Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

8th International Conference
on Electrical Engineering
ICEENG 2012

Direct mapping of digital PID control algorithm to a custom FPGA-based MPSoC, the parallel digital PID (PDPID) controller

By

Mohamed Moanes * Hassan A. Youness* Mahmoud Khaled*

Abstract:

New applications of digital embedded control systems require more advanced techniques that can fulfill increasing control requirements and to meet control constraints, such as reaching RT deadlines, while trying to achieve additional tasks like auto-tuning of the parameters of control algorithm, conducting diagnostic-based operations or executing a fault-tolerance algorithm. Hence, Multiprocessors System on Chip (MPSoC) has been proposed as a promising solution. The main purpose of this paper is to put a step towards enhancing the legacy digital PID control algorithm by exploiting its inherent parallelism. We propose a direct-mapping design of the sequential digital PID to a custom Quad-Core Master-Slave MPSoC design, built-up using an enhanced FPGA Soft-Core microcontroller.

Keywords:

MPSoC, Embedded Control Systems and FPGA

* Faculty of Engineering, Minia University, Egypt

1. 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 system-on-chip (SoC) designs show a clear trend toward integration of multiple processor cores and most of the current embedded applications are migrating from single processor-based systems to multiprocessor systems [2]. Such Multiprocessors SoC (MPSoC) systems are exploited by inherently parallel algorithms leading to improvements in data throughput at reduced clock speeds. In this paper, a direct mapping design is introduced to map the legacy sequential digital PID control algorithm to a custom homogenous Master-Slave Quad-Core Multiprocessor SoC (MPSoC) architecture. Then, a restructured algorithm is proposed from the digital PID control algorithm to be executed as parallel as possible on the proposed architecture. This paper is organized as follows. [Section 2] shows the inherent parallelism in the digital PID control algorithm. [Section 3] describes the hardware architecture of the proposed MPSoC platform.[Section 4] describes the parallel application designed to exploit the parallelism of the sequential algorithm. Further performance improvement is shown in [section 5] using the pipelining approach. Finally, conclusions and future work are presented in [section 6].

2. IT IS INHERENTLY PARALLEL ALGORITHM:

The proportional, integral, derivative, or more popularly, the PID, is probably one of the most popular controllers in use today [3]. The following simplified equation describes the basic operation of the digital PID controller:

$$U(K) = K_p E(K) + K_i I(K) + K_d D(K)$$

The controller input is the measured error signal of the controlled process, E(K). The output of the controller is the control command, U(K). I(K) and D(K) represent integral and derivative of the Kth error sample respectively. Kp, Ki and Kd represents the proportional, integral and derivative controller parameters respectively. Fig.1 describes the structure of the digital PID controller:

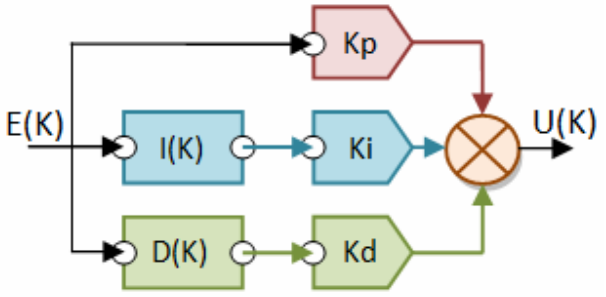

Figure (1): *Structure of the digital PID controller.*

In order to be implemented within a microprocessor or a microcontroller and to be executed iteratively, the system is represented in a sampled-data form. The main three branches of the system consume the most of the calculations of the algorithm. The first branch is computed by simply multiplying the error sample with the Kp parameter. The second and third branches takes more time to execute as they need to compute the integral and derivative of the error sample numerically with an accepted accuracy.
The digital PID control algorithm is usually implemented with a sequential algorithm. Fig.2 describes the operations of the sequential digital PID control algorithm:

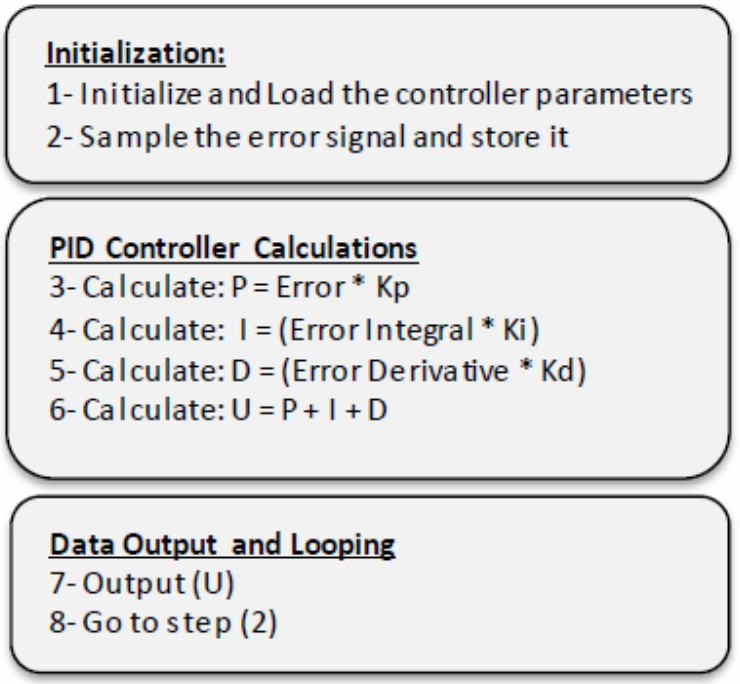

Figure (2): *Operations within the sequential PID algorithm.*

It is clear that the sequential PID controller is inherently parallel algorithm executed sequentially. The main three tasks within the controller are calculations; the P, I and D. These tasks are independent and ready to be parallelized.

3. MPSOC ARCHITECTURE:

Before restructuring the sequential digital PID control algorithm, a customized MPSoC architecture is proposed to manage the parallel execution of new algorithm. Fig.3 shows the top design of the MPSoC system architecture. It consists of two main components (1) The EPM (Enhanced Picoblaze Microcontroller) and (2) The Quad-Port Memories.

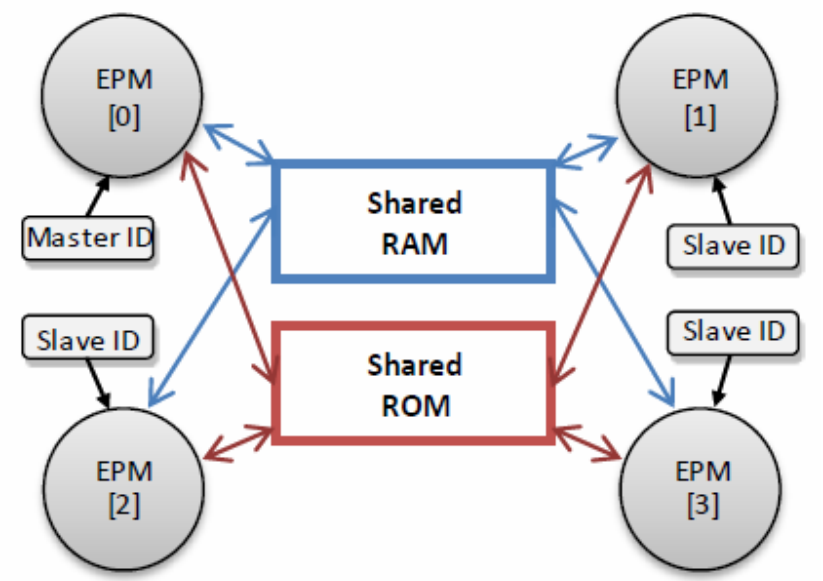

Figure (3): *Top Level Design of inner MPSoC System.*

The EPM is the main building block of the system and will be described in the next subsection. The system contains two types of Quad-Port memories; a RAM for data exchange between the four cores and a ROM for the shared program memory. Each core is assigned a unique HWID (Hardware Identifier) which is used to assign the appropriate task to each core. A Master-Core is required to manage the system work-flow and to synchronize Slave-Cores' tasks. Three Slave-Cores are required to calculate the three parts of the PID algorithm concurrently and to save their results in the shared RAM. The Master-Core should collect resulting calculations and compute the controller output.

A. The EnhancedPicoblaze Core

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 [3]. 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 the 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 Enhanced-PicoBlaze Microcontroller (EPM). The EPM 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 [5], has been enhanced by adding (1) input multiplexing logic for input port extension, (2) output decoding logic for output port extension and (3) a fast four stage pipelined floating-point unit [6] to enable floating-point operations within this microcontroller. The selection of such small microcontroller is duo to the fact that it was designed for efficiency and low deployment cost as it occupies just 96 FPGA slices. 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: (1) Addition, (2) Subtraction and (3) Multiplication.

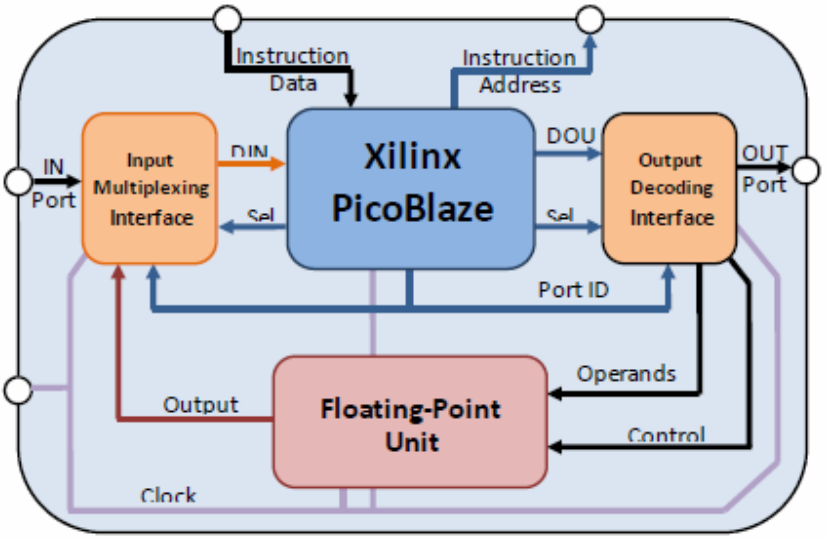

Figure (4): *The Enhanced PicoBlaze Architecture.*

B. Quad-Port Memories

Based on the Xilinx Application-Note #228 on how to create quad-port memories using existing dual-port memories [7], we propose a Quad-Port RAM and a Quad-Port program ROM. The quad-port RAM is used for data exchange and to enable communications within the system with a predefined memory map 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 ROM is designed to be executed by all cores, each with a different behavior, using the predefined hardware identifier (HWID). As stated in Xilinx note, each Quad-Port memory is organized using two Dual-Port memories. Within each Dual-Port memory, the user can Read or Write to and from each port independently (with the exception of simultaneous Read and Write

to the same address). Such limitation does not affect the proposed design, as the shared RAM is organized to enable communication between a maximum of two cores.

C. The Complete Picture

Now, all pieces are connected together to form the customized MPSoC based parallel digital PID (PDPID) controller. Fig.5 describes the complete parallel digital PID (PDPID) controller system architecture. The developed Quad-Core Master-Slave MPSoC is used within the PDPID controller.

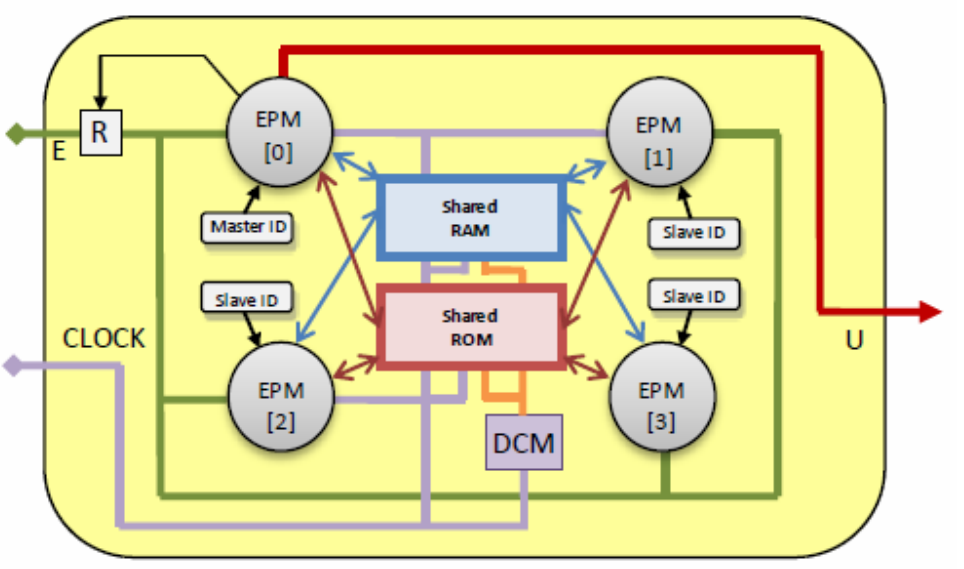

Figure (5): *Top Level Design of The PDPID Controller.*

The input, error signal, is distributed to all cores, allowing each core to read the error sample and accomplish its defined task. As the system manager, the Master-Core can (1) Allow new samples of Error-Signal to be pass, (2) Collect data stored in shared RAM by all Slave-Cores and (3) Calculate and output the control signal (U). 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 (1) enhance the control performance and to (2) save hardware space.

Table.1 describes the used FPGA hardware resources during different stages of system development:

Table (1): *FPGA Resource usage during different stages.*

	Xilinx PicoBlaze	EPM + ROM	PDPID
Slices	92	1630	6511
Slice FFs	76	603	2572
4 input LUTs	181	3134	12465
BRAMs	0	1	2

4. THE SHARED PARALLEL APPLICATION:

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. Instead, additional effort is required at the software level to take advantage of the increased computing power [8].

A new parallel application, settled in a single shared ROM, is designed to perform all control tasks in parallel. Executed by Master-Core and Slave-Cores, the shared parallel application needs a mechanism for task allocation and synchronization. Fig. 6 describes the operations within the shared parallel application algorithm. Using the predefined hardware identifiers (HWIDs), task allocation is accomplished while task synchronization is performed using signals passed between different cores.

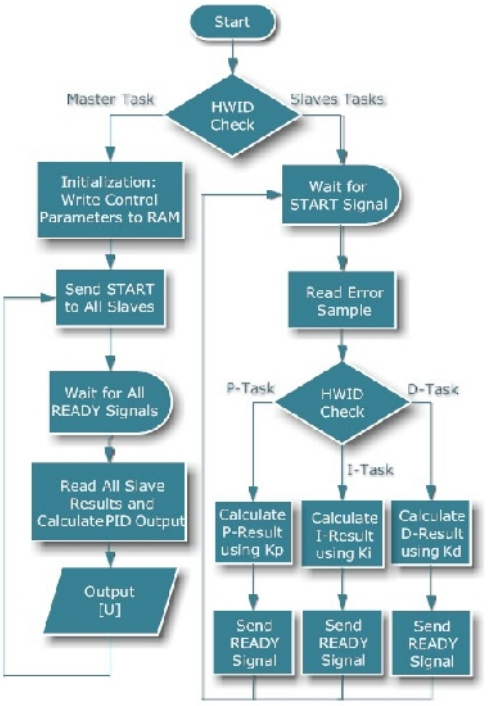

Figure (6): *The Shared Parallel Application..*

The application uses the HWIDs to distinguish between different cores as a mean of task-allocation. First, The HWID is used to coordinate tasks at boot-up. HWIDs are used within Slave-Tasks to identify each Slave-Core's task.
The synchronization between Master-Task and Slave-Tasks is performed with (READY) and (START) signals. The Slave-Task must wait for a (START) signal to read a new Error-Sample and compute its result. The Master-Task will wait for all (READY) signals from all Slave-Tasks. Such behavior will produce an application that is parallel in the phase of Slave-Tasks. The Master-Task is not in parallel with the Slave-Tasks. Fig.7 describes this behavior. The numbering of each task indicates the current Error-Sample. Flags indicate delivery of control outputs.

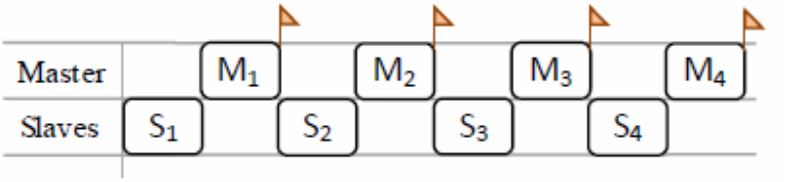

Figure (7): *Master-Task is not in parallel with Slave-Tasks.*

5. THE PIPELINED APPROACH:

Modification within the parallel application has been made to increase the throughput using pipelining. The Task-Loads have been distributed to make all tasks within a close execution time as possible. Within the Master-Task, if (START) signals were sent immediately after receiving all (READY) signals, this will make the slave signals 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 pipelined-based parallelism between Master-Core and Slave-Cores. Fig.8 describes such behavior.

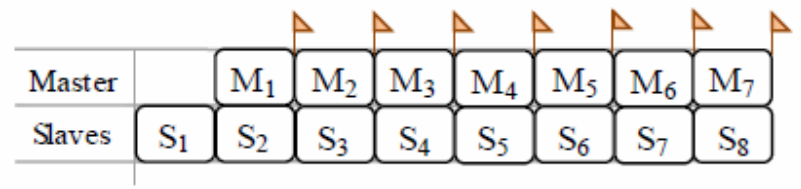

Figure (8): *The Pipelined Approach.*

6. CONCLUSIONS AND RESULTS:

This paper presents the effect of damping constant and rotor inertia constant of the machines on the behavior of electromechanical wave propagation in a one-dimensional ring power system. The analyzed system is continuum, and it is discretized for simplicity of analysis. From the simulation results, it is clear that the higher oscillatory wave vanishes with the increase of damping constant and it suppresses the disturbance wave from its propagation through the entire network. Also, the increase of rotor inertia constant leads to the electromechanical wave propagation velocity decrease.

In this paper, a custom FPGA-based MPSoC architecture is designed to meet the requirements of the new proposed parallel digital PID algorithm. The sequential digital PID algorithm was restructured, because of its inherent parallelism, to run concurrently into the four cores of the proposed architecture. The new parallel algorithm is directly mapped to the customized MPSoC architecture using the means of predefined hardware identifiers. Modifications to the algorithm have been made to propose a two-stage pipelined approach of the system. The system was implemented using VHDL, synthesized and simulated using Xilinx tools. Table.2 shows the simulation results for different implementations during the development process of the PDPID controller. For each implementations of the PDPID controller, the reduction ratio compared to the sequential digital PID controller is shown.

Table (2): *Execution times for different implementations.*

	EPM with RAM/ROM [Sequential]	**PDPID with shared RAM/ROM**		**Pipelined PDPID with shared RAM/ROM**	
Avg. PID Loop Instructions	349	249	-28%	158	-54%
Max. PID Loop Clock Cycles	698	498		316	

For future work, we hope to implement a self-tuning algorithm for control parameters while keeping the system parallelism unaffected. This new self-tuning feature may be implemented within the Master-Task but will affect its performance heavily. We may then replace the Master-Core with a faster microcontroller resulting into a heterogeneous system.

References:

[1] Y. Fong Chan, M. Moallem, andits W. Wang, "Design and implementation of modular FPGA-Basede PID controllers," IEEE Trans. Ind. Electron., vol. 54, no. 4, pp. 1898-1906, Aug. 2007.

[2] Katalin Popovici, Frédéric Rousseau, Ahmed A. Jerraya and Marilyn Wolf, "Embedded Software Design and Programming of Multiprocessor System-on-Chip", Springer, 2010.

[3] Jim Ledin, "Embedded Control Systems in C/C++: An Introduction for Software Developers Using MATLAB", 2004.

[4] Massimo Conti, Simone Orcioni, Natividad Martínez Madrid and Ralf E. D. Seepold, "Solutions on Embedded Systems", Lecture Notes in Electrical Engineering Vol. 81, Springer, 2011.

[5] Xilinx Userguide UG129, "PicoBlaze 8-bit Embedded Microcontroller User Guide", V2.0, 2010.

[6] Rudolf Usselman, "Open Floating Point Unit", The Free IP Cores Projects: http://www.opencores.org, 2000.

[7] Xilinx Application Note, "Quad-Port Memories in Virtex Devices", V1.0, 2002.

[8] Thomas Rauber, Gudula R¨unger, "Parallel Programming For Multicore and Cluster Systems", Springer, 2010