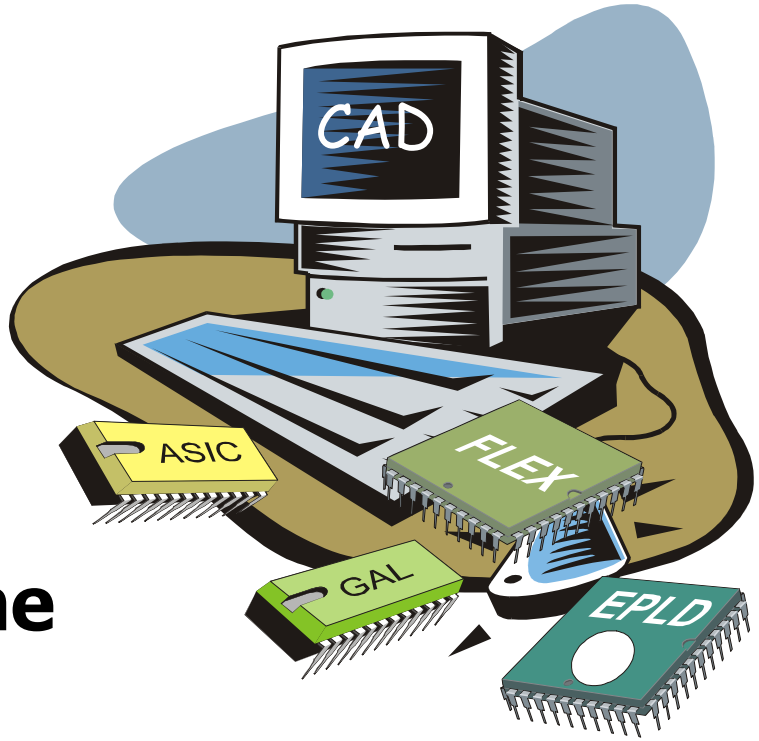
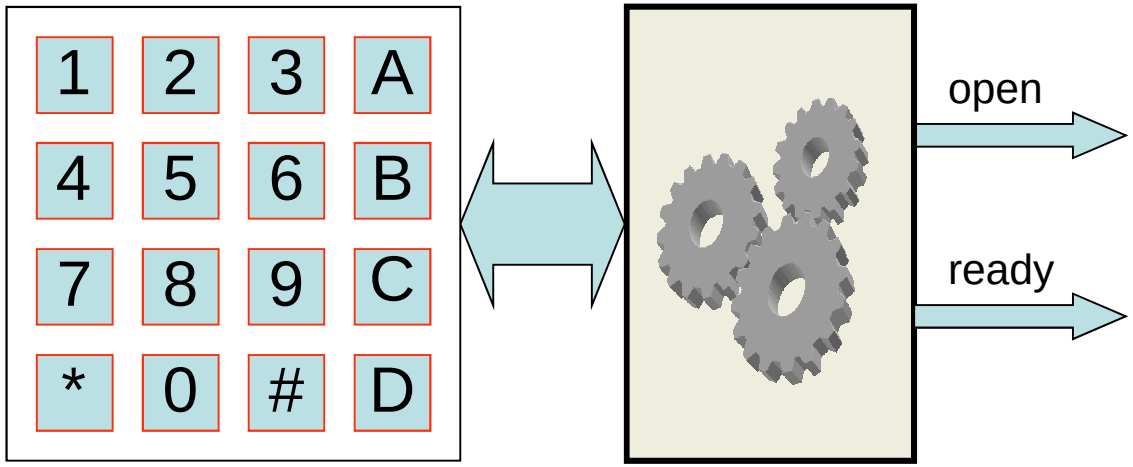
Projektowanie hierarchiczne



Mariusz Rawski
m.rawski@tele.pw.edu.pl
http://rawski.zcb.tele.pw.edu.pl/

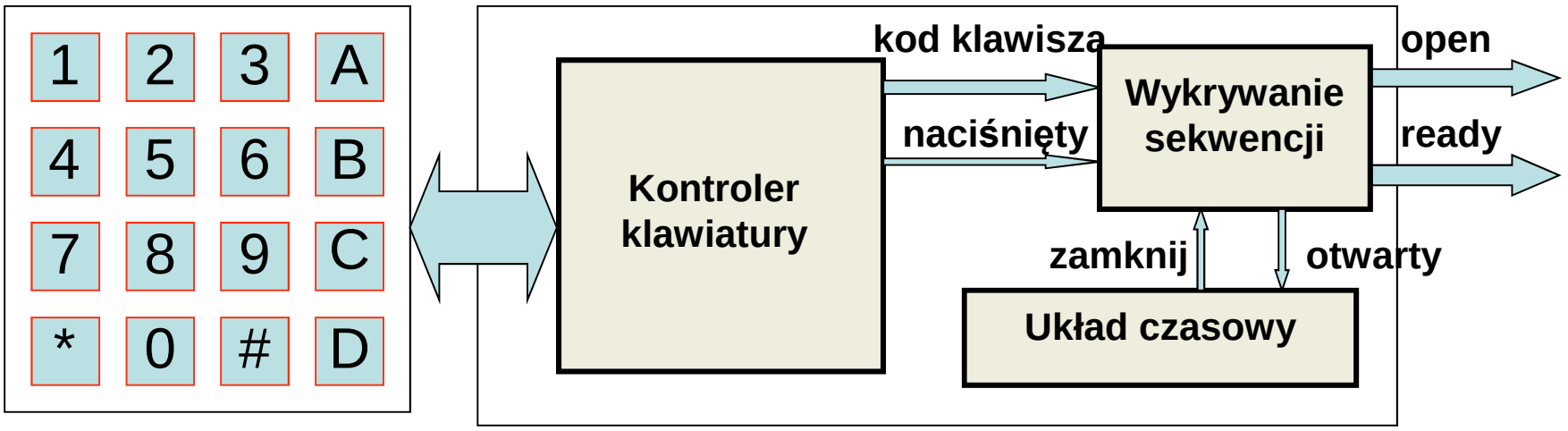
Elektroniczny zamek

- Zamek elektroniczny:
  - ***kod*** – 4 cyfrowy kod wprowadzony z klawiatury
  - ***ready*** – sygnalizacja gotowości pracy, rozpoczęcie wprowadzania kodu zmienia ten sygnał na nieaktywny
  - ***open*** – sygnał otwarcia zamka
- Wprowadzenie poprawnego kodu otwiera zamek na określony czas
- Sterowanie klawiaturą numeryczna



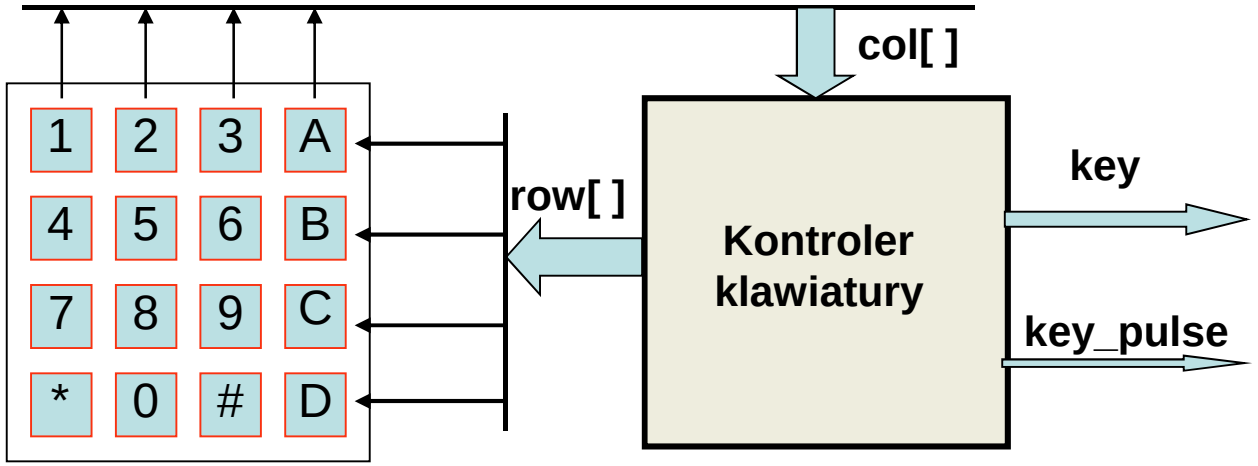
Elektroniczny zamek – schemat blokowy

- Kontroler klawiatury:
  - podaje sygnał naciśnięcia klawisza
  - podaje kod naciśniętego klawisza
- Wykrywanie sekwencji:
  - analizuje kody klawiszy i wykrywa sekwencje kodową otwierającą zamek
- Układ czasowy:
  - odmierza określony czas po otwarciu zamka i wysyła sygnał zamknięcia zamka



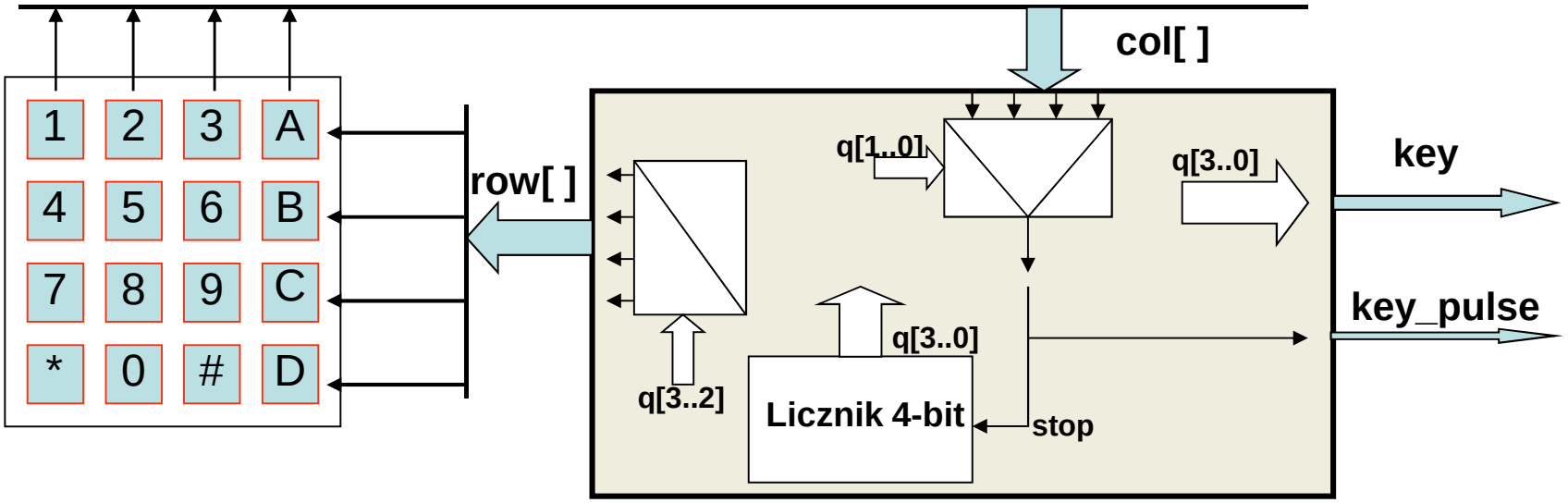
Kontroler klawiatury

- Kontroler klawiatury:
  - podanie sygnału aktywnego na linie kolejnych wierszy
  - sprawdzanie linii kolumn w celu wykrycia sygnału aktywnego pochodzącego ze zwarcia styków klawisza



Kontroler klawiatury – schemat blokowy

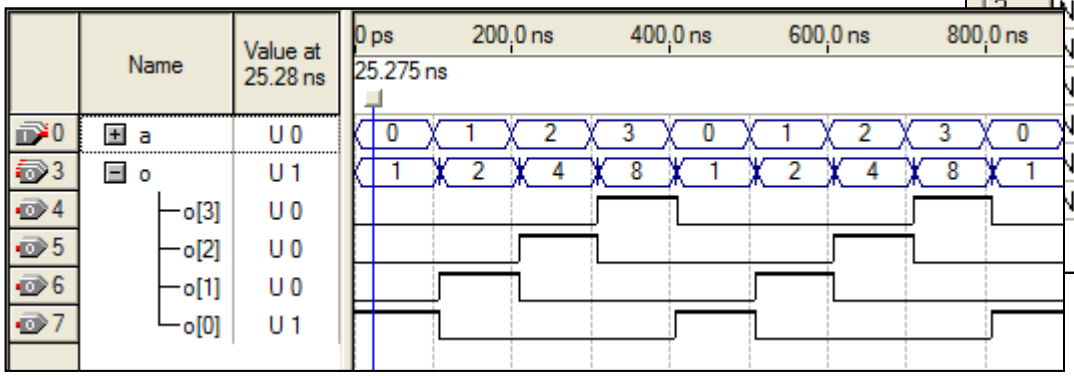
- Kontroler klawiatury:
  - licznik generuje liczby 4 bitowe,
  - 2 najstarsze bity sterują dekoderem, wystawiając sygnał aktywny na kolejne linie wierszy klawiatury
  - 2 młodsze bity sterują multiplekserem „przeglądając” kolejne linie kolumn klawiatury
  - wystąpienie na wyjściu multipleksera sygnału aktywnego sygnalizuje naciśnięcie klawisza – zatrzymuje licznik – kodem klawisza jest wartość licznika –może się okazać konieczne przekodowanie



Dekoder

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

entity dec_2x4 is
        port(
           a    : in std_logic_vector(1 downto 0);
           o    : out std_logic_vector(3 downto 0)
        );
end dec_2x4;
architecture data_flow of dec_2x4 is
begin
        with a select
        o  <= "0001" when "00",
              "0010" when "01",
              "0100" when "10",
              "1000" when "11";
end data_flow;
```

Fitter Summary	
Top-level Entity Name	dec_2x4
Family	Stratix
Device	EP1S10F484C5
Total logic elements	4 / 10,570 (< 1 %)
Total pins	6 / 336 (2 %)

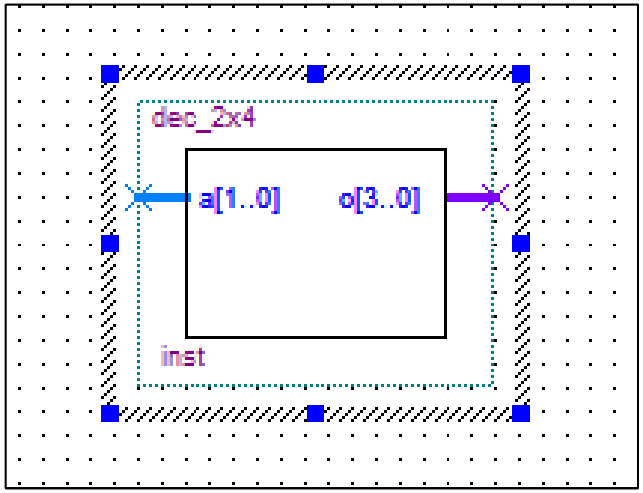


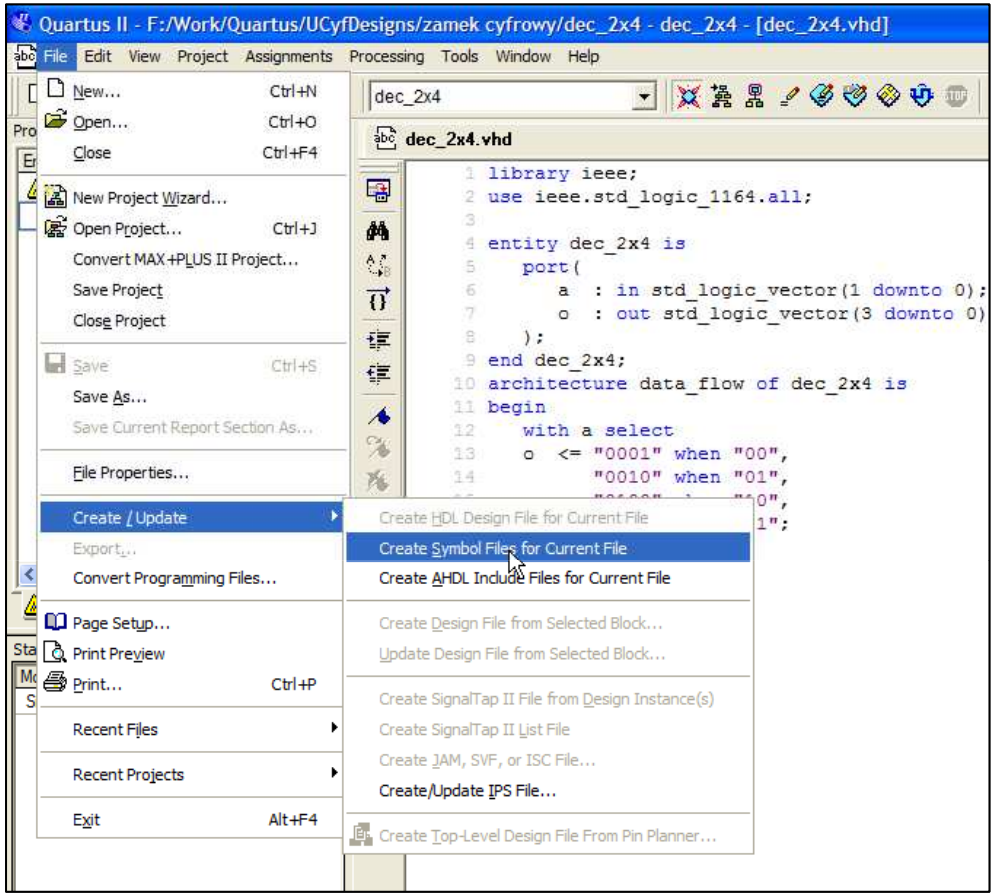
Registered Performance | tpd | tsu | tco | th | Custom Delays

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	8.699 ns	a[0]	o[0]
2	N/A	None	8.694 ns	a[0]	o[3]
	N/A	None	8.694 ns	a[0]	o[1]
	N/A	None	8.691 ns	a[0]	o[2]
	N/A	None	8.416 ns	a[1]	o[1]
	N/A	None	8.411 ns	a[1]	o[0]
	N/A	None	8.410 ns	a[1]	o[2]
	N/A	None	8.406 ns	a[1]	o[3]

Dekoder – umieszczenie w bibliotece modułów

- Stworzenie nowego elementu bibliotecznego reprezentującego zaprojektowany dekoder
- Element reprezentowany jest przez symbol o takiej nazwie jak nazwa projektu
- Wykorzystywany w pliku graficznym





Multiplekser

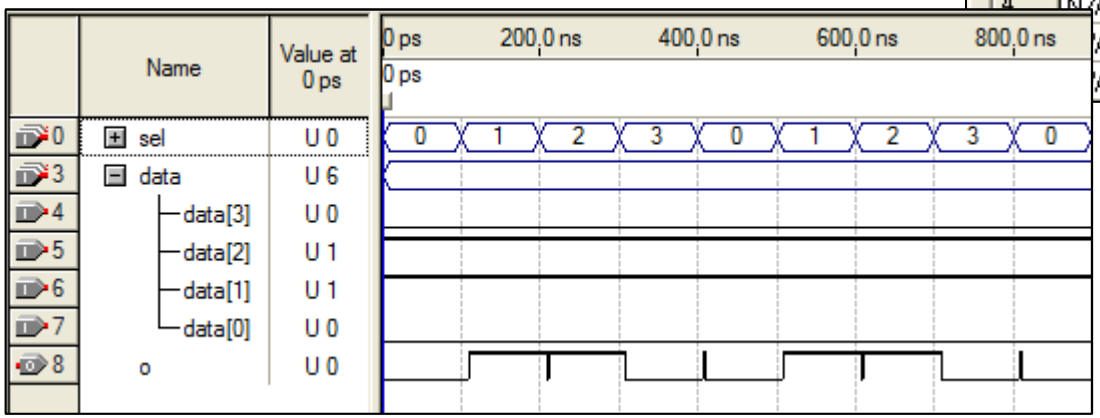
```
library ieee;
use ieee.std_logic_1164.all;

entity mux_2x4 is
        port(
           sel      : in std_logic_vector(1 downto 0);
           data     : in std_logic_vector(3 downto 0);
           o        : out std_logic
        );
end mux_2x4;
architecture data_flow of mux_2x4 is
begin
        with sel select
        o  <= data(0) when "00",
                data(1) when "01",
                data(2) when "10",
                data(3) when "11";
end data_flow;
```

Fitter Summary

Top-level Entity Name	mux_2x4
Family	Stratix
Device	EP1S10F484C5
Total logic elements	2 / 10,570 (< 1 %)
Total pins	7 / 336 (2 %)

Registered Performance | tpd | tsu | tco | th | Custom Delays

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	9.338 ns	sel[0]	o
2	N/A	None	9.234 ns	data[0]	o
3	N/A	None	9.196 ns	sel[1]	o
4	N/A	None	9.046 ns	data[1]	o
	A	None	8.681 ns	data[3]	o
	A	None	8.344 ns	data[2]	o

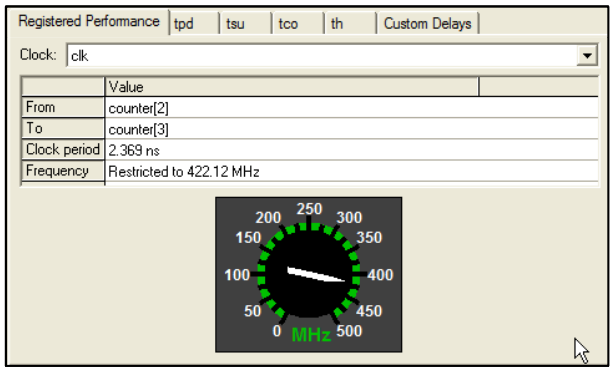


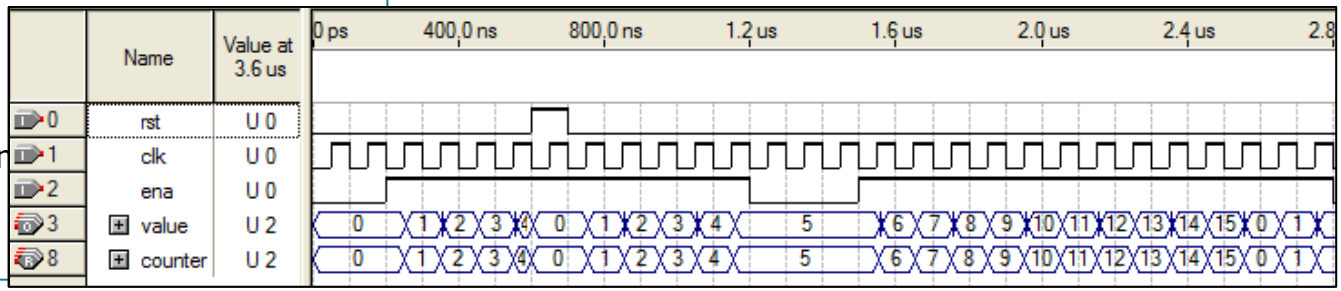
Licznik 4 bitowy

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

entity cnt_4 is
      port(
         rst      : in std_logic;
         clk      : in std_logic;
         ena      : in std_logic;
         value    : out std_logic_vector(3 downto 0)
      );
E`nd cnt_4;
architecture data_flow of cnt_4 is
      signal counter     : unsigned(3 downto 0);
begin
      process(rst, clk)
      begin
         if rst = '1' then
              counter <= "0000";
         elsif rising_edge(clk) then
              if ena = '1' then
                   counter <= counter + 1;
              else
                   counter <= counter;
              end if;
         end if;
      end process;
      value <= std_logic_vector(coun
end data_flow;
```

Fitter Summary

Top-level Entity Name	cnt_4
Family	Stratix
Device	EP1S10F484C5
Total logic elements	5 / 10,570 (< 1 %)
Total pins	7 / 336 (2 %)





Mariusz Rawski

Licznik parametryzowany

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

entity cnt_n is
        generic(
            WIDTH : natural := 8
        );
        port(
            rst       : in std_logic;
            clk       : in std_logic;
            ena       : in std_logic;
            value     : out std_logic_vector(WIDTH - 1 downto 0)
        );
end cnt_n;
architecture data_flow of cnt_n is
        signal counter : unsigned(WIDTH - 1 downto 0);
begin
        process(rst, clk)
        begin
            if rst = '1' then
                counter <= (others => '0');
            elsif rising_edge(clk) then
                if ena = '1' then
                    counter <= counter + 1;
                else
                    counter <= counter;
                end if;
            end if;
        end process;
        value <= std_logic_vector(counter);
end data_flow;
```

Umieszczenie nazwy WIDTH w sekcji GENERIC pozwala na ustalenie tej wartości przez użytkownika

Wielkość licznika zależy od wartości parametru WIDTH

Wartości parametru WIDTH decyduje o budowie licznika

Zapis (others => '0') pozwala uniezależnić się od wielkości licznika

Zalety parametryzacji modułów

- Stworzenie modułu parametryzowanego jest równoznaczne ze stworzeniem szablonu, na podstawie którego można tworzyć wiele bloków funkcjonalnych o różnej budowie
- W odróżnieniu od stałych (CONSTANTS) możliwe jest jednoczesne stworzenie różnych modułów wykorzystywanych w tym samym projekcie
- Do opisu modułów parametryzowanych wykorzystuje się często struktury języka VHDL pozwalające generować warunkowo kod modułu

```
generate_label:
for index_variable in range_low to range_high generate
        statement1;
        statement2;
end generate;
```

```
generate_label:
if expression generate
        statement1;
        statement2;
end generate;
```

Licznik parametryzowany modulo k (1)

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

entity cnt_mod_k is
        generic(
            MOD_VALUE: natural := 10
        );
        port(
            rst         : in std_logic;
            clk         : in std_logic;
            mode        : in std_logic_vector(2 downto 0);
            value       : out std_logic_vector(log2_ceil(MOD_VALUE) - 1 downto 0)
        );
end cnt_mod_k;
architecture data_flow of cnt_mod_k is
        constant    WIDTH  : natural := log2_ceil(MOD_VALUE);

        constant STOP       : std_logic_vector(2 downto 0) := "000";
        constant UP         : std_logic_vector(2 downto 0) := "001";
        constant DOWN       : std_logic_vector(2 downto 0) := "010";
        constant RESET      : std_logic_vector(2 downto 0) := "100";

        signal counter_reg      : unsigned(WIDTH - 1 downto 0);
        signal counter_next     : unsigned(WIDTH - 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;
```

Umieszczenie nazwy MOD_VALUE w sekcji GENERIC pozwala na ustalenie tej wartości przez użytkownika

Funkcja ‘log2_ceil’ jest zdefiniowana w pakiecie ‘my_std_lib’

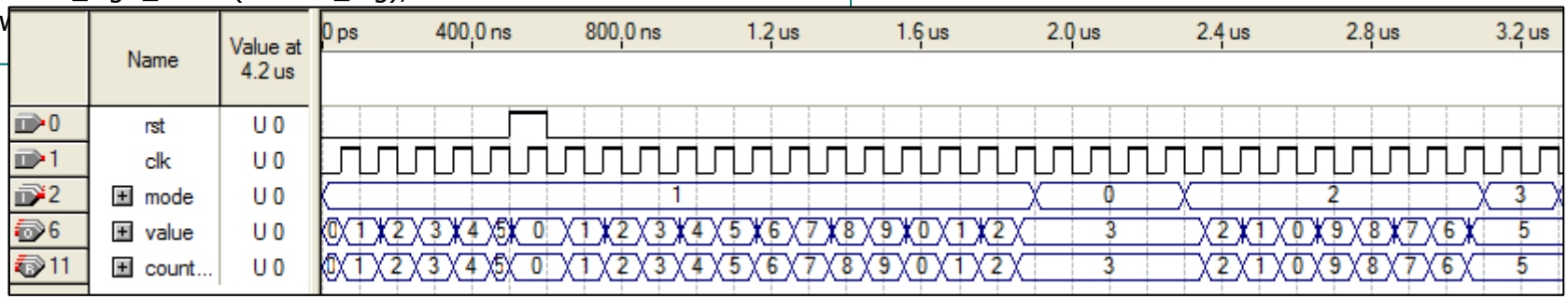
Wartości parametru WIDTH decyduje o budowie licznika i jest obliczana na podstawie MOD_VALUE

Stale opisujące tryby pracy

Licznik parametryzowany modulo k (2)

```
process(counter_reg, mode)
begin
   case mode is
        when STOP =>
             counter_next <= counter_reg;
        when UP =>
             if counter_reg = MOD_VALUE - 1 then
                  counter_next <= (others => '0');
             else
                  counter_next <= counter_reg + 1;
             end if;
        when DOWN =>
             if counter_reg = 0 then
                  counter_next <= to_unsigned(MOD_VALUE-1, WIDTH) ;
             else
                  counter_next <= counter_reg - 1;
             end if;
        when RESET =>
             counter_next <= (others => '0');
        when others =>
             counter_next <= counter_reg;
   end case;
end process;
value <= std_logic_vector(counter_reg);
end data_flow
```

Wartość parametru wykorzystywana jest do prawidłowego opisu modułu



Dekoder parametryzowany

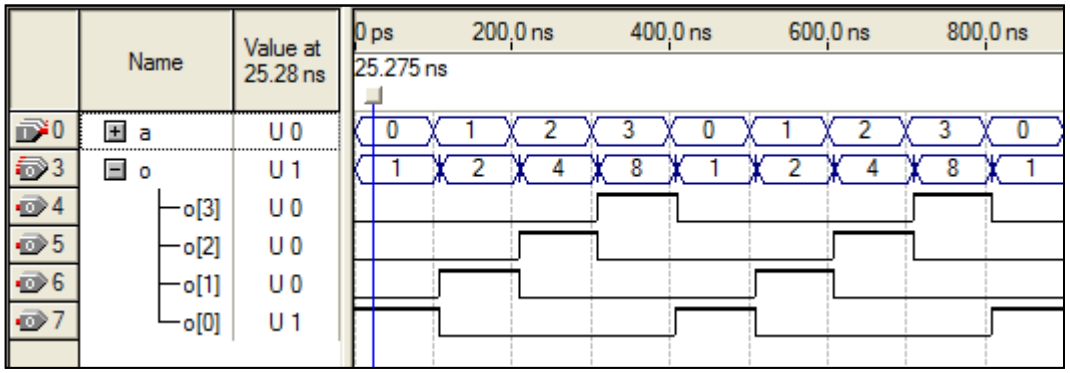
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dec_n is
        generic(
           WIDTH : natural     := 2
        );
        port(
           a    : in   std_logic_vector(WIDTH - 1 downto 0);
           o    : out std_logic_vector(2**WIDTH - 1 downto 0)
        );
end dec_n;
architecture data_flow of dec_n is
begin
        dec_gen:
        for i in 0 to 2**WIDTH - 1 generate
           o(i) <= '1' when a = std_logic_vector(to_unsigned(i, WIDTH)) else '0'
        end generate;
end data_flow;
```

Umieszczenie nazwy WIDTH w sekcji GENERIC pozwala na ustalenie tej wartości przez użytkownika

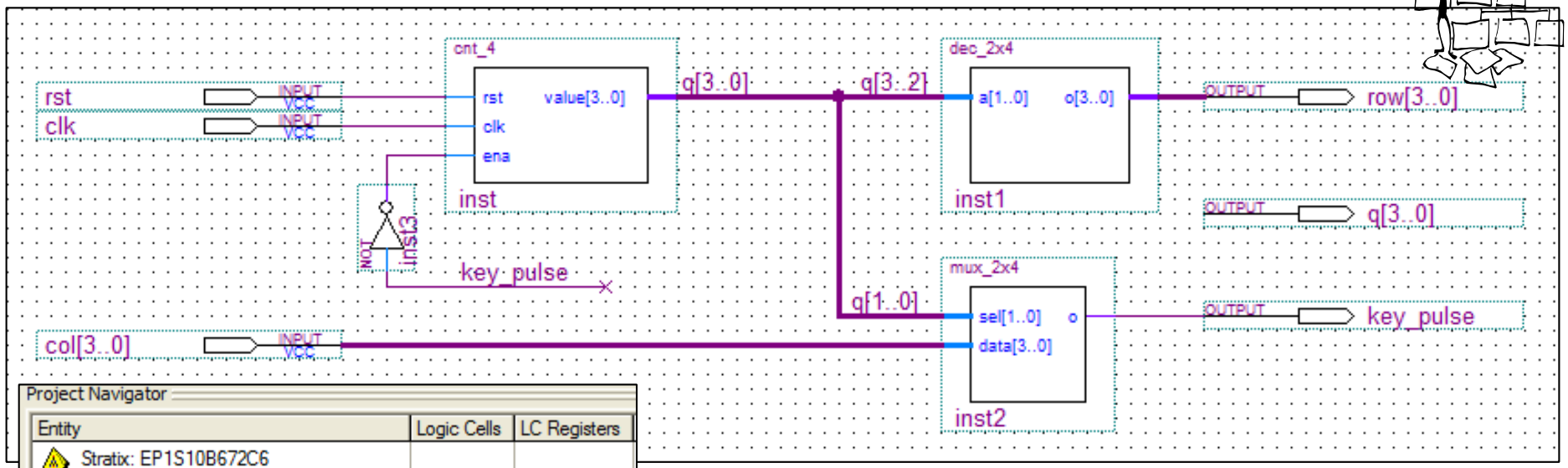
Wyjście ‘o’ ma „szerokość” równą 2 WIDTH

Wykorzystanie konstrukcji warunkowej generacji kodu do opisu działania modułu:

```
o(0) <= '1' when a = std_logic_vector(to_unsigned(0, WIDTH)) else '0';
o(1) <= '1' when a = std_logic_vector(to_unsigned(1, WIDTH)) else '0';
o(2) <= '1' when a = std_logic_vector(to_unsigned(2, WIDTH)) else '0';
o(3) <= '1' when a = std_logic_vector(to_unsigned(3, WIDTH)) else '0';
```



Sterownik klawiatury – realizacja



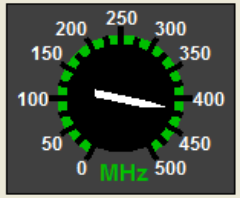
Project Navigator

Entity	Logic Cells	LC Registers
Stratix: EP1S10B672C6		
kb_ctrl_sch	11 (0)	4
cnt_4:inst	5 (5)	4
dec_2x4:inst1	4 (4)	0
mux_2x4:inst2	2 (2)	0

Registered Performance | tpd | tsu | tco | th | Custom Delays

Clock: clk

	Value
From	cnt_4:inst|counter[1]
To	cnt_4:inst|counter[2]
Clock period	2.369 ns
Frequency	Restricted to 422.12 MHz

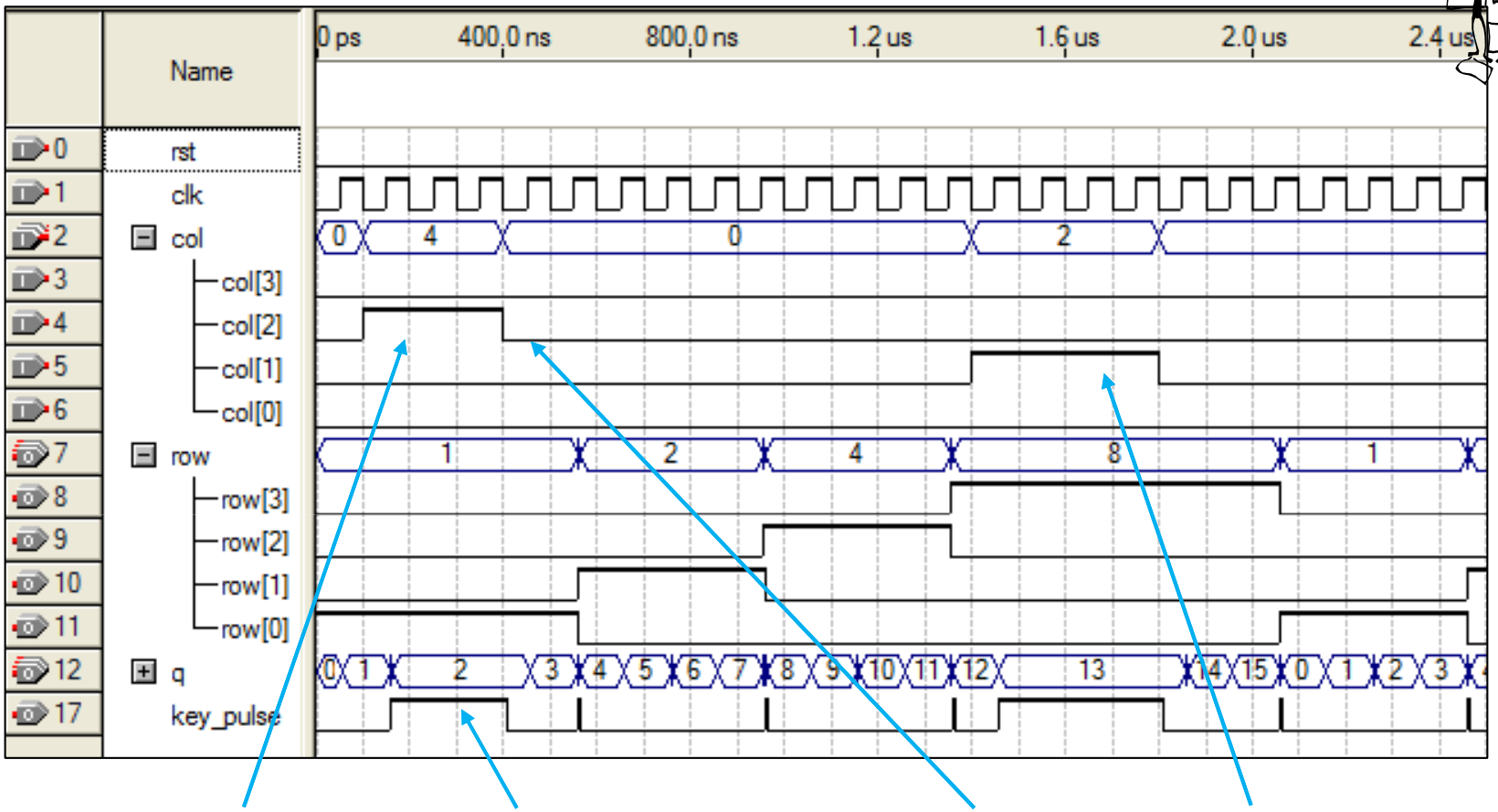


Fitter Summary

Top-level Entity Name	kb_ctrl_sch
Family	Stratix
Device	EP1S10B672C6
Total logic elements	11 / 10,570 (< 1 %)
Total pins	15 / 346 (4 %)

Mariusz Rawski

Sterownik klawiatury – działanie



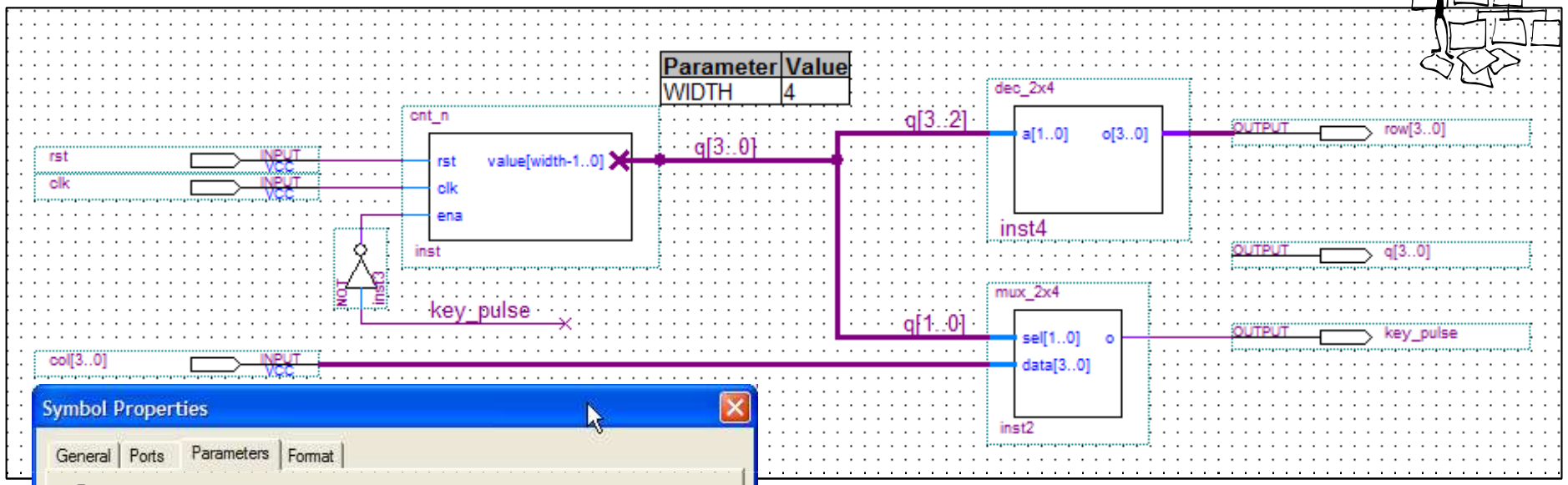
Naciśnięcie klawisza row = 0, col = 2

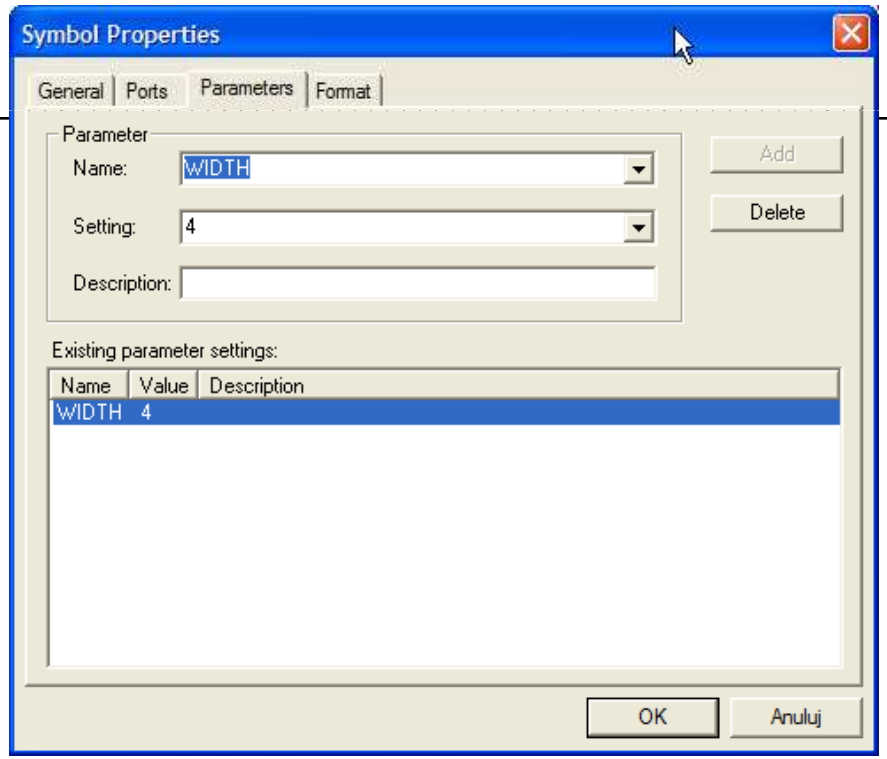
Zatrzymanie licznika i wygenerowanie sygnały naciśnięcia klawisza ‘key_pulse’. Numer klawisza dostępny na ‘q’

Po zwolnieniu klawisza dalsze skanowanie klawiatury

Naciśnięcie klawisza row = 3, col = 1

Sterownik klawiatury – moduły parametryzowane



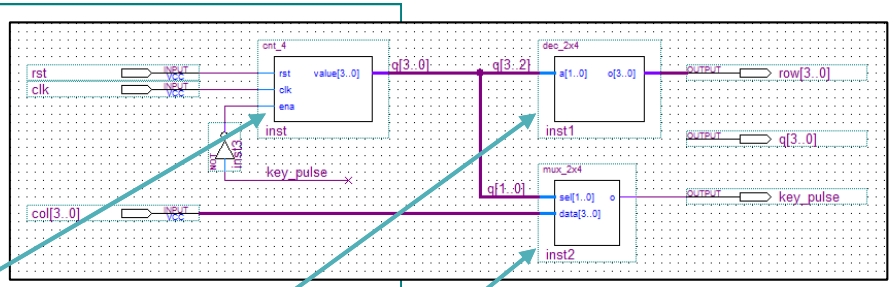


Fitter Summary

Top-level Entity Name	kb_ctrl_p_sch
Family	Stratix
Device	EP1S10B672C6
Total logic elements	11 / 10,570 (< 1 %)
Total pins	15 / 346 (4 %)

Sterownik klawiatury – opis VHDL (1)

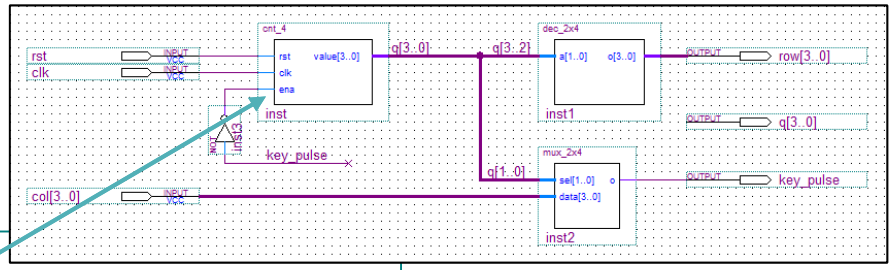
```
library ieee;
use ieee.std_logic_1164.all;

entity kb_ctrl is
        port(
            rst             : in std_logic;
            clk             : in std_logic;
            col             : in std_logic_vector(3 downto 0);
            key_pulse       : out std_logic;
            row             : out std_logic_vector(3 downto 0);
            q               : out std_logic_vector(3 downto 0)
        );
end kb_ctrl;
architecture data_flow of kb_ctrl is
        component cnt_4
            port(
                rst         : in std_logic;
                clk         : in std_logic;
                ena         : in std_logic;
                value       : out std_logic_vector(3 downto 0)
            );
        end component;

        component dec_2x4
            port(
                a           : in std_logic_vector(1 downto 0);
                o           : out std_logic_vector(3 downto 0)
            );
        end component;

        component mux_2x4
            port(
                sel         : in std_logic_vector(1 downto 0);
                data        : in std_logic_vector(3 downto 0);
                o           : out std_logic
            );
        end component;
```



Deklaracja komponentów wykorzystanych w danym module

Sekcja port w deklaracji komponentu musi odpowiadać sekcji port w pliku definicji danego modułu

Sterownik klawiatury – opis VHDL (2)

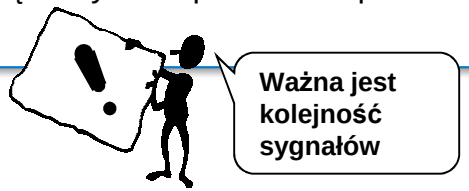



```
        signal key_pulse_i    : std_logic;
        signal q_i            : std_logic_vector(3 downto 0);

begin
        kb_cnt : cnt_4
           port map(
                rst   => rst,
                clk   => clk,
                ena   => (not key_pulse_i),
                value     => q_i
           );

        kb_mux : mux_2x4
           port map(
                q_i(1 downto 0),
                col,
                key_pulse_i
           );

        kb_dec : dec_2x4
           port map(
                a     => q_i(3 downto 2),
                o     => row
           );

        q <= q_i;
        key_pulse <= key_pulse_i;
end data_flow;
```

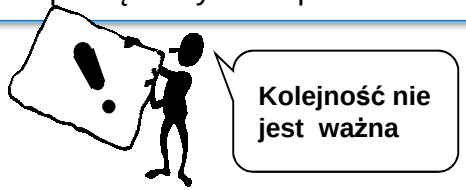
Sygnały wewnętrzne potrzebne do połączenia komponentów

Konkretyzacja komponentu i mapowanie odzwierciedlające połączenie portów komponentu z otoczeniem

Możliwe jest mapowanie przez podanie sygnałów podłączanych do portów bez podania nazw portów

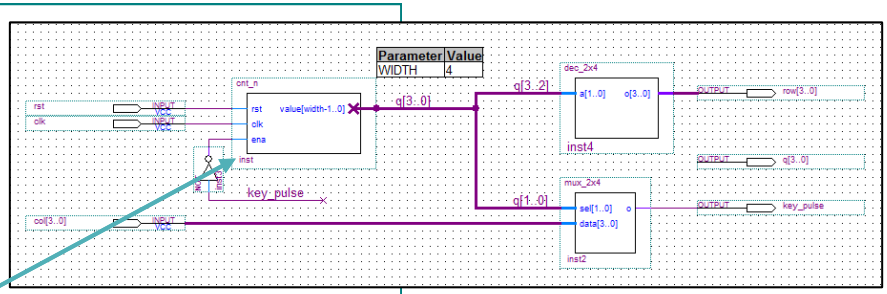


Możliwe jest mapowanie przez podanie nazwy portów i sygnałów podłączanych do portów



Wykorzystanie modułów parametryzowanych (1)

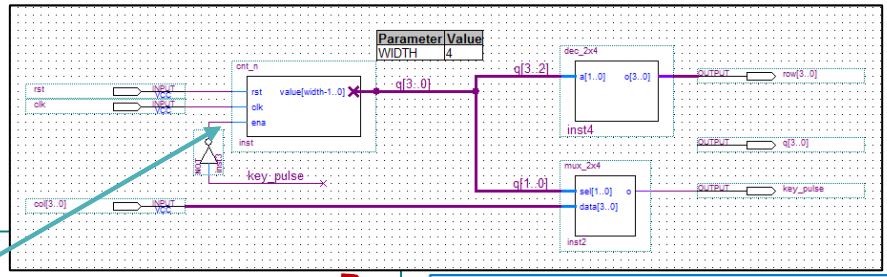
```
library ieee;
use ieee.std_logic_1164.all;

entity kb_ctrl is
        port(
            rst             : in std_logic;
            clk             : in std_logic;
            col             : in std_logic_vector(3 downto 0);
            key_pulse       : out std_logic;
            row             : out std_logic_vector(3 downto 0);
            q               : out std_logic_vector(3 downto 0)
        );
end kb_ctrl;
architecture data_flow of kb_ctrl is
        component cnt_n
        generic(
            WIDTH   : natural := 8
        );
        port(
            rst         : in std_logic;
            clk         : in std_logic;
            ena         : in std_logic;
            value       : out std_logic_vector(WIDTH - 1 downto 0)
            );
        end component;

        component dec_2x4
            port(
                a           : in std_logic_vector(1 downto 0);
                o           : out std_logic_vector(3 downto 0)
            );
        end component;

        •••
```



Deklaracja komponentów wykorzystanych w danym module

Komponent parametryzowany

Wykorzystanie modułów parametryzowanych (2)




```
        signal key_pulse_i    : std_logic;
        signal q_i            : std_logic_vector(3 downto 0);

begin
        kb_cnt : cnt_n
        generic map(
            WIDTH  => 4
        )
        port map(
            rst   => rst,
            clk   => clk,
            ena   => (not key_pulse_i),
            value     => q_i
        );

        kb_mux : mux_2x4
        port map(
            q_i(1 downto 0),
            col,
            key_pulse_i
        );

        •••
```

Sygnały wewnętrzne potrzebne do połączenia komponentów

Konkretyzacja komponentu i mapowanie odzwierciedlające połączenie portów komponentu z otoczeniem jak również mapowanie parametrów

Wykorzystanie pakietów (1)

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

package digit_lock is
        component cnt_n
        generic(
            WIDTH   : natural := 8
        );
        port(
            rst         : in std_logic;
            clk         : in std_logic;
            ena         : in std_logic;
            value       : out std_logic_vector(WIDTH - 1 downto 0)
            );
        end component;

        component mux_2x4
            port(
                sel   : in std_logic_vector(1 downto 0);
                data: in std_logic_vector(3 downto 0);
                o           : out std_logic
            );
        end component;

        component dec_2x4
            port(
                a     : in std_logic_vector(1 downto 0);
                o     : out std_logic_vector(3 downto 0)
            );
        end component;
end digit_lock;

package body digit_lock is

end digit_lock;
```

Plik ‘digit_lock’ definiuje pakiet

Deklaracja komponentów

Ciało pakietu – puste w tym przypadku

Wykorzystanie pakietów (2)

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

entity kb_ctrl_pckg is
        port(
            rst             : in std_logic;
            clk             : in std_logic;
            col             : in std_logic_vector(3 downto 0);
            key_pulse       : out std_logic;
            row             : out std_logic_vector(3 downto 0);
            q               : out std_logic_vector(3 downto 0)
        );
end kb_ctrl_pckg;

architecture data_flow of kb_ctrl_pckg is

        signal key_pulse_i    : std_logic;
        signal q_i            : std_logic_vector(3 downto 0);

begin
        kb_cnt : cnt_n
            generic map(
                WIDTH   => 4
            )
            port map(
                rst   => rst,
                clk   => clk,
                ena   => (not key_pulse_i),
                value     => q_i
            );

        •••
```

Wykorzystanie pakietu ‘digit_lock’

Nie ma potrzeby deklaracji komponentów

Pozostała część bez zmian