Układy Cyfrowe i Systemy Wbudowane 1
Laboratorium 3

Język VHDL: różne poziomy opisu typowych układów kombinacyjnych oraz sekwencyjnych.

Paweł Cyganiuk 263983, Patryk Jurkiewicz 263896

13.11.2023

1. Treść Zadania

3-bitowy licznik dwukierunkowy wariant b)

sekwencja 0 → 1 → 2 → 4 → 3 → 5 → 6 → 7

- symulacja behawioralna: wszystkie 16 stanów X z widocznym zapętleniem - 9 cykli w każdą ze stron(f = 20MHz)
- zawarcie przycisków resetujących i odczytujących (kolejno CLR i CE)

2. Tabela Prawdy

t				t+1			
dir	x_2	x_1	x_0	dir	D_2	D_1	D_0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	0	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	1	0	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0

3. Kod w języku VHDL

W trakcie realizacji zadania podjęliśmy decyzję o przetestowaniu, w jaki sposób język VHDL poradzi sobie z danym problemem. Wcześniej podchodziliśmy do tego samego zadania, wykorzystując bramki logiczne. Teraz skupiamy się na przetestowaniu ewentualnych różnic w podejściu, korzystając z języka VHDL.

Postanowiliśmy zadeklarować dwa sygnały: "Din", który odpowiada za obsługę bitów wchodzących do przerzutnika D, oraz "Dout", który zajmuje się wychodzącymi bitami. W celu utworzenia przerzutników skorzystaliśmy z już istniejących obiektów FDCE dostępnych w bibliotece UNISIM.

```
architecture Behavioral of main is

signal Din : STD_LOGIC_VECTOR (3 downto 0);
signal Dout : STD_LOGIC_VECTOR (2 downto 0);


begin

   Dff1:  FDCE port map (CLR => '0', CE => '1' , D => Dout(0), C => clk, Q=>Din(0));
   Dff2: FDCE port map (CLR => '0', CE => '1' , D => Dout(1), C => clk, Q=>Din(1));
   Dff3: FDCE port map (CLR => '0', CE => '1' , D => Dout(2), C => clk, Q=>Din(2));
```

Następnie, sygnał "Din(3)" jest powiązany z sygnałem "dir", co pokazuje że kierunek jest sterowany przez ten sygnał. Dla wartości 0 idzie on do przodu, dla wartości 1 do tyłu.

Dalej w kodzie występuje konstrukcja "with ... select", która realizuje ustawienie sygnału "Din" na podstawie jego wartości. Oznacza to, że wartość sygnału "Dout" zostanie ustawiona na określoną sekwencję bitów w zależności od aktualnej wartości sygnału "Din". W przypadku, gdy żadna z określonych wartości nie pasuje (others), sygnał "Dout" przyjmuje wartość "---".

Na końcu, wartości poszczególnych bitów sygnału "Din" są przypisywane do odpowiadających bitów sygnału "Q", co oznacza, że stan przerzutników jest bezpośrednio kopiowany do sygnału wyjściowego "Q".

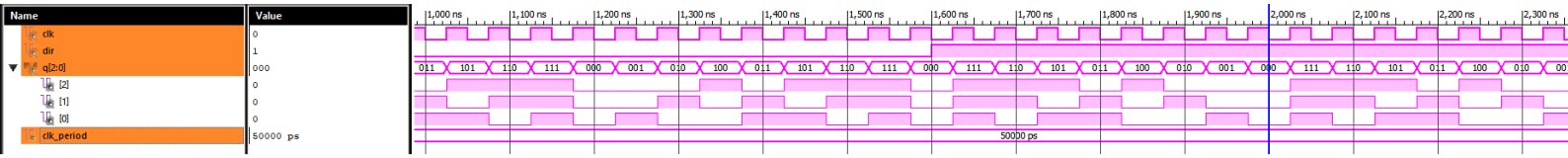
```
   Din(3) <= dir;

   with Din select
   Dout <= "001" when "0000",
   "010" when "0001",
   "100" when "0010",
   "011" when "0100",
   "101" when "0011",
   "110" when "0101",
   "111" when "0110",
   "000" when "0111",
   "111" when "1000",
   "110" when "1111",
   "101" when "1110",
   "011" when "1101",
   "100" when "1011",
   "010" when "1100",
   "001" when "1010",
   "000" when "1001",
   "---" when others;


   Q(0) <= Din(0);
   Q(1) <= Din(1);
   Q(2) <= Din(2);

end Behavioral;
```

4. Symulator ISim

W celu utworzenia testbencha wykorzystaliśmy wbudowaną funkcję programu ISE Project Navigator. Dzięki wcześniejszej deklaracji sygnału Clk, testbench automatycznie wygenerował wszystkie niezbędne elementy do sprawdzenia poprawności kodu. Naszym jedynym wkładem było dodanie odpowiedniego opóźnienia, aby dostosować go do częstotliwości 20 MHz (50 ns), a także zmiana sygnału Dir na 1 po upływie odpowiedniego czasu. Ten proces pozwolił nam skoncentrować się na istotnych aspektach testowania, przy jednoczesnym skorzystaniu z efektywności generowanego automatycznie testbencha.

```
   constant clk_period : time := 50 ns;

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: main PORT MAP (
          clk => clk,
          dir => dir,
          Q => Q
        );

   -- Clock process definitions
   clk_process :process
   begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      wait for clk_period*10;

      -- insert stimulus here

      Dir <= '1' after 1000 ns;

      wait;
   end process;

END;
```

Posiadając wszystko przygotowane, mogliśmy przejść do symulacji, która dała nam następujące wyniki, które zgadzają się ze wcześniej obliczonymi teoretycznymi wynikami.



5. Wnioski

Nasza realizacja projektu przebiegła zgodnie z uprzednimi planami i założeniami. W początkowej fazie prezentowana była oczekiwana sekwencja, a po aktywowaniu przycisku Dir, sekwencja rozpoczynała proces odwracania. Wybrany sposób okazał się znacznie bardziej przyjazny w implementacji po zrozumieniu i opanowaniu operacji bibliotek oraz przerzutników. W porównaniu do schematu logicznego, ten podejście jest bardziej efektywne pod względem czasowym i prostsze w odnalezieniu ewentualnych błędów, gdyż jest przedstawione jako kod, a nie graficzna reprezentacja.