Pico Blaze praxis for solving mathematical functions

Mladen Milushev
Faculty of Mechanical Engineering
TU-Sofia
Sofia, Bulgaria
mcm@tu-sofia.bg

Filip F. Filipov
Faculty of Computer Systems and Control
TU-Sofia
Sofia, Bulgaria
pilif.pi.lif@gmail.com

Abstract—**This paper presents an example use of Pico Blaze, a famous soft-core processor in the education praxis. The mathematical function multiplication is used as a demonstration of combining software and hardware in FPGA.**

Keywords- Pico Blaze; Multiplication; FPGA;

I. INTRODUCTION

Pico Blaze is a compact 8-bit processor soft-core for Xlinx FPGA devices. It is provided as a HDL description and can be synthesized along with other logic. Since he is not intended to be a microcontroller (object oriented) he is compact and flexible and can be used for simple data processing and control or non-time-critical I/O operations. The realization of a multiplication is a good project for exploring these functionalities. FPGAs are the only possibility to implement existing variations of processor cores and to expand them according to the needed functionality.

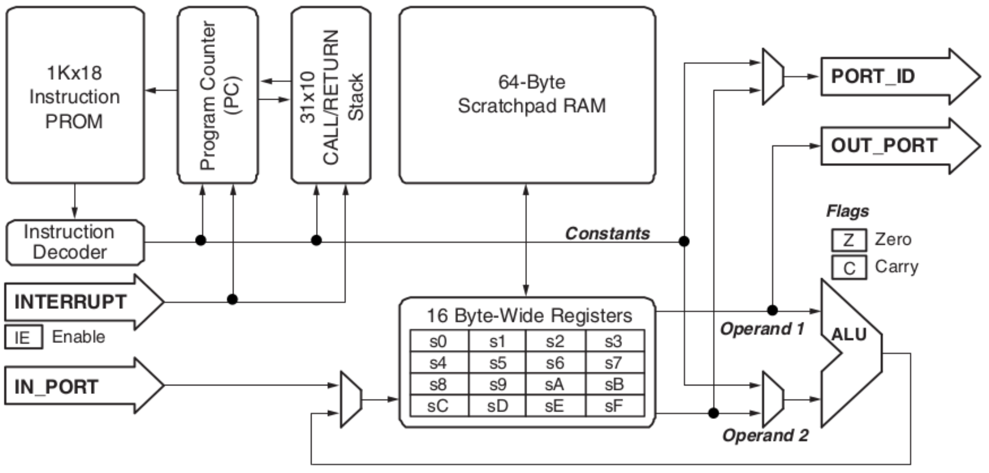


Figure 1. Pico Blaze architecture

II. PICO BLAZE

The main data flow is shown in Fig.1[1] featuring the following characteristics:

- 8 –bit ALU with carry and zero flags
- 64-byte scratch memory
- 256 input and 256 output ports achieved through multiplexing
- 2 clock cycles per instruction

The scratch memory is simply data RAM and is considered to be a reservoir for additional data. Although there is no direct path between the data RAM and the ALU.

For the design process Pico Blaze is organized as two HDL modules shown in Fig.2. KCPSM3 stands for constant(K) coded programmable state machine and is the original name of Pico Blaze. The other module is for the instruction memory – usually the assembly code is configured as ROM in HDL. The signal names on the left side of each module are input and on the right output.

One advantage of using soft-core processor is that the designer can alter his periphery and be familiar with one processor and not to examine the required functionality and based on it to select the proper processor with adequate I/O interface. This way time-critical tasks and many other functionality can be implemented in the FPGA fabric (i.e. hardware) and trough the implemented soft-core all remaining low-speed functions (i.e. software).

The basic development flow with Pico Blaze is shown in Fig. 3 and has the following steps:

- Divide the functionality in software and hardware
- Write the assembly program

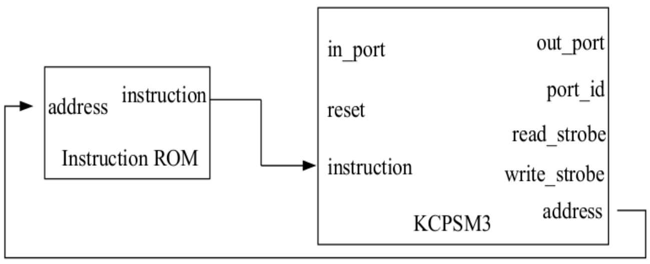


Figure 2. HDL representation of Pico Blaze

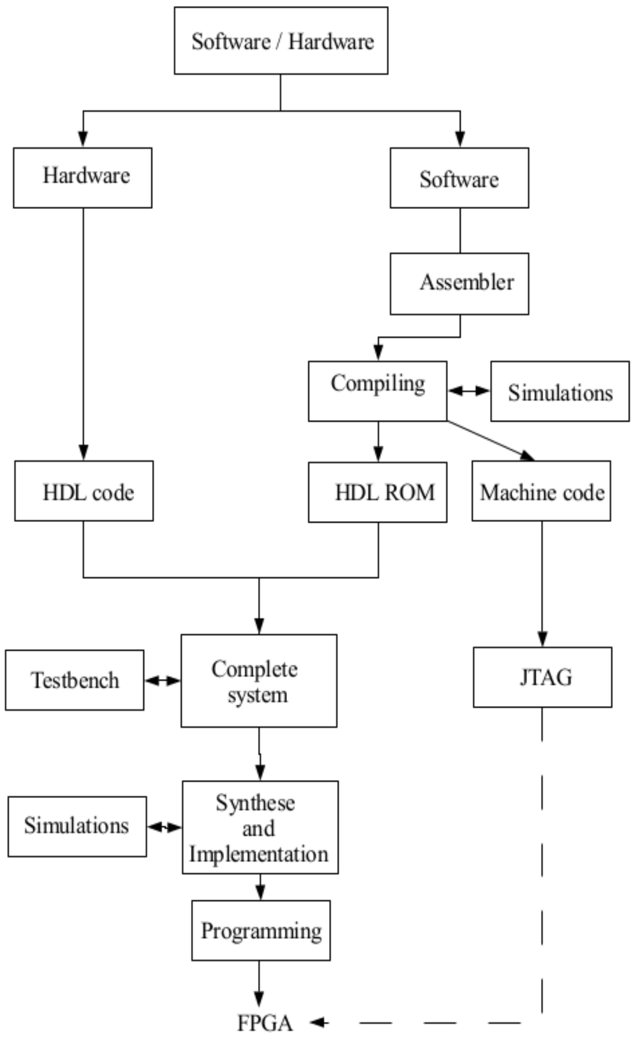


Figure 3. Development flow with Pico Blaze

- Generate an Instruction ROM in a form of a HDL file
- Perform computer simulations on the assembly program
- Write HDL code for the hardware part
- Combine the HDL code with Pico Blaze and instruction ROM
- Perform HDL simulations for the entire system
- Synthesize, implement and then program the FPGA chip
- After the system is synthesized the developer can still change the assembler code.

III. FUNCTION USING PICO BLAZE

Since Pico Blaze does not contain a hardware multiplier the developer has two possibilities to choose from:

- To create a HDL multiplier (hardware) and attach it to Pico Blaze as periphery
- To use a algorithm (software) for multiplication such as shift-and-add

A. HDL multiplier

The multiplication will be only one instruction (i.e. two clock cycles) in the program for the Pico Blaze. The connection with the peripheral requires two output ports and buffers for the two operands and two additional input ports for the 16-bit product.

B. Shift-and-add multiplication

The software multiplication requires a subprogram which in her worst-case scenario would require 60 instructions. The shift-and-add method adds the multiplicand A to itself B times, where B denotes the multiplier. This method is similar to the multiplication performed by paper and pencil, which is to take the digit of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate position to the left. . The shift-and-add method is shown in Fig. 4 [2]. In each iteration the multiplicand B is shifted one position to the left and the multiplier Q one position to the right. If the LSB of the multiplier is 1 then the multiplicand is added to the product A. Table 1 shows the multiplication of 4 (100) and 4 (100) = 10 000 .

TABLE I. MULTIPLICATION EXAMPLE

Step	A	Q	B	Operation
0	0000 0000	100	0000 0100	Start
1	0000 0000	100	0000 1000	Shift left B
	0000 0000	010	0000 1000	Shift right Q
2	0000 0000	010	0001 0000	Shift left B
	0000 0000	001	0001 0000	Shift right Q
3	0001 0000	001	0001 0000	Add B to A
	0001 0000	001	0010 0000	Shift left B
	0001 0000	000 - end	0010 0000	Shift right Q

Multiplication of 4 x 4

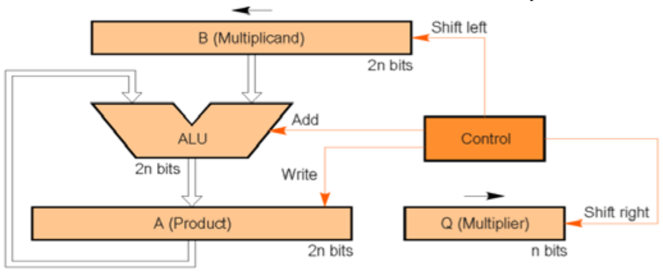


Figure 4. Shift-and-add multiplication

IV. PROJECT DETAIL

The idea of the project is to show how Pico blaze could get two numbers as inputs, perform some function with them and show the result somewhere. For this purpose the following will be used:

- Switch – he will provide the values of the two numbers and select the content of the LED display
- Pushbutton 1 - to load the first number or alternatively the second when pressed
- Pushbutton 2 – to clear the data RAM and relevant registers
- Seven-segment LED – to display the selected 17-bit value in four hexadecimal digits
- UART interface – so that information can be entered and displayed in Windows HyperTerminal which is an alternative to switches and LEDs.

V. INPUT INTERFACE

The similarity between the input and output ports of Pico Blaze is that they all are set as port numbers by the output port *port_id* . So in order to use these ports multiplexing is needed, which can be (like the multiplication) software based or as hardware in form of HDL code. In the project the second alternative is used.

The input interface is shown in Fig. 5. The module *debounce* is used to remove any mechanical glitches when a button is pressed and actually is a simpler filter. After the signal is filtered the *flag FF* module keeps the button-pressed-event asserted until it is retrieved by the Pico Blaze input instruction, which sets the right input port number and clears the event.

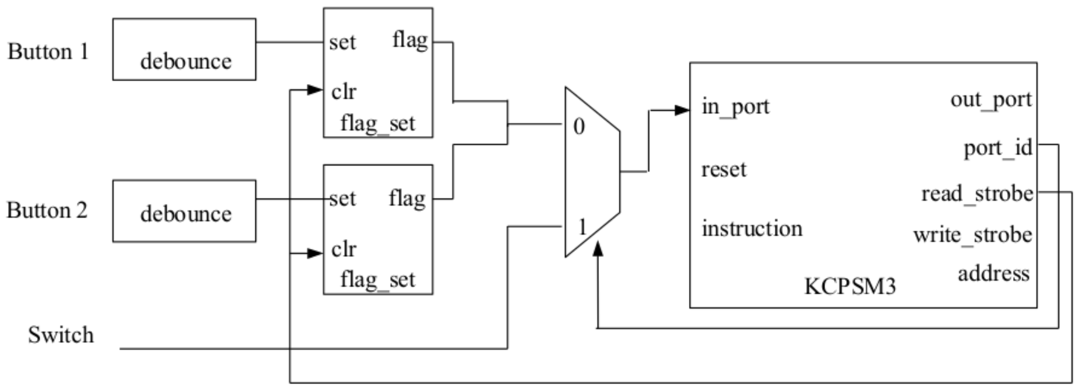


Figure 5. Two button input

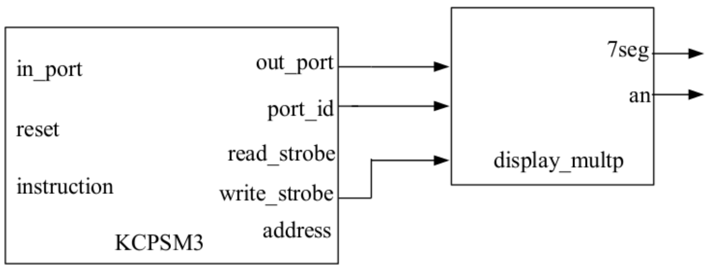


Figure 6. 7Segment output

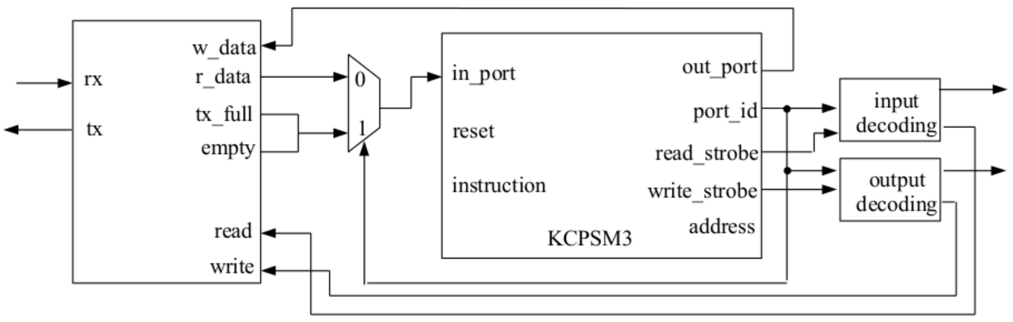


Figure 7. UART and Pico Blaze

VI. LED OUTPUT INTERFACE

It is shown in Fig. 6. There is again decoding of port_id and multiplexing but this time it is time multiplexed, because the four segments share the same input pins. The multiplexing shields the timing and allows them to appear as four independent seven-segment-LEDs. Again it can be software or hardware implemented. We chose to do it hardware in form of VHDL code which is the disp_mux in the figure.

The refresh rate of the segments has to be fast enough to fool our eyes but should be slow enough so that the LEDs can be turned on and off completely. The clock rate on the used prototype board is 50MHz so a 16 bit counter is used to for a refresh rate of 800Hz for each segment.

VII. UART INTERFACE

The interface is shown in Fig. 7. It is written as VHDL code. Since the UART itself has input and output FIFO buffers only decoding and multiplexing for the Pico Blaze I/O ports is needed. The UART signals for FIFO empty (rx_empty) or FIFO full (tx_full) are grouped together like the pushbuttons in the input interface since they are alternative to each other and in this way only one Pico Blaze inport is used.

VIII. RESULTS

For the RS232 connection the Hyper Terminal has to have *bits per second* set to 19200 and a tick on *echo typed Characters*. This way the UART module can generate an SQ prompt shown in Fig.8. . The user can enter four characters:

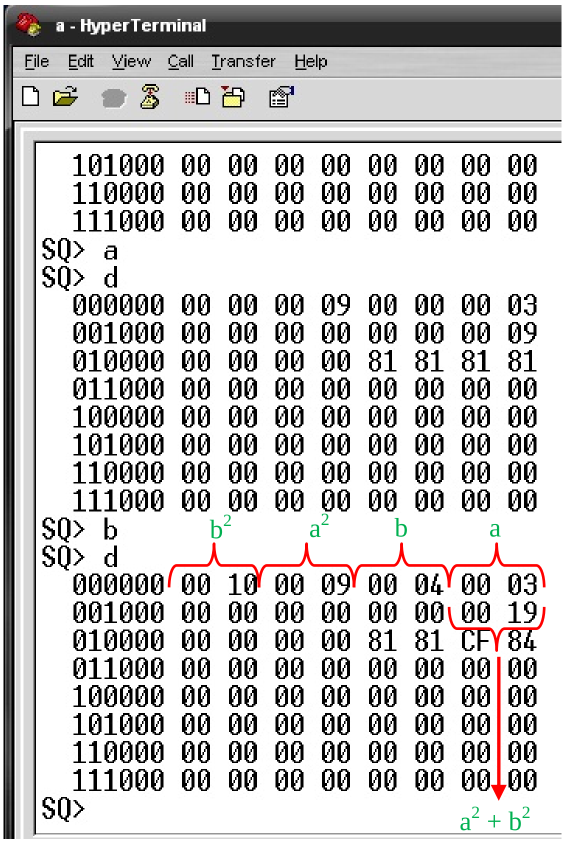
- a and b for the two numbers. When these characters are pressed the value of the 8-bit switch is read.
- c is abbreviation for clear. When pressed the scratch RAM is cleared
- d is abbreviation for dump. When pressed scratch RAM is represented on the screen. In this way the user can observe all steps of the function calculation.

The switch on the Digilent Basys prototype board provides not only values for a and *b* but can also be used to select the content of the segments by changing the values of switches 0 to 2. All combinations and display values are shown in table 2. The board is shown in Fig.9.

TABLE II. COMBINATION OF SWICHES

Display value	SW2	SW1	SW0
a	0	0	0
b	0	0	1
a^2	0	1	0
b^2	0	1	1
a^2+b^2	1	1	1

Changing the displayed results



```
 101000 00 00 00 00 00 00 00 00
 110000 00 00 00 00 00 00 00 00
 111000 00 00 00 00 00 00 00 00
SQ> a
SQ> d
 000000 00 00 00 09 00 00 00 03
 001000 00 00 00 00 00 00 00 09
 010000 00 00 00 00 81 81 81 81
 011000 00 00 00 00 00 00 00 00
 100000 00 00 00 00 00 00 00 00
 101000 00 00 00 00 00 00 00 00
 110000 00 00 00 00 00 00 00 00
 111000 00 00 00 00 00 00 00 00
SQ> b
SQ> d
 000000 00 10 00 09 00 04 00 03
 001000 00 00 00 00 00 00 00 19
 010000 00 00 00 00 81 81 CF 84
 011000 00 00 00 00 00 00 00 00
 100000 00 00 00 00 00 00 00 00
 101000 00 00 00 00 00 00 00 00
 110000 00 00 00 00 00 00 00 00
 111000 00 00 00 00 00 00 00 00
SQ>
```

Figure 8. Communication and results displayed in Hyper Terminal

IX. CONCLUSION

In this paper the realization of a multiplication function combined with a soft-core is explored. The advantages of using this core were shown in case of a realization on FPGA, since Pico Blaze is platform dependent on Xlinx products.

REFERENCES

[1] XLINX, „PicoBlaze 8-bit Embedded Microcontroller User Guide “, UG129 June 22, 2011

[2] Baruch, Z. F., *Structure of Computer Systems* (in English), U. T. PRES, Cluj-Napoca,2002, ISBN 973-8335-44-2

[3] Pong P. Chu, „FPGA prototyping by VHDL examples“, Wiley, 2008

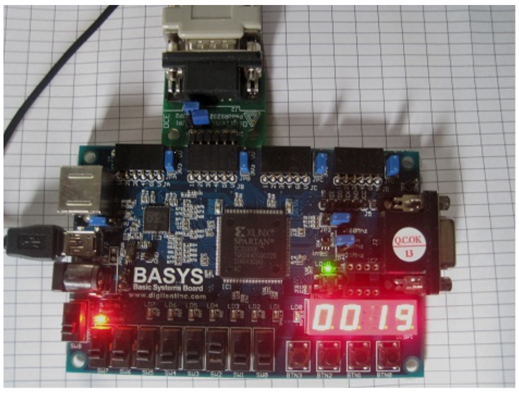


Figure 9. FPGA prototype board