ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Hardware Implementation of Algorithms

10. PicoBlaze 8-bit microcontroller.

Lead Partner: Warsaw University of Technology

Author: Lukasz Mik

University of Applied Sciences in Tarnow

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

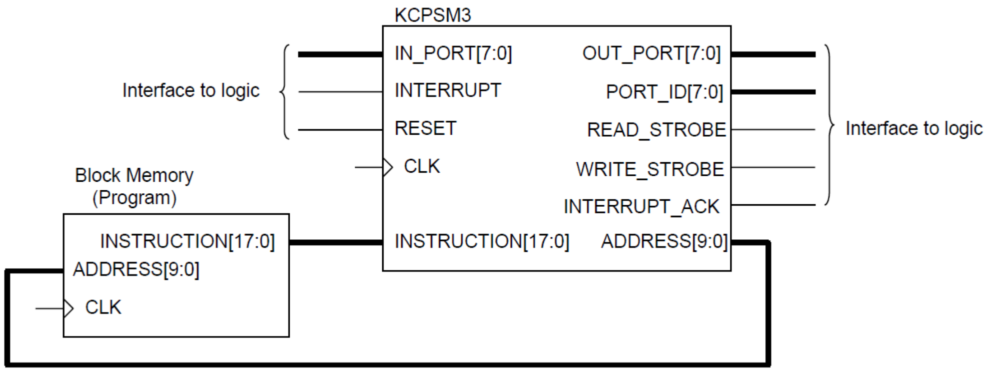
Copyright

Funding Disclaimer

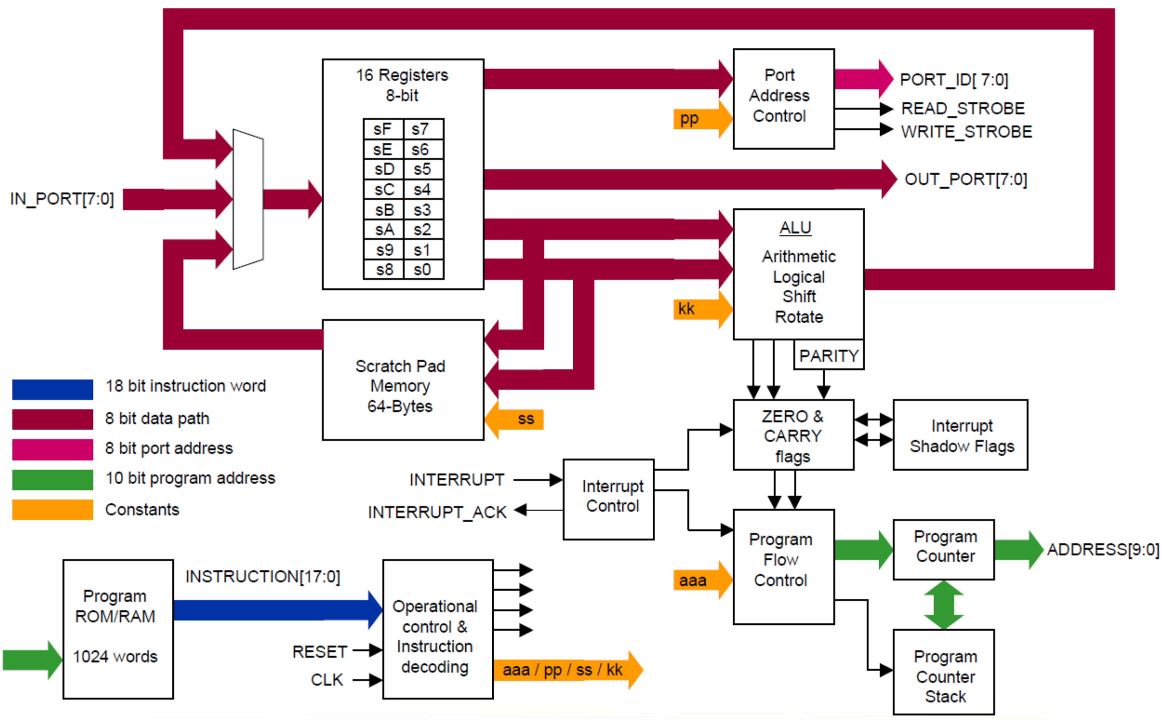
This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

I. What is PicoBlaze?

PicoBlaze (KCPSM3) is a very simple 8-bit microcontroller dedicated to the Spartan-3 family, but is also suitable for Virtex-II and Virtex-IIPRO. Its main purpose is applications requiring a complicated state machine, but not time-critical. That's why it's called the (K)constant Coded Programmable State Machine. The figure below shows the KCPSM3 block with program memory. This is the result of compiling the microcontroller project from source files available on the Xilinx website.

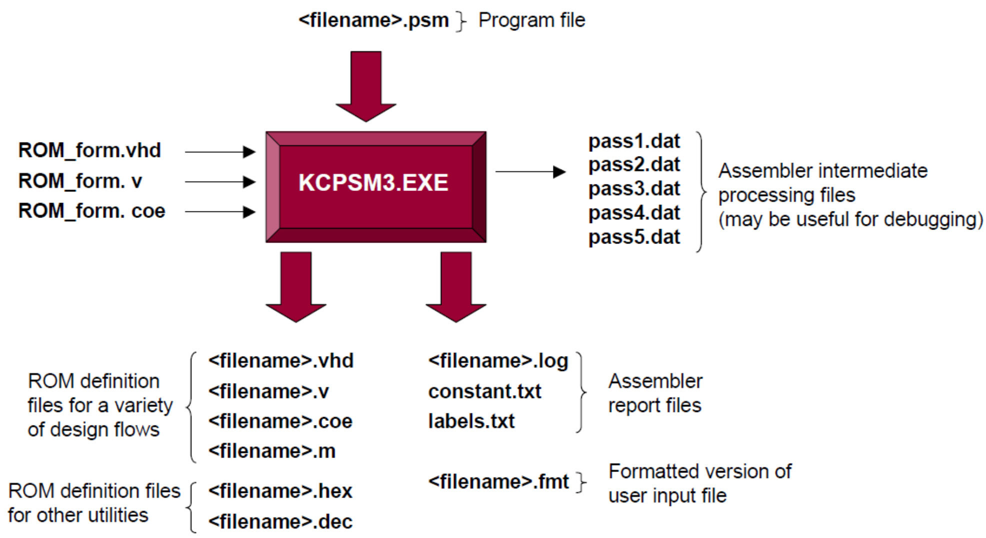


The most important feature of this microcontroller is that it is completely embedded in the FPGA and does not require additional external circuits to operate. Inside the FPGA, it can be connected to any other element that performs advanced logic functions. The KCPSM3 project was written in VHDL. The architecture of the microcontroller is shown in the block diagram below.



The KCPSM3 supports programs up to 1024 instructions long that occupy a single block of ROM. In its architecture, you can extract 16 working registers marked from s0 to sF, which can be assigned their names in assembly language. Each of the registers can be used to the full extent, because the system does not have a built-in accumulator. The ALU block is responsible for performing 8-bit arithmetic and logical operations using working registers. Additional flags set depending on the result of arithmetic and logical operations are Carry and Zero. The statuses of these flags can be used to control the order of instructions to be executed within the program and subroutines. To handle subroutine jumps, a stack is used that allows nesting up to 31 levels within subroutines. PicoBlaze can handle up to 256 input and output ports. Access to the port is done using an 8-bit address, given on PORT_ID. Reading data from the input port or sending data to the output port is done indirectly through one of the working registers. The microcontroller also has a cache memory with a capacity of 64B (Scratch Pad Memory), which can be used to store the contents of registers or data from I/O ports.

The PicoBlaze microcontroller is programmed using the assembler language, the full list of instructions of which can be found in the KCPSM3.zip archive (*KCPSM3_Manual.pdf* file), available for download from the manufacturer's website. The assembler code source file must have the PSM extension. The file with the contents of the program memory is automatically generated by the *KCPSM3.exe* program - assembly language compiler. It works only in the DOS environment, so during the classes it will be necessary to use an emulator of this system, e.g. DOSBox. The method of compiling the source files of the microcontroller is presented below.



II. Download and prepare source files for the exercise

First, we download the *KCPSM3.zip* archive from the Xilinx website: https://www.xilinx.com/products/intellectual-property/picoblaze.html#design. On this page, click on the link with the name *PicoBlaze for Spartan-3, Virtex-4, Virtex-II and Virtex-II Pro FPGAs*. You must log in to download the archive. In the absence of an account on the site https://xilinx.com you must create them to access the files. After unpacking the files from the archive into the KCPSM3 folder, its contents should look like this:

Assembler	04.08.2005 16:51	Folder plików	
DATA2MEM_assistance	04.08.2005 16:51	Folder plików	
JTAG_loader	04.08.2005 16:51	Folder plików	
Verilog	04.08.2005 16:51	Folder plików	
VHDL	04.08.2005 16:50	Folder plików	
kcpsm3.ngc	15.06.2004 13:59	Plik NGC	50 KB
KCPSM3_Manual	10.10.2003 16:06	Microsoft Edge PD...	609 KB
read_me	04.08.2005 16:56	Dokument tekstowy	22 KB
UART_Manual	23.04.2003 10:46	Microsoft Edge PD...	111 KB
UART_real_time_clock	07.10.2003 16:27	Microsoft Edge PD...	316 KB

The user's manual has been made available by the manufacturer in the *KCPSM3_Manual.pdf* file.

In the next step, we create a directory called pico_test1 in the location where we have stored our projects so far, e.g. *C:\Xilinx_work\pico_test1*. This will be the working directory for the rest of this exercise. Copy the following files from the Assembler folder to the *pico_test1* folder:

- KCPSM3.exe
- ROM_form.coe
- ROM_form.v
- ROM_form.vhd

Then we copy the *kcpsm3.vhd* file from the VHDL folder to the *pico_test1* folder.

III. Implementation of a simple program in assembler

As part of the exercise, you will write a program code that will read the status of 8 DIP buttons and display these states on 8 LEDs.

In any text editor, the simple assembler code shown in the listing below:

```
; Simple loop that puts contents of input register
; into the output register
;
; switches DSIN $00
; LEDS DSOUT $80
start: INPUT s0, 00 ; read switches into register s0
OUTPUT s0, 80 ; write contents of s0 to output port 80 - leds.
JUMP start ; loop back to start
```

The program consists of an endless loop, inside which the value from the input port with the address 00h is read into the s0 register. Then the value from the s0 register is sent to the output port with the address 80h. A semicolon is used to mark a comment in the program code.

After editing the code, save the file as **picotest.psm**. Please note that the file name cannot be longer than 8 characters. After saving the file, the contents of the **pico_test1** directory should look like this:

KCPSM3.EXE	05.07.2005 09:33	Aplikacja	89 KB
kcpsm3.vhd	20.07.2005 08:50	Plik obrazu dysku t...	67 KB
picotest.psm	04.05.2023 13:46	Plik PSM	1 KB
ROM_form.coe	25.01.2002 15:17	Plik COE	1 KB
ROM_form.v	04.07.2005 18:05	Plik V	15 KB
ROM_form.vhd	05.07.2005 09:39	Plik obrazu dysku t...	13 KB

Note: If you are using Notepad, you must change the file type from text *.txt to all *.* files

Nazwa pliku:	picotest.psm
Zapisz jako typ:	Wszystkie pliki (*.*)

Otherwise, the program will add the extension *.txt to the name of the picotest.psm file, which will save the file as *picotest.psm.txt*.

IV. Compilation of assembly code using KCPSM3.exe compiler and DOSBox emulator

To compile the program code for the PicoBlaze microcontroller, 4 input files are necessary: *picotest.psm, ROM_form.vhd, ROM_form.v, ROM_form.coe.* ROM_*.* files are templates for block RAM initialization. The assembler compiler is a program that only runs on 32-bit DOS. Windows 32-bit systems can run in command-line mode where KCPSM3.exe can be run. When you try to run this program from a 64-bit Windows command prompt, you will receive a message that the system is not compatible with the 32-bit version of the program.

To avoid this problem, download a 32-bit DOS emulator from the website: http://www.dosbox.com/. After downloading, installing and running the emulator, you will need to mount the **pico_test1** folder as a drive in it, using a letter not used in the system. When DOSBox starts, a command prompt window will appear with the Z drive set by default. So we type the command:

```
mount G C:\Xilinx_work\pico_test1\
```

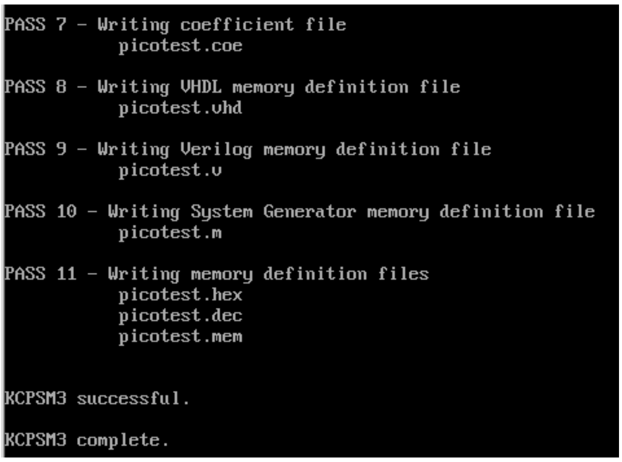
If the entered drive letter G is free and the folder *C:\Xilinx_work\pico_test1* exists, the virtual drive G will be assigned to it.

```
Z:\>mount G C:\Xilinx_work\pico_test1
Drive G already mounted with local directory c:\Xilinx_work\pico_test1\
```

To navigate to the created drive, simply type **G:** followed by **dir** to display the contents of the root directory on that drive.

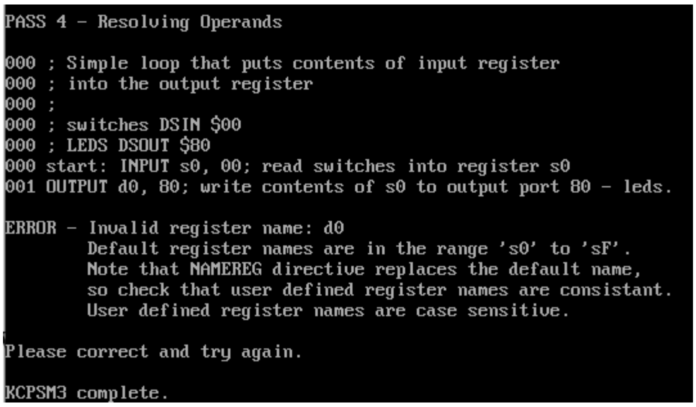
```
Z:\>G:

G:\>dir
Directory of G:\.
.              <DIR>            04-05-2023 13:46
..             <DIR>            04-05-2023 13:44
KCPSM3   EXE            90,308 05-07-2005  9:33
KCPSM3   VHD            67,765 20-07-2005  8:50
PICOTEST PSM               271 04-05-2023 13:46
ROM_FORM COE               857 25-01-2002 16:17
ROM_FORM V              15,275 04-07-2005 18:05
ROM_FORM VHD            12,748 05-07-2005  9:39
    6 File(s)          187,224 Bytes.
    2 Dir(s)       262,111,744 Bytes free.
```

After entering the **KCPSM3 picotest.psm** command, the syntax in the assembler will be checked and then the program code will be compiled to binary format. After successful completion of the compilation process, the *KCPSM3 successful* message will appear.



```
PASS 7 - Writing coefficient file
          picotest.coe

PASS 8 - Writing VHDL memory definition file
          picotest.vhd

PASS 9 - Writing Verilog memory definition file
          picotest.v

PASS 10 - Writing System Generator memory definition file
          picotest.m

PASS 11 - Writing memory definition files
          picotest.hex
          picotest.dec
          picotest.mem


KCPSM3 successful.

KCPSM3 complete.
```

If there is an error in any part of the program code, the compiler will display information about this error.



```
PASS 4 - Resolving Operands

000 ; Simple loop that puts contents of input register
000 ; into the output register
000 ;
000 ; switches DSIN $00
000 ; LEDS DSOUT $80
000 start: INPUT s0, 00; read switches into register s0
001 OUTPUT d0, 80; write contents of s0 to output port 80 - leds.

ERROR - Invalid register name: d0
        Default register names are in the range 's0' to 'sF'.
        Note that NAMEREG directive replaces the default name,
        so check that user defined register names are consistant.
        User defined register names are case sensitive.

Please correct and try again.

KCPSM3 complete.
```

Each subsequent run of the assembler overwrites the previous result files with new ones. After generating the binaries, you can close the DOSBox window by typing **exit**.

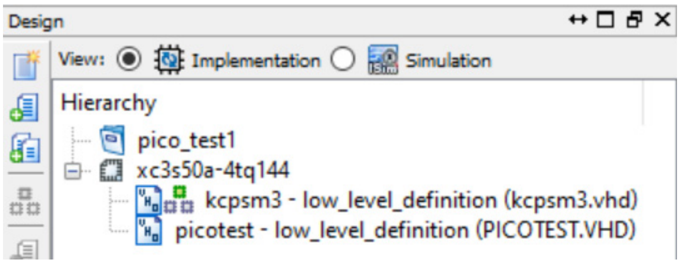
V. PicoBlaze implementation in Spartan-3A.

We run *ISE Design Suite 14.7* and create a new project named **pico_test1**, in the *Xilinx_work* working folder. Then, by default, it will be saved in the same folder where the PicoBlaze microprocessor files are already generated.

W kolejnym kroku tradycyjnie należy podać parametry układu docelowego.

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S50A
Package	TQ144
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

Add source files to the created project by selecting *Project → Add Source…* First, add the **kcpsm3.vhd** and **PICOTEST.VHD** files. After adding them, they should appear in the list of files in the project window.



Double clicking on the *PICOTEST.VHD* file will open it in the program's editing window. In the header part of this file there is a design unit called *picotest*, which is the program memory of the microcontroller.

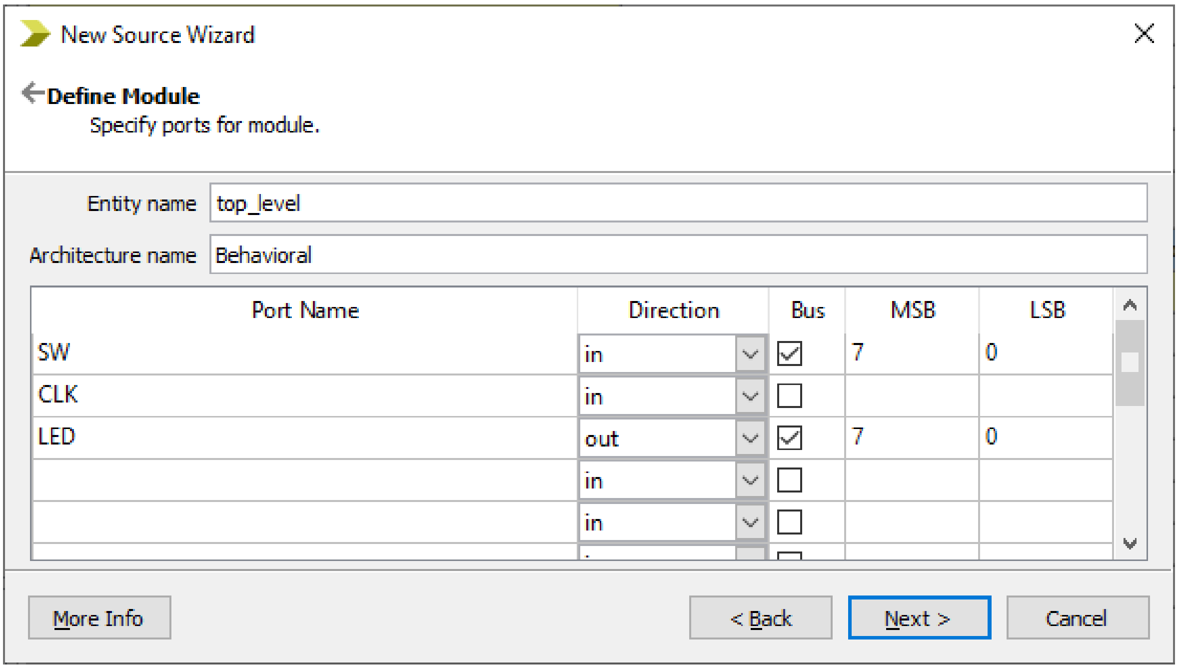
```vhdl
entity picotest is
    Port (      address : in std_logic_vector(9 downto 0);
            instruction : out std_logic_vector(17 downto 0);
                    clk : in std_logic);
    end picotest;
```

Double clicking on the *kcpsm3.vhd* file will load its content in the editing window.

In this file, in the header part, there is a declaration of the main microcontroller design unit.

```vhdl
entity kcpsm3 is
    Port (      address : out std_logic_vector(9 downto 0);
            instruction : in std_logic_vector(17 downto 0);
                port_id : out std_logic_vector(7 downto 0);
           write_strobe : out std_logic;
               out_port : out std_logic_vector(7 downto 0);
            read_strobe : out std_logic;
                in_port : in std_logic_vector(7 downto 0);
              interrupt : in std_logic;
          interrupt_ack : out std_logic;
                  reset : in std_logic;
                    clk : in std_logic);
    end kcpsm3;
```

For the proper operation of the project, it is necessary to connect the added components into a single whole using the top level architecture. To do this, we create a new source file in the project called **top_level.vhd**. When creating a VHDL module, we add 3 ports:



Inside **top_level** architecture we add 2 components: **kcpsm3** –PicoBlaze core and **picotest** – memory with the microcontroller program.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_level is
    Port ( SW : in  STD_LOGIC_VECTOR (7 downto 0);
           clk : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end top_level;

architecture Behavioral of top_level is

-- PicoBlaze core
component kcpsm3
port (address : out std_logic_vector(9 downto 0);
      instruction : in std_logic_vector(17 downto 0);
      port_id : out std_logic_vector(7 downto 0);
      write_strobe : out std_logic;
      out_port : out std_logic_vector(7 downto 0);
      read_strobe : out std_logic;
      in_port : in std_logic_vector(7 downto 0);
      interrupt : in std_logic;
      interrupt_ack : out std_logic;
      reset : in std_logic;
      clk : in std_logic);
end component;

-- program memory
component picotest
port (address : in std_logic_vector(9 downto 0);
      instruction : out std_logic_vector(17 downto 0);
      clk : in std_logic);

end component;
```

It is also necessary to add signals inside the architecture, connecting these components with each other and with the input and output ports of the design unit (ultimately with the pins of the FPGA).

```vhdl
-- Signals used to connect PicoBlaze core to program memory and I/O logic
signal address : std_logic_vector(9 downto 0);
signal instruction : std_logic_vector(17 downto 0);
signal port_id : std_logic_vector(7 downto 0);
signal out_port : std_logic_vector(7 downto 0);
signal in_port : std_logic_vector(7 downto 0);
signal write_strobe : std_logic;
signal read_strobe : std_logic;
signal interrupt_ack : std_logic;
signal reset : std_logic;
-- interrupt input is not used - assigned as inactive value '0'
signal interrupt : std_logic :='0';
```

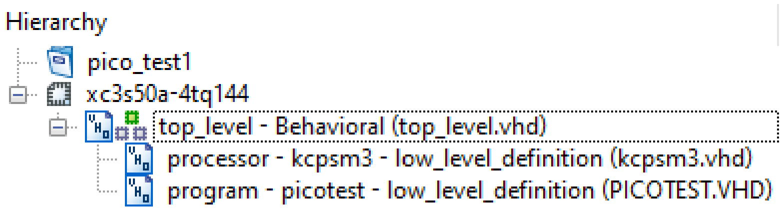
In the description of the architecture, instances of previously added components must be created between the **begin** and **end** keywords. The command **port map** is used for this, which binds the signals within the architecture and the I/O ports of the **top_level** design unit with the ports of the components.

```vhdl
-- Instantiating the PicoBlaze core
processor: kcpsm3
port map (address => address,
       instruction => instruction,
       port_id => port_id,
       write_strobe => write_strobe,
       out_port => out_port,
       read_strobe => read_strobe,
       in_port => in_port,
       interrupt => interrupt,
       interrupt_ack => interrupt_ack,
       reset => reset,
       clk => clk);

-- Instantiating the program memory
program: tutorial
port map (address => address,
       instruction => instruction,
       clk => clk);
```

Two more processes will be added to the project: one to handle the microcontroller's input port and the other to handle its output port.

```vhdl
-- PicoBlaze input port at adress 00h
input_ports: process(clk)
begin
   if clk'event and clk='1' then
      case port_id(1 downto 0) is              --address 00h
         when "00" => in_port <= SW;
         when others => in_port <= "XXXXXXXX"; --other addresses are not used
      end case;
   end if;
end process input_ports;

-- PicoBlaze output port at address 80h
output_ports: process(clk)
begin
   if clk'event and clk='1' then
       if port_id(7)='1' then
          LED <= out_port;
       end if;
   end if;
end process output_ports;

end Behavioral;
```

After saving changes in the project, the change in the project hierarchy - components added to the **top_level** unit will be visible as subordinate.



At this stage, you can validate the syntax and content of the source files by running the *Synthesize XST* process. If the program does not detect errors, before generating the configuration file, you still need to bind the top_level unit ports to the FPGA pins using the UCF file. To do this, add a new source file named **top_level** by selecting the *Implementation Constraints File* option in the file type selection window. In the editing window that will open right after creating the file, paste the following fragment with the assignment of pins to port names.

```
NET "clk"   LOC = P129   | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;

NET "LED[0]"  LOC = P46   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[1]"  LOC = P47   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[2]"  LOC = P48   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[3]"  LOC = P49   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[4]"  LOC = P50   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[5]"  LOC = P51   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[6]"  LOC = P54   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "LED[7]"  LOC = P55   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "SW[0]"  LOC = P70  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[1]"  LOC = P69  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[2]"  LOC = P68  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[3]"  LOC = P64  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[4]"  LOC = P63  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[5]"  LOC = P60  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[6]"  LOC = P59  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SW[7]"  LOC = P58  | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

If the editing window does not open automatically, select the **top_level.ucf** file on the list of project source files and then select the *Edit Constraints (Text)* option in the process window. After saving the project again, you can proceed to its synthesis, implementation and generation of the configuration file (**top_level.bit** or **top_level.bin**). To program the Spartan-3A system with a file generated from the project, the **ElbertV2Config** program should be used as standard. The microcontroller program should start working after the programming of the FPGA is completed.

References

- Ken Chapman – *PicoBlaze: KCPMS3 – 8-bit microcontroller for Spartan-3, Virtex-II and Virtex-II PRO*

- M. Nowakowski – *PicoBlaze. Mikroprocesor w FPGA.* Wydawnictwo BTC, Legionowo 2009.

- UG331 – Spartan-3 Generation FPGA User Guide
  https://docs.xilinx.com/v/u/en-US/ds529