Ćwiczenie 1

Stworzyć moduły, (w oddzielnych plikach lub w jednym), które będą opisywały następujące bramki:

NOT

AND / NAND 2-wejściowe

OR / NOR 2-wejściowe

XOR / XNOR 2-wejściowe

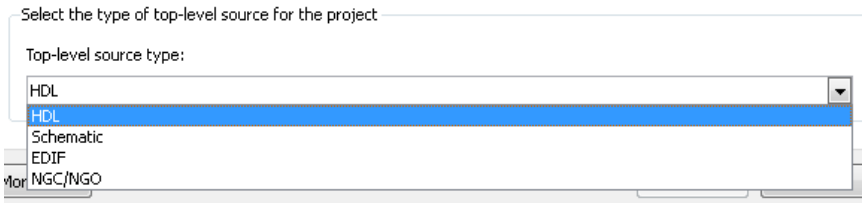
AND / NAND 3-wejściowe

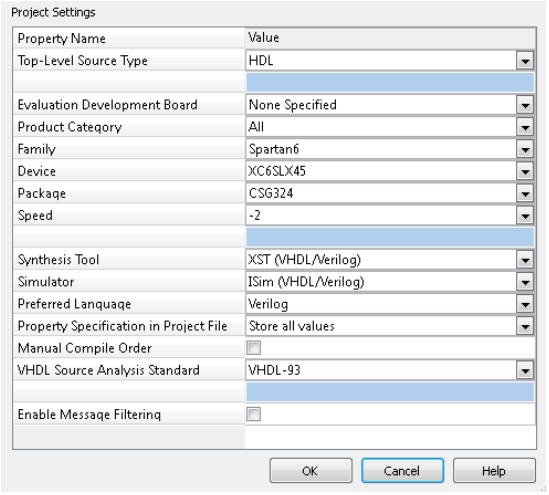
OR / NOR 3-wejściowe

XOR / XNOR 3-wejściowe

Moduły bramek mogą posiadać nazwy wejść/wyjść nieodpowiadające nazwom zawartym w pliku ucf ponieważ będą one dołączane do modułu nadrzędnego (top module). Tylko moduł nadrzędny wymaga opisu zgodnego z plikiem ucf.

Utworzyć nowy projekt wykorzystujący opis strukturalny. Wybrać ustawienia jak na poniższych grafikach.





Dołączyć plik(i) z opisem bramek. Napisać program, który będzie testował działanie bramek z wykorzystaniem dostępnym na płytce przełączników (wejścia) oraz diod led (wyjścia). Struktura proghramu pokazana jest poniżej:

moduły dołączone

moduł nadrzędny

użyty moduł dołączony

opis we/wy
modułu nadrzędnego

proces syntezy

proces implementacji

proces tworzenia pliku bitstream

wgrywanie pliku bitstream

```
1  `timescale 1ns / 1ps
2  `include  "and2.v"
3
4
5  ////////////////////////////////////////////////////////////////////////////////
6  // Company:
7  // Engineer:
8  //
9  // Create Date:    09:47:01 03/11/2022
10 // Design Name:
11 // Module Name:    test
12 // Project Name:
13 // Target Devices:
14 // Tool versions:
15 // Description:
16 //
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 ////////////////////////////////////////////////////////////////////////////////
24 module test(
25     input SW0,
26     input SW1,
27     output LED0
28     );
29
30    and2  bramka1(LED0, SW0, SW1);
31
32 endmodule
33
```

Ćwiczenie 2

Stworzyć moduły, każdy w oddzielnych plikach, które będą opisywały następujące przerzutniki:

- D master-slave wyzwalany zboczem (dowolne) z resetem asynchronicznym
- T master-slave wyzwalany zboczem (dowolne) z resetem asynchronicznym
- RS master-slave wyzwalany zboczem (dowolne) z resetem asynchronicznym
- JK master-slave wyzwalany zboczem (dowolne) z resetem asynchronicznym

Przerzutniki powinny być wykonane z wykorzystaniem odpowiednich bramek (**opis strukturalny**). Niedozwolone jest wykorzystanie bezpośredniego opisu behawioralnego przerzutnika. ~~Do każdego modułu stworzyć moduł testbenchu, który dokona symulacji stworzonego modułu. Należy sprawdzić wszystkie możliwe kombinacje.~~

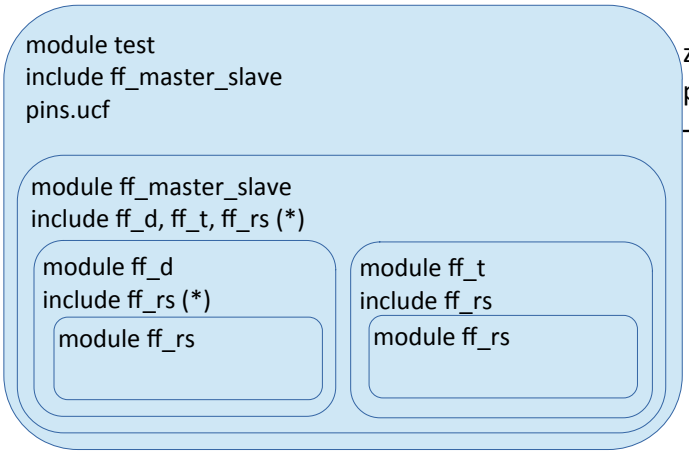
W celu ułatwienia sprawdzania przerzutników wskazane jest etapowe tworzenie przerzutnika:

1. Przerzutnik asynchroniczny RS
2. Modyfikacja przerzutnika asynchronicznego RS na asynchroniczny D i T
3. Przerzutnik synchroniczny D, T i RS
4. Dodanie asynchronicznego resetu
5. Przerzutnik master-slave jako kaskadowe połączenie przerzutnika D/T/RS z RS

Przerzutnik JK należy wykonać oddzielnie z uwagi na sprzężenia zwrotne i brak łatwego łączenia kaskadowego przerzutników synchronicznych.

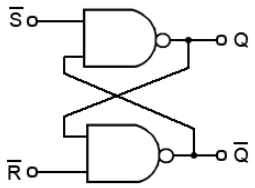
Przetestować działanie przerzutników z wykorzystaniem dostępnym na płytce przełączników (wejścia) oraz diod led (wyjścia). Sygnał zegarowy należy generować ręcznie za pomocą przełącznika. Przetestować działanie resetu przerzutnika czy działa w pełni asynchronicznie.

Struktura programu powinna być następująca:

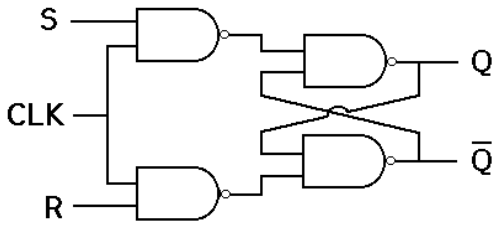


zwrócić uwagę czy nie pojawią się błędy podwójnego dołączenia tego samego modułu – zaznaczone symbolem (*)

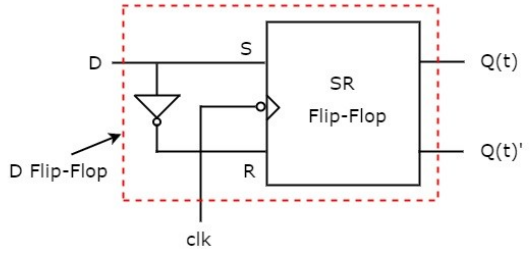
Przerzutnik RS asynchroniczny



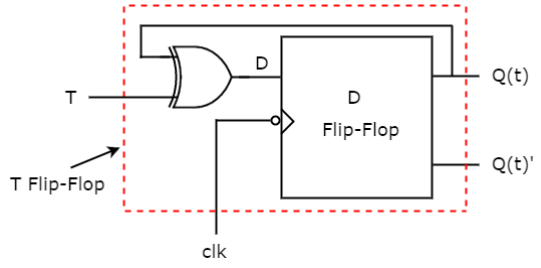
Przerzutnik RS synchroniczny



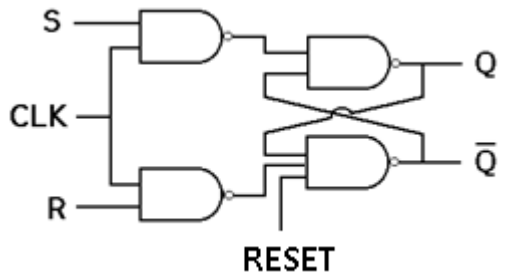
Przerzutnik D



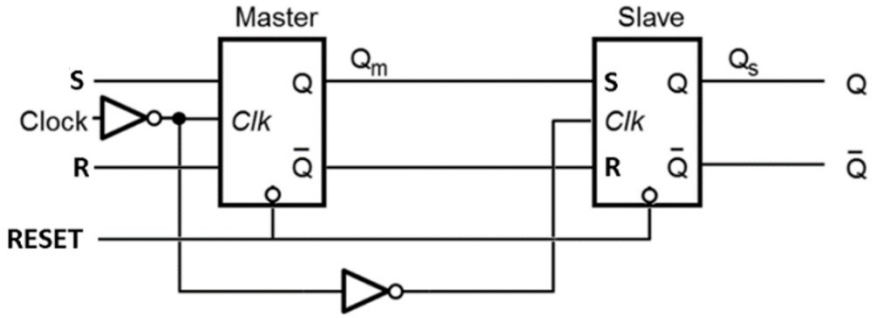
Przerzutnik T



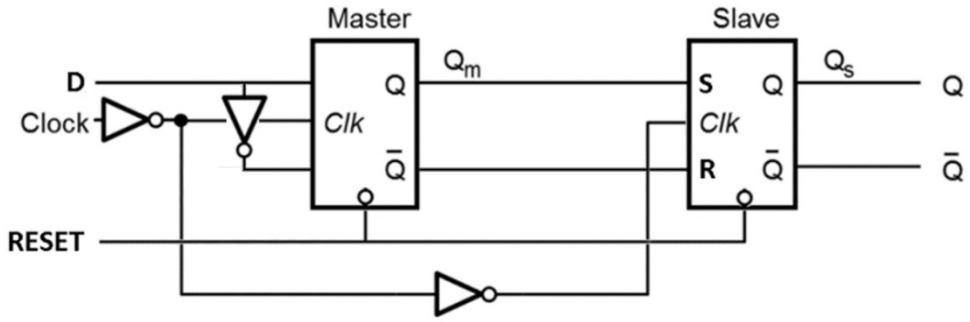
Asynchroniczny reset



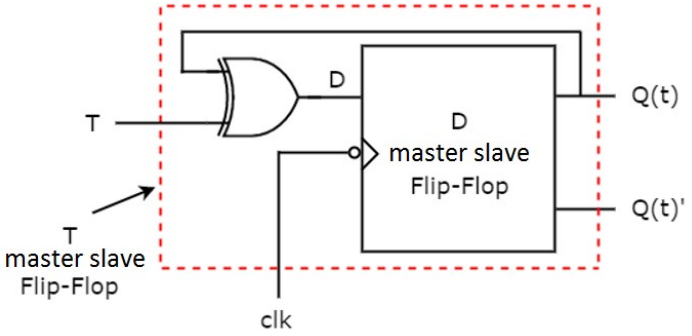
Przerzutnik RS master-slave z resetem asynchronicznym wyzwalany zboczem narastającym



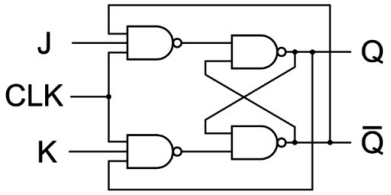
Przerzutnik D master-slave z resetem asynchronicznym wyzwalany zboczem narastającym



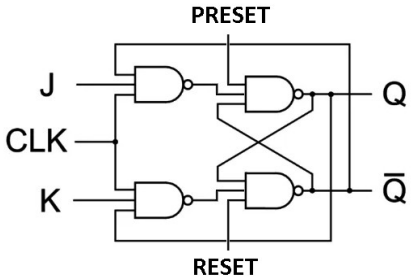
Przerzutnik T master-slave z resetem asynchronicznym wyzwalany zboczem takim jak komórka przerzutnika D master-slave



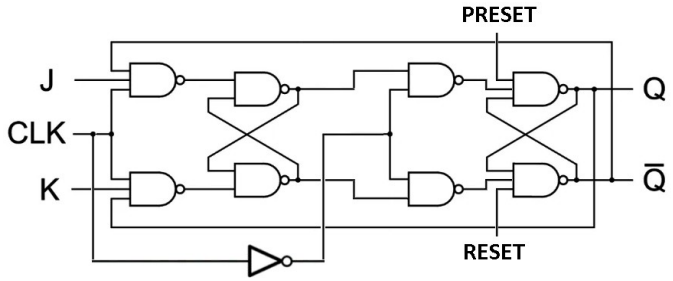
Przerzutnik JK synchroniczny



Przerzutnik JK synchroniczny z resetem asynchronicznym



Przerzutnik JK master-slave z resetem asynchronicznym wyzwalany zboczem opadającym



Uwaga: Wybór zbocza narastającego lub opadającego odbywa się poprzez odpowiednie umieszczenie bramki NOT dla sygnału zegarowego na wejściu komórki.

UWAGA! Z uwagi na ograniczenia struktur w procesorze Spartan 6 może być niemożliwa implementacja sygnału resetu zgodnie z powyższymi schematami. Należy zwrócić uwagę na ostrzeżenia generowane przez proces implementacji. Jeśli pojawi się ostrzeżenie o sygnale wejściowym, który nie ma wymuszenia (a jesteśmy pewni, że sygnał ten jest prawidłowo przypisany):

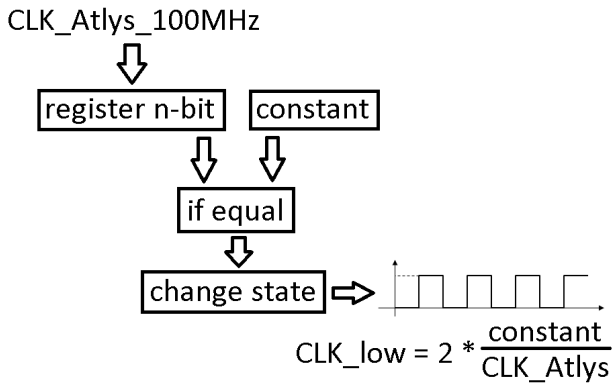
WARNING:Par:288 - The signal ‘name’_IBUF has no load. PAR will not attempt to route this signal.

to należy spróbować wymusić stały sygnał na wejściu reset czyli nieaktywny (1’b1 lub 1’b0). W przypadku przerzutnika master-slave można wymusić sygnał tylko w komórce slave co spowoduje, że reset będzie działał ale nie w pełni asynchronicznie (konieczne będzie wystąpienie zbocza aktywnego dla komórki slave).

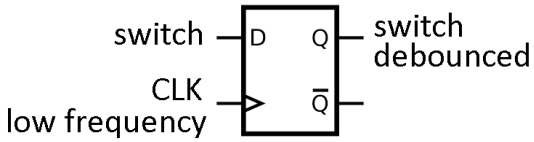
Ćwiczenie 3

Stworzyć dodatkowe moduły, które będą realizowały eliminację drgań na przełącznikach (tzw. debouncing).

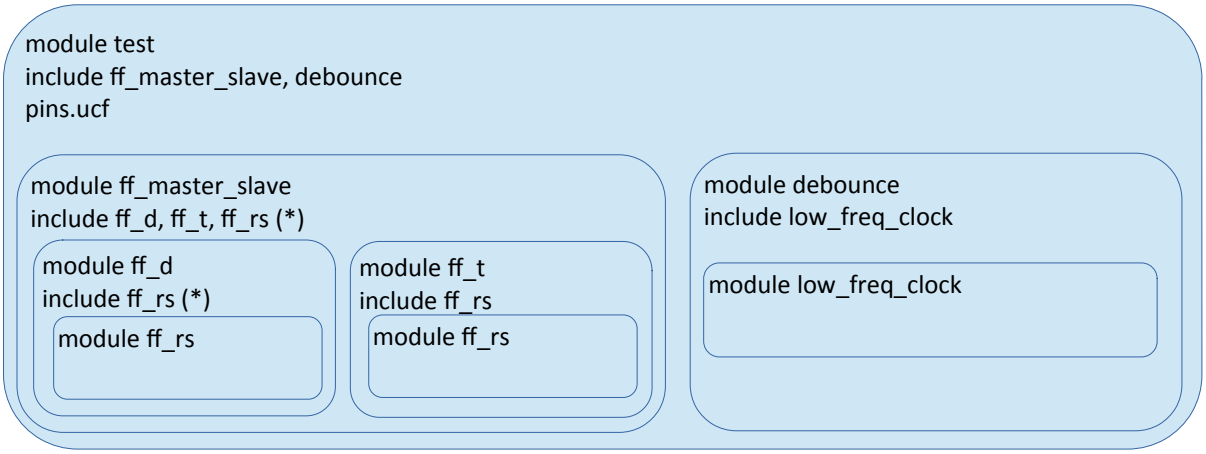
Stworzyć moduł, który wygeneruje sygnał zegarowy o niskiej częstotliwości (rzędu kilku Hz) z zegara dostępnego w układzie FPGA. Do opisu wykorzystać opis behawioralny licznika zmieniającego stan na wyjściu gdy licznik odliczy obliczoną liczbę cykli domyślnego sygnału zegarowego układu FPGA. Wykorzystać wyjście CLK zdefiniowane w pliku ucf. Częstotliwość tego sygnału wynosi 100MHz. Zdefiniować stałą do której ma liczyć licznik za pomocą atrybutu **parameter** służącego do deklarowania stałych. Zmienną zadeklarować jako rejestr o odpowiedniej długości.



Następnie ten moduł wykorzystać do wygenerowania sygnałów wyjściowych dla każdego z przełączników. Wykorzystać przerzutnik D taktowany wygenerowanym sygnałem zegarowym o niskiej częstotliwości, do którego na wejście D podpięty będzie bezpośredni sygnał z przełącznika. Wyjście z przerzutnika będzie „kopią” sygnału z przełącznika ale z wyeliminowanymi drganiami. Wykorzystać opis behawioralny.



Struktura programu powinna być następująca:



Ćwiczenie 4

Stworzyć moduł licznika 4-bitowego opartego na przerzutnikach D/T wyzwalanych zboczem (master-slave) z asynchronicznym resetem zliczającego w zadanej konfiguracji. Przetestować działanie licznika z wykorzystaniem przełączników oraz diod na płytce. Sygnał zegarowy należy generować ręcznie za pomocą przełącznika. **UWAGA!** Jeśli licznik „pomija” niektóre stany to jest to prawidłowe zachowanie z uwagi na generowane drgania na przełącznikach jeśli nie został wykorzystany moduł eliminujący drgania.

Do zaprojektowania należy stworzyć tablice przejścia dla każdego z bitów licznika (jeden przerzutnik odpowiada za jeden bit). Następnie należy zapisać funkcję przejścia minimalizując ją metodą tablicy Karnaugh.

Uwaga: Jeśli licznik nie wykorzystuje wszystkich stanów to należy uwzględnić przejście z tych stanów do pierwszego prawidłowego. Np. licznik 2-12 powinien posiadać możliwość przejścia ze stanów 0,1,13-15 do stanu 2.

Przykład:

Licznik na przerzutnikach D/T zliczający w konfiguracji 2-12.

$(Q_3Q_2Q_1Q_0)_t$ stan poprzedni		$(Q_3Q_2Q_1Q_0)_{t+1}$ Stan następny		$T_3T_2T_1T_0$	$D_3D_2D_1D_0$
				Wymagane stany na wejściach przerzutników	
0	0000	2	0010	0010	0010
1	0001	2	0010	0011	0010
2	0010	3	0011	0001	0011
3	0011	4	0100	0111	0100
4	0100	5	0101	0001	0101
5	0101	6	0110	0011	0110
6	0110	7	0111	0001	0111
7	0111	8	1000	1111	1000
8	1000	9	1001	0001	1001
9	1001	10	1010	0011	1010
10	1010	11	1011	0001	1011
11	1011	12	1100	0111	1100
12	1100	2	0010	1110	0010
13	1101	2	0010	1111	0010
14	1110	2	0010	1100	0010
15	1111	2	0010	1101	0010

Kolorem jasnoszarym zaznaczone są przejścia ze stanów, których licznik nie korzysta. Jak łatwo zauważyć, wartości wejść przerzutnika D są „kopią” stanu następnego. Wartości wejść przerzutnika T są natomiast funkcją XOR stanów poprzednich i następnych.

Minimalizacja funkcji przerzutnika T_0:

$(Q_3Q_2)_t \setminus (Q_1Q_0)_t$	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	0	1	1	0
10	1	1	1	1

Minimalizację wykonujemy poprzez tworzenie możliwie jak największych sklejeń do momentu aż wszystkie jedynki zostaną uwzględnione. Każda jedynka może być użyta w wielu sklejeniach. Jest to nawet wskazane z uwagi na eliminację hazardu.

$$T_0 = Q_0 + \acute{Q}_3 Q_2 + Q_3 \acute{Q}_2 + \acute{Q}_3 Q_1$$

Można również wykonać minimalizację poprzez sklejanie zer, pamiętając o postaci funkcji, która wtedy będzie iloczynem sum. Wybór powinien zależeć od złożoności funkcji, im prostsza tym lepiej (w ćwiczeniu nie będzie to brane pod uwagę).

Ćwiczenie 5

Dodać do licznika z ćwiczenia 4 możliwość ustawienia wartości początkowej ustawianej za pomocą przełączników, która jest aktywowana oddzielnym przełącznikiem. Sposób realizacji dowolny, może być asynchroniczny lub synchroniczny (prostszy). Wskazówka: Najkorzystniej wykorzystać multiplekser na wejściu każdego z przerzutników, do którego podłączone będą dwie ścieżki: normalne liczenie, ustawienie wartości początkowej. Sterowanie multiplekserem za pomocą przełącznika.

Ćwiczenie 6

Zachować program z ćwiczenia 5!

Zmienić źródło sygnału zegarowego na sygnał zegarowy o częstotliwości ok. 1Hz. Wykorzystać wyjście sygnału zegarowego i wygenerować sygnał zegarowy podobnie jak dla modułu debouncingu.

Ćwiczenie 7

Stworzyć moduł licznika 4-bitowego (identycznego jak w ćwiczeniu 6) za pomocą opisu behawioralnego. Zapoznać się z przypisaniami blokującymi oraz nieblokującymi. Można wykorzystać dowolne bloki instrukcji oraz pętle. Poniżej znajduje się przykładowy opis licznika 4-bitowego:

```
module behav_counter( d, clk, clear, load, up_down, qd);

        // Port Declaration
        input   [7:0] d;
        input   clk;
        input   clear;
        input   load;
        input   up_down;
        output  [7:0] qd;

        reg     [7:0] cnt;

        always @ (posedge clk)
        begin
          if (!clear)
            cnt <= 8'h00;
          else if (load)
            cnt <= d;
          else if (up_down)
            cnt <= cnt + 1;
          else
            cnt <= cnt - 1;
        end

        assign qd = cnt;

endmodule
```

Licznik powinien posiadać ustawianie wartości początkowej, debouncing i wykorzystywać sygnał zegarowy jak w ćwiczeniu 5.