A Reconfigurable PID Controller

Sikandar Khan[1], Kyprianos Papadimitriou[2,3],
Giorgio Buttazzo[1], and Kostas Kalaitzakis[2]

[1] Scuola Superiore Sant'Anna, ITALY
[2] Technical University of Crete, GREECE
[3] Technological Educational Institute of Crete, GREECE

Abstract. We survey the Proportional-Integral-Derivative (PID) controller variants and we switch them in runtime via reconfiguration, as the control requirements change. Depending on the PID variant, e.g. P, I, PI, PD, PID, PI-PD, the involved computations to produce the control output are different. We rely on a previous published design to shorten the execution cycle of each controller variant, by increasing the number of arithmetic units operating concurrently. Furthermore, we incorporate a design based on multiplexers that allows for eliminating frequent reconfigurations, which were required in the previous work. Finally, we evaluate our approach in terms of resource utilization and reconfiguration time.

1 Introduction

The PID algorithm has been widely adopted by the industry due to its simplicity of design and implementation. An old work [1] surveying over 11,000 controllers in process industries, reports that more than 97% of regulatory controllers utilize PID. Although designing a PID controller is conceptually intuitive, it is hard in practice, if multiple and often conflicting objectives such as fast transient response and high stability requirements are to be met. Therefore, different PID controller variants have been suggested, each one serving better different control scenarios. These controllers, e.g. P-only, I-only, PI, PD, PID, PI-PD, generate a control output of different nature, ranging from more robust to more stable or accurate [2]. In the present work, we are switching the controller variants via reconfiguration, aiming at achieving a control response that integrates the best feature of all these variants depending on the requirements. The computations required to produce the control output differ, thus we are adjusting them at run-time. Our contributions are:

- the controller type is reconfigured instead of implementing statically all controllers and multiplexing their outputs. This accounts for latency reduction in delivering controller's output, and for resource savings;
- given a number of multipliers and adders, we are examining the overlapped computations so as to parallelize them, while respecting the dependencies and the delay added from the registers holding the intermediate results in each computational unit;

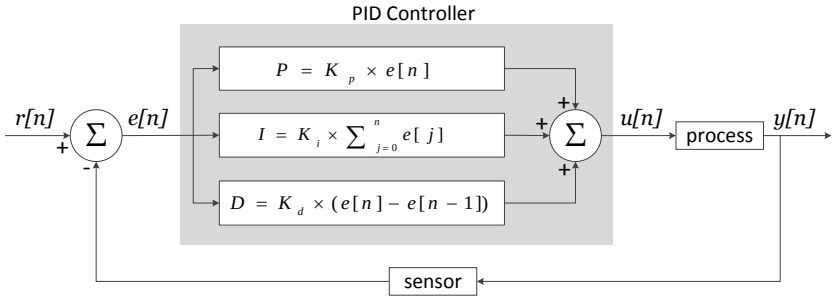

Fig. 1: A PID controller in a feedback loop. $r[n]$ is the desired value.

– we propose a design relying on multiplexers built in each reconfigurable module. This allows for avoiding frequent reconfiguration of the coefficients, i.e. gain parameters, of the controllers, yet respecting their dynamic characteristics.

The rest of the paper is organized as follows. In Section 2 we briefly discuss the PID controller and its variants, and in Section 3 the adaptive controllers. Section 4 presents our approach for switching the controllers. Section 5 has the implementation details along with its evaluation. Section 6 concludes the paper.

2 The PID Controller and its Variants

A PID controller is used in industrial applications for regulating processes as part of a control loop. It receives a setpoint request from the user, e.g. vehicle's desired speed when activating cruise control, and compares it to a measured feedback, e.g. vehicle's current speed. The difference between the setpoint and feedback values is termed error, and the job of the controller is to eliminate it. This process takes place continuously during which the PID controller performs computations to generate an output for eliminating the error. Figure 1 illustrates a generic PID controller. Mathematically it is expressed with Equation 1,

$$u[n] = K_p \times e[n] + K_i \times \sum_{j=0}^{n} e[j] + K_d \times (e[n] - e[n-1]) \tag{1}$$

where $e[n]$ represent the n^{th} sample of the instantaneous error obtained as a difference between the setpoint value $r[n]$, and the measured output of the process $y[n]$ for some physical variable under control, such that $e[n] = r[n] - y[n]$. The PID controller takes the error $e[n]$ as an input and computes its control output $u[n]$ based on its proportional K_p, integral K_i, and derivative K_d gains - termed also coefficients - such that $u[n] = P + I + D$, where P, I, and D refer to the proportional, the integral, and the derivative term, respectively. A closed-loop is inserted in which the process output, $y[n]$, is observed by a sensor, to calculate

the instantaneous error $e[n]$ after each sampling period T. The PID pseudo-code is given below.

```
Em = 0          #Em = e[n-1]
SumEm = 0       #SumEm = e[0]+e[1]+...+e[n-1]
LOOP
wait (T)        #1 execution cycle
    En = error  #En = e[n]
    Un = (Kp*En) + (Ki*(En+SumEm)) + (Kd*(En-Em))
    SumEm = En+SumeEm
    Em = En
end LOOP
```

A PID controller can be used in different combinations to achieve a control response of different nature. The work in [3] analyzes the different control structures. The P controller has fast system response (robust), and decreases system's steady state error (SSE). However, beyond a certain value of the SSE reduction, a further increase in the proportional gain leads to an overshoot of the system response that causes oscillation and leads to instability. Moreover, P controller never eliminates SSE, so it is suitable for systems that can tolerate a constant SSE. On the other hand, the PI controller eliminates the SSE, but it is characterized by slow response (sluggish); the integral term responds to accumulated errors from the past, thus it can cause the present value to overshoot beyond the setpoint, causing instability. The PD controller prevents sudden changes occurring in the control output resulting from sudden changes in the error signal and has good stability. The downside is that the derivative factor directly amplifies the noise. The PID controller has the optimum control dynamics, however, tuning its parameters to respond to different conditions is challenging. For instance, if a PID controller for motor is tuned without load, it will not perform optimally when the load changes. This is why most often a set of parameters is chosen that is working satisfactory in all cases and not necessarily best for any particular case [4]. The control response of PID variants are summarized in Table 1; their advantages can be grasped by contemplating their differences side-by-side.

Table 1: Response characteristics of different controllers.

	Controller type			
Parameter	P	PI	PD	PID
rise time	decreases	minor decrease	no effect	minor decrease
overshoot	increases	increases	decrease	minor decrease
settling time	small change	increases	decrease	minor decrease
steady state error	decreases	eliminates	no effect	minor decrease
stability	degrades	degrades	good	good for small K_d

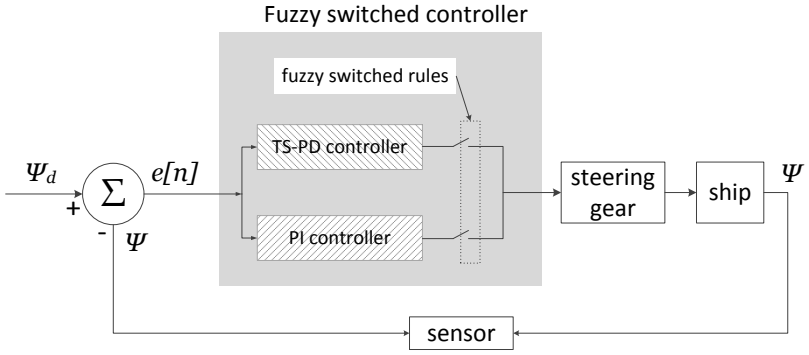

Fig. 2: The proper controller is activated depending on whether faster response or steady accuracy is required.

3 Adaptive Controllers

Non-linear or adaptive controllers operate efficiently in dynamic environments and cover a sufficiently large operating range. Adaptivity is achieved either by combining different controllers and switching amongst them based on their individual operating regime, or, by changing the gain parameters of a single controller as control requirements change.

3.1 Switching Controllers

The concept of dividing the operating envelope of any control process into operating regimes was proposed in [5]. Using this concept, the authors of [6] presented a performance-oriented ship track auto-control that combines the advantages of a fuzzy PD controller and a conventional PI controller. The PD controller is active in the transient regime to deliver fast response, and the PI controller is activated in the steady regime to achieve greater accuracy. The switching decision is based on the rudder angle Ψ of the ship, which is small during straight course (steady regime) but would change by large scale during a course change (transient regime). Figure 2 illustrates the concept of switching the controllers. Other earlier works combining different controllers to integrate both robustness and stability were published in [7, 8]. Similarly, [9] proposes switching amongst multiple co-existing PID controllers in an electrostatic micro-actuator system.

3.2 Reconfigurable Controllers

FPGAs technology offers good-closed performance and its suitability in control applications has been reviewed in [10], which highlights its advantages of speed and low-cost development, flexibility, and limited power consumption. In addition, reconfiguration has been used to modify the control parameters to a new

set of values according to the run-time requirements. To the best of our knowledge, only a few works have proposed reconfiguring the gain parameters [11–13]. The authors of an earlier work [14], proposed a reconfigurable circuit for controlling the passing of the values of gain parameters kept in registers, depending on whether the PID controller executes the P, I or D stage of its execution cycle; this results in frequent reconfigurations. Our approach is different in that it allows partial reconfiguration of the controller type, contrary to changing the gain parameters only. Table 2 summarizes the main attributes of our approach in contrast to previous ones.

Table 2: Static and reconfigurable resources in the proposed controllers. In our design the values of gain parameters are kept in registers that can be changed with a "write" command.

reference	Static	Reconfigurable
[11–13]	controller type type of operations # operations	gain parameters
[14]	controller type type of operations # operations gain parameters	switching of operands
present design	gain parameters	controller type type of operations # operations

4 Designing Effectively the Controller Variants

We are combining both the foregoing adaptive strategies to model a reconfigurable PID controller that can be flexible and fast. Figure 3 illustrates our approach. We are using as vehicle the architecture in [14], and initially we are studying the way it supports all controller variants. We then show how we have parallelized the computations, i.e. multiplications and additions, taking into account the delay introduced by the registers holding the intermediate results in each computational unit. Finally, we show that for switching the gain parameters - depending on the stage of the execution cycle -, we use multiplexers as opposed to frequently partially reconfiguring the fabric, as was done in [14].

4.1 Overlapped and dependent computations

In the original architecture [14], 1 multiplication and 1 addition are carried out in each stage of the execution cycle of the PID controller. The gain parameters are stored in registers and depending on the stage of the execution cycle, they

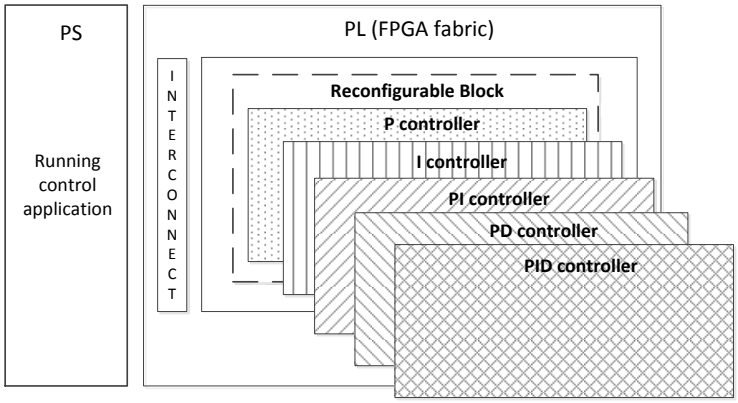

Fig. 3: The partially reconfigurable area is programmed with a different controller depending on the run-time control requirements.

are passed for computation to the PRODUCT and ADDITION blocks via reconfiguring a circuit. We use this original architecture to study its performance for designing other controller variants. Table 3 shows the actual number and type of computations to carry out one complete execution cycle, i.e. generate one control output, along with the dependencies between the computations. In Figure 4(a), we are analyzing the number of stages per execution cycle for the different variants, assuming they are implemented with the original architecture. In Figure 4 we have considered the dependencies of the terms inside the involved computations as well as the cycle delay added from the registers in each computational unit.

Table 3: Number and type of computations carried out in 1 period (T) of execution for the different controllers. Table exposes also the dependencies between the computations.

Controller type	# computations	Involved multiplications	Involved additions
P	1	$K_p \times En = P$	0
PI	4	$K_p \times En = P$	$En + SumEm$
		$K_i \times (En + SumEm) = I$	$P + I$
PD	5	$K_p \times En = P$	$En - Em$
		$-1 \times Em$	
		$K_d \times (En - Em) = D$	$P + D$
PID	8	$K_p \times En = P$	$En + SumEm$
		$-1 \times Em$	$En - Em$
		$K_i \times (En + SumEm) = I$	$P + I$
		$K_d \times (En - Em) = D$	$P + I + D$

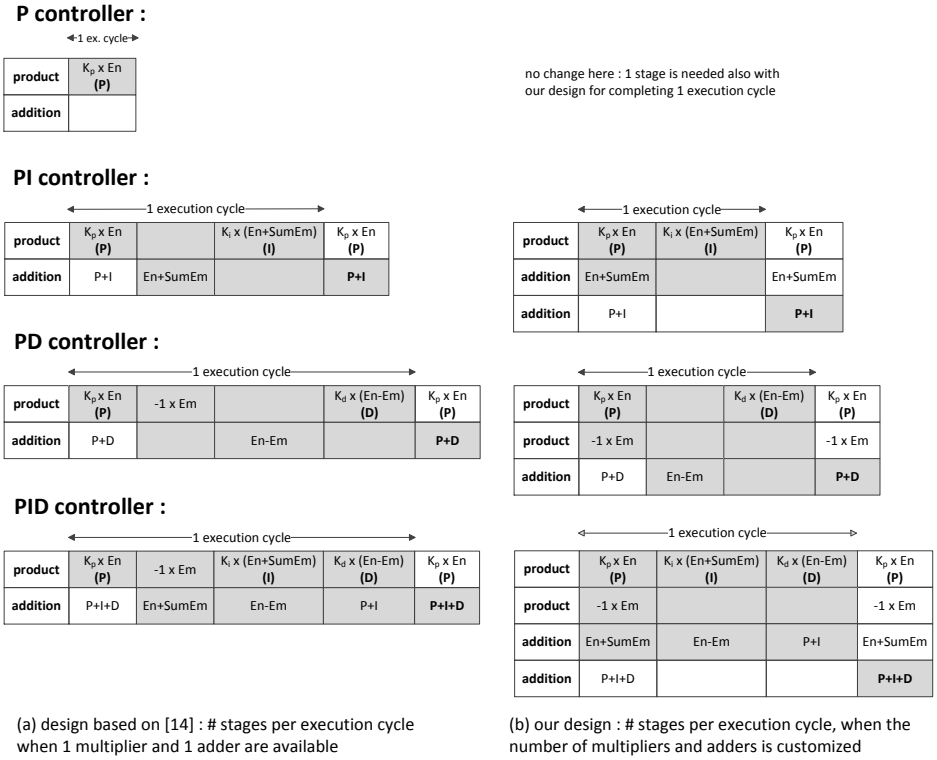

Fig. 4: Comparison of [14] VS. our design in terms of the number of stages per execution cycle. At the end of every execution cycle the output $u[n]$ is computed. Fewer stages can result in smaller latency.

4.2 Reducing the number of stages per execution cycle

We parallelize the arithmetic operations by incorporating additional multipliers and adders; the number of additional computational units differs amongst the controllers. This allows for shortening their execution cycle, which accounts for smaller latency and faster responsiveness. This is a typical requirement in real-time domains. The way the controller variants operate now is shown in Figure 4 (b). It illustrates that in almost all cases 1 stage can be eliminated, e.g. for the PD and the PID controllers the original design [14] completes the execution cycle in 4 stages, while our design completes in 3 stages. In the near future we will study in-depth the trade-offs in terms of the latency from the incorporation of registers in the computational units, i.e. for pipelining and registering their inputs/outputs.

4.3 Switching the gain parameters via multiplexers

In the original design, one reconfiguration occurs in every stage of the execution cycle, e.g. 4 reconfigurations for the PID. If the sampling period (T) for some control variable is $100ms$, roughly speaking, a reconfiguration occurs every $25ms$. We instead create partial bitstreams in which the design for each variant incorporates a fixed set of multiplexers that allow for switching amongst the gain parameters, depending on the stage of the execution cycle. Figure 5 illustrates our design, and its output depending on the stage. A Finite State Machine (FSM) is also implemented in each bitstream for controlling the "SEL" signals of the multiplexers. Depending on the stage of the execution cycle, the FSM enters a different state driving accordingly the "SEL" signals. The operation of the FSM is simple and further details remain out of the scope of this paper. This additional hardware increases the bitstream size, but eliminates reconfiguration in every stage.

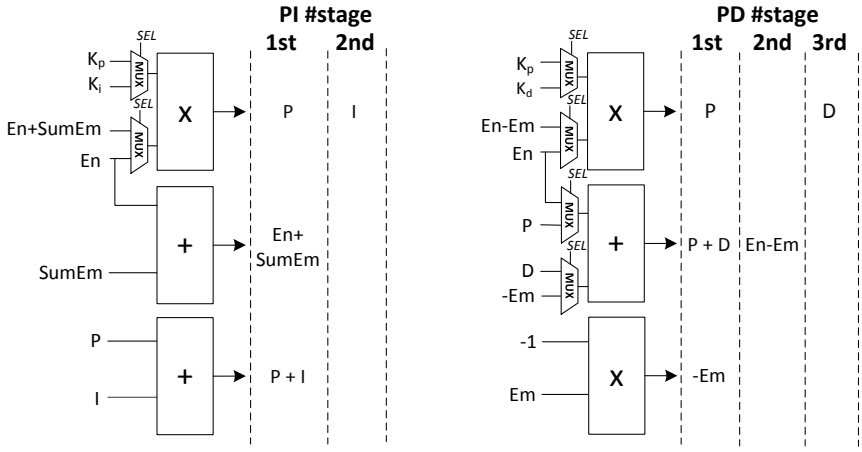

Fig. 5: A fixed number of multiplexers in each RM allows for eliminating partial reconfiguration of the gain parameters, i.e. coefficients, in every stage of the execution cycle.

5 Implementation of a Reconfigurable Arithmetic Block

To demonstrate our approach, we implemented a reconfigurable arithmetic block in the Programmable Logic (PL) of a Zybo platform. We also developed code for the Processor System (PS), which is responsible for I/O data handling, reconfiguration of the block, and runs the control application. This experimental setup

is shown in Figure 6. The functionality of the block PL can be altered by loading the corresponding partial bitstream from a DDR memory into a Reconfigurable Partition (RP) via the PCAP configuration port [15]. One Reconfigurable Partition (RP) in the block can host one Reconfigurable Module (RM) at a time, performing a fixed set of $32-bit$ wide simultaneous operations. A number of RMs have been predesigned and stored in DDR that implement the arithmetic operations of the different variants. The operands to be processed and the results are stored in a fixed set of register array, accessible to the PS through an AXI interface.

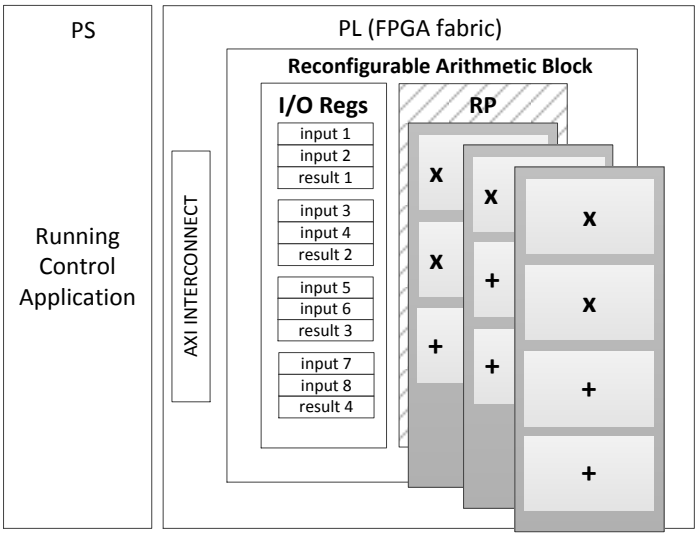

Fig. 6: The proposed reconfigurable arithmetic block, and the experimental setup for testing it.

In each RM, adders are implemented in LUTs, and multiplications in DSP slices. The DSP slice, shown in Figure 7, consists of a fundamental $25 \times 18 - bit$ multiplier with pipelining and extension capabilities [16]. A total of 80 DSP slices are available in the PL of Zybo (Z-7010), distributed equally across 4 Clock Regions (CRs) [15, 16]. It should be noted here that according to Xilinx, restricting modules to one CR accounts for reaching high clock frequencies. To support wider multiplications the DSP slices need to be cascaded. This affects the total number of multiplications than can be created within each CR; in our case the number of multiplications was reduced from 20 (of $25 \times 18 - bit$ each) down to 16 (of $32 \times 32 - bit$ each). Such trade-offs between the number of simultaneous multiplications and data widths of the operands must be taken into

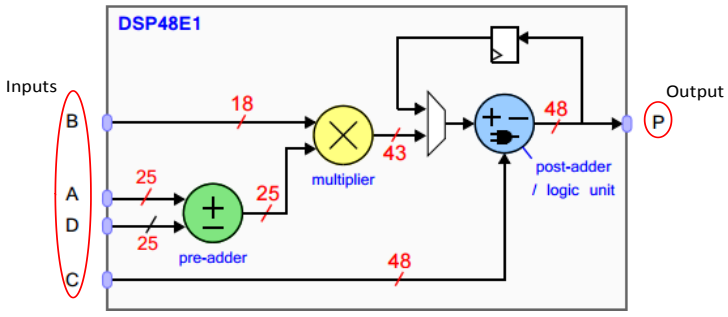

Fig. 7: Xilinx's fundamental DSP48E1 Slice [16]

account when designing any system that involves wider multiplications than the width of the fundamental multiplier. Cascading DSP units also demands more area to be annexed within each Reconfigurable Region (RR). This is why to implement our largest RM - it performs up to 15 simultaneous multiplications with $32 \times 32 - bit$ width - we annexed the entire X0Y1 CR in floorplanning the RP of our design. This region is illustrated with the large Pblock on the left side of Figure 8. However, for the case of the PID controller, a smaller region is required that can implement up to 4 arithmetic units. This is shown on the right side of Figure 8.

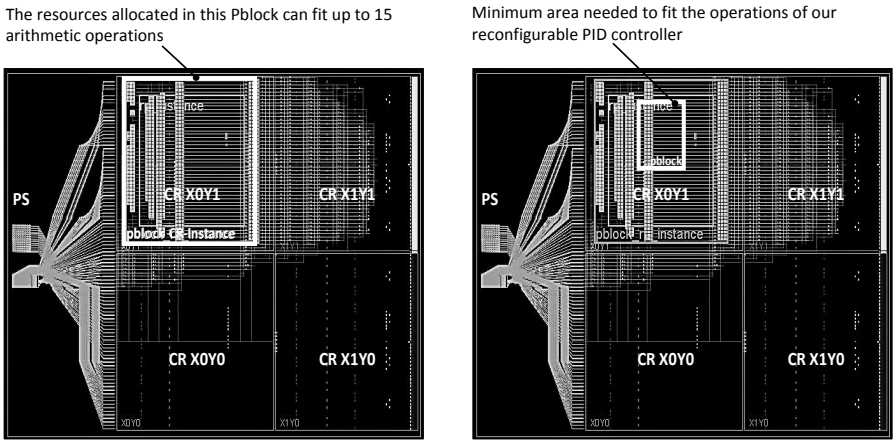

Fig. 8: The large Pblock on the left accommodates up to 15 arithmetic operations. The small Pblock on the right fits the arithmetic operations of the most resource-consuming controller, i.e. PID.

In terms of resource utilization within the Pblock, we obtained that our design is DSP-consuming mainly, while the requirements in flip-flops and LUTs are low. Table 4 summarizes the resource utilization in the larger Pblock of Figure 8, for different Reconfigurable Modules (RMs). Floorplanning to the region indicated by the large Pblock of Figure 8, creates partial bitstreams of 192,343 bytes each. However, by restricting the floorplanning to the smaller region in Figure 8, which has enough resources to implement the required computations for any variant - 4 is the maximum, in the case of the PID controller - the size of partial bitstream created for each PID variant becomes 55,727 bytes. Considering the maximum reconfiguration speed of $400MBps$ for the latest Xilinx devices, a total of 132.8μs are needed to load the partial bitstream into the fabric via PCAP configuration port. Future versions can be built using the ICAP configuration port, accessed by a fast hardware reconfiguration engine [17].

Table 4: Resource utilization in the large Pblock of Figure 8, for different RMs. Each one implements a different number of arithmetic operations. Results are from Xilinx Z-7010 device.

#arithmetic units in the RM	#flip-flops (%)	#LUTs (%)	#DSPs (%)
5	160/5,696 (2.80%)	161/2,848 (5.65%)	5/16 (31.25%)
10	320/5,696 (5.60%)	321/2,848 (11.27%)	10/16 (62.50%)
15	480/5,696 (8.43%)	461/2,848 (16.89%)	15/16 (93.75%)

6 Conclusions

While most works were focused on reconfiguring the gain parameters, we propose reconfiguring the controller type. In this way, the number of active arithmetic units is adjusted at run-time. Our reconfigurable arithmetic block supports up to a max of 15 simultaneous operations (of $32 \times 32 - bit$ each), and we envision its use in adaptive state-space MIMO controllers such as Linear-Quadratic-Gaussian (LQG) control. These controllers have high computational and storage demands as they require several linear algebraic operations including matrix multiplication that needs to store and process more information.

Acknowledgement

This research was partly funded by the EU FP7/2007-2013, under grant agreement no. 610640, DREAMS project.

References

1. L. Desborough and R. Miller, "Increasing Customer Value of Industrial Control Performance Monitoring - Honeywell's Experience," in *AIChE Symposium Series*, 2002.
2. K. H. Ang, G. Chong, and Y. Li, "PID Control System Analysis, Design, and Technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, July 2005.
3. C. C. Hang, K. J. Åström, P. Persson, and W. K. Ho, "Towards Intelligent PID Control," *Automatica*, vol. 28, no. 1, pp. 1–9, 1992.
4. I. Mikkilineni, S. Patel, and C.-H. Tai, "Optimizing a PID Controller for Simulated Single-Joint Arm Dynamics," http://isn.ucsd.edu/classes/beng221/problems/2014, Integrated Systems Neuroengineering Laboratory, University of California San Diego, San Diego, US, Tech. Rep., November 2014.
5. Tor Arne Johansen, "Operating Regime Based Process Modeling and Identification," PhD thesis, University of Trondheim, 1994.
6. B. Jia, H. Cao, and J. Ma, "Design and Stability Analysis of Fuzzy Switched PID Controller for Ship Track-Keeping," *Journal of Transportation Technologies*, vol. 2, no. 4, pp. 334–338, October 2012.
7. B. Jia, G. Ren, and G. Long, "Design and Stability Analysis of Fuzzy Switching PID Controller," in *IEEE World Congress on Intelligent Control and Automation (WCICA)*, 2006.
8. A. Otsubo, K. Hayashi, S. Murakami, and M. Maeda, "Fuzzy Hybrid Control Using Simplified Indirect Inference Method," *Fuzzy Sets and Systems*, vol. 99, no. 3, pp. 265–272, November 1998.
9. M. Vagia, G. Nikolakopoulos, and A. Tzes, "Design of a Robust PID-Control Switching Scheme for an Electrostatic Micro-actuator," *Control Engineering Practice*, vol. 16, no. 11, pp. 1321–1328, November 2008.
10. E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "FPGAs in Industrial Control Applications," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 224–243, March 2011.
11. G. Economakos and C. Economakos, "A Run-Time Reconfigurable Fuzzy PID Controller Based on Modern FPGA Devices," in *Mediterranean Conference on Control and Automation (MED)*, June 2007.
12. R. le Roux, G. van Schoor, and P. van Vuuren, "Block RAM Implementation of a Reconfigurable Real-Time PID Controller," in *IEEE International Conference on High Performance Computing and Communication and IEEE International Conference on Embedded Software and Systems (HPCC-ICESS)*, October 2012.
13. M. Pelc, "Self-Tuning Run-Time Reconfigurable PID Controller," *Archives of Control Sciences*, vol. 21, no. 2, pp. 189–205, November 2011.
14. M. F. Francisco Fons and E. Canto, "Custom-Made Design of a Digital PID Control System," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2006.
15. Xilinx, "Vivado Design Suite User Guide - Partial Reconfiguration, UG909 (v2017.1)," http://www.xilinx.com/, Tech. Rep., April 2017.
16. L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, "The Zynq Book Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc," http://www.zynqbook.com/, Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK, Tech. Rep., July 2014.

17. S. G. Hansen, D. Koch, and J. Tørresen, "High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro," in *Reconfigurable Architectures Workshop (RAW) co-located with IEEE International Parallel and Distributed Processing (IPDPS)*, May 2011, pp. 174–180.