Akademia Górniczo – Hutnicza
im. Stanisława Staszica w Krakowie
Wydział IET
Katedra Elektroniki

Technika Cyfrowa

Wprowadzenie do laboratorium komputerowego część II

Wstęp

W ramach zajęć przedstawione zostaną różne sposoby projektowania przerzutników (flip flops) – począwszy od najprostszego przerzutnika wyposażonego jedynie w wejście synchronizujące (zegarowe CLK), wejście danych (D) i wyjście danych (Q), skończywszy na przerzutnikach wyposażonych w synchroniczne i asynchroniczne wejścia sterujące (zerujące RESET/CLEAR, stawiające SET/PRESET oraz zezwalające CE).

1. Elementarny przerzutnik typu D

Przerzutnik typu D jest najszerzej rozpowszechnionym i najczęściej stosowanym przerzutnikiem ze wszystkich. Jego funkcjonalność sprowadza się do przepisania na wyjście Q stanu wejścia D w momencie wystąpienia zbocza narastającego/opadającego synchronizującego sygnału zegarowego CLK.

Proszę utworzyć **nowy projekt RTL**, podobnie jak to było wykonywane w poprzednim ćwiczeniu (wybierz płytkę: **Digilent Nexys4**). Korzystając z czarodzieja proszę dodać nowy plik VHDL o nazwie *dff*. Plik powinien posiadać dwa porty wejściowe (D i CLK) oraz jeden port wyjściowy (Q).

Moduł *dff* opisywać będzie synchroniczny przerzutnik typu D wyzwalany zboczem narastającym sygnału zegarowego CLK. Opis jego architektury w języku VHDL zamieszczono poniżej. Po zaznajomieniu się z nim (stosowne słowo komentarza umieszczono poniżej), proszę skopiować go w odpowiednie miejsce w pliku *dff.vhd*. Synchroniczność pracy przerzutnika typu D wyraża się poprzez uzależnienie całego procesu od zmian stanu wejścia zegarowego CLK. W przypadku wystąpienia zbocza narastającego w sygnale CLK (funkcja rising edge), na wyjście Q przepisana zostanie wartość z wejścia D. W pozostałych przypadkach, wartość na wyjściu Q nie ulegnie zmianie (stan pamiętania ostatniego stanu wyjściowego). Przerzutnik jest elementarnym cyfrowym elementem z pamięcią.

```
entity dff is
   Port ( clk : in STD_LOGIC;
```

```
        D : in STD_LOGIC;
        Q : out STD_LOGIC);
end dff;

architecture Behavioral of dff is
begin
   process(CLK)
      begin
        if(rising_edge(CLK)) then
           Q <= D;
        end if;
  end process;
end Behavioral;
```

Plik po wprowadzeniu wszystkich zmian proszę zapisać w celu sprawdzenia poprawności opisu modułu w języku VHDL.

Proszę dodać do projektu testbench w języku VHDL i nazwać go stosownie.

Proszę go zmienić w taki sposób, by wyglądał podobnie jak wzorzec zamieszczony poniżej. W pierwszej kolejności proszę zdefiniować okres przebiegu zegarowego period, jako wynoszący 10 ns. Następnie proszę stosownie zmienić zawartość procesu *CLK_process* i usunąć wszystkie nadmiarowe linijki kodu stworzone przez czarodzieja. Należy jeszcze opisać serię wymuszeń na wejściu D przerzutnika. Po wprowadzeniu wszystkich zmian plik należy zapisać.

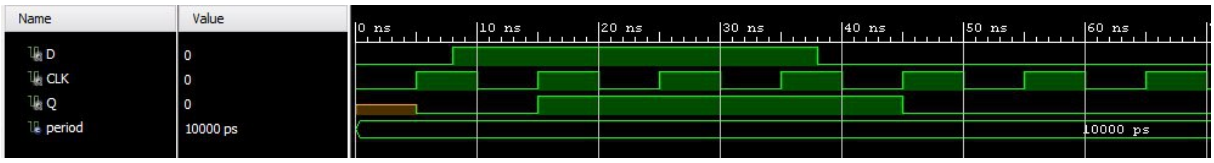
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_dff IS
END tb_dff;

ARCHITECTURE sim OF tb_dff IS
          --Inputs
          signal D : std_logic := '0';
          signal CLK : std_logic := '0';
          --Outputs
          signal Q : std_logic;
         -- Clock period definitions
          constant period : time := 10 ns;
BEGIN
            -- Instantiate the Unit Under Test (UUT)
            uut: entity work.dff PORT MAP (
                  D => D,
                  CLK => CLK,
                 Q => Q );
           -- Clock process definitions
           CLK_process : process
                 begin
                       CLK <= '0';
                       wait for period/2;
                       CLK <= '1';
                       wait for period/2;
         end process;
    -- Stimulus process
    stim_proc: process
      begin
           D <= '0';
          wait for 8 ns;
```

```
        D <= '1';
        wait for 3* period;
       D<= '0';
       wait;
    end process;
END sim;
```

Proszę ustawić czas symulacji na 70 ns i uruchomić symulator. Proszę dopasować szerokość przebiegów do wielkości okna symulatora (Zoom to Full View). Uzyskany widok powinien przypominać przebiegi zamieszczone poniżej.



Przez pierwszych 5 ns na wyjściu *Q* panuje stan nieustalony. Przyczyną jego wystąpienia jest brak zdefiniowanego stanu wyjścia *Q* przed pojawieniem się zbocza narastającego sygnału zegarowego CLK (brak zdefiniowanego stanu początkowego – występuje stan 'U'). Tuż po pojawieniu się pierwszego zbocza narastającego (5 ns), na wyjście *Q* zostaje przerzucony stan z wejścia *D* (w tym przypadku wartość '0'). W 8 ns stan sygnału *D* zostaje zmieniony na '1'. Jednakże zmiana stanu wyjścia *Q* następuje dopiero podczas pierwszego napotkanego narastającego zbocza zegara (15 ns). Następna zmiana wyjścia *Q* następuje w chwili 45 ns kiedy przy narastającym zboczu sygnału zegarowego sygnał *D* = '0'.

UWAGA!

W rzeczywistym układzie reprogramowalnym, na początku symulacji nie pojawia się stan nieokreślonego (ang. uninitialized) 'U' na wyjściu Q. Dzieje się tak ze względu na występowanie tzw. Reset on Configuration, czyli operacji zerowania/ustawiania wszystkich przerzutników w momencie konfiguracji układu FPGA. Zjawisko to nie jest jednak uwzględniane podczas powyższej symulacji.

Zmieniając zawartość jednej linii kodu VHDL opisującej przerzutnik typu D można wyeliminować zjawisko występowania podczas symulacji stanu nieokreślonego na wyjściu *Q* przerzutnika przed pojawieniem się pierwszego zbocza zegarowego, oraz wszystkie niepożądane konsekwencje jakie stany te mogą za sobą pociągnąć zarówno na poziomie symulacji jak i implementacji (wynik implementacji może się różnić od symulacji). W pliku *dff.vhd* proszę dodać do linii opisującej port wyjściowy *Q* zerowy warunek początkowy.

```
Q : out STD_LOGIC := '0'
```

Po zapisaniu wprowadzonej zmiany proszę zapisać plik *dff.vhd*, a następnie ponownie uruchomić symulację. Zgodnie z oczekiwaniami dodanie warunku początkowego wyeliminowało stan nieustalony 'U' na wyjściu *Q* przed wystąpieniem pierwszego zbocza narastającego w sygnale zegarowym CLK.

2. Przerzutnik typu D z asynchronicznymi wejściami sterującymi

Proszę dodać nowy, pusty plik VHDL o nazwie *dff_a* do projektu. Do entity modułu *dff_a* proszę skopiować definicję portów z *entity* modułu *dff*, a następnie poszerzyć je o trzy porty wejściowe – *CLEAR*, *PRESET* i *CE*. Poprawną definicję *entity* modułu *dff_a* przedstawiono poniżej.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dff_a is
  Port ( D : in STD_LOGIC;
         CLK : in STD_LOGIC;
         CE : in STD_LOGIC;
```

```
        CLEAR : in STD_LOGIC;
        PRESET : in STD_LOGIC;
        Q : out STD_LOGIC:= '0');
end  dff_a;

architecture Behavioral of dff_a is
begin
process(CLK, CLEAR, PRESET)
  begin
            if ??? = '1' then
                    Q<= '0';
            elsif ??? = '1' then
                    Q<= '1';
    elsif (rising_edge(CLK)) then
       if ??? = '1' then
          Q <= D;
        end if;
    end if;
  end process;
end Behavioral;
```

Do architektury modułu *dff_a* dodano wejścia asynchroniczne *CLEAR* i *PRESET* oraz synchroniczny sygnał zezwolenia zegara *CE*. Wejścia asynchroniczne mają wyższy priorytet od wejścia zegara i są one wpisywane do listy czułości (lub wrażliwości, ang. sensitivity list – sygnały występujące w nawiasach po słowie kluczowym process). Sygnały występujące wewnątrz procesu są aktualizowane tylko wtedy kiedy następuje zmiana któregokolwiek z sygnałów występujących na liście czułości - zmiana sygnału asynchronicznego wpływa bezpośrednio na stan wyjść. Dlatego na liście czułości wpisywane są wszystkie sygnały asynchroniczne i sygnał zegarowy dla logiki sekwencyjnej oraz wszystkie sygnały wejściowe w przypadku logiki kombinacyjnej. Sygnały synchroniczne np. wejście *D* i *CE* mają najniższy priorytet i są próbkowane tylko dla narastającego / opadającego zbocza sygnału zegarowego, dlatego nie występują one na liście czułości oraz ich logika jest określana po spełnieniu warunku *if(rising_edge(CLK)) then*.

Pewnym problemem jest priorytet sygnałów asynchronicznych *CLEAR*, *PRESET*, czyli co się dzieje w układzie w przypadku kiedy oba te sygnały są aktywne. Warto podkreślić, że dla przerzutnika RS stan taki jest zabroniony. Dlatego w powyższym kodzie można założyć, że stan *CLEAR* ma wyższy priorytet nad sygnałem *PRESET*. Warto podkreślić, że w układach FPGA firmy Xilinx brak jest przerzutników, które posiadają oba te sygnały równocześnie. Powodem tego jest to, że nie zaleca się używać sygnałów asynchronicznych w ogóle oraz równoczesne używanie sygnałów *CLEAR* i *PRESET* jest niezmiernie rzadkie. W powyższym kodzie należy odpowiednio uzupełnić miejsca ???.

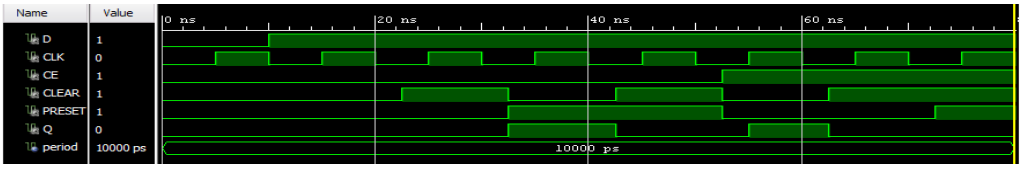
Aby sprawdzić działanie naszego modułu proszę utworzyć nowy testbench dla modułu *dff_a*, przystosować go do symulacji działania testowanego modułu i zdefiniować w nim serię wymuszeń, które umożliwią dokładną obserwację różnych trybów pracy przerzutnika typu D z asynchronicznymi wejściami sterującymi. Przykład wymuszeń umieszczono poniżej.

```
 -- Stimulus process
 stim_proc: process
  begin
    wait for 0.25*period;
      D <= '1';
      CLEAR <= '0';
      PRESET <= '0';
      CE <= '0';
    wait for period;
      CLEAR <= '1';
```

```
    CE <= '1';
  wait for period;
    CLEAR <= '0';
    PRESET <= '1';
  wait for period;
    CLEAR <= '1';
  wait for period;
    CLEAR <= '0';
    PRESET <= '0';
  wait for period;
    CLEAR <= '1';
  wait for period;
    CLEAR <= '0';
  wait for 2*period;
    D <= '0';
  wait for period;
    CE <= '0';
    D<= '1';
  wait;
 end process;

 -- clock generation
 clk_proc: process
  begin
   clk<= not clk;
   wait for  period / 2;
 end process;
```

Proszę zwrócić uwagę na sygnał zegarowy *clk*, czy jego stan początkowy został określony podczas deklaracji sygnału. Proszę ustawić czas symulacji na 150 ns. Po jej uruchomieniu wyniki symulacji powinny przypominać te umieszczone poniżej.



Proszę zwrócić uwagę na fakt stosowania na początku oczekiwania przez ćwiartkę okresu zegara. Dzięki temu sygnały nie zmieniają się w trakcie narastającego zbocza sygnału zegarowego, co może powodować dwuznaczność wyników symulacji.

Brak stanu wysokiego na wejściu zezwalającym CE wstrzymuje normalną pracę przerzutnika - na wyjściu Q utrzymuje się stan poprzedni (sprzed obecnego zbocza narastającego). Stan wysoki tylko na wejściu PRESET wymusza stan wysoki na wyjściu Q przerzutnika. Stan wysoki na wejściu CLEAR wymusza stan niski na wyjściu Q przerzutnika.

3. Przerzutnik typu D z synchronicznymi wejściami sterującymi

Proszę dodać nowy, pusty moduł VHDL o nazwie *dff_s* do projektu. Prócz standardowych portów przerzutnika typu D (D, CLK, Q) entity modułu *dff_s* zawierać powinno definicję synchronicznych wejść sterujących RESET, SET i CE. Poprawną definicję entity modułu *dff_s* przedstawiono poniżej.

UWAGA!

Funkcja synchronicznego portu RESET jest podobna do funkcji asynchronicznego portu CLEAR (zerowanie przerzutnika), podobnie jak funkcja synchronicznego portu SET jest podobna do funkcji asynchronicznego portu

PRESET (ustawianie przerzutnika). Różnica nazw portów jest zwyczajowa i ma na celu podkreślenie ich synchroniczności.

```
entity dff_s is
  Port (D : in STD_LOGIC;
        CLK : in STD_LOGIC;
        CE : in STD_LOGIC;
        RESET : in STD_LOGIC;
        SET : in STD_LOGIC;
        Q : out STD_LOGIC:= '0');
 end dff_s;
```

Do architektury modułu *dff_s* proszę skopiować zawartość architektury modułu *dff*. Proces powinien być zależny **tylko i wyłącznie** od zmian stanów na linii zegarowej CLK. Przekłada się to na pracę wszystkich portów przerzutnika synchronizowaną zegarem CLK.

Wzorując się na projekcie przerzutnika typu D z asynchronicznymi wejściami sterującymi, proszę opisać architekturę przerzutnika typu D z synchronicznymi wejściami sterującymi. Wzorcowy kod w języku VHDL przedstawiono poniżej. Nadrzędnym warunkiem pracy przerzutnika z synchronicznymi wejściami sterującymi jest wystąpienie narastającego zbocza na wejściu zegarowym CLK. Dopiero po wystąpieniu takiego zdarzenia interpretowane są stany pozostałych portów wejściowych. Podobnie jak w przypadku przerzutnika z asynchronicznymi wejściami sterującymi, tutaj również sygnał zerujący (RESET) ma wyższy priorytet nad sygnałem RESET. Sygnał zezwolenia zegara CE ma najwyższy priorytet. .

UWAGA

Priorytet wejść sterujących jest sprawą czysto umowną i może być zdefiniowany przez projektanta. Niemniej wejścia asynchroniczne mają zawsze wyższy priorytet nad sygnałami synchronicznymi.

```
process (CLK) begin
  if(rising_edge(CLK)) then
     if CE = '1' then
       if ??? then
                Q<= '0';
       elsif ???? then
                Q<= ????
       else
                Q<= D;
       end if;
     end if;
  end if;
end process;
```

Moduł proszę zapisać w celu weryfikacji poprawności jego składni.

Proszę utworzyć nowy testbench dla modułu *dff_s*, przystosować go do symulacji i zdefiniować w nim serię wymuszeń identyczną jak w przypadku testbencha dla modułu *dff_a*. Przykład wymuszeń umieszczono poniżej.

```
-- Stimulus process
stim_proc: process
begin
        wait for period;
                D <= '1';
                RESET <= '0';
                SET <= '0';
                CE <= '0';
        wait for period*1.25;
                D <= '1';
```

```
                RESET <= '1';
                SET <= '0';
                CE <= '0';
        wait for period;
                D <= '1';
                RESET <= '0';
                SET <= '1';
                CE <= '0';
        wait for period;
                D <= '1';
                RESET <= '1';
                SET <= '1';
                CE <= '0';
        wait for period;
                D <= '1';
                RESET <= '0';
                SET <= '0';
                CE <= '1';
        wait for period;
                D <= '1';
                RESET <= '1';
                SET <= '0';
                CE <= '1';
        wait for period;
                D <= '1';
                RESET <= '1';
                SET <= '1';
                CE <= '1';
        wait;
end process;
```

UWAGA!

Obecność wejścia zezwalającego CE może wydawać się zbyteczna, jednak port CE umożliwia pracę z mniejszymi częstotliwościami niż sygnał CLK oraz nie wymusza stosowanie lokalnych sygnałów zegarowych co jest niezalecane w przypadku użycia układów FPGA. Podobnie zaleca się stosowanie sygnałów synchronicznych raczej niż sygnałów asynchronicznych. Sygnały asynchroniczne są podatne na występowanie hazardów, powodują że układ działa wolniej oraz zajmuje więcej zasobów sprzętowych.

Studentów zachęca się do zapoznania się z materiałem dostępnym pod adresem: http://www.eetimes.com/document.asp?doc_id=1278998 oraz publikacjami Xilinx: wp275 oraz wp272 (zamieszczone w dodatkowych materiałach na stronie przedmiotu). Należy to jednak uczynić po wykonaniu tego ćwiczenia.

4. Przerzutnik JK

J	K	Q
0	0	$Q_{(t-1)}$
0	1	0
1	0	1
1	1	$\sim Q_{(t-1)}$

Proszę zapoznać się z tablicą prawdy synchronicznego przerzutnika JK, która została zamieszczona obok. Ponadto przerzutnik posiada dwa wejścia asynchroniczne: zerowania R i ustawiania S.

Utwórzmy teraz nowy projekt RTL o nazwie *JKflipflop*. W oknie *Add Sources* za pomocą *Create File* dodajemy nowy plik o nazwie *JKflipflop_main* oraz za pomocą *Add files* dołączony do instrukcji plik *black_box2*. W dalszych oknach postępujemy standardowo: dodajemy czysty plik .xdc, wybieramy moduł *xc7a100tcsg324-1*.

W kreatorze portów dodajemy pięć wejściowych typu *STD_LOGIC* (*clk, J, K, S, R*) oraz jeden wyjściowy typu *STD_LOGIC* o nazwie *Q*. Porty *J,K,S,R,Q* powinny mieć domyślną wartość '0'.

Po zapoznaniu się z poniższymi wskazówkami proszę samodzielnie napisać kod zawierający funkcjonalność układu.

- Kod układu będzie opierał się na konstrukcji *if*. Działanie wejść *J* i *K* musi być uzależnione od sygnału zegarowego *clk*. Działanie wejść *S* i *R* nie może być uzależnione od sygnału zegarowego *clk*. Ponadto, aby uniknąć niejasności podczas stanu wysokiego zarówno na wejściu *S* jak i *R*, proszę zadbać aby wejście *R* miało priorytet wyższy od wejścia *S*.
- Kod będzie sekwencyjny, czyli musi być umieszczony w bloku *process*. Proces powinien być wzbudzany sygnałami *clk, S, R*.
- Sygnał *Q* jest portem wyjściowym. W razie próby odczytu wartości z tego sygnału synteza wykaże błędy w kodzie. Należy utworzyć pomocniczy sygnał *Q_temp* z domyślną wartością '0' i na nim operować. Poza blokiem procesu wystarczy przekazać wartość *Q_temp* do sygnału *Q* (*Q <= Q_temp;*).

5. Symulacja zaprojektowanego układu

Proszę dodać nowy plik symulacji o nazwie *JKflipflop_tb*. Otwieramy plik do edycji i samodzielnie dodajemy komponent *JKflipflop_main*. Następnie dodajemy komponent *black_box2*. Definicję komponentu wraz z portami należy przekopiować:

```
component black_box2 is
   Port ( x : in STD_LOGIC_VECTOR(4 downto 0);
          l : out STD_LOGIC);
end component;
```

Komponent black_box2 jest układem, który posłuży sprawdzeniu poprawności zaprojektowania przerzutnika JK. Podczas symulacji sygnał wyjściowy komponentu *black_box2* zostanie porównany z sygnałem wyjściowym komponentu *JKflipflop_main*. Wszelkie różnice będą wskazywały błędy w zaprojektowanym przerzutniku JK.

Dla celów mapowania proszę dodać do architektury sygnały *clk, Q_tb, Q_correct* typu *STD_LOGIC* oraz 4bitowy sygnał *JKSR* typu *STD_LOGIC_VECTOR*. Sygnały powinny być domyślnie wyzerowane. Proszę wykonać mapowanie komponentu *JKflipflop_main* zgodnie ze wzorem:

```
51 begin
52     JKmapping: JKflipflop_main port map ( clk => clk_tb,
53                                           J => JKSR(3),
54                                           K => JKSR(2),
55                                           S => JKSR(1),
56                                           R => JKSR(0),
57                                           Q => Q_tb
58                                           );
```

Kod mapowania komponentu *black_box2* należy przekopiować:

```
bb2mapping: black_box2 port map( x(4) => clk_tb,
                                 x(3) => JKSR(3),
                                 x(2) => JKSR(2),
                                 x(1) => JKSR(1),
                                 x(0) => JKSR(0),
                                 l => Q_correct
                                 );
```

W pierwszej kolejności zajmiemy się generacją sygnału zegarowego *clk_tb*. W tym celu zdeklarujemy stałą *period* typu TIME o wartości 5 nanosekund.

```
constant period : TIME := 4 ns;
```

Stałą deklarujemy przed słowem kluczowym *begin* w bloku *architecture*.

Do generacji sygnału *clk_tb* napiszemy osobny proces:

```
clk_process: process
begin
        clk_tb<= '0';
        wait for period / 4;
        clk_tb<= '0';
        wait for period / 4;
end process;
```

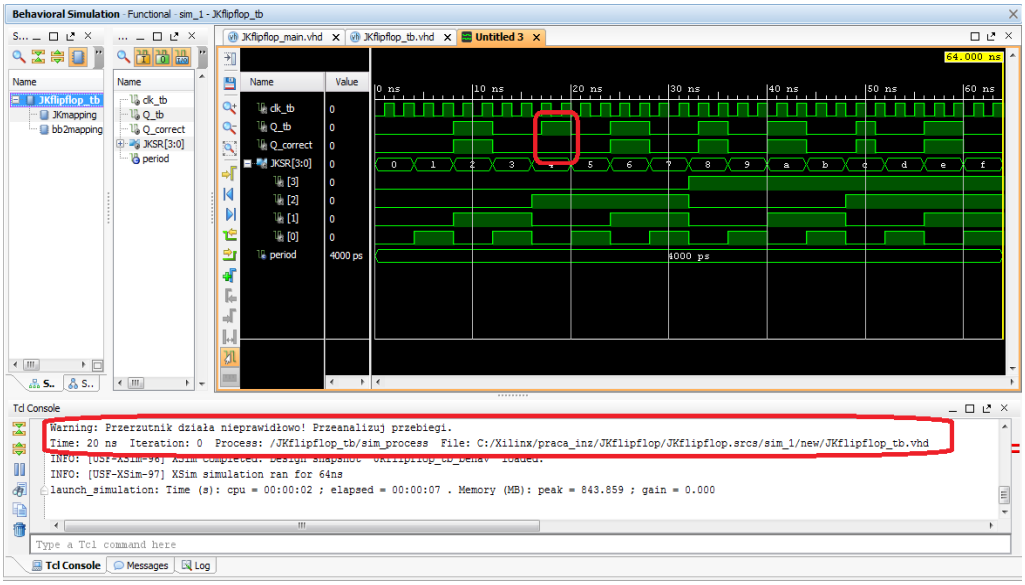
Kod w nim zawarty zmienia wartość sygnału *clk_tb* co 1ns. Okres naszego sygnału zegarowego wynosi więc 2ns. Stała *period* została celowo wydłużona do dwukrotności okresu dla lepszej obserwacji działania przerzutnika.

Proszę zadeklarować bibliotekę *IEEE.STD_LOGIC_UNSIGNED.ALL*. Kod zawierający funkcjonalność symulacji prezentuje się następująco:

```
81      sim_process : process
82      begin
83          wait for period;
84          JKSR <= JKSR + 1;
85          assert (Q_tb = Q_correct)
86          report "Przerzutnik działa nieprawidłowo! Przeanalizuj przebiegi."
87          severity warning;
88      end process;
```

Linie 83-84 powodują inkrementację wektora *JKSR* co stałą *period* (4ns). Aby zapewnić symulację wszystkich 16 kombinacji czas symulacji należy ustawić na 64ns.

Linie 85-87 zawierają konstrukcję *assert*, która sprawdza czy podany warunek jest prawdziwy. Jeśli nie, w konsoli podczas symulacji wyświetlany jest tekst podany po klauzuli *report*. Po klauzuli *severity* podajemy stopień ważności ostrzeżenia (*note, warning, error* lub *failure*). W naszym przypadku *assert* sprawdza czy sygnał wyjściowy komponentu *JKflipflop_main* jest równy sygnałowi wyjściowemu komponentu *black_box2*. Można porównać też przebiegi czasowe tych sygnałów.



W razie wystąpienia błędu (niespełnienia warunku) konsola wyświetli zadany tekst wraz ze wskazaniem dokładnego czasu.

Jeśli konsola nie wyświetla ostrzeżeń można przejść do kolejnego punktu instrukcji. W przypadku wystąpienia ostrzeżeń proszę pracować nad kodem aż do całkowitego ich usunięcia.

6. Uruchomienie zaprojektowanego układu na płytce Nexys4

Proszę zmodyfikować odpowiednio plik .xdc dołączony do projektu. Porty *J,K,S,R* należy przypisać do wybranych przez siebie przełączników, a port *Q* do wybranej diody led. Port *clk* należy przypisać do przycisku, np. środkowego *btnC*. Przyciski nie są przewidziane do generacji sygnałów zegarowych, więc program Vivado przerwie implementację i wyświetli błąd. W naszym przypadku jest to jednak działanie celowe (niemniej niezalecane w praktycznych rozwiązaniach), dlatego w pliku .xdc należy dopisać następujące linijki:

```
11 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
12 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets S_IBUF]
```

Spowodują one zignorowanie tej zasady. Proszę wygenerować *bitstream* i zaprogramować płytkę ewaluacyjną oraz przetestować działanie przerzutnika. Każde wciśnięcie centralnego przycisku generuje jedno narastające zbocze zegarowe.

A jednak układ nie działa do końca prawidłowo – pomimo, że wynik symulacji jest prawidłowy. Najlepiej zaobserwować to przy ustawieniu sygnałów JKSR = ”1100”, gdy każde wciśnięcie przycisku zegara powinno zmienić stan wyjścia (diody) na przeciwny. Tak się nie dzieje, a powodem są drgania styków. Jest to zjawisko czysto mechaniczne. W momencie zwierania i rozwierania przycisku do układu cyfrowego dociera kilka krótkich impulsów, które są uznawane za szybkie naciśnięcia przycisku. Układ FPGA traktuje je jako kilka okresów sygnału zegarowego. W efekcie nigdy nie wiadomo ile razy sygnał wyjściowy został zmieniony na przeciwny.

Aby rozwiązać ten problem musimy powrócić do edycji pliku *JKflipflop_main*. Koniecznych jest kilka zmian.

```
Entity JKflipflop_main is
  Port (  sys_clk : in std_logic;
          J : in std_logic;
          K: in std_logic;
          S: in std_logic;
          R: in std_logic;
          btnC: in std_logic;
          Q: out std_logic);
end JKflipflop_main;

architecture behavioral of  Jkflipflop_main is
          signal Q_temp: std_logic:= '0';  -- auxiliary FF output
          signal clk: std_logic:= '0'; -- clock enable signal (active for one clock when btnC changes)
          signal btn_cnt: integer range 0 to 1000000:= 0;  -- counter mod 10^6
begin
```

W pliku .xdc należy zmienić nazwę portu *clk* na oryginalną nazwę *btnC* oraz dodać (odkomentować) port *sys_clk* przypisany do 100MHz sygnału zegarowego płytki Nexys4.

Następnie w bloku *architecture* należy dodać kod:

```
41      debouncing : process(sys_clk)
42      begin
43          if(rising_edge(sys_clk)) then
44              if (btn_cnt = 0 and btnC = '1') then
45                  btn_cnt <= 1;
46                  clk <= '1';
47              elsif (btn_cnt > 0 and btnC = '0') then
48                  btn_cnt <= btn_cnt + 1;
49                  if (btn_cnt = 1000000) then
50                      btn_cnt <= 0;
51                      clk <= '0';
52                  end if;
53              end if;
54          end if;
55      end process;
```

Proces wzbudzany jest zegarem sprzętowym 100 MHz i z taką częstotliwością się wykonuje. Skupmy się na pierwszym warunku w konstrukcji *if* (linie kodu 44-46). W razie wykrycia naciśnięcia przycisku (czyli stanu wysokiego '1') do licznika *btn_cnt* wpisywana jest wartość 1 oraz sygnał *clk* jest ustawiany na poziom wysoki. W razie wystąpienia drgania styków sygnał *clk* nie będzie więcej zmieniany przez ten warunek, ponieważ licznik *btn_cnt* nie jest już wyzerowany. Teraz przyjrzyjmy się drugiemu warunkowi (linie kodu 47-52). W razie wystąpienia drgań i otrzymaniu przez układ cyfrowy stanu niskiego zwiększana jest wartość licznika. Licznik odlicza do 1 000 000 – biorąc pod uwagę częstotliwość *sys_clk* (100MHz) jest to 10ms. Dopiero po takim czasie (gdy układ będzie otrzymywał stan niski '0') licznik zostanie wyzerowany, a sygnał *clk* ustawiony na '0'. Wtedy możemy uznać, że przycisk został puszczony. Drgania styków to szybkie i krótkotrwałe impulsy - 10 ms jest czasem wystarczającym do ich wyeliminowania.

Ostatnim krokiem jest zmiana linii kodu, który dopisaliśmy wcześniej do pliku .xdc w celu ominięcia zasad implementowania układu. W linii nr 11 ze strony piątej niniejszej instrukcji słowo *clk_IBUF* należy zastąpić *btnC_IBUF*.

Proszę wygenerować *bitstream*, zaprogramować płytkę i sprawdzić działanie układu. Po wyeliminowaniu drgań styków powinien działać w pełni prawidłowo.