Warszawa, 16.01.2023 r.

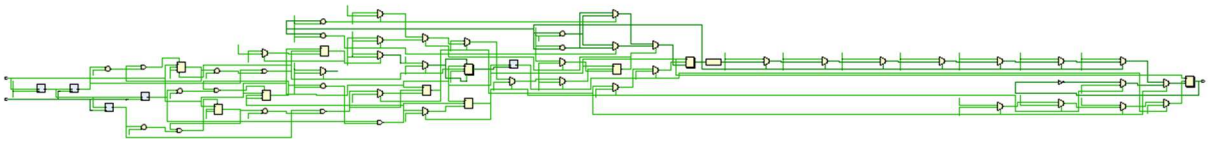
Wykonawcy

Konrad Krupski, 310729

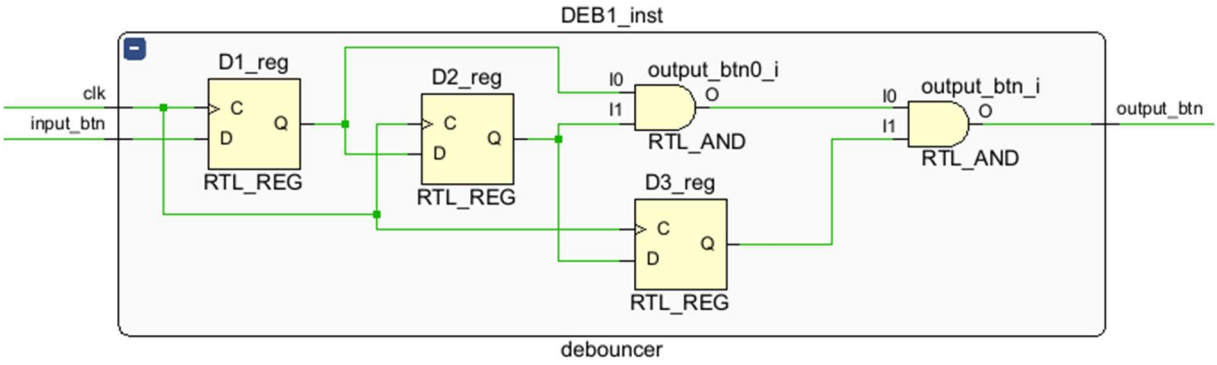
Julia Polak, 310965

Temat: Wariant 3 + Wariant 4. Negowanie wszystkich diod + Wędrująca pojedyncza dioda z przejściem pozycja 8->7 (odbija się)

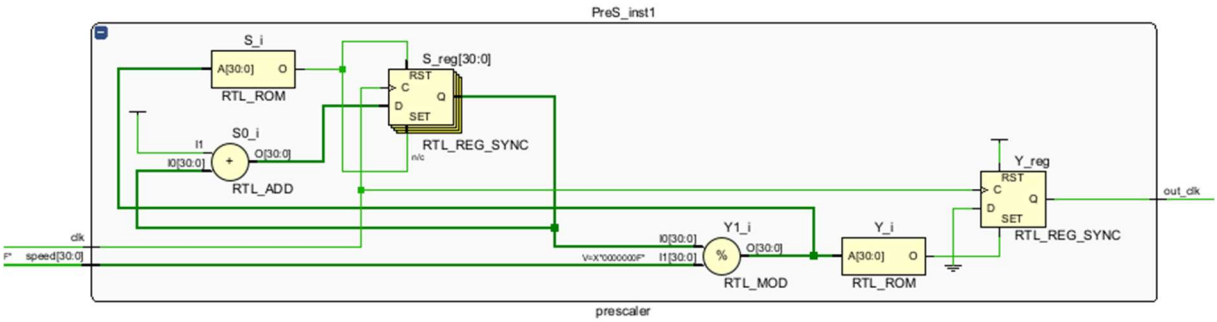
Schemat blokowy układu



Rysunek 1 Schemat blokowy układu po elaboracji



Rysunek 2 Schemat blokowy Debouncer'a po elaboracji



Rysunek 3 Schemat blokowy prescaler'a po elaboracji

Opis działania projektu

Układ będzie uruchamiał się w stanie domyślnym co rozumiemy jako pierwszy wariant świecenia (negowanie diod), podstawowa częstotliwość zegara ustawiona przez prescaler(1 Hz) oraz sygnał reset ustawiony na 0 (czyli brak resetu). Do obsługi zmiany wariantu, resetu czy prescaler'a wykorzystamy przycisku znajdujące się bezpośrednio na płytce Zybo Z7-10. Jako diody wykorzystamy zewnętrzy układ dołączony do portu „je".

Omówienie najważniejszych części kodu

```
34  entity prescaler is
35    port(
36      clk: in bit;
37      speed: in natural;
38      out_clk: out bit
39    );
40  end prescaler;
41
42  architecture Behavioral of prescaler is
43      signal S: natural:=1;
44      signal Y: bit;
45  begin
46  process(clk) is
47      begin
48      if(clk' event and clk='1') then
49              if(S mod speed = 0) then
50              Y <= '1';
51              S <= 1;
52          else
53              Y <= '0';
54              S <= S+1;
55          end if;
56      end if;
57      end process;
58      out_clk <= Y;
59  end Behavioral;
```

Rysunek 4 Kod prescaler'a

Komentarz:

Moduł Prescaler'a został opracowany w oparciu o operację modulo. Jego wejściami są kolejno: sygnał zegara, w naszym przypadku jest to zegar z płytki ZYBO Z7-10, parametr speed, który ustala wartość przez jaką częstotliwość zegara będzie dzielona oraz wyjście out_clk, które jest naszym sygnałem zegarowym po przeskalowaniu.

Dzięki operacji modulo jesteśmy w stanie zmienić stan podawany na wyjście układu prescalera w momencie, gdy zliczymy odpowiednio ustaloną liczbę „tyknięć" zegara doprowadzonego do wejścia. Płytka Zybo Z7-10 charakteryzuje się częstotliwością zegara 125MHz, jednak w celu przeprowadzenia symulacji prescaler został ustawiony a wartość 124. Niestety operacja modulo jest dosyć kosztowana, dlatego w rzeczywistym układzie będziemy spodziewać się pewnych opóźnień.

```
34      entity debouncer is
35          port(
36              input_btn: in bit;
37              clk: in bit;
38              output_btn: out bit
39          );
40      end debouncer;
41
42      architecture Behavioral of debouncer is
43          signal D1, D2, D3: bit;
44      begin
45          process(clk) is
46          begin
47              if(clk' event and clk='1') then
48                  D1 <= input_btn;
49                  D2 <= D1;
50                  D3 <= D2;
51              end if;
52          end process;
53          output_btn <= D1 and D2 and D3;
54
55      end Behavioral;
```

Rysunek 5 Kod układu debouncer'a

Komentarz:

Układ debouncer'a został zrealizowany o połączenie 3 przerzutników typu D, dzięki czemu osiągnęliśmy opóźnienie sygnału o 3 takty zegara. Taki układ w połączeniu z prescaler'em pozwolił nam osiągnąć zamierzano rezultat, czyli wyeliminowanie wpływu drgań używanych przycisków.

Przerzutniki typu D są połączone szeregowo co zostało pokazane w sekcji prezentacji schematu blokowego układu. Sygnał wysoki na wyjściu debouncer'a jest uzyskiwany tylko wtedy, gdy sygnał wejściowy przepropagował się przez każdy z przerzutników.

Do sygnały input_btn przypisujemy przycisk, który ma być obsłużony przez układ debouncer'a. Do wejścia clk dołączamy zegar, którym nasze przyciski mają być sterowane, w naszym przypadku jest to zegar po przeskalowaniu poprzedni opisanym przeze mnie układem debouncer'a. Na wyjściu output_btn uzyskujemy odpowiednio opóźniony stan przycisku.

```
73          if(On_Off_flag = '0') then
74              if(Schemat = '0') then
75              -- obsluga diod Schemat 1 swiecenia Diod
76                  Y <= not Y;
77              else
78              -- obsluga diod Schemat 2 swiecenia Diod
79                  if(UP_DOWN = '0') then
80                      if(Counter < 7) then
81                          Counter <= Counter + 1;
82                      else
83                          UP_DOWN <= not UP_DOWN;
84                          Counter <= Counter - 1;
85                      end if;
86                  else
87                      if(Counter > 0) then
88                          Counter <= Counter - 1;
89                      else
90                          UP_DOWN <= not UP_DOWN;
91                          Counter <= Counter + 1;
92                      end if;
93                  end if;
94              end if;
95          else
96              Y <= "00000000";
97          end if;
```

Rysunek 6 Logika kombinacyjna obsługujące warianty świecenia diod

Komentarz:

W liniach 73 – 76 jest przedstawiony sposób realizacji pierwszego wybranego przez nas wariantu świecenia diod. Wariant ten miał negować diody podłączone do wyjścia płytki, co zostało zrealizowane poprzez zanegowania sygnału pomocniczego Y oraz przypisanie go do wyjścia „je”.

W liniach 79-91 został zrealizowany drugi wybrany przez nas wariant, który polegał na tym, że mieliśmy zaświecić pojedynczą diodą z uwzględnieniem przejścia 8->1. Do realizacji tego zadania wykorzystaliśmy pomocniczy sygnał UP_DOWN, który pozwala nam na ustalenie strony, w którą dioda ma się „poruszać”. Jeżeli jest on równy 0 to poruszamy się zgodnie z następującą kolejnością

1 -> 2 -> 3 -> … -> 8

W przeciwnym wypadku poruszamy się w następujący sposób

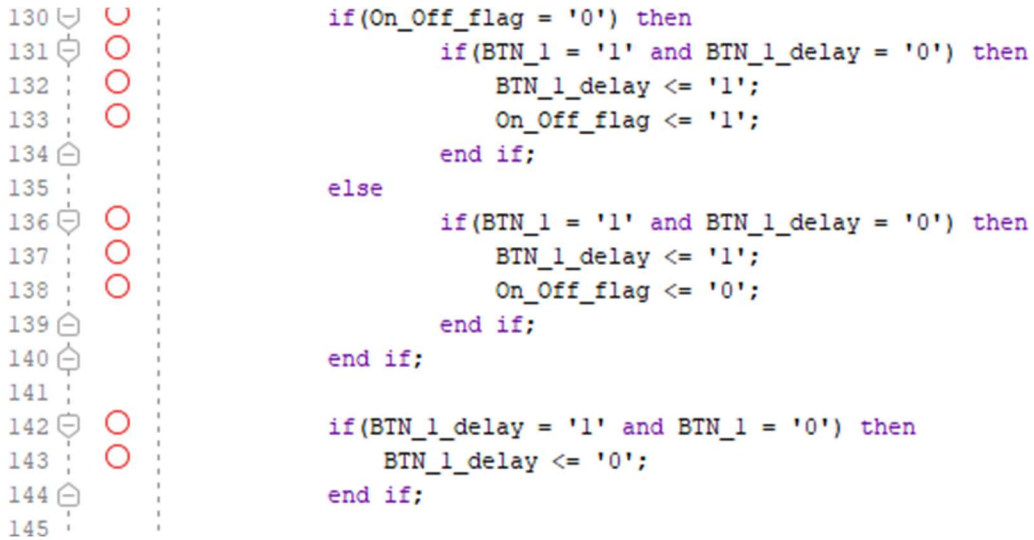
8 -> 7 -> 6 -> … -> 1

Do śledzenia tego, która dioda ma być obecnie zapalona wykorzystaliśmy sygnał pomocniczy Counter oraz odpowiednio napisaną instrukcję warunkową.

```
if (Counter = 0) then
    Y <= "00000001";
elsif (Counter = 1) then
    Y <= "00000010";
elsif (Counter = 2) then
    Y <= "00000100";
elsif (Counter = 3) then
    Y <= "00001000";
elsif (Counter = 4) then
    Y <= "00010000";
elsif (Counter = 5) then
    Y <= "00100000";
elsif (Counter = 6) then
    Y <= "01000000";
elsif (Counter = 7) then
    Y <= "10000000";
else
    Y <= "00000000";
end if;
```

Rysunek 7 Instrukcja warunkowa obsługująca świecenie diody

Możemy zauważyć, że w zależności od wartości sygnału Counter zapalmy diodę wyznaczoną przez wartość binarną, np. wartość dla Counter = 3 czyli 00001000 odpowiada czwartej diodzie.



Rysunek 8 Obsługa przycisku reset

Każdy z przycisków został obsłużony w podobny sposób, dlatego zasada działania zostanie wyjaśniona tylko na jednym z nich. Na początek sprawdzamy obecny stan przycisku (linia 130) i w zależności on tego wykonujemy kolejne instrukcje warunkowe. W tym rozwiązaniu posłużyliśmy się pomocniczym sygnałem BTNx_delay, który zapamiętuje poprzedni stan przycisku, co pozwala na uniknięcie sytuacji, że pojedyncze kliknięcie przycisku wywołuje kilkukrotne wywołanie się instrukcji if. Następnie, jeżeli przycisk został kliknięty a jego poprzednim stanem był stan niski to zmieniamy wartość flagi.

```
150         if(BTN_2 = '1' and BTN_2_delay = '0') then
151             if(Speed_flag = '1') then
152                 Speed <= 64;
153                 Speed_flag <= '0';
154             else
155                 Speed <= 124;
156                 Speed_flag <= '1';
157             end if;
158             BTN_2_delay <= '1';
159         end if;
```

Rysunek 9 Obsługa przycisku do zmiany prędkości

Komentarz:

Przycisk do zmiany prędkości działa w oparciu o parametr Speed, który jest przypisywany do wejścia układu prescaler'a. W zależności od wartości tego parametru ustalamy wartość, przez którą dzielimy zegar. Dzięki temu uzyskujemy szybszą lub wolniejszą pracę.

Opis poszczególnych komponentów porty in/out oraz funkcjonalność

```
port(
    clk: in bit;
    btn: in bit_vector (3 downto 0);
    je: out bit_vector(7 downto 0)
);
```

Rysunek 10 Porty in/out układu

Port clk jest portem sygnału zegara, do którego dołączamy sygnał z pliku testbench albo po prostu z płytki ZYB Z7-10

Port je jest portem, do którego jest dołączony urządzenie zewnętrzne zawierające diody używane w projekcie. Port je posiada wartość od 7 do 0, które odpowiadają poszczególnym diodom.

Port btn jest portem, do którego przypisane są używane przez nas przyciski.

```
19      signal Y: bit_vector(7 downto 0);
20      signal On_Off_flag: bit:='0';
21      signal Speed_flag: bit:='0';
22      signal Schemat: bit:='0';
23      signal Speed: natural:=124;
24      signal Counter: natural:=0;
25      signal UP_DOWN: bit:='0';
26      signal BTN_1, BTN_2, BTN_3: bit;
27      signal BTN_1_delay, BTN_2_delay, BTN_3_delay: bit;
28      signal CLK_PRESCALER: bit;
29      signal CLK_PRESCALER_DB: bit;
```

Rysunek 11 Sygnały pomocnicze

Sygnał Y

Sygnał Y jest sygnałem pośrednich który przechowuje wartość podawaną na wyjście układu, czyli port „je”.

Sygnał On_Off_flag

Sygnał jest odpowiednikiem sygnału reset. Pozwala on na wyłączenie/włączenie układu.

Sygnał Speed_flag

Sygnał, który pozwala na wybor prędkości pracy zegara. Możemy wybrać wartość 0 czyli częstotliwość 1Hz albo wartość 1 czyli częstotliwość 2Hz.

Sygnał Schemat

Sygnał pozwala na wybór wariantu świecenia diod. Wartość 0 odpowiada pierwszemu wariantowi a wartość 1 drugiemu.

Sygnał Speed

Sygnał pośredni, który przechowuje wartość przez jaką ma być dzielona częstotliwość zegara płytki. Sygnał ten jest podawany na wejście układu Prescaler’a.

Sygnał Counter

Sygnał, który śledzi numer diody, która należy włączyć dla wariantu 2.

Sygnał UP_DOWN

Sygnał, który wskazuje kierunek przełączania się diod dla wariantu drugiego.

Sygnały BTN_1 BTN_2 BTN_3

Sygnał, które przechowują wartość reprezentującą stan przycisków odpowiednio opóźniony. Są one również wyjściami układu debouncer’a

Sygnały BTN_1_delay BTN_2_delay BTN_3_delay

Sygnały, które przechowują poprzednie wartości reprezentujące stan przycisków. Sygnały te są używane w momencie, gdy musimy zidentyfikować czy dany przycisk został „puszczony” po uprzednim kliknięciu.

Sygnał CLK_PRESCALER

Sygnał zegarowy, który jest otrzymywany na wyjściu prescaler'a. Wykorzystywany jest do sterowania naszym układem.

Sygnał CLK_PRESCALER_DB

Sygnał zegarowy, który jest otrzymywany na wyjściu prescaler'a. Wykorzystywany jest do sterowania układami debouncer'ów w taki sposób, aby mogły one wprowadzić odpowiednie opóźnienie.

Symulacje (testbench) wraz z omówieniem wyników

```
38      architecture Behavioral of first_tb is
39          signal C: bit;
40          signal Y:  bit_vector(7 downto 0);
41          signal btn: bit_vector(3 downto 0);
42      begin
43          process is
44          begin
45              C <= '0';
46              wait for 1ns;
47              C <= '1';
48              wait for 1ns;
49          end process;
```

Rysunek 12 Kod testbench'a

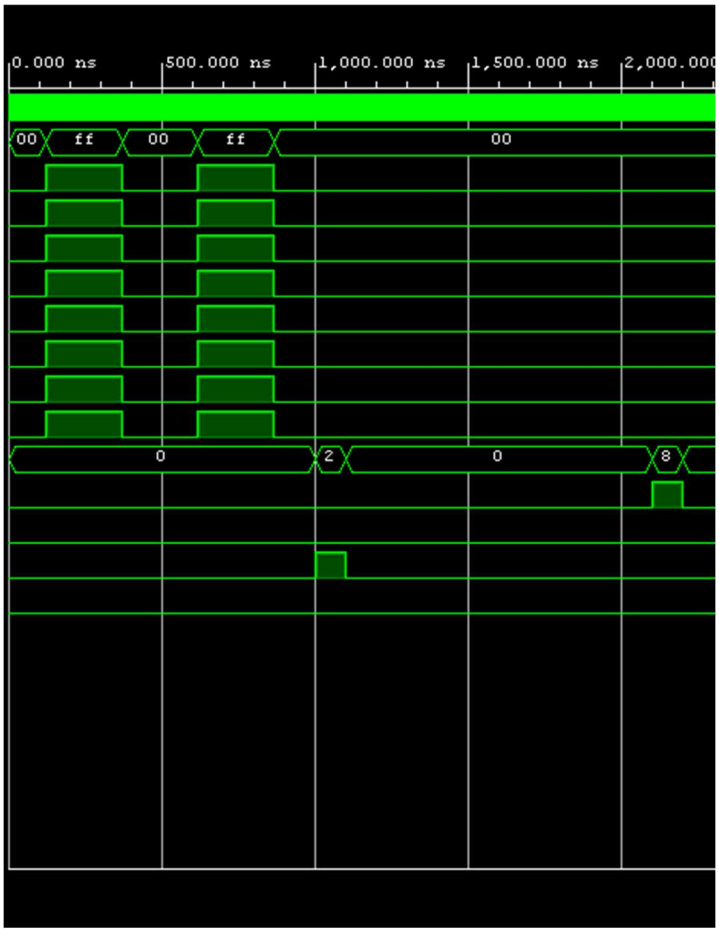
```
51      process is
52      begin
53          btn <= "0000";
54          wait for 1000ns;
55          btn <= "0100"; -- uruchomienie zmiany prędkości
56          wait for 100ns;
57          btn <= "0000";
58          wait for 1000ns;
59          btn <= "1000"; -- uruchomie drugiego scheamtu
60          wait for 100ns;
61          btn <= "0000";
62          wait for 1000ns;
63          btn <= "0010"; -- włącz/ wyłącz
64          wait for 100ns;
65
66      end process;
67
68      inst: entity work.TOP(behavioral)
69      port map (
70          clk => C,
71          btn => btn,
72          je => Y
73      );
74
75  end Behavioral;
```

Rysunek 13 Kod testbench'a c.d.

Komentarz:

Testbench układu został zrealizowany w oparciu o sztucznie wygenerowany zegar o częstotliwości 1GHz. W liniach 53-63 są opisane sygnału odpowiadające poszczególnym przyciskom. Na początku zaczynamy od stanu domyślnego, czyli gdy żaden z przycisków nie jest włączony. Następnie uruchamiamy drugi schemat świecenia i ostatecznie korzystamy z przycisku reset.

Dodatkowo został tutaj instancjonowany projekt zawierający wszystkie wcześniej opisane przeze mnie operacje z odpowiednio zmapowanymi portami.

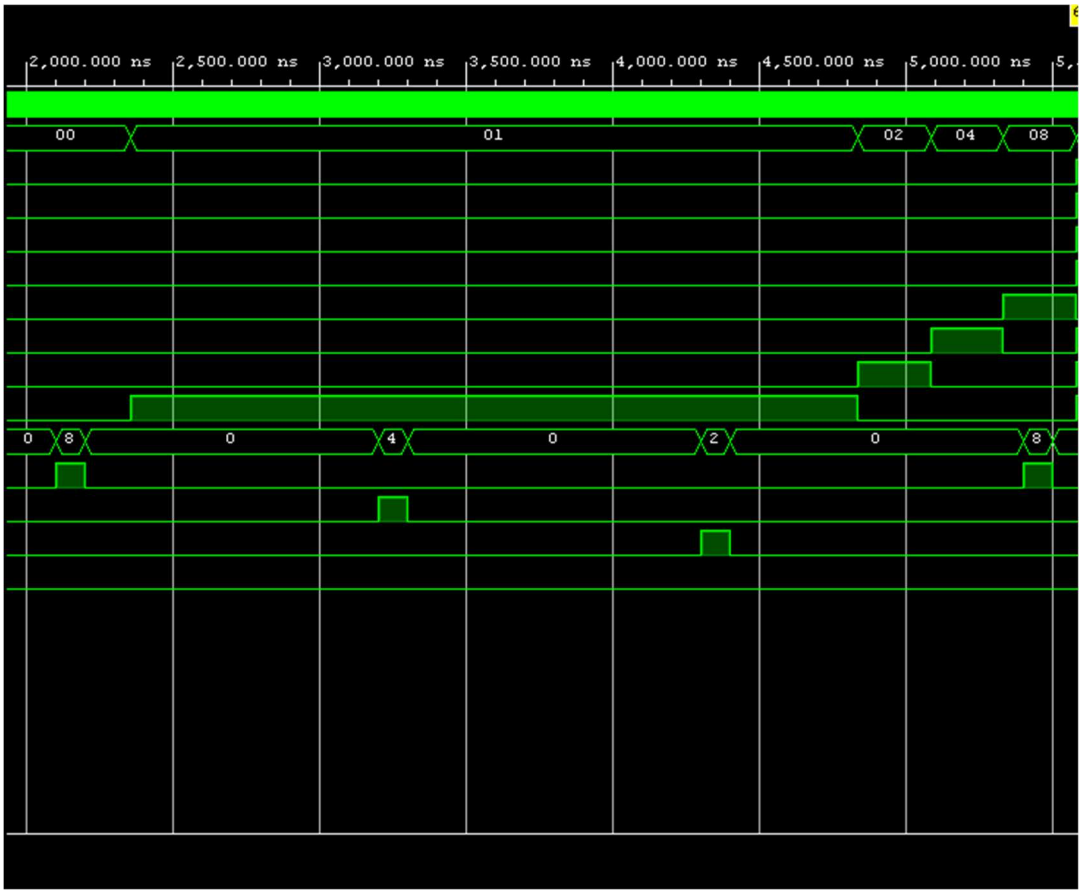


Rysunek 14 Symulacja część 1

Komentarz:

Na początku symulacji zaczynanym w stanie domyślnym, czyli wariancie 1, w którym negujemy diody. Można zauważyć, że ta operacja przebiega pomyślne, gdyż stany na wyjściu zmieniają się odpowiednio.

W dalszej części symulacji klikamy guzik do resetu układu i obserwujemy, że rzeczywiście na wyjściu wtedy stany się nie zmieniają niezależnie od stanu sygnału zegarowego.



Rysunek 15 Symulacja część 2

Komentarz:

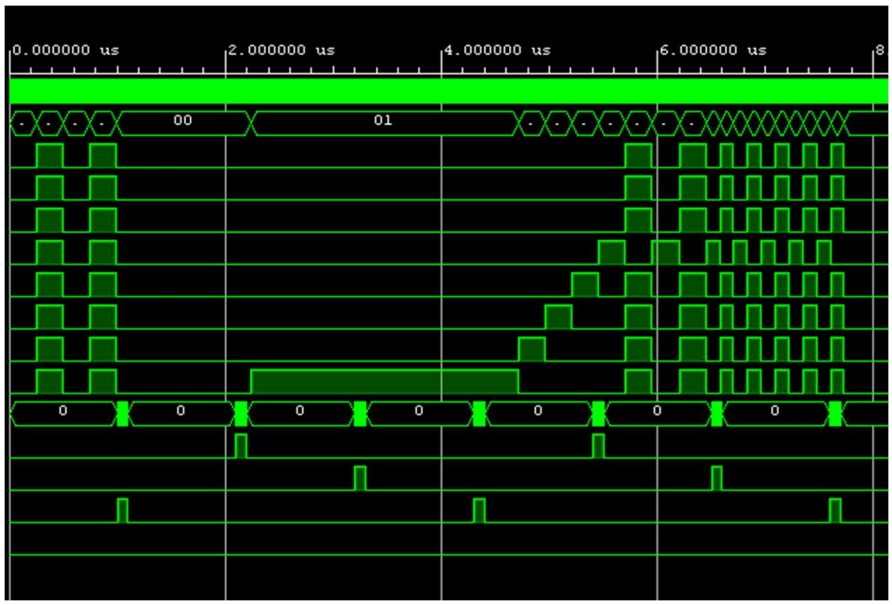
W kolejnym kroku klikamy guzik do zmiany wariantu i obserwuje, że zapaliła się dioda odpowiadająco obecnej wartość sygnału Counter, niemniej jednak praca dalej nie następuje, bo cały czas jesteśmy w stanie reset.

Dalej guzik reset klikamy ponownie i obserwujemy odpowiednie przesuwanie się diody.

Rysunek 16 Symulacja część 3

Komentarz:

Dalej ponownie klikamy guzik do zmiany wariantu i guzik do zmiany prędkości pracy układu. Na przebiegach czasowych możemy zaobserwować, że również ta operacja jest realizowana poprawnie.



Rysunek 17 Pełen cykl pracy zaobserwowany na symulacji