CUSTOM-MADE DESIGN OF A DIGITAL PID CONTROL SYSTEM

F. Fons, M. Fons, E. Cantó

Department of Electronic, Electrical and Automation Engineering
DEEEA-ETSE-URV, Rovira i Virgili University, Tarragona, Spain

ABSTRACT

In the field of real-time signal processing, like most of automatic control systems nowadays present at the industry and focused on PID (Proportional-Integral-Derivative) controllers, it is common to find software-oriented solutions based on powerful 32-bit DSP, RISC or CISC processors. This work deals with the hardware/software co-design of a PID coprocessor, all embedded on a system-on-chip device. The performances reached by a platform composed of an 8-bit MCU and a dynamically reconfigurable FPGA allow scheduling the PID algorithm as a set of tasks executed by both devices concurrently. Moreover, thanks to the flexible hardware characteristics, some modules synthesized into the FPGA are reconfigured at run-time while the rest keeps on active. This cost-effective approach, encouraged by its parallelism, is an alternative to commercial –both general-purpose and specific-purpose– processors in whatever made-to-measure engineering application.

1. INTRODUCTION

Embedded systems often must give solution to engineering problems that simultaneously involve automatic control and signal processing disciplines. A clear example is the design of a digital PID controller: it faces up to not only supervisory and decision-making algorithms but also intensive arithmetic computing. The way of solving this industrial issue has been evolving along the time just as the electronic technology advances:

- Originally, general-purpose microprocessors were designed to execute control-oriented tasks efficiently. Despite this, its modest implementation of multipliers through a series of shifts and adds –spending a significant number of clock cycles– limited its computing power.
- Nonetheless, in 1982, Texas Instruments introduced the first TMS32010 DSP processor, which incorporated a hardware MAC unit to make possible the calculus of a multiply-accumulate operation in a single clock cycle. Conceptually, a DSP algorithm, commonly used in a broad range of applications such as digital filtering or image processing, performs an algebraic sum of products. And from the outset, these computational requirements influenced the architecture of DSP processors [1].
- Traditionally, microcontrollers and DSPs are viewed as standing at opposite extremes of the processing world. While MCUs are best suited for control applications that require low-latency response to unsynchronized events, DSPs have the inverse strengths and they shine in applications where intensive mathematic computing is demanded. A MCU can be used in arithmetic applications but the one-operation-at-a-time nature of its ALU makes such use less than optimal. Similarly, a DSP can be forced into a control application but its internal architecture renders this task inefficient in code and time. Hence, in the past, those applications requiring a fusion of signal processing and control-oriented algorithms were implemented by two separate processors: DSP and MCU, what results a too expensive solution. Some years ago, however, lots of vendors began to offer DSP-enhanced versions of their MCUs as an alternative to that dual-processor option and covered thus the existing gap between general-purpose processors and specialized DSPs. CPU families such as ARM have ported DSP functionality to existing microprocessor designs, borrowing the architectural features of DSPs. Many of these hybrid processors achieve signal processing performances comparable to that of DSPs.
- On the other hand, recently, FPGAs have been gaining considerable relief in high-performance DSP applications and are emerging as ideal coprocessors for standard processors [2]. System-on-Chip or FPGA-based designs bring some key advantages to signal processing: they provide tremendous computational throughput by using highly parallel architectures. In addition, its dynamic partial reconfiguration capability extends much more the cost-effective possibilities thanks to the silicon-area multiplexing of the synthesizable algorithm.

This work describes the Hw/Sw co-design of a PID controller embedded on the Atmel AT94K40 SoC. The reasons of this election are clear: a low-power and reduced 8-bit MCU is enough to carry out the control tasks whereas the arithmetic computing is performed in a 32-bit high-precision data length by a dedicated and flexible FPGA. Both concepts, control and arithmetic, are efficiently fitted and partitioned into two devices while all the system is packaged in a single-chip, what would not be feasible in all other technical approaches discussed previously.

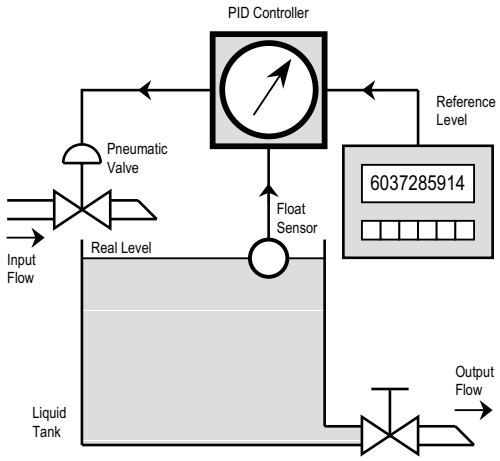

Figure 1. Liquid level control system.

After this introduction, section 2 reviews the PID control theory. Sections 3 and 4 discuss the more outstanding aspects of the Hw/Sw development. The experimental results are compiled in section 5. Finally, section 6 exposes the conclusions of this work.

2. PID CONTROL TECHNIQUE

PID compensation is one of the most common forms of closed-loop control. The general use of digital computers allowed, already several decades ago, to increase the interest for the modeling of continuous dynamic systems in order to, finally, control them in a discrete way. In these systems -for instance the control of the liquid level inside a tank depicted in Fig. 1-, analog signals are converted into discrete digital samples acquired cyclically at a period T and in which calculations must be complete before the next time sample begins [3]. A discrete PID controller can be generically modeled by the following expression:

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

where *e[n]* represents the nth sample of the existing error between the reference and the real value of the physical variable under control, *u[n]* is the resultant stimulus to be applied to the plant in order to compensate this error, and K_p, K_i and K_d are the constant gains assigned to the proportional, integral and derivative components of the controller respectively.

Thus, the PID controller looks for compensating the error between the desired and the effective value of the output signal. A closed loop is inserted for this in which the plant is self-fed by an input consisting of the sum of three

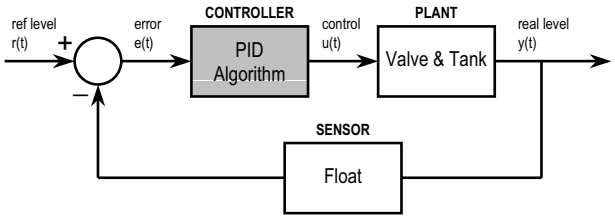

Figure 2. Block diagram of the liquid level control system.

terms: a proportional factor responsible for adjusting the control signal in the same percentage than the instantaneous error, a derivative factor that contributes proportionally to the error rate of change, and an integral term with the role of eliminating the steady state error by means of integrating the instantaneous error along the time. The three terms help to cancel any deviation or disturbance present in the system and maintain therefore the equilibrium, achieving an input tracking with a dynamic response in accordance with the fixed characteristics of the whole plant-controller [4], [5].

3. HARDWARE/SOFTWARE CO-DESIGN

Digital compensation of closed-loop systems is a consolidated application area for embedded MCUs. Although high-scale processors are the habitual choice for the most demanding control applications, often these software-based approaches do not offer enough power for real-time specifications. Even though they are well suited for applications where the format of data to be processed matches their word width, their performance drops in other cases. Moreover, the increase of cost demanded to jump from an 8-bit processor to a bigger one because of computational reasons gets, sometimes, not sufficiently justifiable. Instead, small processors can handle these performances whether they are equipped with peripheral coprocessors to speed up the arithmetic-logic operations: dedicated FPGA-based circuits are able to outperform the bottleneck of processors mainly due to their inherent customization. Under this idea, this work treats the Hw/Sw co-design of a PID controller that, in the presence of two control units, makes possible a concurrent execution: a specific coprocessor assumes the multiply-add computing effort whereas a processor takes charge of handling the data (I/O) and reconfiguring the hardware. In fact, this partitioning pursues to distribute the processing load between both devices in order to look for the best-balanced solution referring to the area-time trade-off. Therefore, a PID cycle is scheduled in four stages making use of only a multiplier and an adder and where their operands are time-multiplexed through a partial reconfiguration of the FPGA.

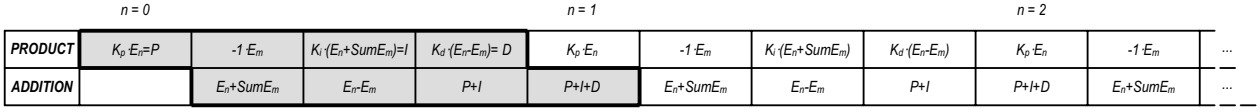

Figure 3. Scheduling of the PID algorithm made with a multiplier and an adder.

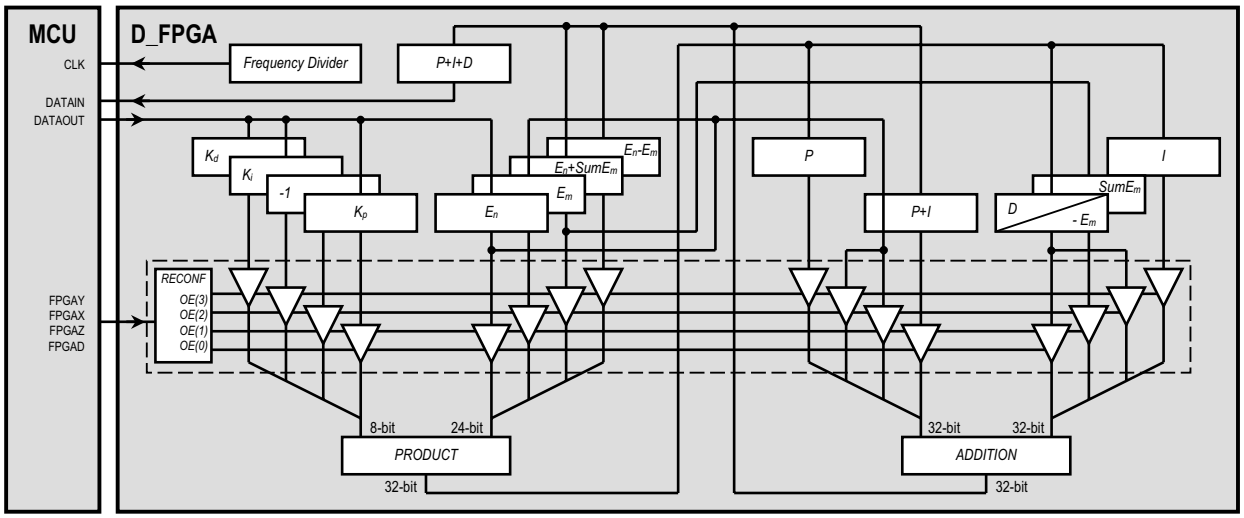

Figure 4. PID coprocessor architecture.

The self-explanatory PID pseudo-code is shown next.

```
Kp=constantP
Ki=constantI
Kd=constantD
Em=0          /*Em=e[n-1]*/
SumEm=0       /*SumEm=e[0]+e[1]+...+e[n-1]*/
Loop
  Wait (T)
  En=error  /*En=e[n], Un=u[n]*/
  Un=Kp*En+Ki*(En+SumEm)+Kd*(En-Em)
  SumEm=En+SumEm
  Em=En
End loop
```

Code 1. PID algorithm.

The software functions involved in this algorithm are the PID initialization and the cyclic computation of the output *u[n]* depending on the instantaneous input *e[n]* and the historic and dynamic evolution of the system.

```
void initPID (char Kp, char Ki, char Kd)
long cyclicPIDTask (char En)
```

Code 2. PID software function prototypes.

The established sampling period *T* can be programmed by the MCU through a cyclic timer interrupt. In order to operate the coprocessor, data are loaded first into the input registers by the MCU. The FPGA coprocessor is then instructed to compute the PID along several partial reconfigurations. Finally, upon completion of the PID cycle, the results may be read from the output register and all this cyclic process is started again and repeated indefinitely.

4. PID COPROCESSOR

The PID controller has been developed in the AT94K40 system-on-chip. This monolithic device, also known as FPSLIC (Field Programmable System Level Integrated Circuit), combines in the same package an 8-bit AVR RISC core processor with its peripherals, a 40 kgates AT40K40 SRAM-based FPGA as well as 36 kbytes Dual-Port SRAM shared between MCU-FPGA [6]. A useful characteristic of this device is its Cache Logic ability, what allows either the same FPGA or the AVR, through an specific configuration interface, to reconfigure on-the-fly whichever part of the FPGA memory, at a bit-level granularity, while the rest of device continues active [7]. Our design takes full advantage of these performances in order to optimize the coprocessor. Concerning the hardware design, the involved computing units -an 8x24-bit multiplier and a 32-bit adder- are hard macros generated by the Atmel IDS Macro Generator tool, which provides optimized technology-dependant modules. Both arithmetic integer units have about the same critical path, what made feasible to schedule a parallel processing of these two operations working at the same frequency.

4.1. Dynamic Partial Reconfiguration

As depicted in Fig. 3, a PID cycle is executed in four steps and a multiplication and an addition are performed in each of them while the operands are multiplexed at run-time by the MCU through partial reconfigurations of the FPGA.

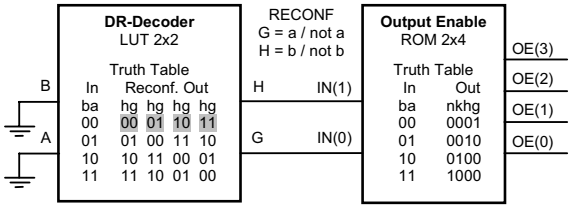

Figure 5. Dynamic multiplexing of the MAC operands.

In static designs, a logic function updates its outputs when any of its inputs changes. Our dynamic design, on the contrary, keeps the inputs tied to ground and the outputs change through reconfiguring the LUTs of the logic cell. In this way, only some logic resources evolve to take effect on the lines *oe* shown in Fig. 4 and Fig. 5.

5. EXPERIMENTAL RESULTS

The PID coprocessor has been described in VHDL and C languages. The hardware design is split into a static skeleton and a dynamic part which is updated each reconfiguration cycle: the PID controller is reconfigured with only a latency of 4 clocks, what allows to switch the operands and the result of both multiplier and adder modules. Thus, a PID cycle is performed in 34 clocks cycles: 16 for dynamic partial reconfiguration and 18 for MCU-FPGA data transfer.

Platform (Op. System)	Time	Development Tools
Pentium 4 @ 2.66 GHz (Windows XP)	1350 ns	Visual C++ 6.0 (Win32)
AMD K6-2 @ 450 MHz (MS-DOS)	1840 ns	Borland C++ 3.1 (MS-DOS)
AMD K6-2 @ 450 MHz (Windows 98)	1550 ns	Visual C++ 6.0 (Win 32)
80C188EB @ 25 MHz (Embedded system)	58000 ns	Borland C++ 3.1 (MS-DOS)
AT94K40 SoC @ 12.5 MHz (Hw/Sw development board)	2720 ns	Atmel System Designer IAR compiler

Table 1. PID cycle performance evaluation.

A study focused on software-based computing showed that the time involved in the integer processing of a PID cycle would be unacceptable in most of high-speed digital control system applications. Instead, a Hw/Sw co-design improves the performance as depicted in Table 1 taking into account the clock frequency of the different tested platforms. Due to the existing area-time trade-off and given that the FPGA cost is made conditional on its size, our design pursues to distribute the load of the MCU and the FPGA, avoiding thus whatever bottleneck and giving a balanced serial-parallel implementation through a four-stages scheduling. In this way, our implementation makes possible to reconfigure the PID coprocessor at run-time spending a few clocks, what keeps practically invariable the overhead of the application and validates the area-saving choice relating to place only a multiplier and an adder. Hardware and software parts have been co-simulated together by the EDA System Designer tool. Finally, the resultant bitstream (MCU program code and FPGA configuration) has been downloaded into a prototype board that we developed for carrying out an automatic test. This board basically consists of the SoC device, an external Eeprom memory that stores the bitstream and a serial RS232 transceiver connected to the UART of the MCU. Through this serial interface, the developer, from a host PC, is able to stimulate the system by entering input data and analyze how the system responds according to its tuned PID parameters. Continuous PID cycles can be executed; at the same time that the system runs, the output data of the PID controller are compared with the same results computed by the PC software application that also implements the PID algorithm to double check and verify the proper behavior of our Hw/Sw design.

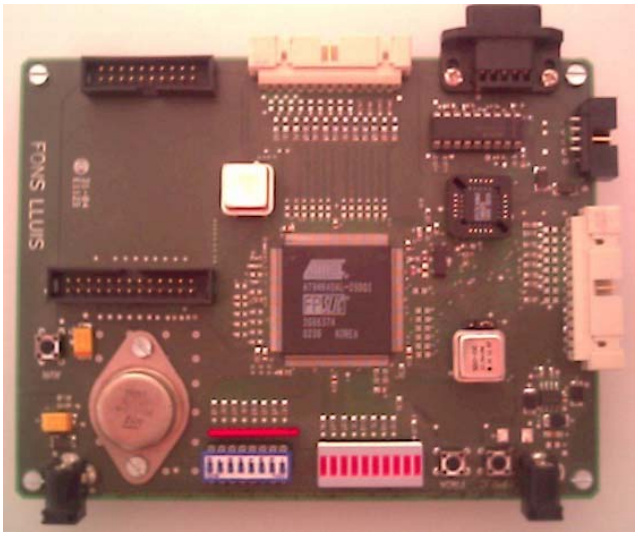

Figure 6. Prototype board developed.

6. CONCLUSIONS

Many field applications demand to merge control and computing processes. While a low-cost CPU can manage the control tasks, a powerful ALU is required to accelerate the computing operations. These opposite requirements are solved in a DSP processor by enlarging the data-path to 16 or 32 bits. Our approach is inspired on SoC technology since it permits to split these control and computing resources more efficiently: an 8-bit MCU handles the data-path whereas a dedicated 32-bit ALU/MAC takes charge of the arithmetic computing. Moreover, flexible FPGAs permit to optimize its area by multiplexing strategies. A generic prototype of PID controller has been implemented, showing all the methodology and design flow. This Hw/Sw solution has been compared with other software-based approaches and the good performance obtained makes it suitable for being ported to real industrial applications.

REFERENCES

[1] Jennifer Eyre, Jeff Bier, "The Evolution of DSP Processors," Berkeley Design Technology Inc., 2000.

[2] Steve Zack, Suhel Dhanani, "DSP Co-Processing in FPGAs: Embedding High-Performance, Low-Cost DSP Functions," www.xilinx.com, 2004.

[3] J. Murphree, B. Brzezinski, J. K. Parker, "Using a Fixed-Point DSP as a PID Controller," American Society for Eng. Education Annual Conf. & Exposition, session 2359, 2002.

[4] Charles L. Phillips, H. Troy Nagle Jr, *Digital Control System Analysis and Design*, Prentice-Hall, 1995.

[5] David Wilson, "16-Bit DSP Servo Control with the MC68HC16Z1," www.freescale.com, 1996.

[6] Atmel Corp., "AT94K Series Field Programmable System Level Integrated Circuit," www.atmel.com, 2002.

[7] Atmel Corp., "AT94K Series Cache Logic® (Mode 4) Configuration," www.atmel.com, 2001.