Custom Computing & Embedded Systems

VHDL – opis układów sekwencyjnych

1. **Zakres tematyczny ćwiczenia:**
   - opis układów sekwencyjnych w języku VHDL,
   - wykorzystanie podprogramów w symulacji,
   - symulacja z wykorzystaniem testbencha VHDL.

2. **Modelowanie układów sekwencyjnych**

W opisie behawioralnym układów sekwencyjnych elementem niezbędnym jest proces. Jest to konstrukcja języka VHDL, która jako całość jest instrukcją współbieżną, natomiast rozkazy wewnątrz procesu uruchamiane są sekwencyjne – zgodnie z kolejnością zapisania.
W czasie symulacji proces może zostać wyzwolony na podstawie listy czułości lub przy pomocy instrukcji 'wait', jednak oba te elementy nie mogą występować w składni procesu równocześnie.

2.1. Opis przerzutnika

Najprostsze elementy sekwencyjne to przerzutniki i zatrzaski. Różnica między nimi polega na sposobie wyzwalania: zatrzask (latch) wyzwalany jest poziomem sygnału a przerzutnik (flip-flop) zboczem sygnału.

Inputs		Outputs	
D	*EN*	*Q*	$\overline{Q}$
0	1	0	1
1	1	1	0
X	0	Q_0	$\overline{Q}_0$

Inputs		Outputs	
D	CLK	*Q*	$\overline{Q}$
0	↑	0	1
1	↑	1	0

↑ = clock transition LOW to HIGH

```
9  entity dlatch is
10     generic (delay: time:=1 ns);
11     Port     (en : in std_logic;
12               rst : in std_logic;
13               d : in std_logic;
14               q : out std_logic);
15 end entity dlatch;
16
17 architecture latch of dlatch is
18
19 begin
20 process(en,rst,d)
21     begin
22     if rst='1' then
23         q<='0' after delay;
24     elsif en='1' then
25         q<=d after delay;
26     end if;
27     end process;
28 end architecture latch;
```

```
9  entity dff is
10     generic (delay: time:=1 ns);
11     Port     (clk : in std_logic;
12               rst : in std_logic;
13               d : in std_logic;
14               q : out std_logic);
15 end entity dff;
16
17 architecture flip_flop of dff is
18
19 begin
20 process(clk,rst)
21     begin
22     if rst='1' then
23         q<='0' after delay;
24     elsif clk='1' and clk'event then
25         q<=d after delay;
26     end if;
27     end process;
28 end architecture flip_flop;
```

Elementy zostały opisane z wykorzystaniem procesu – zwróć uwagę na zawartość listy czułości w obu przypadkach.

[!] Utwórz nowy projekt symulatora Questa i dodaj do projektu pliki .vhd z serwera kursu
[!] Wykonaj kompilację z linii poleceń rozkazem:

```
vcom -2008 -autoorder *.vhd
```

Uruchom symulację zatrzasku i przerzutnika poleceniem:

```
vsim -voptargs=+acc work.f_vs_l_tb
```

Zapisz wszystkie przebiegi czasowe poziomu głównego do późniejszej weryfikacji

[!] Zmodyfikuj źródło dlatch.vhd usuwając port D z listy czułości procesu
Uruchom ponownie kompilację i symulację:

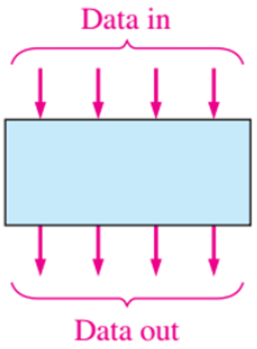
```
vcom -2008 -autoorder *.vhd
vsim -voptargs=+acc work.f_vs_l_tb
```

Porównaj otrzymane przebiegi czasowe z poprzednio zapisanymi

2.2. Opis rejestrów

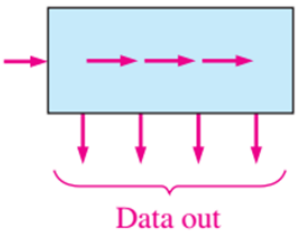
Realizacja rejestru równoległego niewiele odbiega od opisu pojedynczego przerzutnika – konstrukcja architektury jest identyczna, różny jest tylko opis interfejsu.

```
9  entity reg4 is
10     generic (delay: time:=1 ns);
11     Port    (clk : in std_logic;
12              d : in std_logic_vector(3 downto 0);
13              q : out std_logic_vector(3 downto 0) );
14 end entity reg4;
15
16 architecture behav of reg4 is
17
18 begin
19 process(clk)
20     begin
21     if rising_edge(clk) then
22         q <= d after delay;
23     end if;
24     end process;
25 end architecture behav;
```



Podobnie wygląda opis rejestrów z wejściem szeregowym. Poniżej przykład rejestru z wyjściem równoległym, przesuwającego w prawo (shift right).

```
9  entity reg4sr is
10     generic (delay: time:=1 ns);
11     Port    (clk : in std_logic;
12              d : in std_logic;
13              q : out std_logic_vector(3 downto 0) );
14 end entity reg4sr;
15
16 architecture behav of reg4sr is
17     signal q_tmp : std_logic_vector(q'range);
18 begin
19 process(clk)
20     begin
21     if rising_edge(clk) then
22         q_tmp <= d & q_tmp(3 downto 1);
23     end if;
24     end process;
25 q <= q_tmp after delay;
26 end architecture behav;
```



Właściwie wszystkie rodzaje rejestrów realizujących operację przesunięcia lub rotacji można opisać korzystając z operatora konkatenacji (&). Wystarczy zamienić linię kodu 22 na odpowiednią operację:

shift left	`q_tmp <= q_tmp(2 downto 0) & d;`
rotate right	`q_tmp <= q_tmp(0) & q_tmp(3 downto 1);`
rotate left	`q_tmp <= q_tmp(2 downto 0) & q_tmp(3);`

2.3. Opis liczników

W rozwiązaniach bazujących na logice programowalnej stosowane są głównie liczniki synchroniczne. Zasady ich opisu są zbieżne z zasadami opisu rejestrów, gdzie w zależności od rodzaju licznika, operację przesunięcia zastępujemy inkrementacją lub dekrementacją sygnału o odpowiednim typie danych.

Prosty licznik synchroniczny binarny, liczący w przód:

```
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.numeric_std.all;
9   use IEEE.STD_LOGIC_UNSIGNED.all;
10
11  entity b_cntr4 is
12      Port    (clk : in std_logic;
13               rst : in std_logic;
14               q : out std_logic_vector(3 downto 0) );
15  end entity b_cntr4;
16
17  architecture behav of b_cntr4 is
18      signal q_tmp : std_logic_vector(q'range) := x"0";
19  begin
20  process(clk) begin
21      if rising_edge(clk) then
22          if rst='1' then
23              q_tmp <= x"0";
24          else
25              q_tmp <= q_tmp + 1;
26          end if;
27      end if;
28      end process;
29  q <= q_tmp;
30  end architecture behav;
```

Licznik synchroniczny dziesiętny, z wyjściem informującym o końcu zliczania (tc) i kombinacyjnym wyjściem przeniesienia (ceo):

```
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.numeric_std.all;
9   use IEEE.STD_LOGIC_UNSIGNED.all;
10
11  entity d_cntr4ceo is
12      Port    (clk : in std_logic;
13               rst : in std_logic;
14               ce : in std_logic;
15               tc : out std_logic;
16               ceo : out std_logic;
17               q : out std_logic_vector(3 downto 0) );
18  end entity d_cntr4ceo;
19
20  architecture behav of d_cntr4ceo is
21      signal q_tmp : std_logic_vector(q'range) := x"0";
22      signal tci : std_logic;
23  begin
24  process(clk) begin
25      if rising_edge(clk) then
26          if rst='1' then
27              q_tmp <= x"0";
28          elsif ce='1' then
29              if tci='1' then
30                  q_tmp <= x"0";
31              else
32                  q_tmp <= q_tmp + 1;
33              end if;
34          end if;
35      end if;
36  end process;
37  -- outputs
38  tci <= '1' when (q_tmp=9) else '0';
39  ceo <= (tci and ce);
40  tc <= tci;
41  q <= q_tmp;
42  end architecture behav;
```

[!] Uruchom symulację licznika dziesiętnego poleceniem:

```
vsim -voptargs=+acc work.d_cntr4ceo_tb
```

Obserwuj wszystkie przebiegi czasowe poziomu głównego do końca symulacji
Wytłumacz działanie procedury kończącej symulację

[!] Zmodyfikuj źródło *d_cntr4ceo_tb.vhd* w taki sposób aby symulacja kończyła się po dwóch wystąpieniach impulsu *ceo*
Uruchom ponownie kompilację i symulację, zweryfikuj osiągnięte wyniki

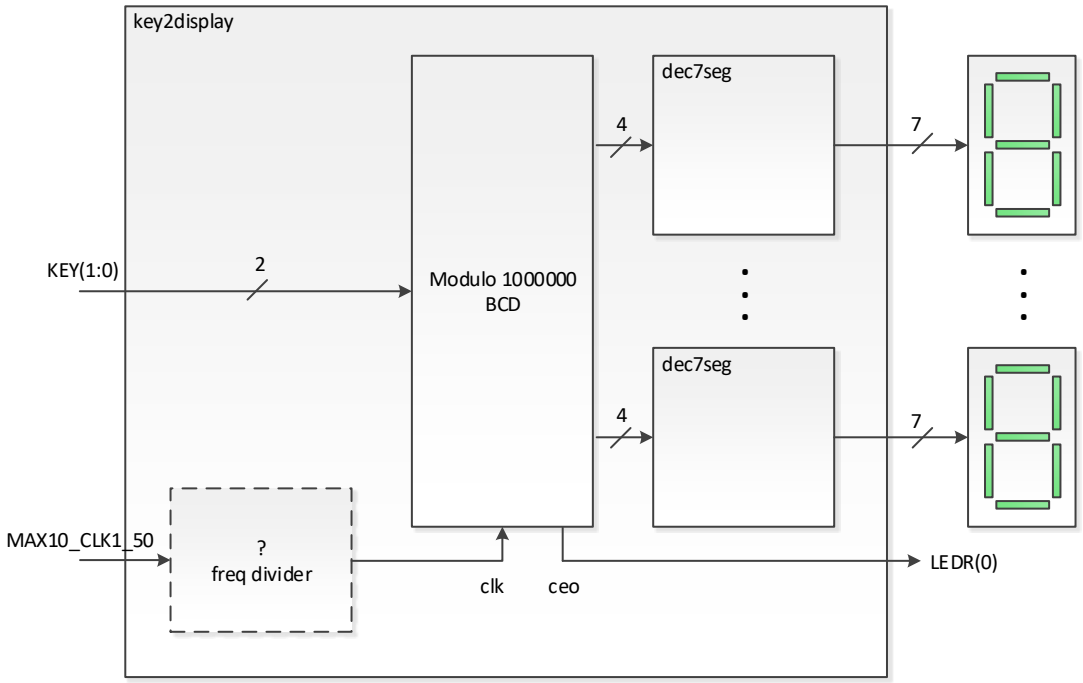
3. Wykorzystanie podprogramów i generacja struktury

Zadanie A

[!] Dodaj do projektu nowy plik o nazwie key2display.vhd
Zdefiniuj w nim jednostkę projektową o podanym niżej interfejsie

```
9   entity key2display is
10      port (
11          max10_clk1_50 : in std_logic;
12          key : in std_logic_vector(1 downto 0);
13          ledr : out std_logic_vector(0 downto 0);
14          hex5,hex4,hex3,hex2,hex1,hex0 : out std_logic_vector(6 downto 0)
15      );
16  end entity key2display;
```

Bazując na poznanych metodach opisu behawioralnego zaprojektuj licznik modulo-1_000_000 zliczający w kodzie BCD. W projekcie wykorzystaj konstrukcję *for…generate* do automatycznej generacji struktury układu.
Projektowane urządzenie powinno wyświetlać wynik na wyświetlaczach 7-segmentowych
Sterowanie za pomocą dwóch przycisków z płyty DE10Lite: key(0) – rst, key(1) – ce
Uwaga: przyciski KEY są monostabilne, ze stabilnym stanem wysokim



[!] Dodaj do projektu nowy plik o nazwie key2display_tb.vhd
Zdefiniuj w nim tesbench proceduralny do weryfikacji działania urządzenia.
Testbench powinien raportować do okna transkrypcji czas trwania impulsu *ceo* oraz wartość wewnętrznej szyny *q* – stan wyjścia liczników w momencie wystąpienia *ceo*=1
Przygotuj makro kompilacji/symulacji i zademonstruj działanie układu

Zadanie B

[!] Korzystając z układów zaprojektowanych w ćwiczeniach lab1 i lab2 zbuduj reklamę świetlną składającą się z co najmniej 10 znaków wyświetlanych na wskaźnikach 7-segmentowych LED.

Wymagania:
- możliwość zmiany kierunku przesuwania tekstu
- możliwość zmiany prędkości ruchu
- możliwość wyboru dwóch różnych tekstów
- preferowane sterowanie przy pomocy przełączników monostabilnych KEY
- demonstracja działania na platformie DE10Lite

Uwaga: w celu wyświetlania dodatkowych znaków należy wykonać własny moduł dekodera wskaźnika 7-segmentowego