A Soft Fixed-Point Digital Signal Processor Applied in Power Electronics

S. de Pablo, J. Cebrián, L. C. Herrero
University of Valladolid
E.T.S.I.I., Paseo del Cauce, s/n
47011 Valladolid (Spain)
sanpab@eis.uva.es

A. B. Rey
Polytechnic University of Cartagena
Edif. Antiguo Hospital de Marina
30202 Cartagena, Murcia (Spain)
alexis.rey@upct.es

Abstract

*This article presents the "DSPuva16" processor, specifically developed for Power Electronics applications. As its name indicates, this is a Digital Signal Processor oriented to calculation: it operates using fixed-point 16-bit numbers extending its precision up to 24 bits. The MAC operations (rD = rD ± rS * rT) are made regularly using only one instruction cycle, that requires four clock cycles. The processor has been physically tested on FPGA working at 40 MHz.*

The application of this processor in Power Electronics (distributed generation systems, AC motor control, active filters) is quite natural, because all working magnitudes are limited in range. The use of several processors is also possible: each one would execute a different regulation loop, with working cycles between 5 and 100 microseconds.

1. Introduction

The first devices used to control electronic power converters (rectifiers, inverters) were analog circuits based on op-amps [7]. When first microprocessors appeared, they were immediately implemented in the control circuits [5] replacing their equivalent analog ones because of their inherent advantages of stability, noise immunity and facility of adjustment. Later we saw the rise of digital signal processors, with greater calculation capacity, which allowed a more precise control and to reach remarkably superior performances [2].

At the moment we are in a point in which, clearly, a processor alone does not suffices for controlling all the system: lots of regulation loops must be controlled, some of them working at high frequency (near the microsecond) [6]. In addition, other tasks such as remote communication or user interface must be attended with increasing behavior demands.

The obvious solution is to use several processors to take care of these diverse tasks. It is not only cheaper and easier to control them that way instead of using only one high performance processor, but also that is the only way to guarantee control in real time.

What happens if the possibility of connecting several processors in a printed circuit board is evaluated? Immediately we get involved in new problems: a high number of nets are required for the high data flow between processors, because the standard serial channels are not usually enough. So additional intermediate elements must be added to accommodate the different data flows. Besides, in general, all used components quickly become obsolete, which forces to redesign all systems once and again. All of these topics lead inevitably to elevated NRE costs.

The panorama is completely different if soft processors are used. Possibly their performance is not as good as their counterpart hard ones, but when they become all integrated in a chip (FPGA or ASIC) all the previous problems disappear immediately. Communication between processors is flexible and immediate using dual-port synchronous memories [4], whose use absolutely does not affect the cost of the equipment [1]. Obsolescence is null since previous designs can be synthesized again, targeting new devices, which will be able to work at higher frequencies.

The main limitation we have, specially in the FPGA domain, is the impossibility of using floating-point units, because their use is absolutely prohibitive if they are not hard-wired on the chip. Nevertheless, it is also possible and it is not excessively complex the use of fixed-point arithmetic. All magnitudes (voltages, currents, gains of regulators, etc.) are naturally or artificially limited by working restrictions [6].

Therefore, in this applications field it would be normal the use of several fixed-point digital signal processors and one or two general purpose processors to take care of communications and user attention. Their communication inside FPGA or ASIC devices is not expensive: there are no restrictions in the amount of communication lines, because they are all internal. The bandwidth is not a problem either, because dual-port memories may work at system frequency and use very low space.

This article presents a fixed-point DSP with enough performance and small size, prepared to handle a reduced set of variables. It has been designed thinking that it could be better to use several small and connected processors solving different loosely coupled tasks, than a huge processor doing all the work. The latter would force designers to use a considerably greater clock frequency and a complicated interruption scheme to attend all the real time demands. On the other hand, several processors may execute different regulation loops employing different working periods, but they are easily coupled as seen on section 7 and demonstrated in [1].

2. Main features of the DSPuva16

The DSPuva16 is a 16-bit fixed-point digital signal processor that extends its precision up to 24 bits. It uses what is called Harvard architecture, because it gets instructions using dedicated buses to its program memory (whose size varies between 256x16 and 4Kx16, according to the processor model) and exchanges data (up to 256 synchronous 16-bits ports) using other buses.

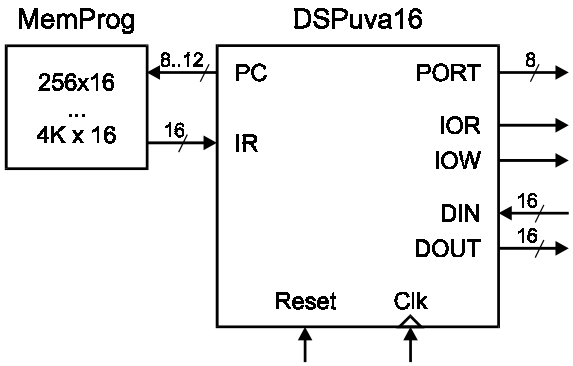

Figure 1. External connections of the DSP.

Its basic operation is the multiplication with accumulation, either positive or negative (rD = rD ± rS * rT), that it regularly executes in an instruction cycle. This one is always made using four clock cycles, as it corresponds to its RISC architecture.

It has 16 registers of 24 bits, denominated from 'r0' to 'r15', that can be used in any operation except for 'r0', which cannot be used as operand. Indeed this characteristic allows us to considerably extend the possibilities of the processor without adding excessive complexity to the internal design. When the 'r0' codification is found in the location of 'rS', its value is immediately annulled, and when it is found in the position of 'rT', its value is replaced by a 16-bit constant that is taken from the program memory. No additional delay is introduced when a constant is called.

This scheme extends, for example, a basic processor instruction as it is the sum (rD = rS + rT). It allows operating with an immediate constant (rD = rS + K) and making direct assignments to any register (rD = 0 + rT; rD = 0 + K). This property is widely used in the programming of digital filters, whose general structure is:

$$y(t) = c_0 \cdot x(t) + c_1 \cdot x(t-T) + \ldots + d_1 \cdot y(t-T) + d_2 \cdot y(t-2T) + \ldots$$

where all coefficients are constant.

External accesses are made through 256 synchronous ports (each access is completed in only one clock cycle) with direct addressing: pN = rS; rD = pM; where N and M are any value between 0 and 255. It could be thought that this addressing mode is less flexible than the indirect one, but in the application field of this processor this does not mean any limitation: it is usual that ports correspond with memory positions used to extend the processor spartan storage capacity, or they may connect with physical devices that allow reading or generating analog measures. Anyway, indirect addressing is always possible using an external pointer, integrated in the same chip; we must remember that we are designing within an FPGA or ASIC.

The control of subroutines is similar to other processors, but the user must dedicate a register (usually 'r0') to keep safe the returning address; later on, the same register can be used to return to the calling point. If nested subroutines are desired, either another register or an external LIFO stack should be used.

3. The instruction set

The instruction set of the DSPuva16, as shown in the table 1, is very simple. It consists only of 17 different instructions, but it reaches a remarkable flexibility thanks to how it uses the codification corresponding to 'r0', as it has been explained in the previous section.

Table 1. DSPuva16 instruction set.

Opcode	Mnemonic	Operation
0000 aaaa aaaa dddd	call (rD) addr	Absolute jump
0001 0xxx xxxx ssss	ret (rS)	Return (pc = rS)
0001 1fff aaaa aaaa	jpFlag addr	Relative jump
0010 dddd nnnn nnnn	rD = pN	Read from port
0011 ssss nnnn nnnn	pN = rS	Write to port
0100 dddd ssss tttt	rD = rS * rT	Normal product
0101 dddd ssss tttt	rD = rS x rT	Shifter product
0110 dddd ssss tttt	rD = rD + rS * rT	Positive MAC
0111 dddd ssss tttt	rD = rD – rS * rT	Negative MAC
1000 xfff dddd tttt	ifFlag rD = rT	Cond. assignment
1001 xfff dddd tttt	ifFlag rD = –rT	Cond. assignment
1010 dddd ssss tttt	rD = rS + rT	Addition
1011 dddd ssss tttt	rD = rS – rT	Subtraction

1100 dddd ssss tttt	rD = rS and rT	Logic AND
1101 dddd ssss tttt	rD = rS or rT	Logic OR
1110 dddd ssss tttt	rD = rS nor rT	Logic NOR
1111 dddd ssss tttt	rD = rS xor rT	Logic XOR

Whenever a subroutine is called the returning address is kept in a register, usually 'r0'. The instruction code dedicates only eight bits to the destination address, so it seems that program length must be limited to only 256 instructions. This is more than enough for many cases, because it is preferred to use several processors executing different regulation loop, so programs are usually short. Anyway, the available program length has been extended using a simple and powerful mechanism: there are up to five different models for this processor (named 'A', 'B', 'C', 'D' and 'E'), with programs of up to 256, 512, 1K, 2K and 4K instructions. When an absolute jump takes place (with the instruction 'call'), it is only possible to jump to even positions when the 'B' model is used, only to one of each four if the model is 'C', and so on. That means that all subroutines must be correctly aligned in the program memory, using the "#align" assembler directive. The processor model may be specified using the "#model" directive.

Conditional jumps are always relative, just to avoid the alignment problem. Like many other processors, these jumps can be made to near positions (in a range of ±128 instructions, more or less), and the typical conditions are the usual ones: 'eq', 'ne', 'gt', 'ge', 'lt', 'le', 'v' and 'nv'. The same conditions can be used to make conditional allocations, as happens in "rN = rN; iflt rN = –rN", that calculates the absolute value of a number kept in a register.

This processor only use the direct addressing mode to read (rD = pN) and to write (pN = rS) on external ports. Indirect addressing is available using external pointers but they are not usually needed in power electronics applications, where only a reduced set of variables is managed and algorithms are executed regularly over the same data set. Physical accesses are synchronous and they are executed in only one clock cycle. This does not mean any limitation either, because external resources are implemented with the same technology, in the same chip, and therefore they can work at the same clock frequency.

Instructions that make products, additions, subtractions and logic operations, have a regular structure that address two operands (any register except 'r0') and a destination register (any one). As said before, the possibilities of the instruction set have been extended allowing the cancellation of the first operand ('rS') and the substitution of the second ('rT') by a 16-bit constant ('K'). In this way, immediate logic masks can be applied to (rD = rS and/or K), the content of a register can be inverted (rD = 0 nor rT) and any register can be initialized with a constant (rD = 0 + K).

The most important operation of this processor, and its reason of being, is the fixed-point product, with or without accumulation. In general, two 16-bit normalized[1] operands are taken and a 24-bit normalized result is produced. All accumulations, additions and subtractions are made with a resolution of 24 bits.

Another kind of product is also available, represented as "rD = rS x rT", which allows to make displacements. The second operand is interpreted in <8.8> format, so that any value can be multiplied by 1/128, 1/64..., 1/4, 1/2, 2, 4..., 32, 64, in addition to other intermediate values, resulting in the desired adjustment.

Other typical instructions have been implemented with "macros" recognized by the assembly language. The 'nop' instruction, that does nothing, is replaced by "r1 = r1 or r1" and the newly created 'break' instruction, which allows setting simulator break-points, is replaced by "r1 = r1 and r1", doing nothing too.

4. The instruction cycle

As indicated above, the internal architecture of this processor is RISC like. That means that it executes all of its instructions regularly, in four clock cycles particularly. Although, from the user's point of view, all instructions are executed in four cycles, certain overlapping between instructions actually exists, and in fact many instructions are finished when already the following one is being executed. In any case, only one latency effect must be considered, and it will be explained later.

All instructions begin reading their 16-bit operation code from the program memory, dedicating to this function two clock cycles. After that, they use other two cycles to read the operands, 'rS' first and then 'rT'. The latter is replaced by a constant read from the program memory if 'r0' is referenced. Finally, in the first clock cycle of the next instruction, the required operation is made and the result is stored in 'rD'.

1) It sends the PC to the program memory.
2) It gathers the operation code on IR.
3) The rS register is read to an 'ACC' register.
4) 'RegT' and 'RegS' get the rT and ACC values.
1') The processor executes rD = RegS op RegT.

The program counter 'PC' is increased during the phase '2' and, if a constant is read from the memory, also during the phase '4'. If the operation requires a jump ('call', 'ret', 'jpFlag'), the program counter is modified during the phase '4'.

Using this simple scheme all instruction except products can be executed. As we'll see in 5.2 section, the internal multiplier of this processor requires four clock

[1] Normalized values are always in the [-1,+1) range. When 16 bits are used to represent these values the highest value can be 0.99996948 and its format is named <1.15>: one integer bit, which contains the sign, and fifteen fractional bits.

cycles to complete its operation, thus requiring a much smaller size[2] and it does not deteriorate the processor working frequency. In order to carry out the products, with or without accumulation, four clock cycles of the next instruction and another two ones from the following one are used:

1) It sends the PC to the program memory.
2) It gathers the operation code on IR.
3) The rS register is read to an 'ACC' register.
4) 'RegT' and 'RegS' get the rT and ACC values.
1') First stage of the MAC using RegS and RegT.
2') Second stage of the MAC using RegS and RegT.
3') Third stage of the MAC using RegS and RegT.
4') Fourth stage of the MAC using RegS and RegT.
1") One cycle because of multiplier segmentation.
2") Reads 'rD' and accumulates the MAC result.

In this way a *latency* is introduced and programmers ought to consider it: the result of a product is not available for general use in the immediately following instruction, but in the later one. However, when it is used for accumulation the latency is avoided, because that operation is carried out a cycle later too. This issue can be understand with the following example:

```
r1 = 0.27          // A value is allocated on r1
r2 = r1 + 0.32     // Right done, because r1 is available
r3 = r1 * r2       // Correct product between r1 and r2
r3 = r3 + r2 * r2  // This use of r3 is correct
nop                // We must wait for the result of r3
p4 = r3        // Now, but not before, r3 is r1*r2+r2*r2
```

Therefore, when the result of a product must be used, except when it is made to accumulate on a register, it is necessary to add a 'nop' or another instruction after the multiplication, in order to give time to the result to be calculated. This one is the only latency that introduces the segmented architecture of this processor.

5. Internal architecture

The DSPuva16 has been designed using Verilog and its based, as schematically shown in figure 2, on a RISC architecture that uses two 24-bit buses, one for operands and another one for results.

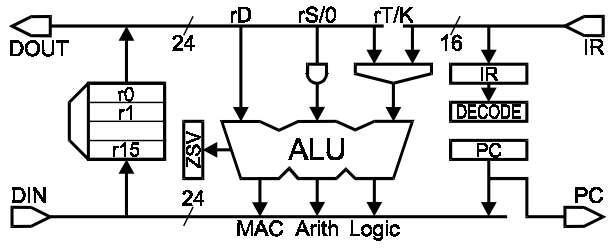

Figure 2. Internal architecture of the DSPuva16.

The program counter ('PC') has 8~12 bits, depending on the chosen processor model. The operation codes are received through 'IR' and they always have 16 bits. Using the registered value of IR a segmented instruction decoder activates each part of the circuit until completing each operation.

The bank of registers, that maintain the values of 'r0' to 'r15', is a 16x24 memory with synchronous writings and asynchronous reading [4]. It requires only 24 basic cells in some FPGA devices, which is equivalent to 5% of all the processor.

The ALU is built using three different units that realize logic operations, arithmetic ones and products. After each instruction, the zero ('Z'), sign ('S') and overflow ('V') flags are updated, which allows carrying out the corresponding allocations and conditional jumps. It must be pointed out that this processor does not need the carry flag ('C'), because it is not necessary for single precision operations.

Two different buses are used to interchange data with external elements, one to send data and another one to receive them. This method avoids additional tri-state buffers that, indeed, are not necessary within a chip.

5.1. Processor based on two internal buses

Many RISC processors have three or more buses at the moment, which even allow to realize all the instruction operations in only one clock cycle.

Since this processor needs four clock cycles to complete each product, and that the use of a third bus would lead to use dual-port memories for the bank of registers[3], it has been chosen to dedicate only one bus for the operands. Through this bus it can be seen the values of 'rD' during phases '1' and '2' and then it is left for 'rS' and 'rT' during the last two phases of each instruction cycle.

The bus for results is dedicated to collect the output of all ALUs and other sources: the input of external data when reading from a port and the value of the program counter when a subroutine is called, to keep the returning address in a register.

5.2. Four step multiplier

The central operation of this processor is the multiplication. It operates on two 16-bit values and generates a 32-bit result. Only 24 bits are available finally. In general the operands are in <1.15> format and the result is in <1.23> format.

In order to build the multiplier it has been decided to divide the operation in four stages[4], multiplying in each step a 16-bit operand by another one of only four bits,

[2] As the calculation of products is divided in four stages the circuit can be reduced to a quarter, approximately. Even so, the multiplier needs about 250 basic cells, that is a half of the processor.

[3] These memories allow two simultaneous asynchronous readings and one synchronous write when the clock cycle is finishing, but they occupy twice the space of the memories used by this processor.

[4] Not all FPGA devices do have embedded multipliers.

emitting a 20-bit intermediate result. This operation can be made with only four adders and an intermediate segmentation register, as shown simplified in figure 3. If we had tried to operate in a single clock cycle we would have needed 15 adders and the latency would have been greater, because of the segmentation registers.

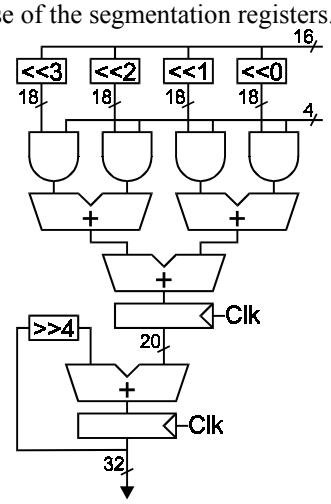

Figure 3. Segmented structure of the multiplier.

6. Development environment

The integrated development environment (IDE) of this processor fully covers its needs, at least while it is applied to the control of power electronics converters. Typical programs executed by the DSPuva16 processor usually have between 50 and 1.000 instructions, which correspond with typical working cycles of 5 to 100 microsecond. Therefore, programming in assembly language is sufficient, and the syntax used by the instructions is quite comfortable, as it has possibly been appreciated above.

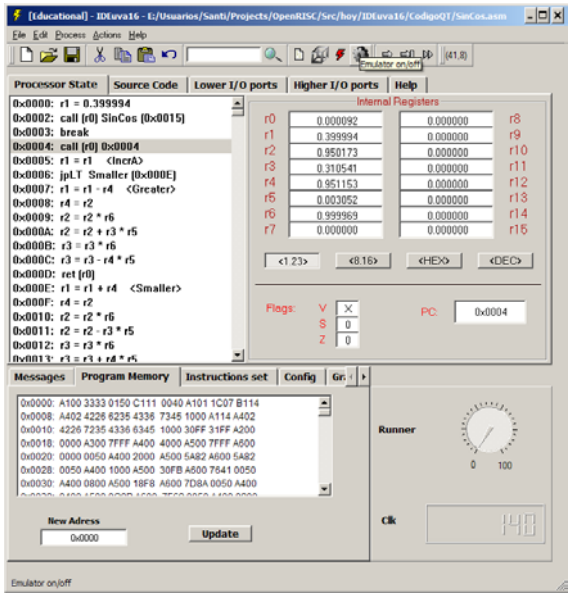

Figure 4. The IDE for the DSPuva16.

The IDEuva16 program is executed in graphics mode[5] and it incorporates a simple text editor, a complete C-like assembler with preprocessor, a simulator and a connection to the physical processor to control it in emulator mode using a suitable interface. Figure 4 displays the result of a small simulation in which the processor calculates "r2 = sin(r1); r3 = cos(r1)" for "r1 = 0.4·p". It needs 14.0 microseconds to complete this operation using a Cordic technique and working at 40 MHz (10 MIPS).

After editing and saving the assembler source code, where directives such as "#include", "#define" and "#ifdef" can be used, the source may be assembled by pressing a button. The process takes few seconds. Then, a step-by-step simulation or a simulation until a 'break' instruction can be done. Thousands or even hundreds of thousands of instructions can be executed in a single step, because simulation times are usually several or tens of milliseconds. The completion of each process on a typical DC/AC application takes few minutes using a 3 GHz computer. During simulations the state of the processor can be seen and intermediate results can be graphically displayed in a dynamic window.

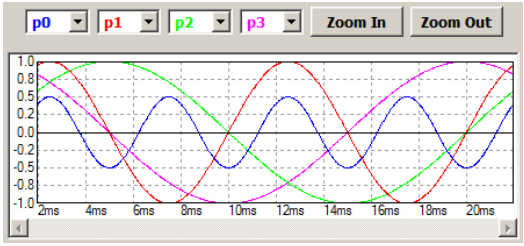

Figure 5. Graphics output of simulations.

When algorithms and assembler codes are stable enough, designers can use the "emulator mode": a real DSPuva16 is synthesized and implemented on an FPGA that is connected to the computer through the parallel port in SPP or EPP mode. The own IDE transfers to the DSP's program memory the result of the assembly, and following its state of reset/run is controlled. When the user stops the processor, the IDE automatically captures the intermediate or final results emitted by the processor through several of its ports, displaying them on the screen of the computer, as can be seen on figure 6.

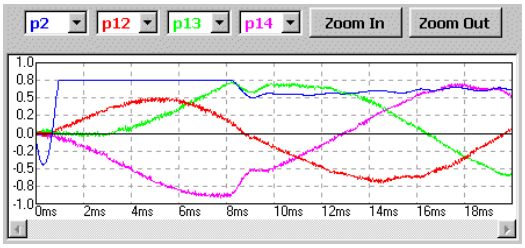

Figure 6. Graphics output of emulations.

[5] Currently English and Spanish versions on Win95/98/2K/XP are available. A Linux version will be released very soon.

The main difference between the simulation and the emulation is that the former is made completely in the computer, whereas the latter is executed in real or almost real time[6] on the physical equipment, implemented at the moment in a Xilinx Spartan-II FPGA[7].

7. Multi-processor systems

The main advantage of this DSP is its reduced size: it only needs about 12% of a 200K-gates FPGA. That means that, although a DSP alone cannot make the entire control task, is not too expensive to add others until the desired performance is reached. In a photovoltaic power application that is under development, and where this DSP is used for control, it is necessary to synchronize the equipment with the mains and to regulate the active and reactive powers; this tasks are covered by a first DSP with a working cycle of 25 microseconds. It is also necessary to control every 5 microseconds the DC/AC output currents and the commutation frequency of the equipment; for this task we have synthesized a second DSP. At the moment, the behaviour of the electrical circuit has been emulated every microsecond by a third DSP that operates in 60 microseconds. All this digital circuit requires about 100K gates (50%) of the FPGA.

For the communication between processors in this applications we use dual port synchronous memories [4]: a DSP can read and write at any time using one port, while another DSP can read, but not write, using the other one. For them, all accesses are as we have seen: direct accesses to ports. Using these memories, which occupy very little space in an FPGA, it is not necessary to worry about conflicts when sharing resources: both processors can use the common memory at any time. It is not necessary to synchronize transfers or signalling them either, because the bandwidth of involved magnitudes is far below the working cycle of these processors, reason why it does not matter that read values by the destination processor correspond with present values or previous ones, emitted by another DSP.

8. Conclusions

This article has shown how the DSPuva16 processor is and how, in spite of its limited features, it can easily solve complex problems of control in real time, simply adding as many processors as needed, but always within a chip, usually a FPGA.

Its main limitation is that it operates with few data and using fixed-point values, but since we have seen these topics do not mean any problem in the control of power electronic converters, because all physical magnitudes are easy to normalize and it is not difficult to keep them in a safe range, except during failure situations that cause the shutdown of the equipment and the stop of control.

[6] Before beginning with the power tests on the real converter its better to physically prove the control of DSPs using an emulated electrical plant. This task can be carried out using another DSP, which calculates how the electrical circuit would behave. Nevertheless, its work cycle is usually much greater than the others, which forces to add additional delays at the others so that they must wait, loosing the pure real time characteristics. Anyway, FPGA emulation finishes in seconds what PC simulation takes several minutes [1].

[7] The results of this article have been proven on a Xilinx XC2S200-PQ208 working at 40 MHz.

9. Acknowledgments

Our thanks to Carmen Cascón [3] and Juan del Barrio [1], who work hard to develop the IDE of this processor and contribute to the first application of this DSP in the control of a photovoltaic power generation system. Thanks also to Le Duc Hung, from the University of Natural Sciences of Ho Chi Minh City (Vietnam) for his comments and improvements on this processor.

References

[1] J. del Barrio, *Desarrollo sobre FPGA de un Emulador de una Planta de Microgeneración Eléctrica*, Final Project at the ETSII, University of Valladolid, Spain, 2004.

[2] B. K. Bose, *Power Electronics and AC drives*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[3] C. Cascón, *Diseño de un entorno de desarrollo para un DSP en coma fija de 16/24 bits integrado en FPGA*, Final Project at the ETSII, University of Valladolid, Spain, 2003.

[4] S. K. Knapp, "XC4000 Series Edge-Triggered and Dual-Port RAM Capability", Xilinx XAPP065, 1996.

[5] B. Norris, *Electronic Power Control and Digital Techniques* (Texas Instruments Electronics Series), McGraw-Hill, 1976.

[6] A. B. Rey, *Control digital vectorial con sliding en fuente de corriente para convertidores CC/CA trifásicos conectados a red*, Ph.D. Thesis, University of Valladolid, Spain, 2000.

[7] J. Schaefer, *Rectifier Circuits: Theory and Design*, John Wiley & Sons, Inc., Library of Congress Catalog Card Number 65-12703, 1965.