ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Sprzętowa implementacja algorytmów

3. Automat stanów w VHDL. Rejestr przesuwny.

Lider projektu: Politechnika Warszawska

Autor: Łukasz Mik

Akademia Nauk Stosowanych w Tarnowie

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

Declaration

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

Funding Disclaimer

This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

I. Automat stanów w VHDL

Automaty skończone stanów składają się ze skończonego zbioru stanów i zbioru przejść pomiędzy stanami. Używa się ich w aplikacjach, w których są wymagane nietypowe sekwencje sygnałów wyjściowych układu sterującego. Istotną cechą takiego automatu jest możliwość przypisania jego stanom wyjściowym dowolnej sekwencji stanów logicznych. Jednym z przykładów zastosowania takiego automatu jest sterownik zmiany świateł na skrzyżowaniu lub licznik nietypowego kodu.

1. Realizacja automatu stanów z wykorzystaniem stałych.

Na poniższym listingu przedstawiony został kod nietypowego licznika zliczającego w górę i dół.

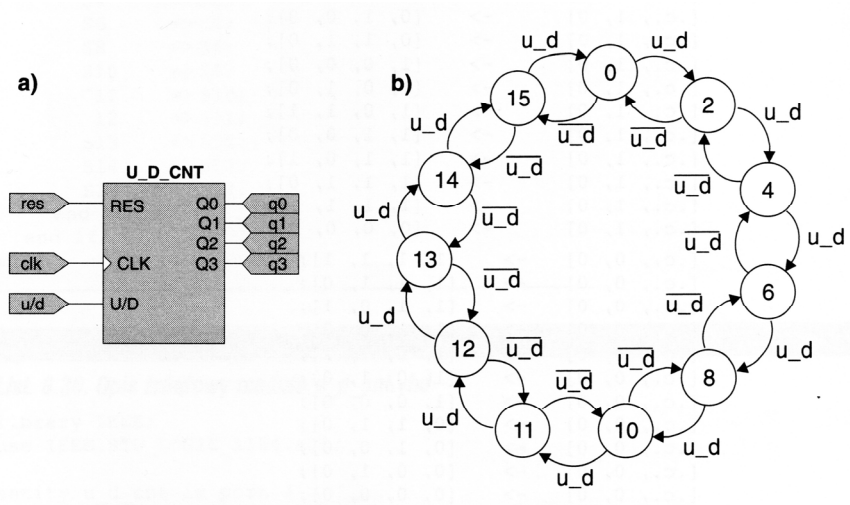
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity u_d_cnt is port (
       clk, res, u_d: in std_logic;
       q: out std_logic_vector(3 downto 0)
       );
end u_d_cnt;

architecture example_arch of u_d_cnt is
      signal state: std_logic_vector(3 downto 0);
      constant S0: std_logic_vector(3 downto 0) := "0000";
      constant S2: std_logic_vector(3 downto 0) := "0010";
      constant S4: std_logic_vector(3 downto 0) := "0100";
      constant S6: std_logic_vector(3 downto 0) := "0110";
      constant S8: std_logic_vector(3 downto 0) := "1000";
      constant S10: std_logic_vector(3 downto 0) := "1010";
      constant S11: std_logic_vector(3 downto 0) := "1011";
      constant S12: std_logic_vector(3 downto 0) := "1100";
      constant S13: std_logic_vector(3 downto 0) := "1101";
      constant S14: std_logic_vector(3 downto 0) := "1110";
      constant S15: std_logic_vector(3 downto 0) := "1111";
begin
process (clk, res) begin
  if (res = '0') then state <= "0000";
      elsif (clk='1' and CLK'event) then
            if u_d = '1' then
               case state is
                   when S0 => state <= S2;
                   when S2 => state <= S4;
                   when S4 => state <= S6;
                   when S6 => state <= S8;
                   when S8 => state <= S10;
                   when S10 => state <= S11;
                   when S11 => state <= S12;
                   when S12 => state <= S13;
                   when S13 => state <= S14;
                   when S14 => state <= S15;
                   when others => stan <= S0;
```

```
            end case;
         elsif u_d = '0' then
            case state is
               when S0 => state <= S15;
               when S2 => state <= S0;
               when S4 => state <= S2;
               when S6 => state <= S4;
               when S8 => state <= S6;
               when S10 => state <= S8;
               when S11 => state <= S10;
               when S12 => state <= S11;
               when S13 => state <= S12;
               when S14 => state <= S13;
               when S15 => state <= S14;
               when others => state <= S0;
            end case;
         end if;
      end if;
 q <= state;
end process;
end example_arch;
```

Kierunek zliczania zależy od stanu na wejściu ***u_d***. Licznik liczy w cyklu: 0, 2, 4, 6, 8, 10, 11, 12, 13, 14, 15, 0, 2 … (w górę)… lub 0, 15, 14, 13, 12, 11, 10, 8, 6, 4, 2, 0, 15, 14 … (w dół). Zmiana stanów licznika następuje pod wpływem impulsów zegarowych podawanych na wejście ***clk***, a zerowanie zapewnia wejście asynchronicznego zerowania ***res***. Symbol graficzny projektu (a) i odpowiadający mu graf przejść (b) został pokazany na poniższym rysunku.



W przypadku definiowania sekwencji stanów wyjściowych automatu należy określić wszystkie występujące w niej stany, a dla każdego stanu bieżącego określić jego stany następne. Gdy stosujemy instrukcję ***case*** warunkiem poprawności opisu jest zdefiniowanie wszystkich możliwych wartości zmiennej selekcyjnej (w przykładzie jest to sygnał ***state***). Układ automatu w przykładzie składa się z 4 przerzutników, które mogą przyjmować 16 stanów. Jak wynika z opisu, w normalnym cyklu pracy automatu wykorzystano tylko 11

stanów, a pozostałe 5 jest niewykorzystanych. Poprawna synteza logiczna opisu VHDL będzie możliwa tylko wtedy, gdy jawnie zostaną określone stany następne po wystąpieniu któregoś ze stanów spoza normalnego cyklu, co najprościej można zrobić za pomocą polecenia `when others => state <= S0;.`

- ***Implementacja automatu stanu z przyciskiem, jako źródłem sygnału zegarowego***

Uruchom program *ISE Design Suite 14.7*, korzystając ze skrótu na pulpicie lub z menu *Start → Programy → Xilinx Design Tools → 64-bit Project Navigator*. Jeśli są otwarte jakieś projekty to zamknij je wszystkie wybierając opcję *File → Close Project*. Następnie utwórz nowy projekt o nazwie *u_d_cnt* i dodaj pliki źródłowe zamieszczone do ćwiczenia: *u_d_cnt.vhd* oraz *u_d_cnt.ucf*.

Następnie zsyntezuj projekt i spróbuj wygenerować plik konfiguracyjny *.bit (*bitstream*). Podczas próby implementacji projektu w układzie docelowym pojawi się błąd, który uniemożliwi wygenerowanie pliku *bitstream*.

```
ERROR:Place:1018 - A clock IOB / clock component pair have been found that are not placed at an optimal clock IOB /
   clock site pair. The clock component <clk_BUFGP/BUFG> is placed at site <BUFGMUX_X2Y1>. The IO component <clk> is
   placed at site <IPAD65>.  This will not allow the use of the fast path between the IO and the Clock buffer. If this
   sub optimal condition is acceptable for this design, you may use the CLOCK_DEDICATED_ROUTE constraint in the .ucf
   file to demote this message to a WARNING and allow your design to continue. However, the use of this override is
   highly discouraged as it may lead to very poor timing results. It is recommended that this error condition be
   corrected in the design. A list of all the COMP.PINs used in this clock placement rule is listed below. These
   examples can be used directly in the .ucf file to override this clock rule.
   < NET "clk" CLOCK_DEDICATED_ROUTE = FALSE; >
```

Jest to spowodowane tym, że przycisk nie jest podłączony do globalnej linii zegarowej GCLK, co program wykryje podczas implementacji. Dlatego w pliku **.ucf*, należy dopisać fragment, który pominie sprawdzanie tej reguły dla linii ***clk***.

```
NET "clk" LOC = P80  | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

Fragment, który należy dodać do opisu linii zegarowej został zaznaczony na zielono.

Po takim zabiegu zamiast błędu podczas implementacji pojawi się tylko ostrzeżenie.

```
WARNING:Place:1019 - A clock IOB / clock component pair have been found that are not placed at an optimal clock IOB /
   clock site pair. The clock component <clk_BUFGP/BUFG> is placed at site <BUFGMUX_X2Y1>. The IO component <clk> is
   placed at site <IPAD65>.  This will not allow the use of the fast path between the IO and the Clock buffer. This is
   normally an ERROR but the CLOCK_DEDICATED_ROUTE constraint was applied on COMP.PIN <clk.PAD> allowing your design to
   continue. This constraint disables all clock placer rules related to the specified COMP.PIN. The use of this override
   is highly discouraged as it may lead to very poor timing results. It is recommended that this error condition be
   corrected in the design.
```

Należy pamiętać, że dla szybkich zegarów używamy wyłącznie linii dedykowanych tj. GCLK.

Po wygenerowaniu pliku konfiguracyjnego *u_d_cnt.bit,* należy zaprogramować układ docelowy przy użyciu programu *ElbertV2Config*, wcześniej wybierając odpowiedni port szeregowy COM.

Podczas testowania projektu na płycie Elbert V2 należy pamiętać, że przyciski SW1 – SW3 zwierają piny wejściowe układu FPGA do masy. Domyślnie występuje na nich stan logiczny 1 (włączone wewnętrzne rezystory podciągające do zasilania).
Można zauważyć, że naciśnięcie przycisku SW1 powoduje często przeskok kilka stanów do przodu. Jest to spowodowane drganiami styków mechanicznych. Przerobimy zatem nasz projekt do postaci, w której źródłem sygnału zegarowego będzie oscylator kwarcowy na płytce drukowanej. Jego częstotliwość podstawową podzielimy tak, aby uzyskać wewnątrz architektury zegar o częstotliwości 1 Hz.

- ***Implementacja automatu stanu z oscylatorem kwarcowym, jako źródłem zegara***

Wewnątrz architektury przed słowem kluczowym *begin* definiujemy dodatkowy sygnał o nazwie *clk_1Hz.*

```
signal clk_1Hz : std_logic := '0';
```

W poprzednim opisie architektury zmieniamy na liście czułości zegar *clk* na zegar *clk_1Hz*.

```
process (clk_1Hz, res)
```

Na początku opisu architektury (po słowie kluczowym *begin*) dodajemy proces, w którym na liście wrażliwości (czułości) podajemy tylko sygnał *clk*. Zadaniem tego procesu będzie wygenerowanie zegara o częstotliwości 1 Hz.

```
process(clk)
variable counter : integer:=0;
begin
if rising_edge(clk) then
      if counter < 6000000 then
            counter := counter+1;
      else
            counter := 0;
            clk_1Hz <= not clk_1Hz;
      end if;
end if;
end process;
```

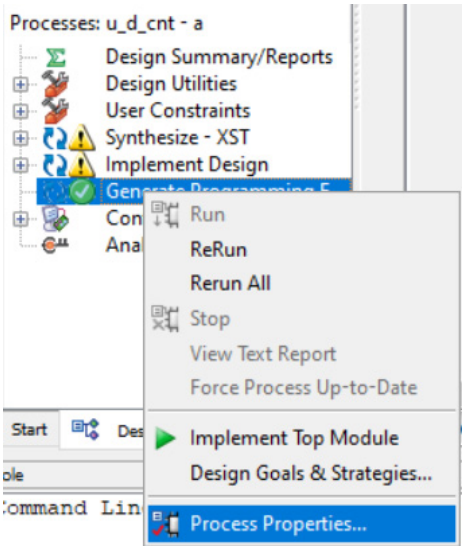
W pliku *constraints* (*.ucf) należy podmienić linię odpowiedzialną za zegar *clk*. Tym razem wskazujemy pin układu FPGA podłączony do rezonatora kwarcowego 12MHz. Usuwamy też linię ignorującą sprawdzanie typu pinu.

```
NET "clk"    LOC = P129   | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
```

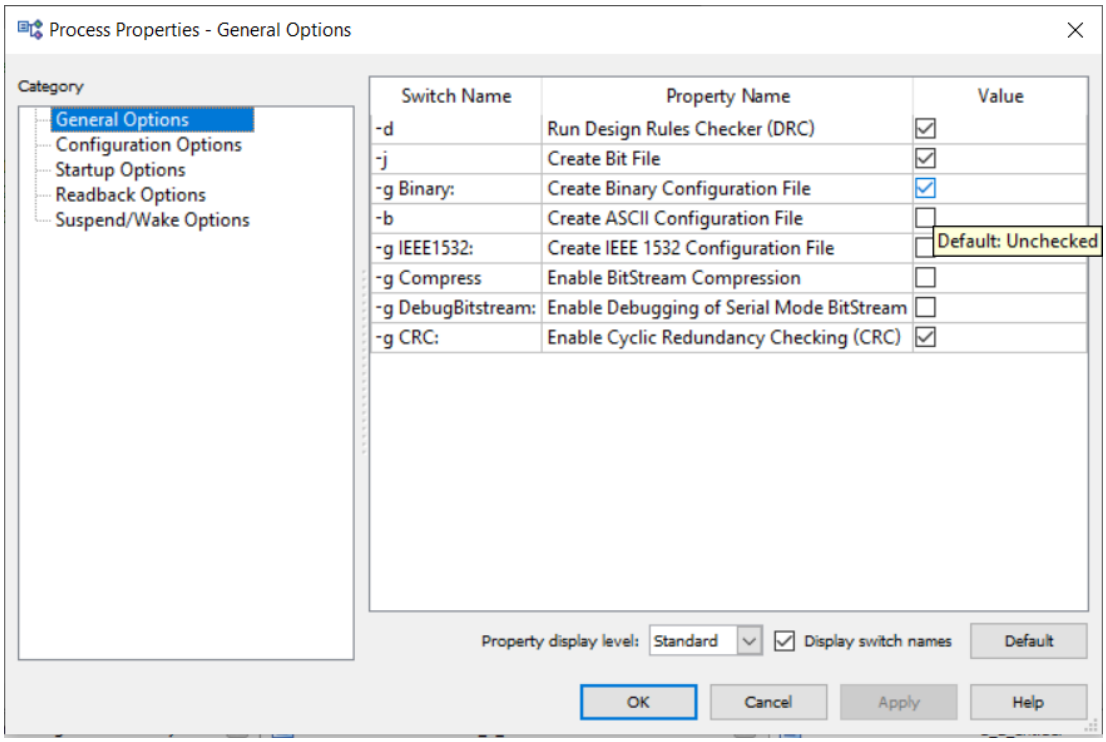
Funkcje przycisków SW2 i SW3 pozostają bez zmian.
W przypadku problemów z błędami powstałymi w wyniku edycji pierwotnego projektu należy wykorzystać pliki źródłowe o nazwie *u_d_cnt_clk_1Hz*, które zostały dołączone do instrukcji.

Program ElbertV2Config, który służy do konfigurowania układu FPGA działa z różnymi rodzajami plików wygenerowanych przez środowisko ISE Webpack. Do tej pory używaliśmy tylko plików o rozszerzeniu *bit*. Zmodyfikujemy ustawienia pakietu WebPack tak, aby poza plikami *bit* było możliwe generowanie plików *bin*.

W oknie *Processes* klikamy prawym klawiszem myszy na opcję *Generate Programming File,* a następnie z rozwijanego menu kontekstowego wybieramy opcję *Process Properties...*



W oknie dialogowym, które się pojawi należy zaznaczyć opcję *Create Binary Configuration File*.



Po zatwierdzeniu zmian ponownie generujemy pliki do zaprogramowania układu FPGA.

W programie ElbertV2Config wskazujemy tym razem plik o rozszerzeniu **.bin* i programujemy (konfigurujemy) układ docelowy.

W zaprezentowanym przykładzie automat stanów został zdefiniowany za pomocą sygnału *state* oraz stałych o nazwach S0, S2, S4, S6, S8, S10, S11, S12, S13, S15 i S15 o zdefiniowanych wartościach. Ten sam efekt można uzyskać za pomocą zdefiniowanej nowego typu zmiennej z jawnie wymienionymi jej wartościami. Tymi wartościami poza liczbami mogą być też ciągi znaków.

2. Realizacja automatu stanu przy użyciu nowego typu danych ze zdefiniowanymi wartościami.

Definiujemy sygnał i przypisujemy do niego nowy typ danych jak poniżej:

```
type STATE_TYPE is (s0, s2, s4, s6, s8, s10, s11, s12, s13, s14, s15);
signal state   : STATE_TYPE;
```

Do każdego stanu, jaki przyjmie sygnał ***state*** musi być przypisana odpowiednia wartość na wyjściu ***q***. Do tego celu użyjemy dodatkowego procesu, na którego liście wrażliwości będzie tylko sygnał ***state***. Procesy odpowiedzialne za automat stanu zostały przedstawione na poniższych listingach.

```
process (clk_1Hz, res) begin
  if res = '0' then state <= S0;
       elsif rising_edge(clk_1Hz) then
             if u_d = '1' then
              case state is
                    when S0 => state <= S2;
                    when S2 => state <= S4;
                    when S4 => state <= S6;
                    when S6 => state <= S8;
                    when S8 => state <= S10;
                    when S10 => state <= S11;
                    when S11 => state <= S12;
                    when S12 => state <= S13;
                    when S13 => state <= S14;
                    when S14 => state <= S15;
                    when others => state <= S0;
              end case;
             elsif u_d = '0' then
              case state is
                    when S0 => state <= S15;
                    when S2 => state <= S0;
                    when S4 => state <= S2;
                    when S6 => state <= S4;
                    when S8 => state <= S6;
                    when S10 => state <= S8;
                    when S11 => state <= S10;
                    when S12 => state <= S11;
                    when S13 => state <= S12;
                    when S14 => state <= S13;
                    when S15 => state <= S14;
                    when others => state <= S0;
              end case;
             end if;
```

```
      end if;
end process;
process (state)
begin
   case state is
      when S0 => q <= "0000";
      when S2 => q <= "0010";
      when S4 => q <= "0100";
      when S6 => q <= "0110";
      when S8 => q <= "1000";
      when S10 => q <= "1010";
      when S11 => q <= "1011";
      when S12 => q <= "1100";
      when S13 => q <= "1101";
      when S14 => q <= "1110";
      when S15 => q <= "1111";
      when others => q <= "0000";
   end case;
end process;
```

Ważną cechą języka VHDL oraz jego interpreterów jest to, że przełamanie linii kodu nie wpływa na jego interpretację i syntezę. Przykładowo zapis `when S0 => q <= "0000";` jest interpretowany tak samo jak zapis:

```
when S0 =>
   q <= "0000";
```

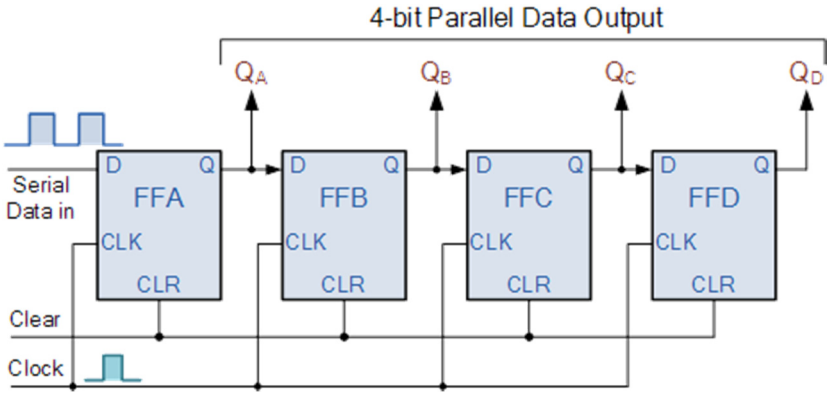
Po zsyntezowaniu projektu i wygenerowaniu pliku konfiguracyjnego należy sprawdzić poprawność jego działania.

II. Rejestr przesuwny.

Rejestry przesuwne są używane do przechowywania i przenoszenia danych w systemach cyfrowych lub do konwersji danych z postaci szeregowej na równoległą i odwrotnie. Tworzy się je za pomocą szeregowo połączonych przerzutników, najczęściej typu D. Długość rejestru w bitach odpowiada liczbie przerzutników w tym rejestrze.

- ***Implementacja 4-bitowego rejestru przesuwnego z wyjściem szeregowym i równoległym***

Przykładowy, 4-bitowy rejestr przesuwny z wejściem szeregowym i wyjściem równoległym został pokazany na poniższym rysunku.



Jeśli chcielibyśmy wykorzystać taki rejestr przesuwny jako SISO (z szeregowym wejściem i szeregowym wyjściem) to wówczas wyjściem rejestru byłoby wyjście ostatniego z przerzutników czyli Q_D.

Przykład implementacji rejestru przesuwnego został przedstawiony na poniższym fragmencie kodu w języku VHDL.

```
process (clk_1Hz)
begin
  if (clk_1Hz'event and clk_1Hz = '1') then
            s_reg(2 downto 0) <= s_reg(3 downto 1);
            s_reg(3) <= din;
  end if;
end process;
```

Przy każdym zboczu narastającym zegara *clk_1Hz* następuje przesunięcie 3 najstarszych bitów o 1 pozycję w prawo (`s_reg(2 downto 0) <= s_reg(3 downto 1)`) z jednoczesnym przypisaniem bitu z wejścia *din* do najstarszego bitu w rejestrze. Instrukcja `led <= s_reg` przypisanie stanów z wyjścia równoległego rejestru do pinów wyjściowych układu FPGA,

podłączonych do diod LED (D5 do D8). Instrukcja `q <= s_reg(0)` przypisuje wyjście szeregowe rejestru do diody LED (D1).
Pliki z kodami źródłowymi o nazwach *shift_reg.vhd* i *shift_reg.ucf* zostały udostępnione wraz z instrukcją do zajęć.

ZADANIA:

1. Utwórz wektory testowe do projektu rejestru przesuwnego, a następnie przeprowadź symulację jego działania przy użyciu narzędzia ISim z pakietu ISE Webpack.

2. Na podstawie przerobionych podczas tego ćwiczenia przykładów utwórz projekt rejestru przesuwnego, który będzie działał na zasadzie automatu stanu.

References

- P. Zbysiński, J. Pasierbiński – *Układy programowalne – pierwsze kroki.* Wydawnictwo BTC, Warszawa 2004

- J. Majewski, P. Zbysiński – *Układy FPGA w przykładach.* Wydawnictwo BTC, Legionowo 2007.

- Electronics Tutorial – *Sequential Logic: The Shift Register* https://www.electronics-tutorials.ws/sequential/seq_5.html