

ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Sprzętowa implementacja algorytmów

10. 8-bitowy mikrokontroler PicoBlaze.

Lider projektu: Politechnika Warszawska

Autor: Łukasz Mik

Akademia Nauk Stosowanych w Tarnowie

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 Where other published and unpublished source materials have been used, these have been acknowledged.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

© Copyright 2021 - 2023 the [ENGINE](#) Consortium

Warsaw University of Technology (Poland)

International Hellenic University (IHU) (Greece)

European Lab for Educational Technology- EDUMOTIVA (Greece)

University of Padova (Italy)

University of Applied Sciences in Tarnow (Poland)

All rights reserved.



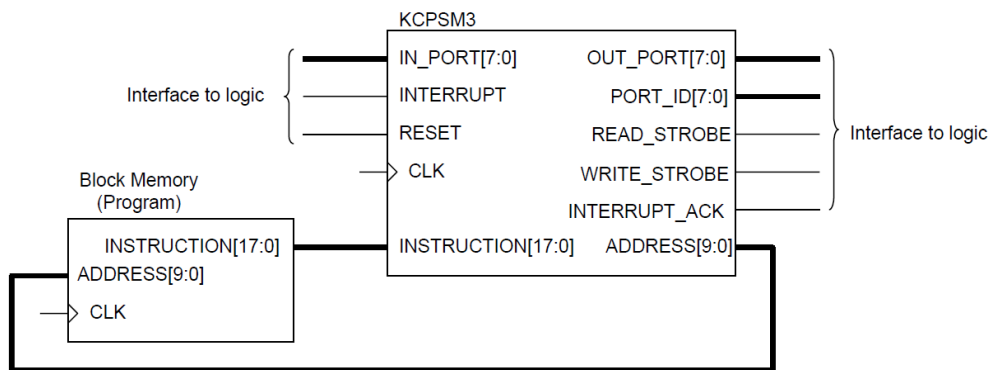
This document is licensed to the public under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Funding Disclaimer

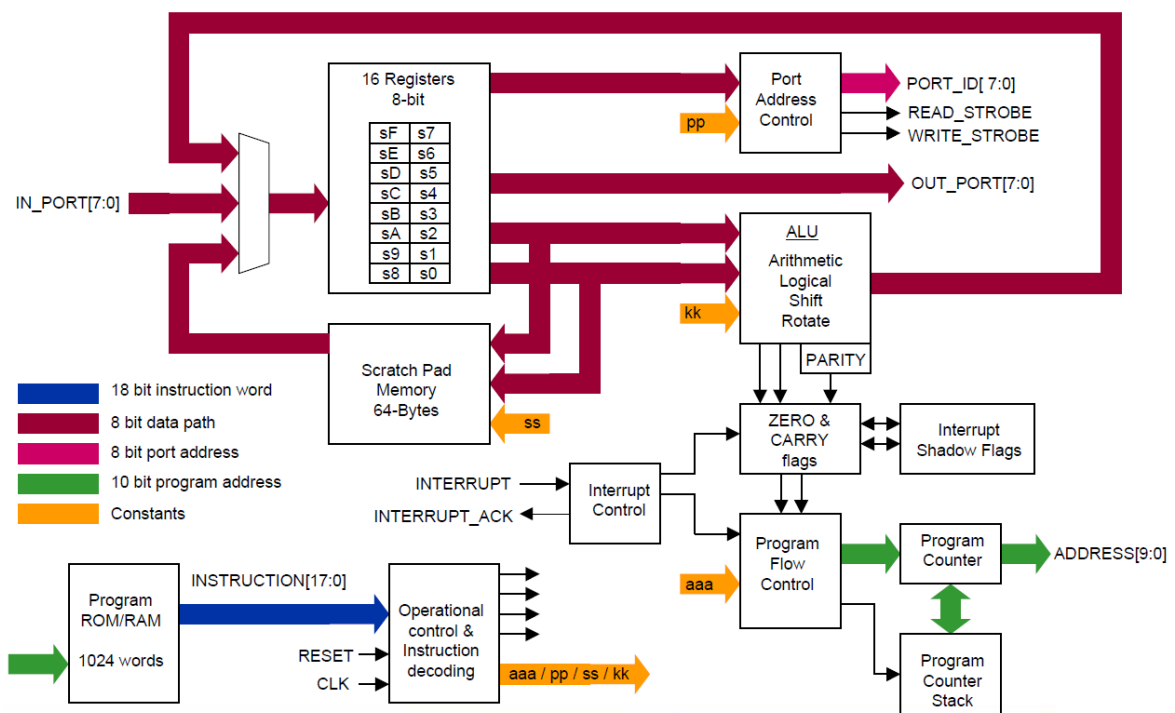
This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

I. Czym jest PicoBlaze?

PicoBlaze (KCPSM3) jest bardzo prostym 8-bitowym mikrokontrolerem dedykowanym dla układów z rodziny Spartan-3, ale jest również odpowiedni dla układów Virtex-II i Virtex-IIPRO. Jego głównym przeznaczeniem są aplikacje wymagające skomplikowanego automatu stanów, ale nie krytyczne czasowo. Dlatego nosi nazwę (K)constant Coded Programmable State Machine. Na poniższym rysunku został pokazany blok KCPSM3 wraz z pamięcią programu. Jest to wynik kompilacji projektu mikrokontrolera z plików źródłowych, udostępnionych na stronie firmy Xilinx.

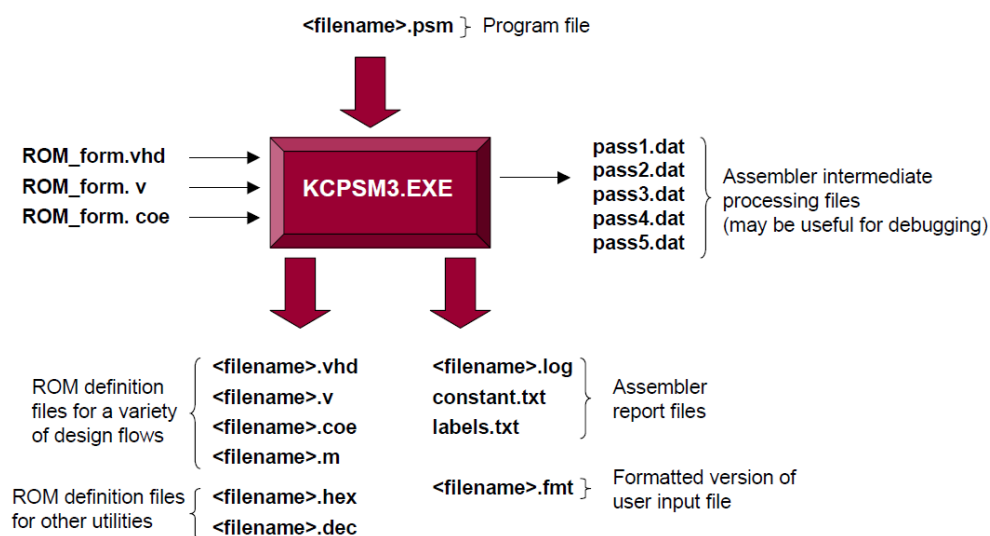


Najważniejszą cechą tego mikrokontrolera jest to, że jest całkowicie osadzony w układzie FPGA i do działania nie wymaga dodatkowych układów zewnętrznych. Wewnątrz układu FPGA można go podłączyć do dowolnego innego elementu, realizującego zaawansowane funkcje logiczne. Projekt KCPSM3 został napisany w języku VHDL. Architektura mikrokontrolera została przedstawiona na poniższym schemacie blokowym.













KCPSM3 obsługuje programy o długości do 1024 instrukcji, które zajmują pojedynczy blok pamięci ROM. W jego architekturze można wyodrębnić 16 rejestrów roboczych oznaczonych od s0 do sF, którym można przypisać swoje nazwy w języku assembler. Każdy z rejestrów można wykorzystać w pełnym zakresie, ponieważ układ nie ma wbudowanego akumulatora. Blok ALU jest odpowiedzialny za wykonywanie 8-bitowych operacji arytmetyczno-logicznych przy użyciu rejestrów roboczych. Dodatkowymi flagami ustawianymi w zależności od wyniku operacji arytmetyczno-logicznych są Carry i Zero. Statusy tych flag mogą być wykorzystane do sterowania kolejnością instrukcji, które mają być wykonywane w ramach programu i podprogramów. Do obsługi skoków do podprogramów wykorzystywany jest stos, który umożliwia zagnieżdżenie do 31 poziomów w ramach podprogramów. PicoBlaze może obsłużyć do 256 portów wejściowych i wyjściowych. Dostęp do portu odbywa się za pomocą 8-bitowego adresu, podawanego na PORT_ID. Odczytywanie danych z portu wejściowego lub wysyłanie danych na port wyjściowy odbywa się pośrednio przez jeden z rejestrów roboczych. Mikrokontroler posiada dodatkowo pamięć podręczną o pojemności 64B (Scratch Pad Memory), którą można wykorzystać do przechowywania zawartości rejestrów lub danych z portów we/wy.

Do programowania mikrokontrolera PicoBlaze używa się języka assembler, którego pełna lista instrukcji znajduje się w archiwum KCPSM3.zip (plik *KCPSM3_Manual.pdf*), do pobrania ze strony producenta. Plik źródłowy z kodem assemblera musi mieć rozszerzenie *PSM*. Plik z zawartością pamięci programu jest automatycznie generowany przez program KCPSM3.exe – kompilator języka assembler. Działa on tylko w środowisku DOS więc w trakcie zajęć będzie konieczne użycie emulatora tego systemu np. DOSBox. Sposób kompilacji plików źródłowym mikrokontrolera został przedstawiony poniżej.



II. Pobranie i przygotowanie plików źródłowych do ćwiczenia

W pierwszej kolejności pobieramy archiwum *KCPSM3.zip* ze strony Xilinx: <https://www.xilinx.com/products/intellectual-property/picoblaze.html#design>. Na tej stronie należy kliknąć link z nazwą *PicoBlaze for Spartan-3, Virtex-4, Virtex-II and Virtex-II Pro FPGAs*. Do pobrania archiwum konieczne jest zalogowanie się. W przypadku braku konta na stronie <https://xilinx.com> trzeba je utworzyć, żeby mieć dostęp do plików. Po rozpakowaniu plików z archiwum do folderu *KCPSM3*, jego zawartość powinna wyglądać tak:

 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

Podręcznik użytkownika został udostępniony przez producenta w pliku *KCPSM3_Manual.pdf*.

W kolejnym kroku tworzymy katalog o nazwie *pico_test1* w lokalizacji, gdzie do tej pory przechowywaliśmy swoje projekty np. *C:\Xilinx_work\pico_test1*. Będzie to katalog roboczy dla dalszej części tego ćwiczenia. Z folderu *Assembler* kopiujemy do folderu *pico_test1* następujące pliki:

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

Z folderu *VHDL* do folderu *pico_test1* kopiujemy plik *kcpsm3.vhd*.

III. Implementacja prostego programu w assemblerze







W ramach ćwiczenia zostanie napisany kod programu, który będzie odczytywał stan 8 przycisków typu DIP i wyświetli te stany na 8 diodach LED.

W dowolnym edytorze tekstu prosty kod programu w assemblerze, zaprezentowany na poniższym listingu:

```
; 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
```

Program składa się z niekończącej się pętli, wewnątrz której odczytywana jest wartość z portu wejściowego o adresie 00h do rejestru s0. Następnie wartość z rejestru s0 jest przesyłana na port wyjściowy o adresie 80h. Za pomocą średnika jest oznaczany komentarz w kodzie programu.

Po zakończeniu edycji kodu zapisujemy plik jako *picotest.psm*. Należy pamiętać, że nazwa pliku nie może być dłuższa niż 8 znaków. Po zapisaniu pliku zawartość katalogu *pico_test1* powinna wyglądać tak:

 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

Uwaga: jeśli korzystasz z programu Notepad, należy zmienić typ pliku z pliku tekstowego *.txt na wszystkie pliki *.*

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

W przeciwnym wypadku program dopisze do nazwy pliku *picotest.psm* rozszerzenie *.txt, co spowoduje zapisanie pliku jako *picotest.psm.txt*.

IV. Kompilacja kodu asemblera przy użyciu kompilatora KCPSM3.exe i emulatora DOSBox

Do kompilacji kodu programu dla mikrokontrolera PicoBlaze niezbędne są 4 pliki wejściowe: *picotest.psm*, *ROM_form.vhd*, *ROM_form.v*, *ROM_form.coe*. Pliki *ROM_*.** są szablonami do inicjalizacji blokowej pamięci RAM. Kompilator asemblera jest programem, który działa tylko w 32-bitowym systemie DOS. Systemy Windows w wersji 32-bitowej mogą pracować w trybie wiersza polecenia, w którym można uruchomić program KCPSM3.exe. Przy próbie uruchomienia tego programu w wierszu polecenia 64-bitowego systemu Windows, pojawi się komunikat o niezgodności systemu z 32-bitową wersją programu.

Żeby ominąć ten problem należy pobrać emulator 32-bitowego systemu DOS ze strony <http://www.dosbox.com/>. Po pobraniu, zainstalowaniu i uruchomieniu emulatora konieczne będzie zamontowanie w nim folderu *pico_test1* jako dysku, przy użyciu nie wykorzystywanej w systemie litery. Gdy uruchomi się DOSBox, pojawi się okno wiersza poleceń z domyślnie ustawionym dyskiem Z. Wpisujemy zatem polecenie:

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

Jeśli wpisana litera dysku **G** jest wolna, a folder *C:\Xilinx_work\pico_test1* istnieje to do niego zostanie przypisany wirtualny dysk G.

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

Żeby przejść na utworzony dysk wystarczy wpisać komendę **G:** a następnie komendę **dir**, aby wyświetlić zawartość katalogu głównego na tym dysku.

```
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.
```

Po wpisaniu komendy **KCPSM3 picotest.psm** nastąpi sprawdzenie składni w asemblerze a następnie kompilacja kodu programu do formatu binarnego. Po prawidłowym przejściu procesu kompilacji pojawi się komunikat KCPSM3 succesful.

```
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 succesful.

KCPSM3 complete.
```

Jeśli w jakimś fragmencie kodu programu pojawi się błąd to kompilator wyświetli informacje o tym błędzie.

```
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.
```

Każde kolejne uruchomienie asemblera powoduje nadpisanie poprzednich plików wynikowych nowymi. Po wygenerowaniu plików binarnych można zamknąć okno DOSBox wpisując komendę **exit**.

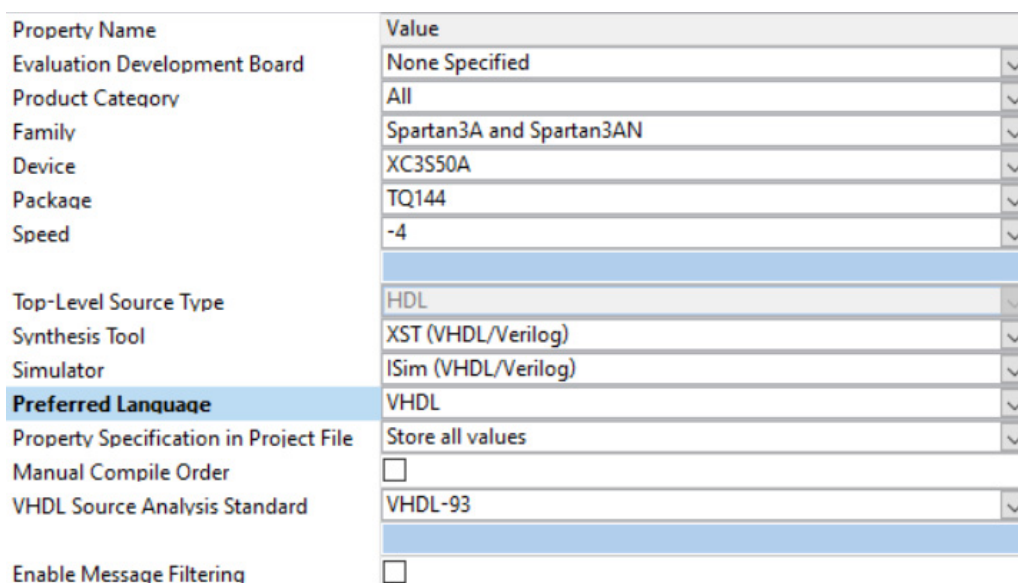
V. Implementacja PicoBlaze w układzie Spartan-3A.

Uruchamiamy *ISE Design Suite 14.7* i tworzymy nowy projekt o nazwie *pico_test1*, w folderze roboczym *Xilinx_work*. Wtedy domyślnie zostanie ona zapisany w tym samym folderze, w którym są już wygenerowane pliki mikroprocesora PicoBlaze.



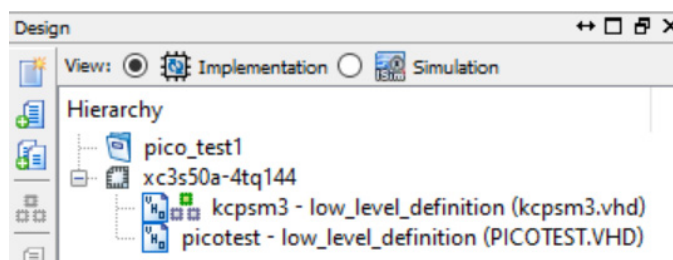
Name:	pico_test1
Location:	C:\Xilinx_work\pico_test1
Working Directory:	C:\Xilinx_work\pico_test1

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	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

Do utworzonego projektu dodajemy pliki źródłowe wybierając opcję *Project* → *Add Source...* W pierwszej kolejności dodajemy pliki *kcpsm3.vhd* i *PICOTEST.VHD*. Po ich dodaniu powinny się pojawić na liście plików w oknie projektu.



Podwójne kliknięcie na plik *PICOTEST.VHD* spowoduje otwarcie go w oknie edycyjnym programu. W części nagłówkowej tego pliku znajduje się jednostka projektowa o nazwie *picotest*, która stanowi pamięć programu mikrokontrolera.

```

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;

```

Podwójne kliknięcie na plik *kcpsm3.vhd* załaduje jego zawartość w oknie edycyjnym. W tym pliku w części nagłówkowej znajduje się deklaracja głównej jednostki projektowej mikrokontrolera.

```

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;

```

Do prawidłowego działania projektu niezbędne jest podłączenie dodanych komponentów w jedną całość za pomocą architektury najwyższego poziomu (top level). W tym celu stworzymy w projekcie nowy plik źródłowy o nazwie *top_level.vhd*. Podczas tworzenia modułu VHDL dodajemy 3 porty:

New Source Wizard

← Define Module
Specify ports for module.

Entity name: top_level

Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
SW	in	<input checked="" type="checkbox"/>	7	0
CLK	in	<input type="checkbox"/>		
LED	out	<input checked="" type="checkbox"/>	7	0
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

More Info < Back Next > Cancel

Wewnątrz architektury *top_level* dodajemy 2 komponenty: **kcpsm3** – rdzeń PicoBlaze oraz **picotest** – pamięć z programem mikrokontrolera.

```

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;

```

Niezbędne jest też dodanie sygnałów wewnątrz architektury, łączących te komponenty ze sobą i portami wejścia wyjścia jednostki projektowej (docelowo pinami układu FPGA).

```

-- 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';

```

W opisie architektury, pomiędzy słowami kluczowymi **begin** i **end** muszą zostać utworzone instancje dodanych wcześniej komponentów. Służy do tego polecenie **port map**, które wiąże ze sobą sygnały wewnątrz architektury i porty we/wy jednostki projektowej **top_level** z portami komponentów.

```

-- 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);

```

W projekcie zostaną dodane jeszcze 2 procesy: jeden do obsługi portu wejściowego mikrokontrolera i drugi do obsługi jego portu wyjściowego.

```

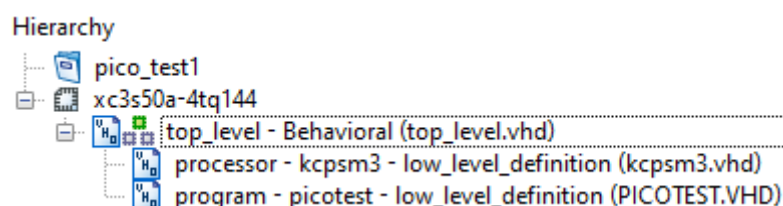
-- PicoBlaze input port at adress 00h
input_ports: process(clk)
begin
  if clk'event and clk='1' then
    case port_id(1 downto 0) is
      when "00" => in_port <= SW;
      when others => in_port <= "XXXXXXXX"; --other addresses are not used
    end case;
  end if;
end process input_ports;

-- 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;

```

Po zapisaniu zmian w projekcie zmiana w hierarchii projektu – komponenty dodane do jednostki **top_level** będą widoczne jako podrzędne.



Na tym etapie można sprawdzić poprawność składni i zawartości plików źródłowych przez uruchomienie procesu syntezy (*Synthesize - XST*). Jeśli program nie wykryje błędów to przed wygenerowaniem pliku konfiguracyjnego trzeba jeszcze powiązać porty jednostki **top_level** z pinami układu FPGA za pomocą pliku UCF. W tym celu dodajemy nowy plik źródłowy o nazwie **top_level** wybierając w oknie wyboru typu pliku opcję *Implementation Constraints File*. W oknie edycyjnym, które się otworzy zaraz po utworzeniu pliku należy wkleić poniższy fragment z przypisaniem pinów do nazw portów.

```
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;
```

Jeśli okno edycyjne samo automatycznie się nie otworzy należy zaznaczyć plik *top_level.ucf* na liście plików źródłowych projektu a następnie w oknie procesów wybrać opcję *Edit Constraints (Text)*. Po ponownym zapisaniu projektu można przystąpić do jego syntezy, implementacji i wygenerowania pliku konfiguracyjnego (*top_level.bit* lub *top_level.bin*). Do zaprogramowania układu Spartan-3A plikiem wygenerowanym z projektu należy standardowo użyć programu **ElbertV2Config**. Program mikrokontrolera powinien zacząć działać po zakończeniu programowania układu FPGA.

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>