2 presents some features related to the implementation of the two considered controllers. These implementation results are based on a Xilinx FPGA Virext2pro XC2VP30.

The control structures use a clock frequency of 100MHz. Figure 7 illustrates a comparison between the structures when having steps at the reference value of the speed wref. The curve represents the controlled speed driven by P-Controller at start-up then driven by PI- Controller at stationary state.

In fact, at t1 = 0.75s a reconfiguration order is given and as it shown that for 0s<t< t1 the output is done by P-Controller and from t≥ t1 till the end only PI-Controlled is implemented.

Table 1: Features of realized structures

Structure	Static Part	P controller	PI controller
Slices	6741 (49%)	755 (≈5%)	857 (≈6%)
Reconfiguration Time (µs)	-	701	701

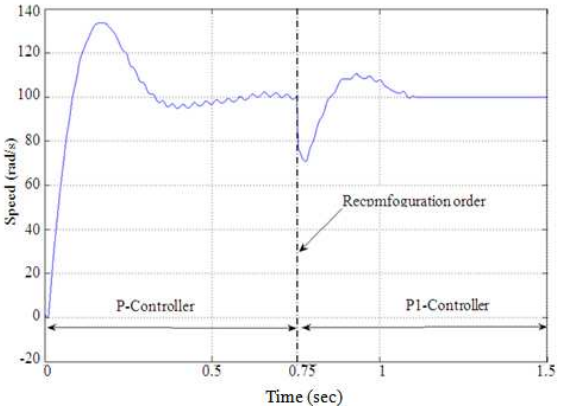

Fig. 7: Speed wave form with Partial bitstream implmentation

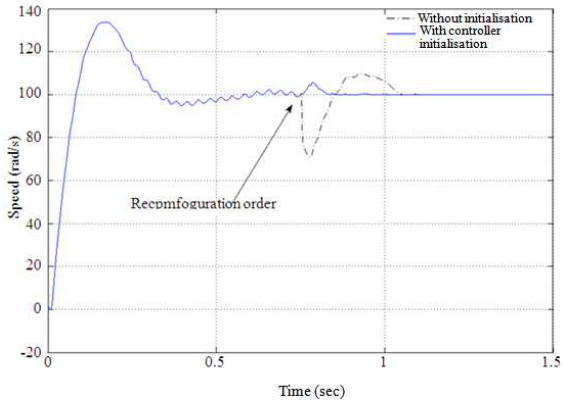

Fig. 8: Comparison of the control schemes with and without internal state initialization

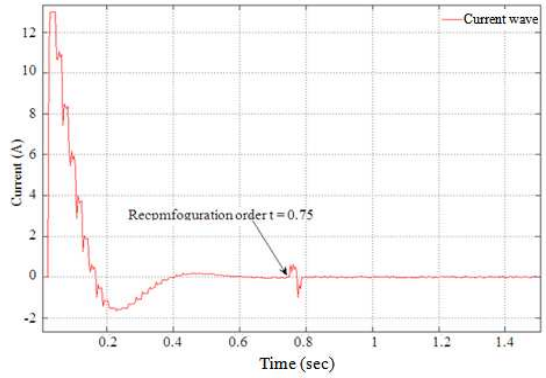

Fig. 9: Current wave form Partial bitstream implementation at 0.75 with internal state initialization

The Fig. 8 shows a comparison between the system response with and without initialisation for RM. We obtain here a more satisfying result.

The Fig. 9 shows the controlled current for the system considered with a partial bitstream implementation at t1 = 0.75s.

DISCUSSION

Using the DPR for implementing two types of DC motor controllers was successful and gave us different results described earlier.

The Fig. 7 illustrates the switching instant between HW P-controller and HW PI-controller. As shown on this figure, the new configuration is implemeted at t=0.75s. While, the P-controller used produces low noise but also a steady state response, the PI-controller gives better steady state response, but requires as it is given in Table 1 more resources for its implementation.

The controlled speed output shows a disturbance at the time of implementing the new partial bitstream. This disturbance could be avoided by initialising the internal state of the controller (eg. by saving it on BRAM).

At reconfiguration time we obtain a disturbance on system response. In order to reduce it and according to (Schulz *et al*., 2007), we proposed to maintain the different comupting variables (error, I, iref) while switching between configuration. Thus, in Fig. 8 and 9, we notice the less disturbance obtained by initialising the different computing variables (error, I, iref) of the controller implemented.

Future trends: Several extensions and applications of the system are in progress. The first of these is to produce a variant of the original system in which a PowerPc, hard microprocessor, will replace Microblaze softprocessor. The new system will be used to control the reconfiguration operation through ICAP bus meaning some switching conditions.

The second is to use AC motor and to develop its own controllers such as Field Oriented Control (FOC) and Direct Torque Control (DTC).

The third foregoing work may be oriented towards implementing control systems on reconfigurable circuit systems which are particularly involved by the MPSoC architectural solutions (Othman and Salem, 2008) and enhanced by Real-Time Operating System (RTOS) (Salem *et al*., 2008).

CONCLUSION

FPGA technology has become an attractive alternative to implement digital control systems, because it offers an interesting trade-off between performance, design effort and cost for various embedded applications fields.

HW solutions offer reliability and better speed performance than SW ones. But they suffer from the lack of flexibility. Thanks to the reconfigurability and particularly the DPR such HW solutions become more flexible. This methodology has given the FPGA the capability to modify its internal structure on the fly, while it is turned on.

In this same context, a design flow of a current controller for DC motor is given. A MPSoC architecture is used to implement on the same chip the motor emulator and the controller. A partial reconfigurable region was defined for the controller.

So, different types of controller (P and PI) can be used and implemented on the fly when the supervisor decides it, depending on the desired performance.

A predictive computation of the controller state has permitted a commutation with reduced bumping phase during the new partial bitstream implementation. The experimental results have been satisfying and have confirmed the adopted methods and approaches.

REFERENCES

Abdelkrim, H. and S.B. Saoud, 2008. Reconfigurable system On Chip (RSOC) concepts and applications. Proceedings of the International Conference on Embedded Systems and Critical Applications, (ISESCA' 08), Gammarth, Tunisia.

Blodget, B., S. McMillan and P. Lysaght, 2003. A lightweight approach for embedded reconfiguration of FPGAs. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, (DATECE' 03), IEEE Xplore Press, pp: 399-400. DOI: 10.1109/DATE.2003.1253642

Brown, S. and J. Rose, 1996. FPGA and CPLD architectures: A tutorial. IEEE Design Test Comput., 13: 42-57. DOI: 10.1109/54.500200

Donlin, A., J. Becker and M. Hubner, 2005. I models and tools for the dynamic reconfiguration of FPGAs. Proceedings of the IEEE International SOC Conference, Sept. 19-23, IEEE Xplore Press, Herndon, VA., pp: 313-316. DOI: 10.1109/SOCC.2005.1554518

Gajski, D.D., 2000. SpecC: Specification Language and Methodology. 1st Edn., Springer, ISBN-10: 0792378229, pp: 313.

Gambier, A., 2008. Digital PID controller design based on parametric optimization. Proceedings of the IEEE International Conference, Control Application, Sept. 3-5, IEEE Xplore Press, San Antonio, pp: 792-797. DOI: 10.1109/CCA.2008.4629671

Goldstein, S.C., H. Schmit, M. Budiu, S. Cadambi and M. Moe *et al*., 2000. Pipe Rench: A reconfigurable architecture and compiler. IEEE Comput., 33: 70-77.

Hagemeyer, J., B. Kettelhoit, M. Koester and M. Porrmann, 2007. Design of homogeneous communication infrastructures for partially reconfigurable FPGAS. Proceedings of the 2007 International Conference on Engineering of Reconfigurable Systems and Algorithms, Jun. 25-28, Las Vegas, USA.

Hauck, S., 1998. The roles of FPGAs in reprogrammable systems. Proc. IEEE, 86: 615-638. DOI: 10.1109/5.663540

Imecs, M., P. Bikfalvi, S. Nedevschi and J. Vasarhelyi, 2000. Implementation of a configurable controller for an AC drive control: A case study. Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, (FPCCM' 00), IEEE Xplore Press, Napa Valley, CA, USA., pp: 323-324. DOI: 10.1109/FPGA.2000.903437

Jerrraya, A.A., 2004. Long term trends for embedded system design. Proceedings of the Euromicro Symposium on Digital System Design, Aug. 31-Sept. 03, IEEE Xplore Press, pp: 20-26. DOI: 10.1109/DSD.2004.1333254

Kastrup, B., A. Bink and J. Hoogerbrugge, 1999. ConCISe: A compiler-driven CPLD-based instruction set accelerator. Proceedings of the 7th annual IEEE Symposium on Field-Programmable Custom Computing Machines, Apr. 21-23, IEEE Xplore Press, Napa Valley, CA, USA., pp: 92-101. DOI: 10.1109/FPGA.1999.803671

Kessal, L., D. Demigny, N. Boudouani and R. Bourguiba, 2000. Reconfigurable hardware for real time image processing. Proceedings of the International Conference on Image Processing, Sept. 10-13, IEEE Xplore Press, Vancouver, BC, Canada, pp: 110-113. DOI: 10.1109/ICIP.2000.899307

Mermoud, G., A. Upegui, P. Carlos-andres and E. Sanchez, 2005. A dynamically-reconfigurable FPGA platform for evolving fuzzy systems. Comput. Intell. Bioinspired Syst., 3512: 296-359. DOI: 10.1007/11494669_70

Othman, S.B. and A.K.B. Salem, 2008. MPSoC design of RT control applications based on FPGA SoftCore processors. Proceedings of the 15th IEEE International Conference on Electronics, Circuits and Systems, Aug. 31-Sept. 3, IEEE Xplore Press, St. Julien's, pp" 404-449. DOI: 10.1109/ICECS.2008.4674876

Othman, S.B., A.K.B. Salem and S.B. Saoud, 2010b. Hw_acceleration for FPGA-based drive controllers. Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE), Jul. 4-7, IEEE Xplore Press, Bari, pp: 196-201. DOI: 10.1109/ISIE.2010.5637591

Othman, S.B., A.K.B. Salem, H. Abdelkrim and S.B. Saoud, 2010a. RT control systems based on FPGA solutions: Softcore processors associated to HW accelerator. Int. J. Discrete Event Control Syst., 1: 171-182.

Paiz, C. and M. Porrmann, 2007. The utilization of reconfigurable hardware to implement digital controllers: A review. Proceedings of the IEEE International Symposium on Industrial Electronics, Jun. 4-7, IEEE Xplore Press, Vigo, Spain, pp: 2380-2385. DOI: 10.1109/ISIE.2007.4374979

Salem, A.K.B. and S.B. Othman, 2010. Field programmable gate array -based system-on-chip for real-time power process control. Am. J. Applied Sci., 7: 127-139.

Salem, A.K.B., S.B. Othman and S.B. Saoud, 2008. Hard and soft-core implementation of embedded control application using RTOS. Proceedings of the IEEE International Symposium on Industrial Electronics, Jun. 30-Jul. 2, IEEE Xplore Press, Cambridge, pp: 1896-1901. DOI: 10.1109/ISIE.2008.4677261

Saoud, S.B. and J.C. Hapiot, 2000. Parallel architectures applied real time emulation [of power apparatus]. Proceedings of the IEEE International Conference on Industrial Electronics, Control and Instrumentation, (IECON' 00), IEEE Xplore Press, Nagoya, Japan, pp: 1719-1724. DOI: 10.1109/IECON.2000.972535

Schulz, B., C. Paiz, J. Hagemeyer, S. Mathapati and M. Porrmann *et al*., 2007. Run-time reconfiguration of FPGA-based drive controllers. Proceedings of the European Conference on Power Electronics and Applications, Sept. 2-5, IEEE Xplore Press, Aalborg, pp: 1-10. DOI: 10.1109/EPE.2007.4417686

Sklyarov, V., 2000. Synthesis of control circuits with dynamically modifiable behavior on the basis of statically reconfigurable FPGAs. Proceedings of the 13th Symposium on Integrated Circuits and Systems Design, (ICSD' 00), IEEE Xplore Press, Manaus, Brazil, pp: 353-358. DOI: 10.1109/SBCCI.2000.876054

Tanougast, C. and S. Weber, 2001. Methodologie de partitionnement applicable aux systemes sur puce à base de FPGA, pour l'implantation en reconfiguration dynamique d'algorithmes flot de donnees. PHD. Nancy, France, Henri Poincare University.

Vuillemin, J.E., P. Bertin, D. Roncin, M. Shand and H. Touati *et al*., 1996. Programmable active memories: Reconfigurable systems come of age. IEEE Trans. Very Large Scale Integ. Syst., 4: 56-69. DOI: 10.1109/92.486081

Wirthlin, M.J. and B.L. Hutchings, 1995. A dynamic instruction set computer. Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Apr. 19-21, IEEE Xplore Press, Napa Valley, CA, USA., pp: 99-107. DOI: 10.1109/FPGA.1995.477415

Zhang, X. and K.W. Ng, 2000. A review of high-level synthesis for dynamically reconfigurable FPGAs. Microprocessors and Microsystems 24: 199-211. DOI: 10.1016/S0141-9331(00)00074-0