Układy Cyfrowe – laboratorium

Przykład realizacji ćwiczenia nr 5 (wersja 2015)

1. Temat

Realizacja iteracyjnego algorytmu wyznaczania przybliżonej wartości pierwiastka funkcji. Metoda Newtona, zwana również metodą Newtona-Raphsona lub metodą stycznych, polega na przybliżaniu w kolejnych krokach algorytmu miejsca zerowego.

Metoda Newtona przyjmuje następujące założenia dla funkcji f:

- 1. W przedziale [a, b] znajduje się dokładnie jeden pierwiastek.
- 2. Funkcja ma różne znaki na krańcach przedziału, tj. $f(a) \cdot f(b) < 0$.
- 3. Pierwsza i druga pochodna funkcji mają stały znak w tym przedziale.

W pierwszym kroku metody wybierany jest punkt startowy x_0 (zazwyczaj jest to wartość a, b, 0 lub 1, z którego następnie wyprowadzana jest styczna w $f(x_0)$. Odcięta punktu przecięcia stycznej z osią OX jest pierwszym przybliżeniem rozwiązania (ozn. x_1). Jeśli to przybliżenie nie jest satysfakcjonujące, wówczas punkt x_1 jest wybierany jako nowy punkt startowy i wszystkie czynności są powtarzane. Proces jest kontynuowany, aż zostanie uzyskane wystarczająco dobre przybliżenie pierwiastka. Kolejne przybliżenia są dane rekurencyjnym wzorem: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$.

2. Zadanie

Znaleźć pierwiastek trzeciego stopnia liczby 123 z wykorzystaniem metody Newtona. Punkt startowy oraz wynik ma być zapisany 8-bitową liczbą ze znakiem. Zakładamy 10 kroków iteracji.

3. Przykładowe rozwiązanie

Pierwiastek 3. stopnia z liczby 123 można obliczyć znajdując miejsce zerowe następującej funkcji $f(x) = x^3 - 123$. Pochodna tej funkcji jest $f'(x) = 3x^2$. Stąd szukane równanie ma postać $x_1 = x_0 - \frac{x_0^3 - 123}{3x_0^2}$.

3.1. Funkcja $f_{next}(x)$

Należy zrealizować funkcję kombinacyjną $f_{next}(x) = x_0 - \frac{x_0^3 - 123}{3x_0^2}$ obliczającą następne przybliżenie miejsca zerowego. Z powyższego wzoru mamy do realizacji operacje mnożenia, dzielenia i odejmowania, pamiętając o kolejności wykonywanych operacji oraz o szerokościach ścieżek danych w kolejnych krokach.

Realizujemy kolejno: mnożenie x * x, mnożenie x * x * x czyli potęgę 3. stopnia, odejmowanie $x^3 - 123$, mnożenie $3 * x^2$, dzielenie $(x^3 - 123)/3 * x^2$ i odejmowanie $x - (x^3 - 123)/3 * x^2$. Diagram działań przedstawia rysunek:



Rys. 1. Funkcja $f_{next}(x)$

W wyniku mnożenia dwóch liczb 8-bitowych otrzymany iloczyn ma długość 16bitów. W wyniku mnożenia przez stałą +3, zwiększamy długość wyniku o 3 bity. Ponieważ założenie projektowe jest, aby wynik był 8-bitowy, należy na wyjście funkcji przekazać tylko 8 bitów z 24-bitowego wektora ilorazu. W realizacji funkcji arytmetycznych należy posłużyć się funkcjami bibliotecznymi systemu Quartus2. Założeniem projektowym jest, że funkcja $f_{next}(x)$ jest blokiem kombinacyjnym, dlatego wszystkie użyte funkcje składowe mają być blokami kombinacyjnymi.

Realizacja potęgowania x^2 z użyciem funkcji *alt_mult_add* składa się z następujących kroków:

 Tworzymy nową instancję wybierając z narzędzi Quartus2 Tools > Megawizard Plugin Manager



 Wybieramy biblioteczną funkcję mnożenia *altmult_add*, podajemy nazwę instancji *mult2.vhd* (można użyć funkcji *lpm_mult*, UWAGA! nie można użyć trybu potęgowania w funkcji LPM_MULT ponieważ źle symuluje w ModelSim "Multiply dataa input by itself (squaring operation))")

MegaWizard Plug-In Manager [page 2a]	
MegaWizard Plug-In Manager [page 2a] Which megafunction would you like to customize? Select a megafunction from the list below Installed Plug-Ins Attras SOPC Builder AttracCUMULATE AttracCUMULATE ALTRP_ADD_SUB ALTFP_ADD_SUB ALTFP_CONVERT ALTFP_CONVERT ALTFP_CONVERT ALTFP_CONVERT ALTFP_CONVERT ALTFP_LOG ALTFP_LOG ALTFP_LOG ALTFP_LOG ALTFP_LOG ALTFP_LOG ALTFP_LOG ALTFP_SQRT ALTMEMMULT ALTMULT_ACCUM (MAC) ALTMULT_ACCUM (MAC) ALTSUT_COMPLEX ALTSUT_COMPLEX ALTSUT_COMPLEX ALTSUT_COMPLEX ALTSUT_COMPLEX ALTSUT_COMPLEX ALTSUT_COMPLEX	Which device family will you be using? Which type of output file do you want to create? AHDL YHDL Verilog HDL What name do you want for the gutput file? Browse d\mult2.vhd Return to this page for another create operation Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Dytions dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu). Your current user library directories are:
LPM_ABS	
	Cancel < <u>B</u> ack <u>N</u> ext > <u>F</u> inish

 Ustawiamy szerokość sygnałów wejściowych na 8 bitów i reprezentację liczb ze znakiem *signed*, sygnał wyjściowy został automatycznie ustawiony na 16 bitów Zakład Cyberbezpieczeństwa, Instytut Telekomunikacji, Politechnika Warszawska, 2015.

MegaWizard Plug-In Manager [page 3 of 7]				
🎭 ALTMULT_ADD				
1		[<u>A</u> bout D	ocumentation
Parameter 2 EDA 3 Summary Settings				
General > Extra Modes > Saturation > Rounding > Multipliers	\geq			
		Currently selected device family:	Cyclone IJ	[🗸
mult2			🗹 Match r	project/default
	<u></u>	General	1 .	multipliers
clock0 면 datab: Sign dataa. Sign	ed ed	How wide should the <u>A</u> input buses be? How wide should the <u>B</u> input buses be?	8 🗸	bits bits
		How wide should the 'result' output bus be?	16 🗸	bits
		Create a 4th asynchronous clear input option This forces all registers to have an associated asynchronous clear input Create an associated clock gnable for each cloce	k	
		Input Representation		
		What is the representation format for A inputs? What is the representation format for B inputs?	Sig Mor Sig	ned re Options ned
Resource Usage 1 dsp_9bit + 16 reg	-	Cancel <	Eack Next	> Einish

• Wyłączamy dodatkowe rejestry wyjściowe

MegaWizard Plug-In Manager [page 4 of 9]	
🎲 ALTMULT_ADD	
V	<u>About</u> <u>D</u> ocumentation
1 Parameter 2 EDA 3 Summary Settings	
General Extra Modes Saturation Rounding Multipliers	\rangle
	Outputs Configuration
mult2	Create a shiftout output from <u>A</u> input of the last multiplier More Options
	Create a shiftout output from B input of the last multiplier
dataa_0(70) MULTU	Add extra register(s) at the output More Options
	Adder Operation
a01	What operation should be performed on
clock0 2 datas Sign	ed
	What operation should be performed on outputs of the second pair of multipliers?
	More Options
	Implementation
	Which multiplier-adder implementation should be used?
	Use the default implementation
	Use dedicated multiplier circuitry (Not available for all families)
Resource Usage	
1 dsp_9bit	Cancel < <u>B</u> ack <u>M</u> ext > <u>F</u> inish

- Strona 5. i 6. bez zmian
- Wyłączamy rejestry na wejściu i wyjściu bloku mnożącego

Zakład Cyberbezpieczeństwa, Instytut Telekomunikacji, Politechnika Warszawska, 2015.

MegaWizard Plug-In Manager [page 7 of 9]	
altmult_add	About Documentation
Parameter ZEDA Summary Settings Copyral Evits Modes Solution Devecting Multiplicat	
	ALL MULTIPLIERS
mult2	Input Configuration
dataa_0[70] MULTO	Register input <u>A</u> of the multiplier More Options
	Register input <u>B</u> of the multiplier More Options
datab: Signed dataa: Signed	What is the input A of the multiplier connected to? Multiplier input
	Use the same signal to control the source for input A of all multipliers
	What is the input B of the multiplier connected to? Multiplier input
	Use the same signal to control the source for input B of all multipliers
	Output Configuration
	Register output of the multiplier More Options
Resource Usage 1 dsp_9bit	
	Cancel < Back Next > Einish

• Informacje na temat bibliotek, w pliku *fnext.hvd* należy dodać bibliotekę *altera_mf*

MegaWizard Plug-In Manager [page 8 of 9] EDA	
ALTMULT_ADD	About Documentation
Parameter 2 EDA 3 Summary	
mult2 detae_0(7.0) detab_0(7.0) detab_0(7.0) detab: Sign detab: Sign	Simulation Libraries To properly simulate the generated design files, the following simulation model file(s) are needed File Description altera_mf Altera megafunction simulation libray Timing and resource estimation Generates a netlist for timing and resource estimation for this megafunction. If you are synthesist root source estimation for this megafunction. If you are synthesis tools support library synthesis tool, using a timing and resource estimation. Note I bitle party synthesis tools support his feature - check with the tool vendor for complete support information. Note: Netletic generation can be a time-intensive process. The size of the
Resource Usage 1 dsp_9bit	design and the speed of your system affect the time it takes for netlist generation to complet. Generate netlist Cancel < Back Next > Einish

 Zaznaczamy pola do wygenerowania plików z szablonem instancji mult2_inst.vhd oraz z komponentem użytej funkcji mult2.cmp, pliki te posłużą do skopiowania zawartości w odpowiednich sekcjach Zakład Cyberbezpieczeństwa, Instytut Telekomunikacji, Politechnika Warszawska, 2015.



Analogicznie tworzymy pozostałe bloki arytmetyczne.

Realizację bloku funkcji $f_{next}(x)$ przedstawia wydruk *fnext.vhd*:

```
library ieee;
use ieee.std logic 1164.all;
use ieee.numeric std.all;
-- Altera Megafunctions
library altera mf;
use altera_mf.altera_mf_components.all;
library lpm;
use lpm.all;
entity fnext is
      port
       (
             data : in std logic vector(7 downto 0);
             result: out std logic vector(7 downto 0)
      );
end fnext;
architecture arch fnext of fnext is
      constant sqrt3
                         : integer := 123;
      -- sygnaly blokow operacyjnych
      -- wymagane do polaczenia
      signal mult2_result : std_logic_vector(15 downto 0);
      signal mult3_result : std_logic_vector(23 downto 0);
      signal mult_3_result : std_logic_vector(18 downto 0);
      signal div_result : std_logic_vector(23 downto 0);
      signal sub result : std logic vector(23 downto 0);
      -- funkcje zewnetrzne
      component mult2
             PORT
             (
                    dataa 0 : IN STD LOGIC VECTOR (7 DOWNTO 0) := (OTHERS => '0');
```

```
datab 0 : IN STD LOGIC VECTOR (7 DOWNTO 0) := (OTHERS => '0');
                    result : OUT STD LOGIC VECTOR (15 DOWNTO 0)
             );
      end component;
      component mult3
             PORT
             (
                    dataa 0 : IN STD LOGIC VECTOR (7 DOWNTO 0) := (OTHERS => '0');
                    datab 0 : IN STD LOGIC VECTOR (15 DOWNTO 0) := (OTHERS => '0');
                    result : OUT STD LOGIC VECTOR (23 DOWNTO 0)
             );
      end component;
      component mult 3
             PORT
             (
                    dataa 0 : IN STD LOGIC VECTOR (15 DOWNTO 0) := (OTHERS => '0');
                    datab 0 : IN STD LOGIC VECTOR (2 DOWNTO 0) := (OTHERS => '0');
                    signb : IN STD LOGIC := '0';
                    result : OUT STD LOGIC VECTOR (18 DOWNTO 0)
             );
      end component;
      component div
             PORT
             (
                    denom : IN STD LOGIC VECTOR (18 DOWNTO 0);
                    numer : IN STD LOGIC VECTOR (23 DOWNTO 0);
                    quotient : OUT STD LOGIC VECTOR (23 DOWNTO 0);
                    remain : OUT STD LOGIC VECTOR (18 DOWNTO 0)
             );
      end component;
begin
      mult2 inst : mult2 PORT MAP (
             dataa 0 => data,
             datab 0 => data,
             result => mult2 result
      );
      mult3 inst : mult3 PORT MAP (
             dataa 0 => data,
             datab 0 => mult2 result,
             result => mult3 result
      );
      mult 3 inst : mult 3 PORT MAP (
             dataa 0 => mult2 result,
             datab 0 => "011",
             signb => '1', -- 1 dla signed, 0 dla unsigned
             result => mult_3_result
      );
      sub result <= std logic vector(signed(mult3 result) - to signed(sqrt3, 24));</pre>
      div inst : div PORT MAP (
             denom => mult 3 result,
             numer => sub result,
             quotient => div result --,
             --remain => remain sig
      );
      result <= std_logic_vector(signed(data) - signed(div result(7 downto 0)));</pre>
end arch fnext;
```

Kompilacja i weryfikacja

Projekt jest realizowany na płytkach uruchomieniowych DE2-115 z układem fpga CycloneIVE EP4CE115F29C7, dla którego należy ustawić projekt (ustawiamy wersje na *Revision fnext*).

Schemat blokowy (rys. 2) można uzyskać za pomocą narzędzia *Tools>Netlist Viewers>RTL Viewer*, który obrazuje diagram z rysunku 1.



Rys. 2. Schemat blokowy funkcji $f_{next}(x)$ po kompilacji

Weryfikacja poprawności działania bloku $f_{next}(x)$ jest przeprowadzona na drodze symulacji, wywołując ModelSim z programu Quartus: *Tools>Run EDA Simulation Tool >EDA RTL Simulation*. Uruchamiamy symulację dla *fnext* z biblioteki *rtl_work* dwukrotnie na nią klikając lub wykonując komendę:

ModelSim> vsim rtl_work.fnext

Wektory testowe można podać wykonując przykładowy skrypt *test_fnext.do*, który został utworzony (lub skopiowany) w podkatalogu *simulation\modelsim* i wykonany za pomocą komendy:

VSIM> do test_fnext.do

Skrypt testu przedstawia poniższy wydruk:

```
restart -nowave
add wave -radix decimal *
force data 10#0 0
run 100
force data 10#1 0
run 100
force data 10#2 0
run 100
force data 10#3 0
run 100
force data 10#4 0
run 100
force data 10#5 0
run 100
force data 10\#6 0
run 100
force data 10#7 0
run 100
force data 10#8 0
```

run 100
force data 10#9 0
run 100
force data 10#10 0
run 100
... i tak dalej dla kolejnych wektorow testowych...

Przykładowy widok wyniku testu:

*	Msgs															
∎-� /fnext/data	29	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
💶 🛶 /fnext/result			42	12	7	6	5	6	6	6	7	8	8	9	9	10
	841	0	1	4	9	16	25	36	49	64	81	100	121	144	169	196
		0	1	8	27	64	125	216	343	512	729	1000	1331	1728	2197	2744
		0	3	12	27	48	75	108	147	192	243	300	363	432	507	588
			-41	-10	-4	-2	0		1	2	2		3		4	4
Image: Image		-123	-122	-115	-96	-59	2	93	220	389	606	877	1208	1605	2074	2621
⇔≣ ⊙ Now	3 ns) ns 0.1	liiiiliiii Lns 0.2	ns 0.3	ins 0.4	huuluu Ins 0.5	ns 0.6	ns 0.7	/ns 0.8	liiiliii Ins 0.9	ns 1	huuduuu ns 1.1	liiniliini Lins 1.2	liiniliini 2ns 1.3	liiiiliii 3ns 1.4	huuluu Ins 1
Gursor 1	0.00 ns	0.00 ns														
	< >	•														
0 ps to 2 ns																

Weryfikację poprawności działania funkcji $f_{next}(x)$ można także dokonać z wykorzystaniem formularza np. MSExcel:

💌 so	🔄 sqrt3.xls [Tryb zgodności]															
	A B	С	D	E	F	G	Н		J	K	L	M	N	0	P	Q 🚡
1																
2	sqrt=	123	123	123	123	123	123	123	123	123	123	123	123	123	123	12
3	data=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
4	×2=	1	4	9	16	25	36	49	64	81	100	121	144	169	196	22
5	×3=	1	8	27	64	125	216	343	512	729	1000	1331	1728	2197	2744	337
6	×3-sqrt=	-122	-115	-96	-59	2	93	220	389	606	877	1208	1605	2074	2621	325
7	×2*3=	3	12	27	48	75	108	147	192	243	300	363	432	507	588	67
8	div=	-41	-10	-4	-2	0	0	1	2	2	2	3	3	4	4	
9	sub=	42	12	7	6	5	6	6	6	7	8	8	9	9	10	1 👻
14 4	🕨 🗛 🕹								[4						▶ 1.;;

3.2. Realizacja bloku ASMD

Diagram algorytmu przedstawiony jest na rysunku:





Wykorzystują metodę ASMD dzielimy algorytm na bloki ASM (rys. 4).



Rys. 4. Algorytm podzielony na bloki ASM

Przykładowa realizacja algorytmu podana jest na wydruku:

```
library ieee;
use ieee.std logic 1164.all;
use ieee.numeric std.all;
entity lab5 is
      port
      (
             clk, reset : in std_logic;
             start, load : in std_logic;
             data : in std_logic_vector(7 downto 0);
             result: out std logic vector(7 downto 0);
             ready : out std logic
      );
end lab5;
architecture arch lab5 of lab5 is
      type state type is (idle, s0, s1, s2, s3);
      signal state_reg, state_next : state_type;
      signal cnt reg, cnt next : unsigned(3 downto 0);
      signal in res reg, in res next : signed(7 downto 0);
      signal acc_reg, acc_next : signed(7 downto 0);
      signal ready reg, ready next : std logic;
```

```
signal fnext result : std logic vector(7 downto 0);
       component fnext
              port
              (
                     data : in std logic vector(7 downto 0);
                     result : out std_logic_vector(7 downto 0)
              );
       end component;
begin
       -- automat
       process (clk, reset)
       begin
              if reset = '1' then
                    state reg <= idle;</pre>
              elsif rising edge(clk) then
                    state reg <= state next;</pre>
              end if;
       end process;
       -- licznik
       process (clk, reset)
       begin
              if reset = '1' then
                    cnt reg <=(others => '0');
              elsif rising edge(clk) then
                    cnt reg <= cnt next;</pre>
              end if;
       end process;
       -- akumulator i rejestry
       process (clk, reset)
       begin
              if reset = '1' then
                    acc reg <=(others => '0');
              elsif rising edge(clk) then
                    acc reg <= acc next;</pre>
              end if;
       end process;
       -- rejestr wejsciowy
       process (clk, reset)
       begin
              if reset = '1' then
                    in res reg <= (others => '0');
              elsif rising edge(clk) then
                    in res reg <= in res next;</pre>
              end if;
       end process;
       -- flaga
       process (clk, reset)
       begin
              if reset = '1' then
                    ready reg <= '0';</pre>
              elsif rising edge(clk) then
                    ready_reg <= ready_next;</pre>
              end if;
       end process;
       -- tablica przejsc automatu
       process(state reg, load, start, cnt reg)
       begin
              state next <= state reg; --domyslny</pre>
```

```
case state reg is
                     when idle =>
                            state_next <= s0;</pre>
                     when s0 => -- ustawienie danych
                            if load = '1' then
                                   state next <= s1;</pre>
                             end if;
                     when s1 => -- załadowanie i start obliczeń
                             if start = '1' then
                                  state next <= s2;</pre>
                             end if;
                     when s2 => -- obliczenia
                             if cnt reg = "1001" then -- liczba iteracji
                                   state next <= s3;</pre>
                             end if;
                     when s3 => -- prezentacja wyniku, załadowanie nowych danych
                             if load = '1' then
                                   state next <= s1;</pre>
                             end if;
                     when others =>
                            state next <= idle;</pre>
              end case;
       end process;
       -- sygnaly wyjsciowe automatu Moore'a
       process(state reg, acc reg, cnt reg, in res reg, start, load, data,
ready reg, fnext result)
       begin
              acc next <= acc reg; -- wartosci
              cnt next <= cnt reg; -- domyslne</pre>
              in res next <= in res reg;</pre>
              ready next <= '0';</pre>
              case state reg is
                     when idle =>
                     when s0 \Rightarrow -- ustawienie danych
                            if load = '1' then
                                   in res next <= signed(data);</pre>
                             end if;
                     when s1 => -- załadowanie i start obliczeń
                             if start = '1' then
                                   acc next <= in res reg;</pre>
                                    cnt_next <= (others => '0');
                             end if;
                     when s2 => -- obliczenia
                            cnt next <= cnt reg + 1;</pre>
                             acc next <= signed(fnext result);</pre>
                             if cnt reg = "1001" then -- liczba iteracji
                                    ready next <= '1';</pre>
                            end if;
                     when s3 => -- prezentacja wyniku, załadowanie nowych danych
                            ready next <= ready reg;</pre>
                             if load = '1' then
                                    in res next <= signed(data);</pre>
                                    ready next <= '0';</pre>
                            end if;
                     when others =>
              end case;
       end process;
       -- sygnaly wyjsciowe ukladu
       ready <= ready_reg;</pre>
       result <= std_logic_vector(acc_reg);</pre>
       b1: fnext port map
```

Kompilacja i weryfikacja

Projekt należy stworzyć dla układu CycloneIVE EP4CE115F29C7 (ustawiamy wersje na *Revision lab5*).

Uruchamiamy symulację dla *lab5* z biblioteki *rtl_work* dwukrotnie na nią klikając lub wykonując komendę:

ModelSim> vsim rtl_work.lab5

Wektory testowe można podać wykonując przykładowy skrypt *test_lab5.do*, który został utworzony (lub skopiowany) w podkatalogu *simulation\modelsim* i wykonany za pomocą komendy:

VSIM> do test_lab5.do

Skrypt testu przedstawia poniższy wydruk:

```
restart -nowave
add wave -radix decimal *
add wave -radix unsigned cnt_reg
add wave -radix unsigned cnt_next
force clk 0 0, 1 {50 ps} -r 100
force reset 1 0, 0 100
force load 0 0, 1 200, 0 300
force start 0 0, 1 300, 0 400
force data 10#1 0
run 1400
```

Przykładowy widok wyniku testu:



3.3. Realizacja na zestawie uruchomieniowym DE2

Przyjęto założenia dotyczące sterowania układu:

- Wejście danych *data*: przełączniki *sw7..sw0*
- Wejście sterujące start: przycisk key3
- Wejście sterujące *load*: przycisk *key2*
- Wejście sygnału reset: przycisk key0
- Wyjście danych wyniku result: diody ledr17..ledr10
- Wyjście pokazujące status danych wejściowych *data*: diody *ledr7..ledr0*
- Wyjście pokazujące status sygnału ready: dioda ledg7
- Wyjście pokazujące status sygnału reset: dioda ledg0
- Wyjście pokazujące status sygnału div_clk: dioda ledg1

Projekt należy stworzyć dla układu CycloneIVE EP4CE115F29C7 (ustawiamy wersje na *Revision top_lab5*).

Należy zauważyć, że przyciski *key* w stanie zwolnionym mają wartość logiczną 1, natomiast po wciśnięciu mają wartość logiczną 0 (patrz: dokumentacja DE2-115). Moment przyciśnięcia jest sygnalizowany wygenerowaniem pojedynczego sygnału na wyjściu bloku *edge_detect*. Sygnał *sys_clk* został przypisany do sygnału zegarowego o częstotliwości 50 MHz, który następnie jest dzielony a uzyskany sygnał *div_clk* steruje pozostałymi blokami układu.

Realizacja projektu pokazana jest na wydruku:

```
library ieee;
use ieee.std logic 1164.all;
use ieee.numeric std.all;
entity top lab5 is
      generic(
             d : integer := 25 -- dlugosc podzielnika czestotltiwosci
             );
      port
      (
             sys clk, n reset : in std logic;
             start, load : in std logic;
             data : in std logic vector(7 downto 0);
             data out : out std logic vector(7 downto 0);
             result: out std logic vector(7 downto 0);
             ready : out std logic;
            div clk out : out std logic;
            reset out : out std logic
      );
end top lab5;
architecture arch lab5 of top lab5 is
      signal cnt reg, cnt next : unsigned(d-1 downto 0);
```

```
signal div clk : std logic;
       signal start tick : std logic;
      signal load_tick : std_logic;
component lab5 is
      port
       (
             clk, reset : in std_logic;
             start, load : in std_logic;
             data : in std logic vector(7 downto 0);
             result : out std_logic_vector(7 downto 0);
             ready : out std logic
      );
end component;
component edge detect is
      port(
             clk, reset : in std logic;
             level : in std logic;
             tick : out std logic
      );
end component;
begin
      -- licznik
      process (sys clk, n reset)
      begin
             if n reset = '0' then
                   cnt reg <=(others => '0');
             elsif rising edge(sys clk) then
                    cnt reg <= cnt next;</pre>
             end if;
      end process;
             process(cnt reg)
      begin
             cnt next <= cnt reg + 1;</pre>
       end process;
       -- dzielnik czestotliwosci
      div clk <= cnt reg(d-1);
       -- obserwacja danych na diodach led
      data out <= data;</pre>
       reset out <= n reset;</pre>
      div clk out <= div clk;
b1: lab5
      port map (clk => div clk, reset => not(n reset), start => start tick, load =>
load tick, data => data,
             result => result, ready => ready);
b2: edge detect port map(clk => div clk, reset => not(n reset), level =>
not(start), tick => start tick);
b3: edge detect port map(clk => div clk, reset => not(n reset), level => not(load),
tick => load tick);
end arch lab5;
```

Realizacja detektora zbocza sygnału podana jest na wydruku:

-- z ksiazki: "Fpga prototyping by VHDL examples" Pong P.Chu library ieee; use ieee.std_logic_1164.all; entity edge_detect is

```
port(
              clk, reset : in std logic;
              level : in std logic;
              tick : out std logic
       );
end edge detect;
architecture edge detect arch of edge detect is
       type state type is (zero, edge, one);
       signal state reg, state next : state type;
begin
       process (clk, reset)
       begin
              if reset = '1' then
                    state reg <= zero;</pre>
              elsif rising edge(clk) then
                     state reg <= state next;</pre>
              end if;
       end process;
       process(state_reg, level)
       begin
              state next <= state reg;</pre>
              tick <= '0';
              case state reg is
                     when zero =>
                            if level='1' then
                                  state next <= edge;</pre>
                            end if;
                     when edge =>
                            tick <= '1';
                            if level = '1' then
                                   state next <= one;</pre>
                            else
                                   state next <= zero;</pre>
                            end if;
                     when one =>
                            if level='0' then
                                   state next <= zero;</pre>
                            end if;
              end case;
       end process;
end edge detect arch;
```

Po kompilacji należy rozlokować sygnały na płytce za pomocą narzędzia *Assignments>Pin Planner*. Lista przypisań sygnałów do nóżek układu (patrz: dokumentacja DE2-115) z pliku *top_lab5.qsf*:

```
set_location_assignment PIN_AB28 -to data[0]
set_location_assignment PIN_AC28 -to data[1]
set_location_assignment PIN_AC27 -to data[2]
set_location_assignment PIN_AD27 -to data[3]
set_location_assignment PIN_AC26 -to data[4]
set_location_assignment PIN_AD26 -to data[6]
set_location_assignment PIN_AD26 -to data[7]
set_location_assignment PIN_N21 -to load
set_location_assignment PIN_G21 -to ready
set_location_assignment PIN_J15 -to result[0]
set_location_assignment PIN_H16 -to result[1]
set_location_assignment PIN_J16 -to result[2]
```

```
set location assignment PIN H17 -to result[3]
set location assignment PIN F15 -to result[4]
set location assignment PIN G15 -to result[5]
set location assignment PIN G16 -to result[6]
set location assignment PIN H15 -to result[7]
set_location_assignment PIN_R24 -to start
set_location_assignment PIN_Y2 -to sys clk
set_location_assignment PIN_E21 -to reset out
set location assignment PIN E22 -to div clk out
set location assignment PIN M23 -to n reset
set_location_assignment PIN_G19 -to data out[0]
set_location_assignment PIN_F19 -to data out[1]
set location assignment PIN E19 -to data out[2]
set location assignment PIN F21 -to data out[3]
set location assignment PIN F18 -to data out[4]
set location assignment PIN E18 -to data out[5]
set location assignment PIN J19 -to data out[6]
set location assignment PIN H19 -to data out[7]
```

Literatura i materiały pomocnicze:

- 1. Plansze do wykładu UCYF
- Literatura podana na wykładzie, ze szczególnym uwzględnieniem rozdz. 6 książki "Programowalne układy przetwarzania sygnałów i informacji"
- 3. Dokument PDF: "DE2-115 User Manual"

Opracowanie wewnętrzne ZCB IT, PW, grudzień 2015: dr inż. P. Tomaszewicz