Układy Cyfrowe – laboratorium

Przykład realizacji ćwiczenia nr 6 (wersja 2015)

1. Wstęp

1.1. Algorytm

Realizacja algorytmu wyznaczania wyrazów ciągu $F(n)$ w języku VHDL z zastosowaniem podziału projektu na moduły: FSM i Data Path.

Definicja 1. Niech będzie dany ciąg:

$$F(0) = a;\ F(1) = b; F(n) = F(n-1) + F(n-2) dla\ n > 1.$$

Dla $F(0) = 0$ i $F(1) = 1$ będzie to ciąg Fibonacciego.

Uwaga 1. Fibonacci – Leonardo z Pizy żył w latach 1170-1250. Jacques Philippe Maria Binet (1786-1856) podał wzór jawny służący do wyznaczania *n*-tego wyrazu ciągu:

$$F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

1.2. Sekwencyjna specyfikacja algorytmu w języku CPP

```
void fib ( int count , int x , int y ) {
	int first = y , second = x ; // rejestry , multipleksery
	while ( count--) // dekrementacja , porównanie z 0
	{
		int tmp = first + second ; // sumator
		first = second ; // multipleksery
		second = tmp ; // multipleksery
		printf ( „%d , ” , second ) ;
	}
}
```

Uwaga 2. Implementacja ta ma złożoność czasową $O(n)$. Istnieje inna, bardziej efektywna metoda wyznaczania *n*-tego elementu ciągu, w czasie $O(\log(n))$.

1.3. Specyfikacja algorytmu w języku VHDL

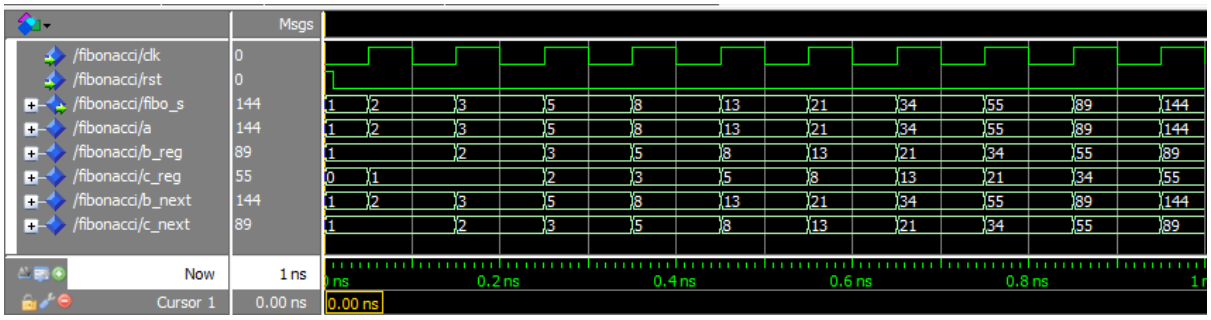
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity fibonacci is
	generic(N :integer :=16);
	port(
		clk, rst	: in std_logic;
```

```
              fibo_s       : out std_logic_vector(N-1 downto 0)
);
end fibonacci;
architecture fib_arch of fibonacci is
       signal a :unsigned(N-1 downto 0);
       signal b_reg :unsigned(N-1 downto 0);
       signal b_next :unsigned(N-1 downto 0);
       signal c_reg :unsigned(N-1 downto 0);
       signal c_next :unsigned(N-1 downto 0);
       begin
       process(clk, rst)
       begin
              if(rst='1')then
                     b_reg <= to_unsigned(1, N);
                     c_reg <= to_unsigned(0, N);
              elsif(clk'event and clk='1')then
                     c_reg <= c_next;
                     b_reg <= b_next;
              end if;
       end process;
       process(a, b_reg, c_reg)
       begin
              b_next <= a;
              c_next <= b_reg;
              a <= b_reg + c_reg;
       end process;
       fibo_s <= std_logic_vector(a);
end fib_arch;
```

Symulacja

Skrypt *test_fibo.do*

```
restart -nowave -force
add wave clk
add wave rst
add wave -radix unsigned fibo_s
add wave -radix unsigned a
add wave -radix unsigned b_reg
add wave -radix unsigned c_reg
add wave -radix unsigned b_next
add wave -radix unsigned c_next
force clk 0 0, 1 {50 ps} -r 100
force rst 1 0, 0 10
run 1000
```



2. Implementacja

W implementacji dokładamy możliwość swobodnego zdefiniowania wartości elementów początkowych $F(0)$ i $F(1)$ oraz liczby wykonanych kroków.

2.1. Moduł *Data Path*

```
library ieee;
use ieee.std_logic_1164.all;

entity dp is
generic(
	N : integer := 4
);
port(
	rst, clk	: in  std_logic;
	first, second, count	: in  std_logic_vector(N-1 downto 0);
	first_sel, second_sel : in  std_logic;
	first_load, second_load, counter_load : in  std_logic;
	comp_out	: out std_logic;
	output : out std_logic_vector(N-1 downto 0)
);
end dp;

architecture dp_arch of dp is

component mux is
generic(
	N : integer :=4
);
port(
	sel	: in std_logic;
	x, y	: in  std_logic_vector(N-1 downto 0);
	output : out std_logic_vector(N-1 downto 0)
);
end component;

component reg is
generic(
	N : integer := 4
);
port(
	rst, clk, load		:in  std_logic;
	input			:in  std_logic_vector(N-1 downto 0);
	output			:out std_logic_vector(N-1 downto 0)
```

```
);
end component;

component add is
generic(
	N : integer := 4
);
port(
	x, y   : in std_logic_vector(N-1 downto 0);
	output : out std_logic_vector(N-1 downto 0)
);
end component;

component comp is
generic(
	N : integer := 4
);
port(
	x            :in  std_logic_vector(N-1 downto 0);
	output       :out std_logic
);
end component;

component counter is
generic(
	N : integer := 4
);
port(
	clk, rst  :in  std_logic;
	input  :in  std_logic_vector(N-1 downto 0);
	load   :in  std_logic;
	output :out std_logic_vector(N-1 downto 0)
);
end component;

	signal first_reg, second_reg, add_out, counter_r, first_r, second_r :
std_logic_vector(N-1 downto 0);
begin
	b1:    mux generic map(N => N) port map(sel => first_sel, x => first, y =>
second_r, output => first_reg);
	b2:    mux generic map(N => N) port map(sel => second_sel, x => second, y =>
add_out, output => second_reg);
	b3: reg generic map(N => N) port map(rst => rst, clk => clk, load =>
first_load , input => first_reg, output => first_r);
	b4: reg generic map(N => N) port map(rst => rst, clk => clk, load =>
second_load , input => second_reg, output => second_r);
	b5: add generic map(N => N) port map(x => first_r , y => second_r, output =>
add_out);
	b6: counter generic map(N => N) port map(rst => rst, clk => clk, input =>
count, load => counter_load , output => counter_r);
	b7:    comp generic map(N => N) port map(x => counter_r, output => comp_out);
	output <= second_r;
end dp_arch;
```

2.2. Komponenty modułu Data Path

Multiplekser

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
generic(
	N : integer :=4
);
port(
	sel    : in std_logic;
	x, y   : in  std_logic_vector(N-1 downto 0);
	output : out std_logic_vector(N-1 downto 0)
);
end mux;

architecture mux_arch of mux is
begin
	output <= x when (sel = '0') else
		y;
end mux_arch;
```

Rejestr

```
library ieee;
use ieee.std_logic_1164.all;

entity reg is
generic(
	N : integer := 4
);
port(
	rst, clk, load      : in std_logic;
	input  : in std_logic_vector(N-1 downto 0);
	output : out std_logic_vector(N-1 downto 0)
);
end reg;

architecture reg_arch of reg is
	signal reg_reg, reg_next : std_logic_vector(N-1 downto 0);
begin
	process(rst, clk)
	begin
		if rst = '1' then
			reg_reg <= (others => '0');
		elsif rising_edge(clk) then
			reg_reg <= reg_next;
		end if;
	end process;

	process(load, input, reg_reg)
	begin
		if load = '1' then
			reg_next <= input;
		else
			reg_next <= reg_reg;
```

```
            end if;
      end process;

      output <= reg_reg;
end reg_arch;
```

Sumator

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add is
generic(
      N : integer := 4
);
port(
      x, y   : in std_logic_vector(N-1 downto 0);
      output : out std_logic_vector(N-1 downto 0)
);
end add;

architecture add_arch of add is
begin
      output <= x + y;
end add_arch;
```

Licznik

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
generic(
      N : integer := 4
);
port(
      clk, rst  :in  std_logic;
      input  :in  std_logic_vector(N-1 downto 0);
      load   :in  std_logic;
      output :out std_logic_vector(N-1 downto 0)
);
end counter;

architecture cnt_arch of counter is
      signal counter_reg, counter_next : unsigned(N-1 downto 0);
begin
      process(rst, clk)
      begin
            if rst = '1' then
                  counter_reg <= (others => '0');
        elsif rising_edge(clk) then
                  counter_reg <= counter_next;
            end if;
    end process;
```

```
	process(load, input, counter_reg)
	begin
		if load = '1' then
			counter_next <= unsigned(input);
		elsif counter_reg = 0 then
			counter_next <= counter_reg;
		else
			counter_next <= counter_reg - 1;
		end if;
	end process;

    output <= std_logic_vector(counter_reg);
end cnt_arch;
```

Komparator

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity comp is
generic(
	N : integer := 4
);
port(
	x		: in std_logic_vector(N-1 downto 0);
	output : out std_logic
);
end comp;

architecture comp_arch of comp is
begin
	process(x)
	begin
		if x = 0 then
			output <= '0';
		else
			output <= '1';
		end if;
    end process;
end comp_arch;
```

2.3. Moduł FSM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fsm is
port(
	rst, clk	: in std_logic;
	comp, start  : in std_logic;
	first_sel, second_sel : out  std_logic;
	first_load, second_load, counter_load : out  std_logic;
```

```
      ready  : out std_logic
);
end fsm;

architecture fsm_arch of fsm is
      type state_type is (init, s1, s2, last);
      signal state_reg, state_next : state_type;
begin
      process(rst, clk)
      begin
            if rst = '1' then
                  state_reg <= init;
            elsif rising_edge(clk) then
                  state_reg <= state_next;
            end if;
      end process;

      process(state_reg, comp, start)
      begin
            ready <= '0';

            first_sel <= '0';
            second_sel <= '0';
            first_load <= '0';
            second_load <= '0';
            counter_load <= '0';

            case state_reg is
                  when init =>
                        if start = '0' then
                              state_next <= init;
                        else
                              state_next <= s1;
                        end if;
                  when s1 =>
                        first_load <= '1';
                        second_load <= '1';
                        counter_load <= '1';
                        state_next <= s2;
                  when s2 =>
                        first_sel <= '1';
                        second_sel <= '1';
                        first_load <= '1';
                        second_load <= '1';

                        if comp = '1' then
                              state_next <= s2;
                        else
                              state_next <= last;
                        end if;
                  when last =>
                        ready <= '1';
                        state_next <= init;
                  when others =>
                        state_next <= init;
      end case;
```

```
  end process;
end fsm_arch;
```

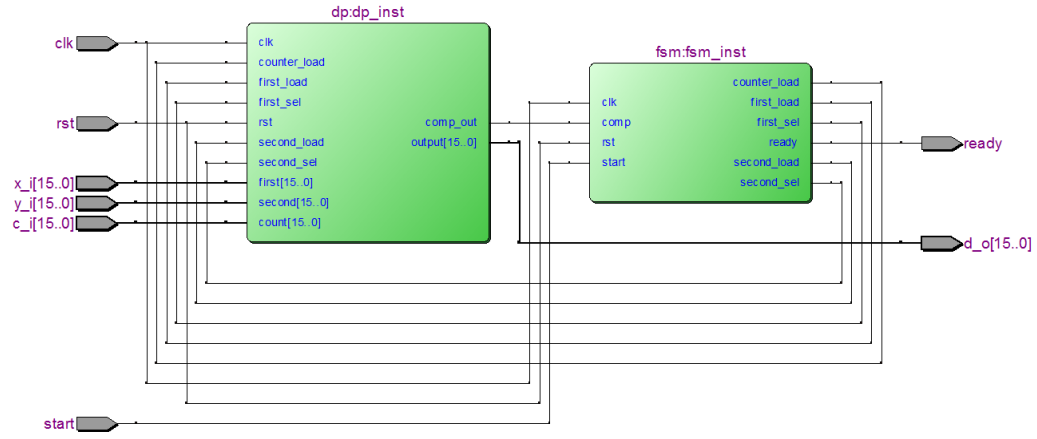
2.4. Integracja modułów FSM i Data Path

```
library ieee;
use ieee.std_logic_1164.all;

entity fib is
generic(
      N : integer := 16
);
port(
      rst, clk, start     : in std_logic;
      x_i, y_i, c_i: in std_logic_vector(N-1 downto 0);
      ready : out std_logic;
      d_o    : out std_logic_vector(N-1 downto 0)
);
end fib;

architecture fib_arch of fib is

component fsm is
port(
      rst, clk     : in std_logic;
      comp, start  : in std_logic;
      first_sel, second_sel : out  std_logic;
      first_load, second_load, counter_load : out  std_logic;
      ready  : out std_logic
);
end component;

component dp is
generic(
      N : integer := 4
);
port(
      rst, clk     : in  std_logic;
      first, second, count      : in  std_logic_vector(N-1 downto 0);
      first_sel, second_sel : in  std_logic;
      first_load, second_load, counter_load : in  std_logic;
      comp_out     : out std_logic;
      output : out std_logic_vector(N-1 downto 0)
);
end component;

      signal f_load, s_load, c_load, f_sel, s_sel, comp :std_logic;
begin

      fsm_inst :fsm port map(rst => rst, clk => clk, comp => comp, start => start,
first_sel => f_sel, second_sel => s_sel, first_load => f_load, second_load =>
s_load, counter_load => c_load, ready => ready);

      dp_inst : dp generic map(N => N) port map(rst => rst, clk => clk, first =>
x_i, second => y_i, count => c_i, first_sel => f_sel, second_sel => s_sel,
```

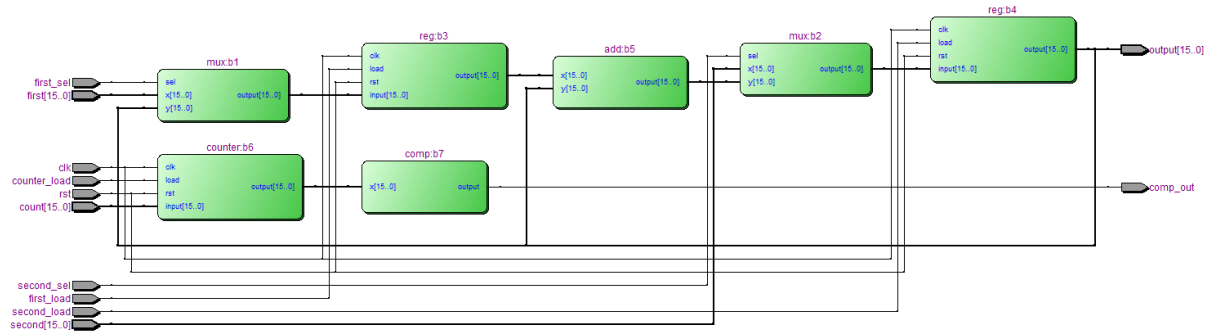
```
first_load => f_load, second_load => s_load, counter_load => c_load, comp_out =>
comp, output => d_o);

end fib_arch;
```

3. Wyniki kompilacji i symulacji

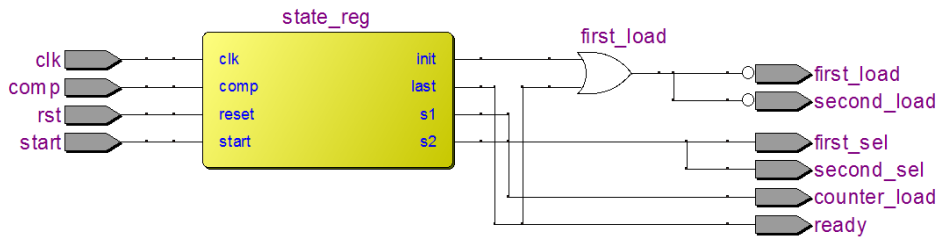
Moduł FIB



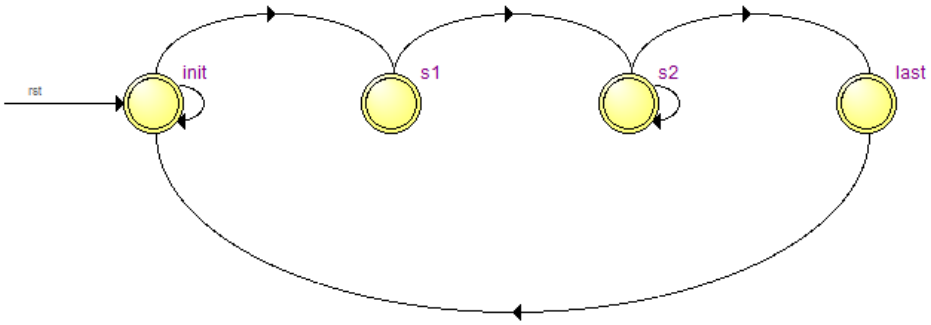
Moduł Data Path



Moduł FSM



Realizacja automatu



State Table

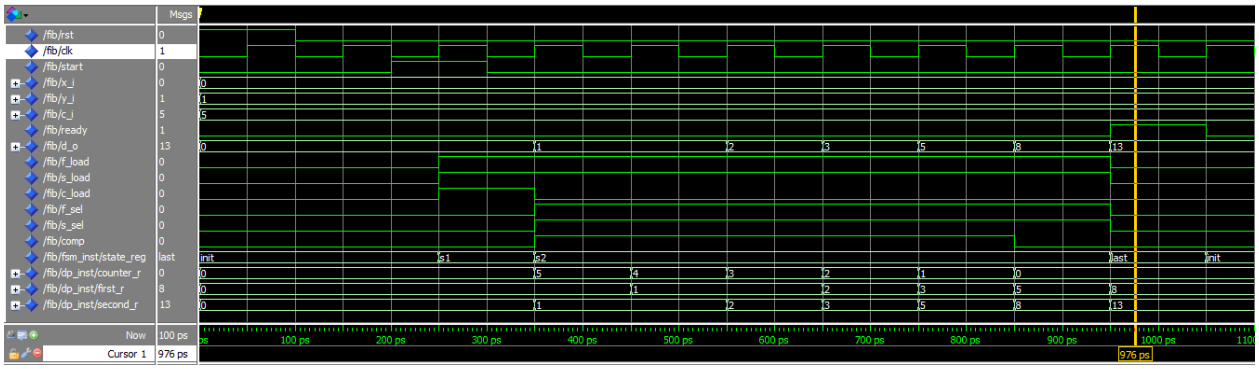
	Source State	Destination State	Condition
1	init	init	(!start)
2	init	s1	(start)
3	last	init	
4	s1	s2	
5	s2	last	(!comp)
6	s2	s2	(comp)

Transitions | Encoding

Symulacja

Skrypt *test_fib.do*

```
restart -nowave
add wave -radix unsigned *
add wave /fib/fsm_inst/state_reg
add wave -radix unsigned \
/fib/dp_inst/counter_r  /fib/dp_inst/first_r  /fib/dp_inst/second_r
force clk 0 0, 1 {50 ps} -r 100
force rst 1 0
force start 0 0
force x_i 10#0 0
force y_i 10#1 0
force c_i 10#5 0
run
force rst 0 0
run
force start 1 0
run
force start 0 0
run 800
```



Literatura i materiały pomocnicze:

1. Plansze do wykładu UCYF
2. 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:
P. Sapiecha, P. Tomaszewicz