UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/907

DOI: https://doi.org/10.14483/22487638.12976

INVESTIGACIÓN

8-bit softcore microprocessor with dual accumulator designed to be used in FPGA

Microprocesador softcore de 8 bits con doble acumulador diseñado para ser usado en FPGA

William Sáenz Rodríguez[1], Fernando Rivera Sánchez[2], Fernando Martínez Santa[3]

Fecha de recepción: 3 de octubre de 2017 **Fecha de aceptación:** 23 de febrero de 2018

Cómo citar: Sáenz R., W., Rivera S., F.y Martínez S., F. (2018). Dual-accumulator softcore 8-bit microprocessor designed to be used on FPGAs. *Revista Tecnura, 22*(56), 40-50. DOI: https://doi.org/10.14483/22487638.12976

Abstract

Context: This paper is presents the design and implementation of an 8-bit softcore RISC microprocessor able to be run on space-optimized FPGA, in order to be used for embedded applications.

Method: The design of this microprocessor was developed in Verilog hardware description language and can be implemented in FPGA from different manufacturers; therefore, the user has only to define the input and output ports according to the type of FPGA. This is an accumulator-type processor, but it has two different accumulators that can be used as pointers for indirect addressing. The processor is Harvard with a RAM of 8x256 bits, and a ROM that can be resized from 17x252 bits to 17x8K bits. Additionally, it has one 8-bit input port, one 8-bit output port, and one 8-bit address port, which means that the processor can address more than 256 8-bit output and input ports/devices.

Results: The developed processor, named "ZASUA," was compared with PICOBLAZE softcore and other three similar processors of free distribution in the Web, and some improvements over those were found. Criteria such as the Flip Flops used, occupied LUTs, Slices in use, and maximum delay of each processor were analyzed, all these results were obtained from the implementation of the processors in the Xilinx FPGAs.

Conclusions: The designed architecture is composed by two accumulators, which can be used either as source or destination for the operation of the ALU. This fact gives some flexibility to the design, doing it better than a single-accumulator processor, and getting it closer to the register-based processors.

Keywords: Embedded microprocessor, Harvard Architecture, RISC, Softcore, FPGA, Verilog, Dual Accumulator.

Resumen

Contexto: Se diseñó e implementó un microprocesador softcore RISC de 8 bits para que funcionara sobre dispositivos FPGA, y que estuviera optimizado en espacio con el fin de usarlo en aplicaciones embebidas.

Método: El diseño de este microprocesador se desarrolló en el lenguaje de descripción de *hardware* Verilog, y puede ser implementado en FPGA de

1 Ingeniero Electricista, ElectroSanchez & Cía. Ingeniería Eléctrica S.A.S., ingeniero de diseño. Bogotá, Colombia. Contacto: proyectos1@electrosanchez.com.co

2 Ingeniero Electricista, Alcaldía de Soacha, Ingeniero Alumbrado Público. Bogotá, Colombia. Contacto: hrivera.cto@alcaldiasoacha.gov.co

3 Magíster en Ingeniería Electrónica y de Computadores, ingeniero en Control Electrónico e Instrumentación. Profesor asistente Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. Contacto: fmartinezs@udistrital.edu.co

diferentes fabricantes, de tal forma que el usuario solo tenga que definir los puertos de entada y de salida, según el FPGA utilizado. El procesador desarrollado es de tipo acumulador, pero tiene dos diferentes acumuladores que pueden ser usados como apuntador para direccionamiento indirecto. El procesador es Harvard, con una RAM de 8x256 bits y una ROM que puede ser redimensionada desde 17x252 bits hasta 17x8K bits. También, tiene un puerto de entrada de 8 bits, uno de salida de 8 bits y otro de direcciones de 8 bits, lo que significa que puede direccionar hasta 256x8 bits puertos o dispositivos de salida y la misma cantidad de entrada.

Resultados: El procesador, denominado *ZA-SUA*, fue comparado con el *softcore Picoblaze* y con otros tres procesadores similares de libre distribución en la Web, y se alcanzaron algunas mejoras sobre ellos. Se analizaron criterios como número de *flip flops* usados, LUT ocupadas, *slices* en uso y retardo máximo de cada procesador, todos estos resultados fueron obtenidos de la implementación de los procesadores en FPGA de Xilinx.

Conclusiones: La arquitectura diseñada está compuesta por dos acumuladores, los cuales pueden ser usados como fuente o destino de las operaciones de la ALU. Este hecho da cierta flexibilidad al diseño, haciéndolo mejor que un procesador con un solo acumulador, y acercándolo más a un procesador basado en registros.

Palabras clave: procesador embebido, arquitectura Harvard, RISC, *softcore*, FPGA, Verilog, acumulador dual.

INTRODUCTION

At present time, more than 95% of the electronic chips produced are used for embedded systems (Narayanan and Xie 2006). Most electronic devices that surround us such as televisions, radios, cars, and aircraft, among others, contain embedded systems. In general, embedded systems are characterized by being subject to size requirements, having low power consumption, and being economically cheap. Embedded systems have a hardware and software part, which are used to create specific applications (Henzinger and Sifakis 2006; Plavec 2004). In the market you can get embedded systems, such as: FPGAs, CPLDs, microcontrollers, microprocessors, DSP, and others; and all these electronic devices are designed to be programmed by the user. FPGAs and CPLD have shown a huge flexibility for designing custom applications (Garzón, Bareño, and Jacinto 2010; Gómez, Plazas, and Restrepo 2015; Martínez Sarmiento and Giral Ramírez 2017; Riaño, Ladino, and Martínez 2012).

The creation of FPGAs and CPLDs started the term "soft-core processor," which consists of writing a processor in a hardware description language (HDL), and can be adapted to fulfill a certain function. These types of processors offer several advantages, such as reducing the cost, improving flexibility, and more immunity to the obsolescence (Tong, Anderson, and Khalid 2006).

In FPGAs, any type of processor can be implemented regardless of the type of architecture and instructions, whether it is CISC (Complex Instruction Set Computer) (Appel and George 2001) or RISC (Reduced Instruction Set computer) (Hu et al. 2009). On the other hand, these architectures can be implemented according to the distribution of memory: they can be Harvard (Trivedi and Tripathi 2015) or Von Newman (Pastor and Sánchez 1997). The RISC processors have been implemented as Design and Performance Analysis of 8-bit RISC Processor using Xilinx Tool (Uma 2012), designing a low power 8-bit Application Specific Processor (Samal and Samal 2014), and FPGA Implementation of MIPS RISC Processor (Kelgaonkar and Kodgire n.d.), FPGA Implementation of an 8-bit Simple Processor (Ayeh et al. 2008), Asynchronous 8-Bit Processor Mapped into an FPGA Device (Herrera and Viveros 2014). Regarding CISC processors, the design of an 8-bit CISC CPU based on FPGA has

been developed (Zhang and Bao 2011). It is important to note that only the number of logical gates the developed processor occupies determines the capacity of the FPGA to be implemented.

Today several soft-core 8-bit processors have been developed as mentioned above, and there are 8-bit processors that have public or commercial domain, such as Picoblaze (Xilinx 2011), V8-uRISC, and Free-RISC8 (Santana Hernandez 2004), among others. There is limited information that can be found on the Internet on free code of softcore processors with enough information to reproduce them. Besides, they are fully functional and can be implemented in any FPGA without importing the manufacturer.

One of the best known 8-bit softcore processors with reproducible code is PICOBLAZE. Its documentation can be downloaded from the Xilinx Website. This processor was developed in a high-level language, which can only be run on Xilinx devices, and it is developed in Verilog and VHDL. It contains 16 data records, 64-position data memory, 8-bit ALU, and has 1Kbits ROM (Xilinx 2011).

Another processor is the Free-RISC8, which is a model designed in Verilog synthesizable in a simple 8-bit microcontroller and is compatible with the code of the controller 16C57 Microchip company. It has a variety of software development tools making this processor attractive for educational purposes or even to use in an FPGA.

Regarding the V8-uRISC, it is a general-purpose processor designed and optimized specifically for programmable logic. It combines a small number of gates with execution to a single cycle of clock for many instructions; its objective is to deliver high performance of the 8-bit microprocessor while occupying very little space in its implementation. It was developed in VHDL and Verilog (Hays and Jshamlet 2016).

A more general solution for open source processors is the OpenCores.org Web page. It is the world's largest community site for the development of open source hardware cores. OpenCores.org takes the source code for different digital projects and supports users with different tools, platforms, forums, and other useful information. The drawback with this information is that most projects are not fully functional and therefore cannot be reproduced. Or sometimes they are reproducible but do not present the respective documentation to be understood.

The following is the design and results of the implementation of an 8-bit softcore processor optimized in size, which work with devices of many manufacturers, and will be published in the Web for free: the code will be left open with the respective guide information so that it can be reproduced and modified by any user. (Clayton 2014; Crabtree 2009; Guzman 2012; Hays and Jshamlet 2016; Riedel 2009).

Methodology

General Description

The softcore 8-bit processor ZA-SUA is a Harvard RISC processor, featuring a new dual-accumulator design; This processor contains 28 instructions which can be used to perform direct, indirect, or immediate addressing; it also has the ability to develop external interruption. ZA-SUA means "night or day," referring to both accumulators used in the design, which can be used as simple accumulators or as index registers indifferently. The name ZA-SUA was taken from the dead language Muysc cubun, in homage to the Muisca (usually called Chibcha) indigenous community of the central region of Colombia.

The double accumulator is represented by the letters A and B (See Figure 1), which allows to save the results that come from the Arithmetic Logic Unit (ALU) or the reading data of a port. They allow to select the operand of the ALU if A or B is desired, and these two accumulators can also be used like pointers for instructions with indirect addressing.

On the other hand, the instruction encoding is 17 bits length, which is stored in the instruction

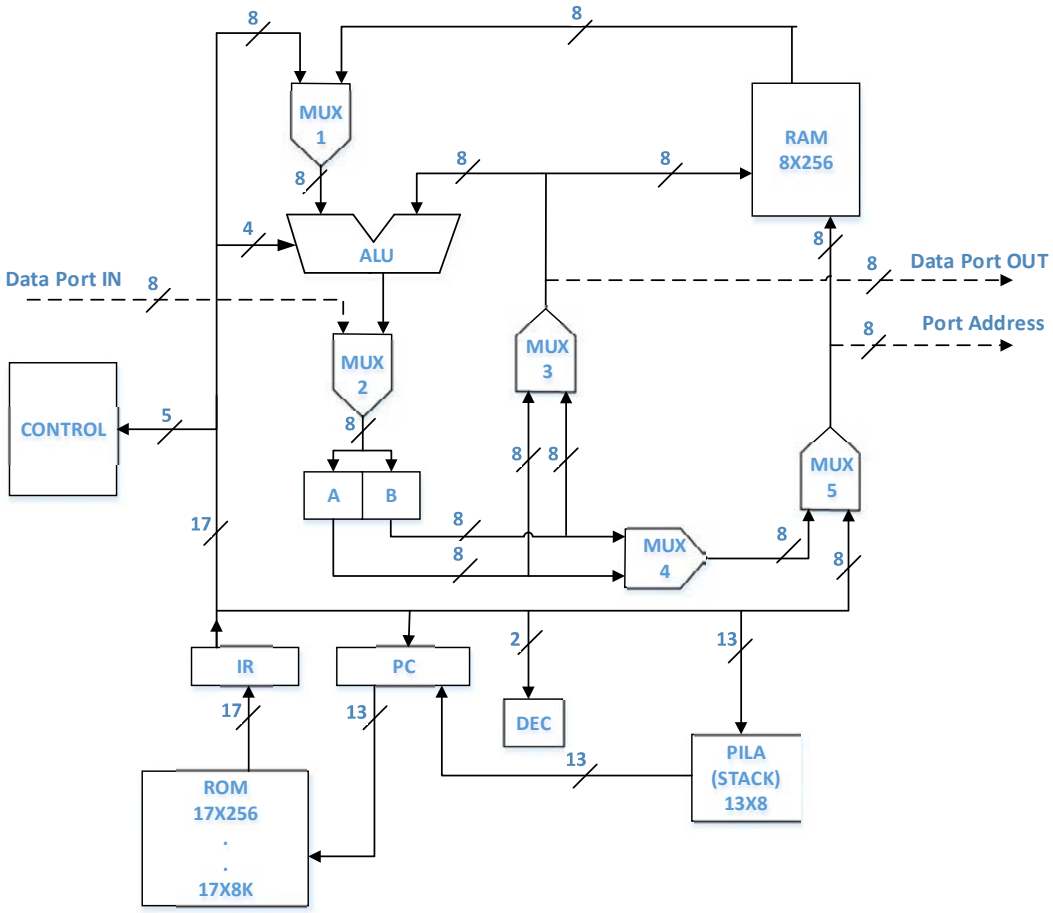

Figure 1. Block diagram of the ZA-SUA 8-bit soft-core processor

Source: own work.

register (IR-Instruction Register); the ROM (Read Only Memory) is scalable and can be varied from 256, 512, 1024, 2048, 4096 and 8192 positions; it also contains a stack (STACK) of 13 bits and 8 positions. Finally, the RAM (Random Access Memory) is 8-bits length and has 256 positions.

The "DEC" is a decoder to control three "MUX" multiplexers, which are: MUX 1, MUX 4 and MUX 5. The others are controlled by the IR. The "PC" (Program Counter) is the program counter. The continuous cables shown in Figure 1 are the internal connections of the processor, whereas non-continuous cables are some of the main input and output ports. The MUX 1 performs the function of letting the instruction register data pass if an instruction that handles an immediate or literal addressing is activated and if it does not let the read data of the RAM pass; this occurs when an instruction has an address direct. With respect to MUX 4, it works when an instruction has an indirect addressing and it discovers with which it wants to develop it with A or B. MUX 5 is used to let the output data of the multiplexer 4 (MUX 4) pass if there is indirect addressing; if it does not, the multiplexer lets the data of the instruction register go through to develop a task directly. MUX 2 selects whether to store the ALU or the value that is present on the processor input port. If a task is performed with the ALU using the MUX 3, the operand is either A or B. Regarding the ALU, it has 16 instructions, which are linked to variables of ZERO and CARRY; this allows the accumulation and storage of data in the operations it performs.

Instruction Set

The processor has a total of 28 instructions. Table 1 shows the instructions that the ALU handles. A brief description is given to explain what each instruction does, its respective encoding, and mnemonic. Also, if the instruction affects carry (C) and zero (Z), it is represented with '1', and if it does not modify, it is represented with '0'. Table 2 shows the instructions not handled by the ALU, a brief description of them, and their respective coding and mnemonic.

Coding instructions

The instruction encoding is 17 bits (detailed in Table 3). The most significant bits are those containing the instruction, the following 8 bits of "General", point to a Constant, a RAM address, or a port address, which vary according to the type of instruction. The 2 bits of "Address" refer to the type of address that has the instruction. The "Source" bit selects the value of the accumulator A or B with which you want to develop an operation of the ALU or write a data in a position of the RAM. The least significant bit is the "Destination", which is where you want to save the data, which can be stored in the accumulator A or in the accumulator B. The instruction encoding is defined for all instructions as indicated in Table 3, minus the jump instructions. JFIZ, JFIC, JUMP and CALL are organized as shown in Table 4. As evidenced by the 4 most significant bits is the instruction and the least significant is the number that replaces increasing or decrementing the PC, which refers to the number of lines you want to skip. Table 5 shows other instructions and the bits needed to define them, as well as the not needed bits, which are crossed out. The OUTPUT and INPUT instructions can support 256 input and output ports; this is used to connect different modules in the ZASUA processor.

Table 1. ALU Instructions

Description	Mnemonic	Coding	Affects	
			Z	C
Addition	ADD	00000	1	1
Addition with carry	ADDC	00001	1	1
Subtraction	SUB	00010	1	1
Subtraction with carry	SUBC	00011	1	1
Increment	INC	00100	1	1
Decrement	DEC	00101	1	1
Shift left (without carry)	SHL	00110	1	0
Shift right (without carry)	SHR	00111	1	0
Rotate left (with carry)	ROL	01000	1	1
Rotate right (with carry)	ROR	01001	1	1
Logic operations	AND	01010	1	0
	OR	01011	1	0
	XOR	01100	1	0
	NOT	01101	1	0
Load a value in an accumulator	LOAD	01110	1	0
Move values between accumulators	MOVE	01111	1	0

Source: own work.

Table 2. Non-ALU Instructions

Description	Mnemonic	Coding
Jump if Zero flag is true	JIFZ	1000
Jump if Carry flag is true	JIFC	1001
Unconditional jump	JUMP	1010
Subroutine calling	CALL	1011
Stores an accumulator in RAM	STORE	11000
Subroutine return	RETURN	11001
Port reading	INPUT	11010
Port writing	OUTPUT	11011
Enable interrupts	EINT	11100
Disable interrupts	DINT	11101
Interruption return	RETI	11110
Relative jump	JUMPR	11111

Source: own work.

Table 3. Instructions coding

Instructions Coding				
5 bits	8 bits	2 bits	1 bit	1 bit
Instruction	General	Addressing	Source	Destination

Source: own work.

Table 4. Jump Instruction coding

Jump Instruction coding	
4 bits	13 bits
Instruction	Absolute program memory address (PC)

Source: own work.

Table 5. Coding for all the instructions

ALU				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	Addressing	Source	Destination
JIFZ, JIFC, JUMP y CALL				
4 bits	*13 bits*			
Instruction	Absolute program memory address (PC)			
OUTPUT				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	Addressing	Source	Destination
STORE				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	Addressing	Source	Destination
RETURN y RETI				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	Addressing	~~Source~~	Destination
INPUT				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	Addressing	Source	Destination
JUMPR				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	~~Addressing~~	Source	Destination
EINT y DINT				
5 bits	*8 bits*	*2 bits*	*1 bit*	*1 bit*
Instruction	General	Addressing	Source	~~Destination~~

Source: own work

Addressing modes

The 8-bit soft-core processor has 3 addressing modes: direct, indirect, and immediate. The following are the modes of addressing, associated with their correspondent instructions:

- Indirect addressing: the instructions that develop it are STORE, OUTPUT, and all the ALU instructions.
- Immediate addressing: they are developed by the instructions found in the ALU and by the instructions RETI and RETURN.
- Direct addressing: the instructions that develop it are STORE, INPUT, OUTPUT, and all the instructions handled by the ALU.

The 2 bits of "Address" are decoded as shown in Table 6. As for the coding of "Source" and "Destination", both of them present a homogeneous organization: the 0 directs to the accumulator A and the 1 directs to the accumulator B.

Table 6. Addressing bits coding

Coding	Description
00	Direct addressing
01	Immediate addressing
10	Indirect addressing with the accumulator A as index
11	Indirect addressing with the accumulator B as index

Source: own work.

Processor state machine

Figure 2 shows the state machine of the Softcore 8-bit Processor "ZA-SUA". The first state is the RESET. In each clock cycle it is checked to know if it is active; if it is, reset all the main registers as: program counter, stack address counter, and the accumulators. In the state of SEARCH, the instruction to be executed is read. In DECODE, each instruction is decoded to enable and disable the processes that need to be executed for each instruction. INSTRUCTIONS state executes the instruction

process. The processes that are enabled and disabled during the DECODE state must remain active during the INSTRUCTIONS state because there is a delay in the clock cycle.

The delay occurs because the processor executes tasks in parallel in each cycle of clock. If a process like saving a value in the pile is activated in the first cycle of a clock, this process cannot be executed. This happens because the pile was not active at the beginning of the first clock cycle, it will only be executed in the second clock cycle if this process is still active during this clock cycle, for this reason each task activated or deactivated has to be enabled the state before it will be used.

In the INSTRUCTIONS state (see Figure 2) the following instructions are executed: all ALU's, JFIZ, JFIC, JUMP, CALL, STORE, RETURN, INPUT, OUTPUT, EINT, DINT, RETI and JUMPR.

In the DECODE state (Figure 2) three lines emerge. These represent the thirteen states mentioned above (for practical purposes).

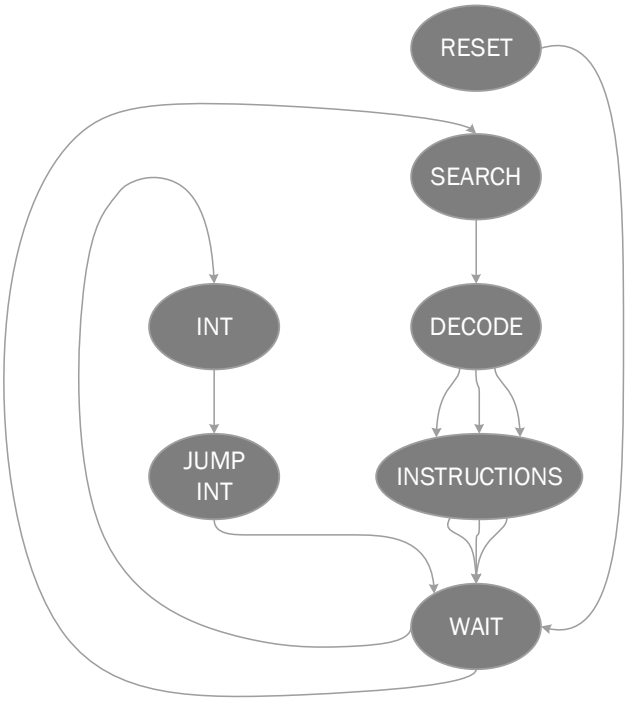

Figure 2. Finite state machine of the control unit

Source: own work.

It should be noted that the thirteen states emerge from the INSTRUCTIONS state but converge in the WAIT state (Figure 2) since it is mandatory that these states pass through the state to initiate the next state.

As noted in the above listing, all ALU instructions (Table 1) are controlled by this state. It is worth mentioning that it is not necessary to develop a decoder to select the instructions presented by the ALU since the instructions that arrive at DECODE are the same delivered by it at the end, for which it does not perform decoding.

The WAIT state is the state of waiting for the instructions to be executed, which enables the reading of the following instruction (ROM). After exiting the WAIT state, it is checked whether or not there was an interruption (INT). If there is no interruption, the program will go to the SEARCH state. On the contrary, if an interruption occurs, the task of storing the program counter value in the stack is enabled during this state. In the JUMP INT state, it jumps to the memory location of the ROM, where the tasks that are performed during the interruption started.

In order for the interruption to develop (INT), it must have passed the state of EINT and will only be executed when the instruction is finished. During this process the CARRY is temporarily archived. If the user wishes to save the value of the accumulators, he must store them at the beginning of the interruption (INT) at the desired RAM position. When the RETI instruction is executed, the CARRY returns to the value it had prior the interrupt.

JUMPR is used to perform relative jumps for selecting a value from a records table, which is designed at will of the user, who will have to use the RETURN instruction to achieve implementation. Each state is executed in a clock cycle, all internal processor instructions are developed in 4 clock cycles, except the external ones as shown in Figure 2, the RESET is executed in two clock cycles and the interrupt (INT) is executed in 3 clock cycles.

RESULTS

Comparison of Technical Criteria

To make an appropriate comparison, five criteria were chosen for four processors: one is PICOBLAZE and the other three were taken from the OpenCores.org page (see Table 7).

Comparative of the results

The following are the results obtained from the implementation of the "ZA-SUA" processor and the other chosen processors. It is noteworthy that the processors were implemented in a SPARTAN-3AN Starter Kit FPGA card, and the results below are obtained from the "Design Summary" of the Xilinx 14.7 software. On the other hand, the ROM, RAM, ALU, STACK, IR, and Control, in the processors PICOBLAZE, Tiny 8, and NATALIUS, are divided into blocks; while in the ZA-SUA and TISC processors all parts are integrated in a single code, which helps to give a better idea of the total space occupied by the processor. The information obtained in the tables described above is summarized below (Table 8).

It can be observed that the processor that occupies less resources is the TISC (in terms of flip-flops used), but it should be clarified that it does not have RAM and is the processor with less instructions.

Regarding LUTs (LOOKUP TABLE), or search tables used, it was found that the processor that uses less resources is the PICOBLAZE, using only 1.49% of the total of the available tables because it is written in a language of High level, which ensures that it can only be used in Xilinx devices.

By analyzing the occupied SLICES, it is found that the processor that occupies less resources is the PICOBLAZE again and the one with the most resources is the TINY8 (See Figure 3).

Table 7. Technical specifications of the compared processors

	ZA-SUA	PICOBLAZE[15]	NATALIUS[21]	TINY8[20]	TISC[19]
Bus size [Bits]	8	8	8	8	8
RAM [Bytes]	32	64	4	32	-
Instructions number	28	57	29	-	14
Bits per instruction	17	18	16	16	12
ROM [Words]	256-8K	1K	-	-	-

Source: own work.

Table 8. General comparative results

	Flip-Flops		LUTs		Slices		Frequency	
Processor	Used	[%]	Used	[%]	Used	[%]	Maximum delay [us]	Maximum freq. [MHz]
ZA-SUA	75	0,64	421	3,58	223	3,79	1076	929
PICOBLAZE	76	0,65	176	1,49	98	1,66	1220	817
NATALIUS	158	1,34	416	3,53	270	4,58	1130	888
TINY8	301	2,55	1493	12,67	823	13,97	1100	910
TISC	54	0,46	189	1,61	111	1,88	1070	936

Source: own work.

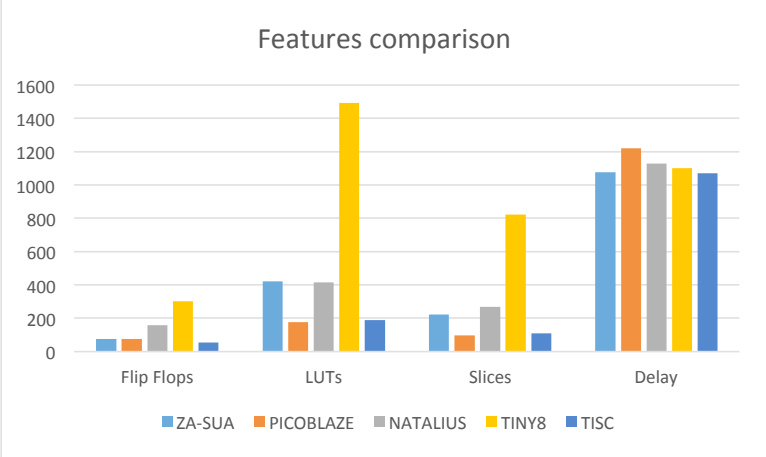

Figure 3. Size and performance criteria comparison among processors

Source: own work.

Finally, by comparing the maximum execution frequencies of the processor, it was obtained that the one with the highest performance is the TISC and, in contrast, the one with the lowest performance is the PICOBLAZE (Figure 3).

CONCLUSION

According to Figure 3, it can be concluded that the ZA-SUA processor occupies less flip-flops than NATALIUS and TINY8, but it presents almost the same number of flip-flops as the PICOBLAZE, which affirms that the ZA-SUA processor is optimized in size compared to these processors.

On the other hand, the ZA-SUA processor occupies less LUTs than TINY8, fewer SLICES than NATALIUS and TINY8, and has higher performance than the PICOBLAZE, NATALIUS, and TINY8 processors (see Figure 3). ZA-SUA guarantees the superiority in the design of the processor in comparison to these processors.

Although the results indicate that the PICOBLAZE processor has better performance in general terms, it can only be implemented in Xilinx devices; contrary to the ZA-SUA processor, which is more versatile because it can be reproduced in FPGAs from different manufacturers.

Although the TISC processor provides favorable results regarding the use of resources, it should be clarified that it has no RAM and has 14 instructions less than the ZA-SUA processor.

Finally, it is analyzed that the NATALIUS processor, although having an instruction more than the ZA-SUA, does not have INTERRUPT instruction, which limits the possibility of executing basic tasks as this is a fundamental requirement for the development.

Future work

In first instance, a C-language compiler is to be developed for the 8-bit softcore processor ZA-SUA, with the aim that the user needing the processor can program it easily without having to resort to the source code of the processor. Secondly, various applications will be implemented with the compiler to experiment with all available processor resources. Performance benchmarking software (Benchmark) will be performed to study the maximum capacities of the ZA-SUA processor. Finally, it is possible to modify the code so that the capacity of the processor can be increased to 16 bits.

References

Appel, Andrew W. and Lal George. 2001. "Optimal Spilling for CISC Machines with Few Registers." Pp. 243–53 in *ACM SIGPLAN Notices*, vol. 36. ACM.

Ayeh, E., K. Agbedanu, Y. Morita, O. Adamo, and P. Guturu. 2008. "FPGA Implementation of an 8-Bit Simple Processor." Pp. 1–5 in *Region 5 Conference, 2008 IEEE*. IEEE.

Clayton, Jhon. 2014. "risc16f84 :: Overview." *OpenCores*. Retrieved (http://opencores.org/project,risc16f84).

Crabtree, Vincent. 2009. "Tiny Instruction Set Computer :: Overview." *OpenCores*. Retrieved (https://opencores.org/project,tisc).

Garzón, Víctor Alonso Bravo, Jesús Jair Navarro Bareño, and Edwar Jacinto. 2010. "Diseño E Implementación de Un Codec Digital de Audio Con FPGA, En Formato PCM, de 2 Canales Con Interfaz Para Usuario." *Tecnura: Tecnología y Cultura Afirmando el Conocimiento* 14(26):56–68. Retrieved (http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/article/view/6687/8270).

Gómez, Edwar Jacinto, Donovan Camilo Platero Plazas, and Mario Fernando Robayo Restrepo. 2015. "Voltimetro True-Rms Sobre Fpga Basado En Algoritmo Cordic." *Revista Tecnura* 19:129–36. Retrieved (http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/article/view/9619/10827).

Guzman, Fabio. 2012. "Natalius 8 Bit RISC :: Overview." *OpenCores*. Retrieved (http://opencores.org/project,natalius_8bit_risc).

Hays, Kirk and Jshamlet. 2016. "Open8 uRISC :: Overview." *OpenCores*. Retrieved (http://opencores.org/project,open8_urisc).

Henzinger, Thomas A. and Joseph Sifakis. 2006. "The Embedded Systems Design Challenge." Pp. 1–15 in *International Symposium on Formal Methods*. Springer.

Herrera, Moises and Francisco Viveros. 2014. "Asynchronous 8-Bit Processor Mapped into an FPGA Device." Pp. 1–7 in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. IEEE.

Hu, Weiwu et al. 2009. "Godson-3: A Scalable Multicore RISC Processor with x86 Emulation." *IEEE micro* 29(2).

Kelgaonkar, Pranjali S. and Shilpa Kodgire. n.d. "Pipelined 32bit RISC MIPS Processor on Spartan-6 FPGA." *International Journal of Science, Engineering and Technology Research (IJSETR), ISSN* 2278–7798.

Martínez Sarmiento, Fredy Hernán and Diego Armando Giral Ramírez. 2017. "OpenRRArch: Una Arquitectura Abierta, Robusta Y Confiable Para El Control de Robots Autónomos." *tecnura* 21(51):96–104.

Narayanan, Vijaykrishnan and Yuan Xie. 2006. "Reliability Concerns in Embedded System Designs." *Computer* 39(1):118–20.

Pastor, Enric and Fermín Sánchez. 1997. "La Máquina Rudimentaria: Un Procesador Pedagógico." *III Jornadas de Enseñanza Universitaria sobre Informática (JENUI'97), Madrid, Spain* 395–402.

Plavec, Franjo. 2004. *Soft-Core Processor Design*. University of Toronto.

Riaño, José, César Ladino, and Fredy Martínez. 2012. "Implementación de La Transformada FFT Sobre Una FPGA Orientada a Su Aplicación En Convertidores Electrónicos de Potencia." *Tekhnê* 9:21–32. Retrieved (http://revistas.udistrital.edu.co/ojs/index.php/tekhne/article/view/8925/10297).

Riedel, Ulrich. 2009. "tiny8 :: Overview." *OpenCores*. Retrieved (http://opencores.org/project,tiny8).

Samal, Lopamudra and Chiranjibi Samal. 2014. "Designing a Low Power 8-Bit Application Specific Processor." Pp. 1–5 in *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on*. IEEE.

Santana Hernandez, Gladis Elizabeth. 2004. "Diseño de Un Procesador Usando Lenguajes de Descripción de Hardware." Instituto Politécnico Nacional, Mexico DF.

Tong, Jason G., Ian D. L. Anderson, and Mohammed A. S. Khalid. 2006. "Soft-Core Processors for Embedded Systems." Pp. 170–73 in *Microelectronics, 2006. ICM'06. International Conference on*. IEEE.

Trivedi, Priyanka and Rajan Prasad Tripathi. 2015. "Design & Analysis of 16 Bit RISC Processor Using Low Power Pipelining." Pp. 1294–97 in *Computing,*

Communication & Automation (ICCCA), 2015 International Conference on. IEEE.

Uma, R. 2012. "Design and Performance Analysis of 8–bit RISC Processor Using Xilinx Tool." *International Journal of Engineering Research and Applications* 2(2):53–58.

Xilinx. 2011. "PicoBlaze 8-Bit Embedded Microcontroller User Guide." *IP documentation* 1–120. Retrieved (http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf.).

Zhang, Yunjie and Lei Bao. 2011. "The Design of an 8-Bit CISC CPU Based on FPGA." Pp. 1–4 in *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. IEEE.