Quad-Core MPSoC Architecture for PID-Based Embedded Control Systems

Hassan A. Youness, Mahmoud Khaled, and Mohamed Moness

***Abstract*—Modern embedded control systems require new techniques to fulfill the rapid increase in control requirements and constraints. Multiprocessor systems have been proposed as a promising solution for modern digital control systems. Embedded control systems need to meet hard real-time deadlines, while conducting additional tasks like tuning of control parameters or executing a fault-tolerance algorithm. In this paper, a novel low-cost custom FPGA-based quad-core multiprocessor system-on-chip (MPSoC) architecture prototype is introduced to enhance the performance of legacy digital PID controllers. Due to its inherent parallelism, the PID controller can be mapped directly to the proposed architecture which is built-up using four soft-core microprocessors. The digital PID algorithm is restructured to fit to the introduced system. The main contribution of this paper is a throughput-oriented high-performance low-cost digital PID controller. Results showed remarkable reductions in the execution time of the control loop of the introduced parallel PID controller.**

***Index Terms*—Embedded control systems, multiprocessors, PID control, SoC.**

I. Introduction

Recently, Field-Programmable Gate Arrays (FPGAs) have become an alternative solution for the realization of digital control systems, which were previously dominated by general purpose microprocessor systems [1]. Modern SoC (System-on-Chip) designs show a clear trend towards the integration of multiple processor cores. Furthermore, most of the current embedded applications are migrating from single processor-based systems to multiprocessor systems [2]. Such multiprocessor systems are exploited by inherently parallel algorithms leading to improvements in data throughput at reduced clock speeds. In this paper, a custom quad-core homogenous MPSoC system is introduced to enhance the performance of legacy digital PID controllers. The introduced architecture is based on a small-sized soft-core microcontroller, the PicoBlaze microcontroller. The proposed architecture is benchmarked to review its abilities to execute parallel algorithms. The legacy sequential digital PID control algorithm is restructured to form a new parallel PID algorithm that is executed as parallel as possible on the proposed architecture. The restructured algorithm is directly mapped to the new multiprocessor architecture. The main contribution of this paper is a novel real parallel PID controller. As we will show during the next sections, the controller is designed to for high-performance and it is throughput-oriented.

The rest of this paper is organized as follows. In the next section, we review related work. Section III provides information on the inherent parallelism of the legacy digital PID control algorithm. Section IV describes the hardware architecture of the proposed quad-core MPSoC platform. Section V describes the restructured parallel PID algorithm. Further performance improvement is introduced in Section VI. Section VII shows how the parallel PID algorithm is simulated and tested. Finally, conclusions and future work are presented in Section VIII.

Manuscript received February 21, 2013; revised April 25, 2013.

The authors are with Faculty of Engineering, Minia University, Egypt (e-mail: hassan_youness@eng.miniauniv.edu.eg, eng.mk@mu.edu.eg, m.moness@mu.edu.eg).

II. Related Work

Embedded digital controllers have quantifiable requirements, such as energy consumption, performance (hard real-time computation), and implementation costs. Therefore, the implementation of a controller based on an embedded device differs from a controller based on a general purpose computing platform, where those factors are not implicit.

The use of reconfigurable hardware for digital control applications, not only as prototyping platform but as final target architecture, has been reported since the early 90's. However, it is only until recently that researchers have started to show a greater interest in this technology, because of higher computational demands of digital control systems, and the fast evolution undergone by FPGAs in the last decades [3].

Many control applications are implemented directly within FPGAs rather than using a microcontroller within the FPGA. Performance in an FPGA is more flexible. For example, an algorithm can be implemented sequentially or completely in parallel, depending on the performance requirements. A completely parallel implementation is faster but consumes more FPGA resources. Microcontrollers and FPGAs can successfully implement any digital logic function. However, each has unique advantages in cost, performance, and ease of use. On the other side, microcontrollers are well suited to control applications, especially with widely changing requirements [4].

In this section, we present a discussion of related work on how reconfigurable computing can be used for control applications with the focus on FPGA technology. Monmasson *et al.* [5] reviewed the state of the art of FPGA design methodologies with a focus on industrial control system applications. Nakamura, T. *et al*. [6] proposed a PID-based controller of an electro static levitation system. The controller is implemented within an FPGA using soft-core microprocessor and PID software. Hu Yue-li *et al*. [7] proposed a novel quad-core master-slave architecture for

DOI: 10.7763/IJCTE.2013.V5.822

computer vision systems. The four cores are linked to a shared bus/memory system. Using this system to implement the proposed parallel PID algorithm will face the problem of larger area cost and communication overhead. Hence, we use a simpler direct communication between the four cores. We will also use small soft-cores to minimize the cost and area. Ben Othman *et al*. [8] proposed new dual-core architecture for control applications. They aimed at designing a full speed real-time motor control drive algorithms for FPGA based MPSoC. However they split different control loops and tasks over different cores, we propose a generic parallel PID controller where the algorithm is restructured to run in a parallel form. [1], [9]-[11] proposed different FPGA-based digital PID controllers. They are implemented directly on FPGAs using LUTs (Look-Up-Tables). Such design would be perfect for standard control applications without any requirements changes. Joao Lima *et al*. [12] proposed mixed FPGA and microcontroller design where the PID is implemented directly within the FPGA logic and a microcontroller is used for parameter tuning.

Rather than using a single microcontroller, we propose a quad-core system running a parallel PID algorithm to enhance the performance and to keep the benefits of using a microcontroller within an FPGA.

III. The Legacy PID Algorithm

The proportional, integral, derivative, or more popularly, the PID, is probably one of the most popular controllers in use today [13]. Equation (1) describes the basic operation of the digital PID controller:

$$U(k) = K_p E(k) + K_i I(k) + K_d D(k) \quad (1)$$

The system is represented in a sampled-data form. $E(K)$,the input to the controller and the K^{th} sample of the error signal. The output of the controller is called control command, $U(K)$. $I(K)$ represents the integral the K^{th} error sample while $D(K)$ represents the derivative of the same error sample. K_p, K_i and K_d are the proportional, integral and derivative controller parameters. Fig.1 describes the structure of the digital PID controller.

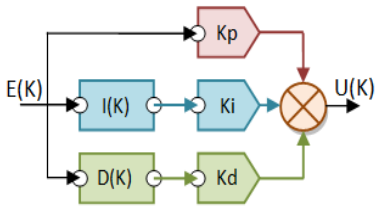

Fig. 1. Structure of the digital PID controller.

The three branches of the system are distributed calculations. The first branch is computed by simply multiplying the error sample with the *Kp* parameter. The second and third branches take more time to execute as they need to compute the integral and derivative of the error sample. The digital PID control algorithm is usually implemented with a sequential algorithm. Fig.2 describes the operations of the sequential digital PID control algorithm. The sequential PID controller is inherently a parallel algorithm but it is executed sequentially. The three major tasks within the controller are the calculation of *P*, *I* and *D* parts. These tasks are independent and ready to be executed simultaneously.

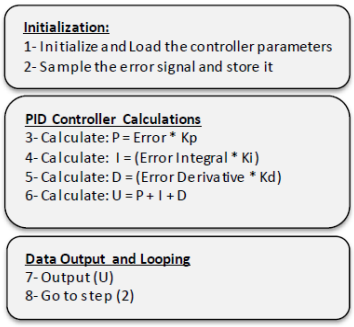

Fig. 2. Operations within the sequential PID algorithm.

IV. The Proposed MPSoC Architecture

Before restructuring the sequential digital PID control algorithm, a customized MPSoC architecture is proposed to handle the new parallel algorithm. Fig. 3 shows the top-design of the proposed MPSoC architecture. It consists of two main components: The Enhanced Picoblaze Microcontroller (EPM) and The Quad-Port Memories.

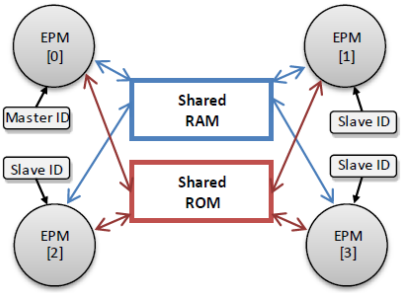

Fig. 3. Top level design of inner MPSoC system.

EPMs are the main building blocks of the system and will be described in the next subsection. The system contains two types of QP (Quad-Port) memories; a QP-RAM for data exchange between the four cores and a QP-ROM for the shared program memory. Each core is assigned a unique 2-bit HWID (Hardware Identifier) which is used to execute the appropriate task. Master-core manages the system work-flow and synchronize slave-cores' tasks and may execute additional task like parameter tuning. Three slave-cores calculate the three parts of the PID algorithm concurrently and save their results in the shared QP-RAM. Master-core should collect calculations results and computes the controller output.

A. The Enhanced PicoBlaze Microcontroller

Digital controllers may be implemented on dedicated microcontrollers. Programming control sequences in software is often easier than creating similar structures in hardware, but microcontrollers are typically limited by performance. Each instruction executes sequentially. As an application increases in complexity, the number of instructions required to implement the application grows and system performance decreases accordingly [4]. By contrast, performance in an FPGA is more flexible. For example, an algorithm can be

implemented sequentially or completely in parallel, depending on the performance requirements. A completely parallel implementation is faster but consumes more FPGA resources. A microcontroller embedded within an FPGA provides the best of both worlds. The microcontroller implements non-timing crucial complex control functions while timing critical or data path functions are best implemented using FPGA logic.

Fig. 4 shows the main building component of the system, the EPM. It is shown in its simplest form, a single input port and a single output port while the used version incorporates many input/output ports. Xilinx PicoBlaze, a soft-core microcontroller [4], has been enhanced by adding input multiplexing logic for input port extension, output decoding logic for output port extension and fast four-stage pipelined FPU (Floating-Point Unit) [14] to enable floating-point operations. The selection of such small microcontroller is due to the fact that it was designed for efficiency, low deployment cost and power conservation. Even with such resource efficiency, it performs a respectable 44 to 100 million instructions per second (MIPS) depending on the target FPGA family and speed grade. The connected FPU is modified to only perform the following basic floating-point operations: Addition, Subtraction and Multiplication.

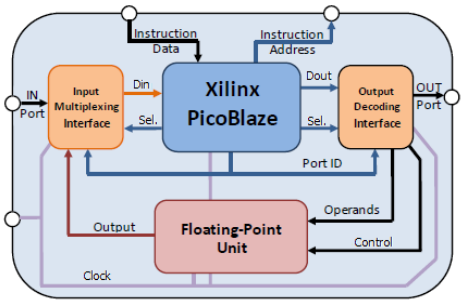

Fig. 4. The enhanced PicoBlaze.

B. Quad-Port Memories

Based on Xilinx application note on how to create quad-port memories using existing dual-port memories [15], we propose a quad-port RAM and a quad-port program ROM. The quad-port RAM is used for data exchange and for communications within the system using predefined memory map that is known to all cores. Instead of using a separate ROM for each core of the system, a single shared quad-port ROM is used by all cores. The application embedded within the ROM is designed to be executed by all cores, each with a different behavior, using the predefined HWIDs.

C. Quad-Port Memories

All pieces are connected together to form this custom MPSoC Parallel Digital PID (PDPID) controller. Fig. 5 shows the complete PDPID controller architecture. The developed MPSoC is used within the PDPID controller.

The input, $E(k)$, is distributed to all cores. This allows each core to read the error sample and accomplish its defined task. As the system manager, the master-core allows new samples of error-signal to pass, collects data stored in shared RAM and calculates the control signal $U(K)$. The system was designed to only allow the master-core to handle output of the PDPID controller. Although this design limits future fault-tolerant benefits, design requirements were to enhance the control performance and to save hardware space and power. Table I describes FPGA resources that are used for different stages of system development.

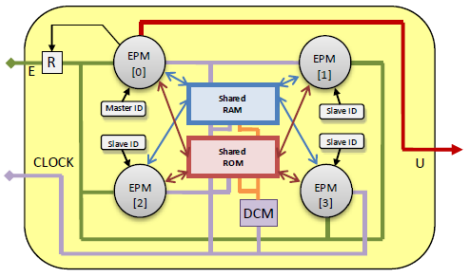

Fig. 5. Top level design of the PDPID controller.

TABLE I: FPGA RESOURCE USAGE

Resources	PicoBlaze	EPM	PDPID
Slices	96	1630	6511
Slice FFs (Flip Flops)	76	603	2572
4-Input LUTs	181	3134	12465
BRAMs	0	1	2

In order to gain a clear picture of benefits of duplicating microcontroller cores within the system, the system is benchmarked. Benchmarking of systems that employ small microcontrollers is limited by their resources. Therefore, no heavy benchmarking applications could be executed on the system. Thanks to Křivka's work to develop a C-Compiler for the PicoBlaze [16], we created several applications to benchmark the system. The compiler is based on SDCC (Small Device C-Compiler) and it is called PBCC (PicoBlaze C-Compiler). Enhancements are made to the PBCC to extend its functionality to generate parallel applications that can run efficiently on the proposed MPSoC architecture. Applications are written in C language and compiled twice to run on both the single-core PicoBlaze microcontroller and the MPSoC architecture. Fig. 6 describes a complete development suite used to generate applications to run on the proposed architecture.

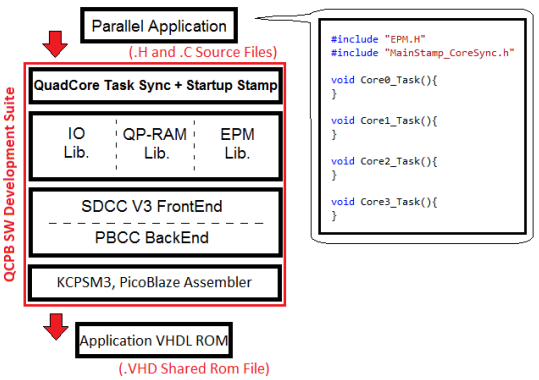

Fig. 6. Software development flow for the proposed architecture.

The development suite consists of: 1) Xilinx PicoBlaze Assembler (KCPSM3), 2) Zbyněk Křivka's PBCC, 3) Libraries to handle the EPM ports, QP-RAM access and IO operations and finally 4) Library to manage Task Synchronization between the four cores. Software developer will not schedule tasks by hand. The header file

"MainStamp_coreSync.h" will do the job. The developer will only have to place the code each task within each function of the four functions: Core0_Task() to Core3_Task(). Table II and Table III show the results obtained from running different benchmarking applications on both the single-core PicoBlaze (EPM) system and the quad-core PicoBlaze (QCPB) system.

The first table describes the tests applied where the second table shows the results of each one. For each test, the number of arithmetic and floating-point operations (A/F Opr.) is shown. Then, for each test, the execution time in terms of clock cycles (Cyc.) and in terms of number of instructions (Ins.) is shown. The last two columns emphasize the reduction ratio (RR) when QCPB is compared with the EPM and the ROM usage (RU) in percentage.

TABLE II: APPLIED BENCHMARKING TESTS

Category	ID	Test
ALU	1	12-elements 8-Bit integer Array Add.
	2	12-elements 8-Bit integer Array Mul.
	3	12-elements 8-Bit integer Array Div.
	4	12-elements 16-Bit integer Array Add.
	5	12-elements 16-Bit integer Array Mul.
	6	12-elements 16-Bit integer Array Div.
RAM	7	16-Byte RAM data transfer
	8	32-Byte RAM data transfer
	9	64-Byte RAM data transfer
FPU	10	32-Bit (floating point) 2x2 Matrix element-by-element Add./Sub./Mul./Div.
	11	Two 4D 32-Bit float vector product

TABLE III: RESULTS OF APPLIED BENCHMARKING TESTS

ID	A/F Opr.	EPM		QCPB		RR [%]	RU
		Ins.	*Cyc.*	*Ins.*	*Cyc.*		
1	12/0	203	406	55	110	73.0%	17%
2	48/0	980	1690	247	494	74.4%	23%
3	72/0	1156	2312	295	590	74.4%	23%
4	24/0	281	562	71	124	74.3%	28%
5	84/0	1932	3864	493	986	74.4%	34%
6	144/0	3025	6050	760	1520	74.8%	39%
7	0/0	1612	3224	398	796	75.3%	25%
8	0/0	3070	6140	762	1524	75.1%	25%
9	0/0	5982	11964	1490	2980	75.0%	25%
10	0/4	1244	2488	311	622	75.1%	87%
11	0/7	2177	8708	1249	2498	71.3%	97%

Four conclusions are extracted from the previous results: 1) the system can greatly enhance the performance of parallel-ready algorithms, 2) FPU operations consume ROM heavily due to the code overhead in C language, 3) as PID algorithm will exhibit more floating point computations, that cannot be implemented using this C-Compiler due to ROM limitations and 4) the proposed PID algorithm should be implemented in assembly language to reduce the code-size and execution time.

V. THE PARALLEL PID ALGORITHM

For software developers, the new hardware development toward multicore architectures is a challenge, since existing software must be restructured toward parallel execution to take advantage of the additional computing resources. In particular, software developers can no longer expect that the increase of computing power can automatically be used by their software products. Hence, additional effort is required at the software level to take advantage of the increased computing power [17].

A new parallel application, settling in the shared QP-ROM is designed to perform all control tasks in parallel. Executed by both master-core and slave-cores, the shared parallel application needs a mechanism for task allocation and synchronization. Fig.7 (a) describes the parallel tasks within the proposed algorithm. Task allocation is accomplished using the predefined hardware identifiers (HWIDs). Task synchronization is performed using hardware signals between different cores.

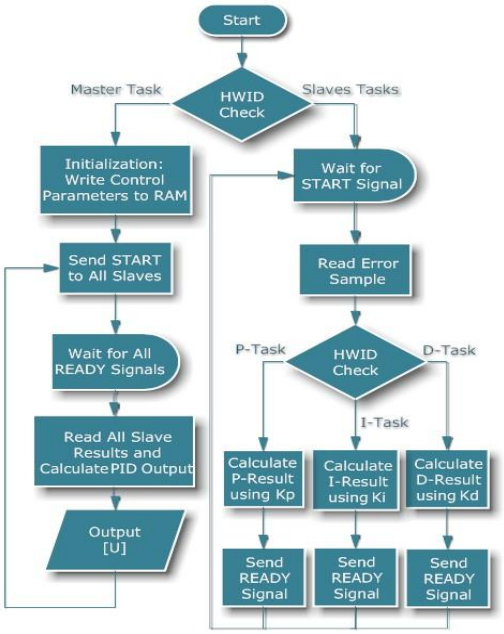

Fig. 7 (a). The shared parallel application.

The application uses the HWIDs to distinguish between different cores as a mean of task-allocation. The HWID is used to coordinate tasks at boot-up. Synchronization between master-task and slave-tasks is performed with (READY) and (START) signals. A slave-task must wait for a (START) signal to read a new error-sample and compute its result. Master-task will wait for all (READY) signals from all slave-tasks. Such behavior will produce an application that is only parallel in the phase of slave-tasks. The master-task is not in parallel with the slave-tasks. Fig. 7 (b) describes this behavior. The numbering of each task indicates the sample number. Flags indicate delivery of control outputs.

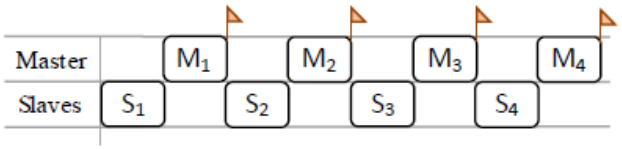

Fig. 7 (b). Master is not in parallel with slaves.

The application is implemented in assembly language for best optimization in both execution time and code-size. Xilinx PicoBlaze assembler is used to generate VHDL file of shared ROM from the assembly code. The shared ROM file is included within the PDPID MPSoC, simulated and debugged

as will be described in the next sections.

VI. The Pipelined Parallel PID Controller

As described in Fig. 7 (a), master-task is not in parallel with slave-tasks. Modification within the parallel application has been made to increase the throughput using software-pipelining. Task-loads have been measured and redistributed to make all tasks within a close execution time. Within master-task, if (START) signals were sent immediately after receiving all (READY) signals, this will make the slave-tasks continue with the next error-sample while the master-task continues with calculations of previous output. When the master-task finishes its output computations of sample (*K*), and because of the close execution time, the slave-tasks will have been finished computing their results of sample (*K*+1). They will meet again to synchronize when the master-task is waiting for all (READY) signals. Such behavior results in software-pipelined parallelism between master-task and slave-tasks. Fig. 8 describes this new behavior.

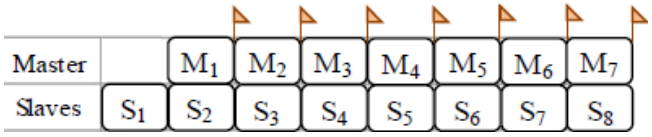

Fig. 8. Effect of pipelined approach.

VII. Simulation and Testing

The system is proposed as a prototype that may be implemented using any preferred technology. The proposed PDPID system is modeled using two different languages: VHDL and SystemC. VHDL is used to describe main parts of the system like EPM, QP-RAM and QP-ROM while SystemC is used to connect parts with SystemC HW channels. SystemC is also used to drive the simulation process and to generate clocks. MODELSIM is used to perform the simulation as it has a powerful mixed-language simulation kernel.

A. Debugging the Parallel Algorithm

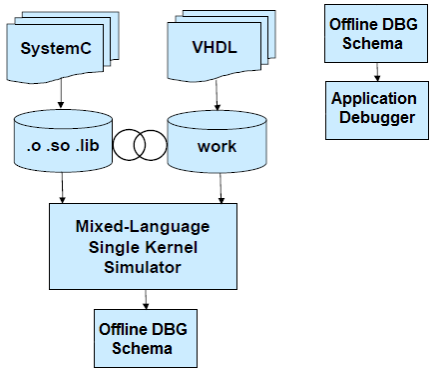

Fig. 9. Offline debugging technique.

The parallel application contains four simultaneously running assembly instruction threads. Regular debugging techniques such as cycle based HW debugging using Modelsim and instruction based simulation using PicoBlaze ISS (Instruction Set Simulator) are painful. The first is clearly impossible and the latter is only able to simulate and debug a single PicoBlaze microcontroller. Here we introduce a new technique for debugging such systems; it is an offline debugging technique. Fig. 9 describes the technique.

The process consists of two phases: generation of offline DBG (Debug) schema and application debugging phase. The first phase is to generate a schema-file that contains the required information needed for debugging each core of the system. This file is generated from MODELSIM using a TCL (Tool Control Language) script that captures instructions, register values and ports during the execution of a single loop of the parallel PID algorithm and writes them to the schema file. The second phase is responsible of using this file to debug the system in an offline manner. The second phase is a high level GUI (Graphical User Interface) application that parses the schema file and enables offline debugging of the application. Fig. 10 is a screenshot of the quad-core offline debugger.

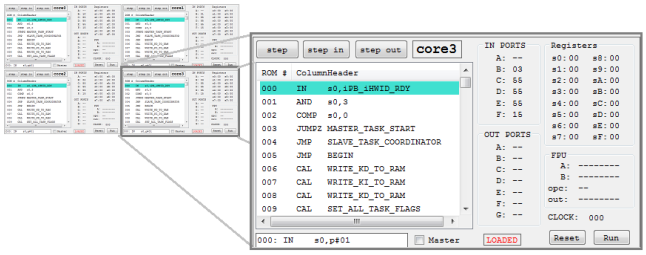

Fig. 10. Offline debugging application.

The main benefits of this technique are: 1) delayed disassembling of instruction codes, 2) forward or backward instruction stepping in zero-time and 3) multi-core synchronous execution and stepping.

B. Simulation Results

TABLE IV: Simulation Results

	EPM PID Controller [Sequential]	PDPID [Parallel]	Pipelined PDPID [Parallel]
PID Loop (Cycle.)	T_s = 698	T_m = 498	T_m = 316
		RR = -28%	RR = -55%
ROM Usage	68%	73%	74%
$S_{\%}(4)$	--	35%	55%

Table IV shows the simulation results for different implementations during the development process of the proposed parallel PID controller. For each implementation, the PID loop time is shown in the term of the number of clock cycles needed to execute a single loop. Moreover, the execution Reduction Ratio (RR) is computed against the sequential case. The calculation of PID-loop speed-up is based on the "Equal Duration Model" [18]. The speedup factor of a parallel system can be defined as the ratio between the time taken by a single processor to solve a given problem instance to the time taken by a parallel system consisting of (n) processors to solve the same problem instance. Equation.2 describes how it is computed:

$$S(n) = \frac{T_s}{T_m} \quad (2)$$

where, *S(n)* denotes the speed-up calculated for *n* processors, T_s denotes the time taken by the single processor and T_m denotes the time taken by the parallel system. In order to scale the speedup factor to a value between 0% and 100%, we divide it by the number of processors, *n*=4.

It may look odd that the results obtained from benchmarking the system looks different from the results obtained from applying the parallel PID on the system. However most of the benchmarking showed a reduction of about 73% to 75%, the reduction obtained from the parallel PID algorithm is almost about 54%. The benchmarking tested the system ability to enhance parallel-ready operations like ALU, FPU and RAM operations while the parallel PID algorithm contains more than just parallel-ready operations. Task synchronization and communications between the four cores should be taken into consideration. Although the system managed to execute the algorithm in a parallel form, it also increased the execution time and code size as it added many task synchronization code.

VIII. Conclusions and Future Work

In this paper, a custom FPGA-based MPSoC architecture is designed to meet the requirements of proposed parallel digital PID scheme. The main contribution of the paper is a novel throughput-oriented high-performance parallel digital PID controller. The proposed architecture is benchmarked to review the benefits of duplicating cores within the system. The sequential digital PID algorithm was restructured, because of its inherent parallelism, to run concurrently inside the four cores of the. The new parallel algorithm is directly mapped to the MPSoC architecture using the means of predefined hardware identifiers. Modifications to the algorithm have been made to propose a two-stage software pipelined approach of the system. Results obtained from simulating and debugging of the system showed great enhancement in the performance of the digital PID controller. The software pipelining technique applied enhanced the throughput of the system .The system represents a prototype of a real parallel PID controller and it can be modeled and implemented using many different system design fields. Embedded systems are very power constrained and this prototype faces challenges when power and space conservations are considered. We hope to extend the work to include a study of the power consumption and tuning for the system.

References

[1] Y. F. Chan *et al.*, "Design and implementation of modular FPGA-based PID controllers," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1898-1906, Aug. 2007.

[2] K. Popovici, F. Rousseau, A. A. Jerraya, and M. Wolf, *Embedded Software Design and Programming of Multiprocessor System-on-chip*, Springer, 2010.

[3] C. V. P. Gatica, "Dynamically reconfigurable hardware for embedded control systems," *Cognitive Interaction Technology*, Exzellenzcluster, Dissertation, Dr.Eng., 2012.

[4] Xilinx, *PicoBlaze 8-bit embedded microcontroller user guide*, vol. 2, 2010.

[5] E. Monmasson and Y. Chapuis, "Contributions of FPGAs to the control of electrical systems, a review," *IEEE Industrial Electronics Society Newsletter*, December 2002, vol. 49, no. 4, pp. 8-15.

[6] T. Nakamura, Y. Awa, H. Shimoji, and H. Karasawa, "Control system of electrostatic levitation furnace," *Acta Astronautica*, 2002, vol. 50. pp. 609-614.

[7] Y.-L. Hu and D. Qian, "Design of an architecture for multiprocessor System-on-Chip (MPSoC)," in *Proc. Conference on High Density Microsystem Design and Packaging and Component Failure Analysis*, 2006, pp. 63-66.

[8] S. B. Othman, A. K. B. Salem, and S. B. Saoud, "MPSoC design of RT control applications based on FPGA softcore processors," in *Proc. ICECS 2008*, pp. 404-409, 2008.

[9] L. Samet, N. Masmoudi, M. W. Kharrat, and L. Kamoun, "A digital PID controller for real time and multi loop control: a comparative study," in *Proc. IEEE International Conference on Electronics, Circuits and Systems*, 1998, vol. 1, pp. 291-296.

[10] M. S. M. Siddiqui, A. H. Sajid, and D. G. Chougule, "FPGA based efficient implementation of PID control algorithm," in *Proc. International Conference on Control, Automation, Communication and Energy Conservation*, June 2009, pp. 4-6.

[11] R. A. Flores, F. R. Gutierrez, and C. Jeanniton, "Qualitative evaluation of a PID controller for autonomous mobile robot navigation implemented in an FPGA card," in *Proc. Seventh International Conference on Natural Computation (ICNC)*, July 2011, vol. 3, pp. 1753-1757.

[12] J. Lima, R. Menotti, J. M. P. Cardoso, and E. Marques, "A methodology to design FPGA-based PID controllers," in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2006, vol. 3, pp. 2577-2583.

[13] J. Ledin, *Embedded Control Systems in C/C++: An Introduction for Software Developers Using MATLAB*, CMP Books, January 12, 2003.

[14] R. Usselman. (2012). Open floating point unit. *The Free IP Cores Projects.* [Online]. Available: http://www.opencores.org

[15] *Quad-port memories in virtex devices*, Xilinx Application Note, vol. 1, no. 228, 2002.

[16] ZbynekKrivka, "PBCC: PicoBlaze C Compiler," *Rev.*, vol. 2, 2010.

[17] T. Rauber and G. Runger, *Parallel Programming For Multicore and Cluster Systems*, Springer, 2010.

[18] H. E. Rewini and M. A. E. Barr, *Advanced Computer Architecture and Parallel Processing*, John Wiley & Sons, Inc., 2005.

Hassan A. Youness received his B.Sc. and M.Sc. degrees from Assiut University, Assiut, Egypt, and Ph.D. from Graduate School of Information Science and Technology, Osaka University with the cooperation of Ain Shams University, Egypt. He worked for IBM Company and Mentor Graphics in Egypt. He is currently an assistant professor at Minia University, Computers and Systems Eng. Department. His research interests include Integrated System Design, Fault Tolerance, HW/SW Co-design, Parallel Computers, and MPSoCs.

Mahmoud Khaled was born in Minia, Egypt, on June 22, 1987. He received his B.Sc. from Faculty of Engineering, Department of Computer and Systems, Minia University, Egypt, in 2009.

He is a teaching assistant and M.Sc. student in Faculty of Engineering, Department of Computer and Systems, Minia University, Egypt. His research interests are in the area of Embedded Control Systems and MPSoCs.

Mohamed Moness received his B.Sc., M.Sc. and Ph.D. degrees from Assiut University, Assiut, Egypt. He worked as a head of Computer and Systems Engineering Department, from 1999-2010, and worked as a dean for the faculty of engineering for two times respectively. He is currently a professor at Minia University, Computers and Systems Eng. Department. His research area is Automatic Control Systems.