International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 6 (December, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Using the Xilinx PicoBlaze Soft Processor in FPGAs

Rustem Popa
Associate Professor, Department of Electronics and Telecommunications, Dunarea de Jos University, Galati, Romania
Rustem.Popa@ugal.ro

Abstract*: We have implemented in Spartan-3E FPGA an 8-bit microcontroller using PicoBlaze sources from Xilinx. A simple multiplier algorithm has been implemented in the memory of the microcontroller and then the whole circuit has been tested. Other implementations of the same algorithm in FPGA, without using PicoBlaze, have been compared with the first case. Although the hardware of the microcontroller uses some of the FPGA resources, they are relatively constant and only few memory locations are need for the implementation of the algorithm, less than any other solution discussed in this paper.

Keywords: FPGA, Spartan 3E, PicoBlaze, IP core, Verilog HDL, ISE, multiplier, microprogramming.

I. INTRODUCTION

The PicoBlaze processor is a compact 8-bit microcontroller core for Xilinx FPGAs. It is provided as a cell-level HDL description (both VHDL and Verilog) and can be synthesized along with any programmable logic. This description is known as "soft core" or "IP core", which means "Intellectual Property core". This 8-bit soft microcontroller IP has a very small footprint (a few tens of slices plus program memory). It can easily be integrated into a larger system and adds another dimension of flexibility in FPGA-based designs ([1], [2]).

In the next section we talk about the architecture of the PicoBlaze microcontroller and its functional blocks, including some considerations about processor instruction set. Section III describes the multiplier algorithm and its implementation in microcontroller. Section IV presents three different implementations of the same algorithm, without using a microcontroller: first of them uses a Verilog description of the multiplier ([3]), another solution is proposed in [4] and is based on the serial/parallel registers driven by a finite state machine, and the last solution uses microprogramming, a synthesis method described in [5]. Section V describes the results of our simulations to compare all these methods, and finally, Section VI contains the concluding remarks.

II. DESCRIPTION OF THE PICOBLAZE

As shown in the block diagram in Fig. 1, the PicoBlaze microcontroller contains 16 byte general-purpose data registers, an 8 bit Arithmetic and Logic Unit (ALU) with CARRY and ZERO indicator flags, 64-byte internal scratchpad RAM, 1K x 18 Instruction PROM which is automatically loaded during FPGA configuration, a 10 bit program counter, an automatic 31-location CALL/RETURN stack, and 256 input and 256 output ports.

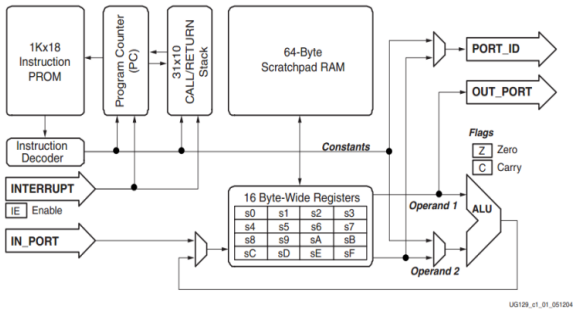

Fig. 1 PicoBlaze Embedded Microcontroller Block Diagram ([6])

There is no dedicated accumulator, all register operations are interchangeable and no registers are reserved for special tasks. ALU performs all microcontroller calculations, including basic arithmetic operations (such as addition and subtraction), logic operations (such as AND, OR, XOR), arithmetic compare and bitwise test operations, and comprehensive shift and rotate operations. ALU operations affect the ZERO and CARRY flags. Internal general-purpose 64-byte scratchpad RAM is addressable from the register file using the STORE and FETCH instructions. The Program Counter (PC) increments to the next instruction to be executed. Only some instructions, like JUMP, RETURN, CALL, RETURNI, and the Interrupt and Reset events modify its default behaviour. The CALL/RETURN hardware stack is implemented as a separate cyclic buffer and it stores up to 31 instruction addresses. The Input/Output ports extend the PicoBlaze microcontroller's capabilities and allow the microcontroller to connect to other FPGA logic. An optional INTERRUPT input allows to handle asynchronous external events ([6]).

As we can see in Fig. 2, this IP is described by 2 HDL modules: kcpsm3, which represents the processor, and prog_rom, which is 1K x 18 instruction memory.

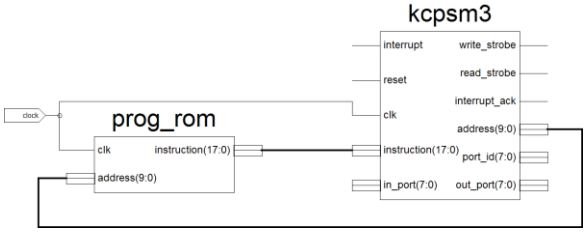

Fig. 2 Microcontroller PicoBlaze and Instruction PROM

KCPSM is the acronym from the Konstant Coded Programmable State Machine, the first name of the PicoBlaze, and the version 3 was optimized for Spartan-

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 6 (December, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

3 FPGAs ([2]). The PicoBlaze program ROM is used within a VHDL or Verilog design flow. The PicoBlaze assembler generates a VHDL or Verilog file in which a block RAM and its initial contents are defined.

All data processing instructions operate on any of the 16 general-purpose registers. Only the data processing instructions modify the ZERO or CARRY flags as suitable for the instruction. The data processing instructions consists of logic instructions, arithmetic instructions, test and compare instructions, and shift and rotate instructions. PicoBlaze instruction set with a complete functional description of each instruction is given in [6].

III. The Algorithm and Its Implementation

We have used, as an example, a multiplier for positive binary numbers, proposed in [5]. The two 4-bit numbers, multiplicand and multiplier, are stored in two registers, X and Y. The product register P is an 8-bit register and it contains the result of the multiplication. This register serves as an accumulator to accumulate the sum of the partial products. The counter I counts from 0 to 4. If an operand is multiplied by 2, then it is shifted to the left with one position. If the operand is divided by 2, then it is shifted to the right with one position. Notation [Y] signifies the integer part of the operand stored in register Y (Fig. 3).

This algorithm is implemented in the assembly language of the microcontroller PicoBlaze, using instructions from the instruction set of the microcontroller. This program is given below:

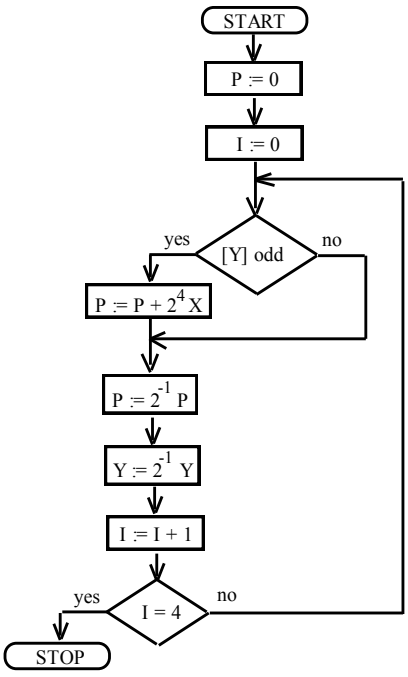

Fig. 3 The multiplier algorithm for two 4-bit numbers ([5])

```
; a multiplier for positive two 4-bit numbers
beginning:  load s3, 07      ;multiplicand
            output s3, 80
            load s4, 0f      ;multiplier
            output s4, 80;
            load s5, 00      ;product
            load s6, 00      ;counter
start:      test s4, 01      ;multiplier is even?
            jump z, next     ;if yes, then go to
next
            load s7, s3
            sl0 s7           ;multiply s3 with 2^4
            sl0 s7
            sl0 s7
            sl0 s7
            addcy s5, s7
next:       sr0 s5           ;product is divided
by 2
            sr0 s4           ;multiplier is divided
by 2
            add s6, 01       ;counter is
incremented
            compare s6, 04   ;compare with 4
            jump nz, start
            output s5, 80    ;display the product
            jump beginning
```

The two 4-bit numbers have been loaded in registers s3 and s4, and the product from the register s5 is sent to the output port 80 of the microcontroller. In this example, multiplicand is 7, and multiplier is 15 (or 0F in hexadecimal). The 8-bit product is 105 (or 01101001 in binary representation).

This text file has been saved with the name mult.psm in the same folder with the assembler kcpsm3.exe. One of the automated generated files is mult.v, which contains the instructions from the 1k x 18 Instruction PROM of the microcontroller. This Verilog file is renamed as prog_rom.v and, as we can see in Fig. 2, it contains 10 bits of address input and 18 bits of data output for instructions. Memory contains 18432 bits and information in the file is organized in 72 rows, each of them having 256 bits, represented with hexadecimal digits. Our example needs about 336 bits, as we can see below:

```
module prog_rom (address, instruction, clk);
input [9:0] address;
input clk;
output [17:0] instruction;
RAMB16_S18 ram_1024_x_18(
. . . . . .)
defparam ram_1024_x_18.INIT_00 =
```

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 6 (December, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

256'h040E050EB570070607060706070607061730500E240106000500C480040FC3800307;

defparam ram_1024_x_18.INIT_01 =

256'h0000000000000000000000000000000000000000000004000C580540646048601;

. . . . . .

endmodule

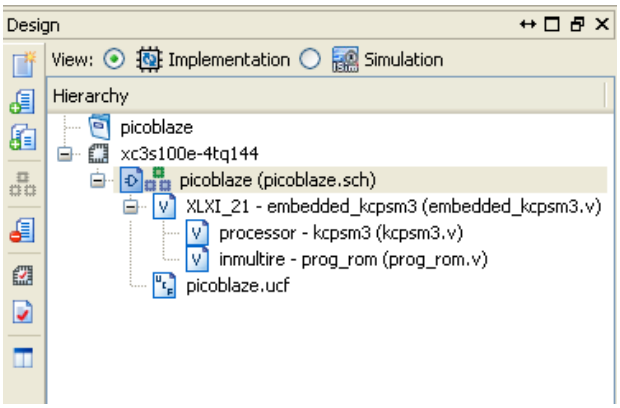

Fig. 4 Hierarchy of the project "picoblaze" in the window "Design" from ISE

In Fig. 4 we can see a new project, called "picoblaze", implemented in a Xilinx Spartan-3E FPGA from the Basys 2 board. The circuit has 144 pins and about 100,000 logic gates. picoblaze.sch is a schematic file, which contains the graphic symbol generated from the file embedded_kcpsm3.v. This symbol includes both kcpsm3.v and prog_rom.v files, that is the whole structure from the Fig. 2.

Fig. 5 represents the content of the picoblaze.sch. We have used two counters with the aim of decrease the clock frequency from 25 MHz to about 1,5 Hz, in order to see what happens with the outputs of the microcontroller, using LEDs from the Basys 2 board. CB16CE is a 16 bit counter with Clock Enable (CE), and CB8CE is a 8 bit counter of the same type. If the MSB of the port_id is 1 (port address 80), then information from the output out_port(7:0) may be visualized on the LEDs of the Basys 2 board.

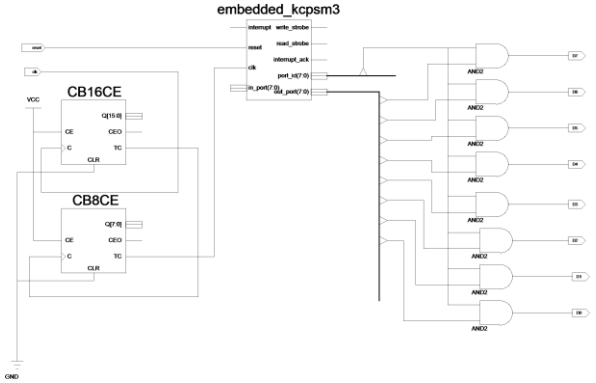

Fig. 5 Schematics using an embedded microcontroller PicoBlaze

IV. Other Implementations of the Algorithm

In this section we discuss other implementations of the multiplier, without using a soft processor like PicoBlaze.

A. *Implementation using Verilog HDL*

A very simple code can be written using Verilog language, like in the following example:

```
module product (input [3:0] x, y,
                output [7:0] p);
assign p = x * y;
endmodule
```

If the FPGA contains some embedded multipliers, then one of them will be used for our project. Our circuit, xc3s100e contains 4 multipliers for operands up to 18 bits in a single column. Dedicated 18 x 18 multipliers are fast and efficient for sign or unsigned multiplication of operands up to 18 bits. We have analysed also the case when there are no dedicated multipliers in FPGA, and the synthesis of a multiplier has been done only with Configuration Logic Blocks (CLBs), using Look Up Tables (LUTs) and Flip-Flops (FFs).

B. *Implementation using wired logic*

This solution was proposed in [4]. As indicated by the arrows of the diagram in Fig. 6, 4 bits from the accumulator register and 4 bits from the multiplicand are connected to the adder inputs. When an ADD signal occurs, the adder outputs (5 bits, including carry) are stored in the register by the next rising clock edge. The multiplier is stored in the lower 4 bits of the product register, which are initially unused. The LOAD signal loads the multiplier into the lower 4 bits, and clears the upper 5 bits of the 9 bit-register. The SHIFT signal causes the contents of the product register to be shifted one place to the right on the next rising clock edge. After a START signal has been received, the control circuit generates the proper sequence of the ADD and SHIFT signals. If the current multiplier bit (M) is 1, the multiplicand is added to the accumulator followed by a right shift; if the multiplier bit is 0, the addition is skipped and only the right shift occurs.

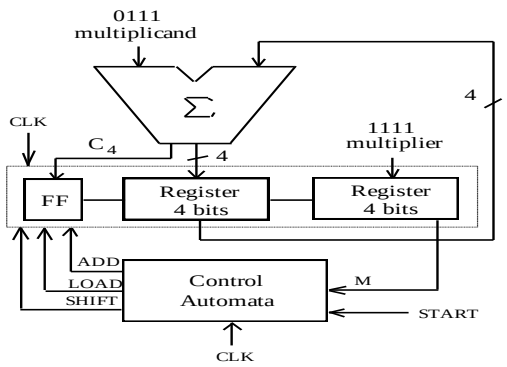

Fig. 6 Block diagram for binary multiplier ([4])

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 6 (December, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

C. *Implementation using microprogramming*

This solution was proposed in [5]. A microprogrammed system has two main units: RALU (Registers and ALU), which processes the information based on an algorithm, and CROM (Controller and ROM), which generates the signals for RALU, like the Control Automata in the previous implementation. The block diagram for RALU is very similar with the diagram from Fig. 6 (without Control Automata) and the block diagram for CROM is represented in Fig. 7. Only two instructions are need to implement our algorithm and the whole program contains 7 memory locations, which are stored in the memory M. These instructions are: IF T=0 THEN GO TO ADR0, ELSE GO TO ADR1, and DO OPi AND GO TO ADR1. Variable T is selected using a multiplexer, and it may be a start signal, E, the LSB of the multiplier Y, Y_0, and the MSB of the counter I, I_2. AM is an address multiplexer (which selects one of the addresses ADR0 and ADR1), and AR is the address register (which selects the next instruction form the memory M).

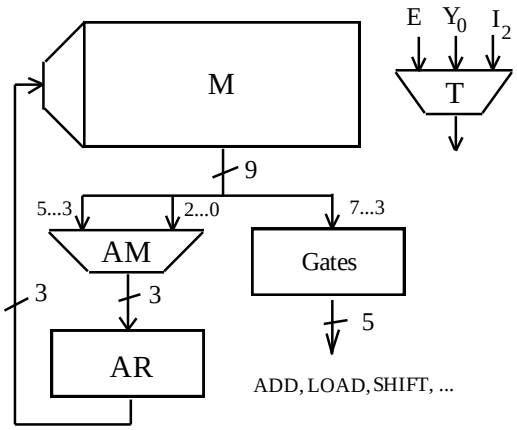

Fig. 7 Block diagram for CROM ([5])

V. EXPERIMENTAL RESULTS

Soft processor PicoBlaze has been implemented on a Basys 2 board, and Fig. 8 shows the Device Utilization Summary in Xilinx ISE. It uses 179 LUTs, 68 flip-flops, 93 slices and one block of RAM memory (with 10 addresses and 18 data bits). These resources are constant for other applications, so we are interested only in the memory used for our algorithm. A comparison with other solutions is given in Table I.

In Fig. 9 there is an example of simulation. The two operands have been loaded in two registers of the PicoBlaze, and after 114 clocks, the result has been obtained. Then a jump to the first address runs again the algorithm.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	68	1,920	3%	
Number of 4 input LUTs	177	1,920	9%	
Number of occupied Slices	93	960	9%	
Number of Slices containing only related logic	93	93	100%	
Number of Slices containing unrelated logic	0	93	0%	
Total Number of 4 input LUTs	179	1,920	9%	
Number used as logic	109			
Number used as a route-thru	2			
Number used for Dual Port RAMs	16			
Number used for 32x1 RAMs	52			
Number of bonded IOBs	10	108	9%	
Number of RAMB16s	1	4	25%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	4.00			

Fig. 8 Resources used by PicoBlaze IP in xc3s100e FPGA

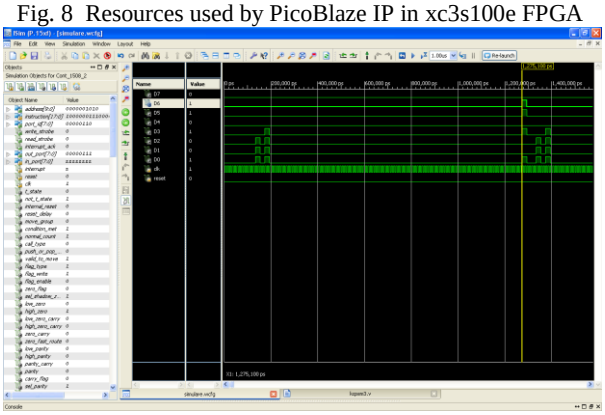

Fig. 9 Simulation Screen for Multiplying of 0111 with 1111

Final result for the two proposed operands, 7 and 15, is 105, that is 01101001 in binary. This result can be seen on the output LEDs from the Basys 2 board in Fig. 10.

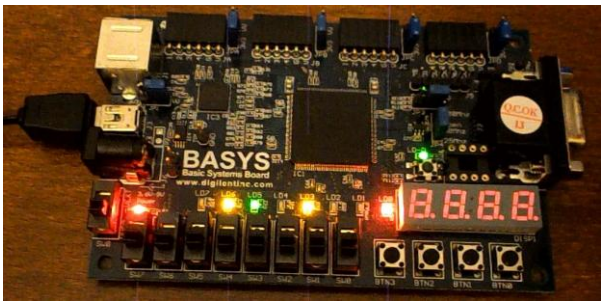

Fig. 10 Final Result on a Basys2 Board

PicoBlaze uses only 336 bits of instruction memory, but total time is about 1140 ns, at a clock frequency of 100 MHz. If a dedicated 18 x 18 multiplier is used, then implementation of the algorithm needs 542 bits of memory configuration, but the propagation time is only 8.850 ns, the best from all our solutions. If we don't use a dedicated multiplier in FPGA, then we use less memory (only 394 bits), but the time is longer (12.564 ns). Our examples of wired logic and microprogrammed logic, are sequential solutions, as good as the solution with PicoBlaze, thus the total times are greater than in Verilog solutions (hundreds of ns). All these results are illustrated in Table I.

Table I. A Comparison in Hardware Resources for Different Solutions

Implementation	4 Input LUTs	FFs	Bits of Memory	Time (ns)
with PicoBlaze (only memory)	-	-	336	1140
using Verilog with MULT18x18SIO	32	32	542	8.850
using Verilog with CLBs	23	26	394	12.564
using wired logic	26	13	429	120
using micro-programmed logic	23	20	451	350

VI. CONCLUSIONS

Results from the Table I suggest that combinational circuits, generated using Verilog HDL, offer the highest speed (time is less than 12.564 ns). Their configurations in FPGA need usually more than 400 bits of memory. Sequential circuits used to implement the same algorithm need also more than 400 bits of configuration memory, but they are slower, total time depends on the clock frequency (hundreds of ns). The advantage of implementation in PicoBlaze is the smallest amount of memory (only 336 bits of memory) for this algorithm, even though this solution offers the lowest speed (only 1140 ns). If we take into account the hardware resources to implement PicoBlaze, the number of bits of memory are larger, but these resources are constant for other applications. So, for a complex system with not so high speed, implementation using soft IPs seems to be the best solution.

VII. REFERENCES

[1] 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, 1st. ed., Strathclyde Academic Media, Glasgow, Scotland, UK, 2014.

[2] P. P. Chu, FPGA Prototyping by Verilog Examples. Xilinx Spartan 3 Version, Wiley, New Jersey, USA, 2008.

[3] P. J. Ashenden, Digital Design. An Embedded Systems Approach using Verilog, Morgan Kaufmann, Burlington, USA, 2008.

[4] C. H. Roth, jr., and L. L. Kinney, Fundamentals of Logic Design, 7th. ed., Cengage Learning, Stamford, USA, 2014.

[5] A. Stauffer, Systèmes numériques cables et microprogrammés, Presses Polytechniques Romandes, Lausanne, Suisse, 1989.

[6] PicoBlaze 8-bit Embedded Microcontroller User Guide for Extended Spartan-3 and Virtex-5 FPGAs Introducing PicoBlaze for Spartan-6, Virtex-6, and 7 Series FPGAs (UG129), Xilinx, 2011. Available: http://www.xilinx.com/

[7] Spartan-3 Generation FPGA User Guide (UG331), Xilinx, 2008. Available: http://www.xilinx.com/