Generowanie sygnałów testowych VHDL

Wariant współbieżny (bez procesu):

sygnał <= stan początkowy, stan_1 **after** time_1, stan_2 **after** time_2, ... ;

time_n - punkty czasowe podawane narastająco

Wariant sekwencyjny (wewnątrz procesu):

sygnał <= stan_1; **wait for** delay_1;
sygnał <= stan_2; **wait for** delay_2;

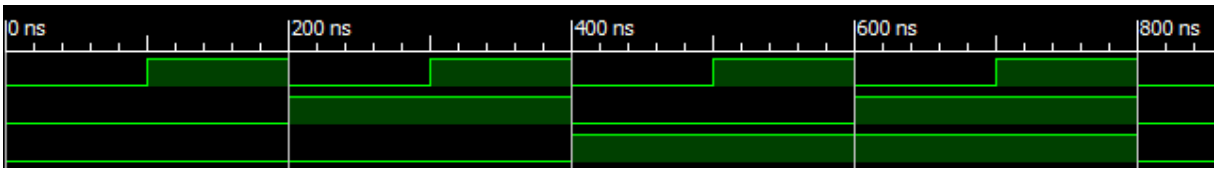
delay_n - opóźnienie pomiędzy kolejnymi zmianami

W wariancie z wektorem i pętlą **for** wykonywane są konwersje typów, wymagające użycia odpowiedniej biblioteki. Należy sprawdzić, czy w nagłówku pliku znajduje się instrukcja:

USE ieee.numeric_std.ALL;

Przykład 1

Aby w pełni przetestować układ o trzech wejściach IN_0, IN_1 i IN_2 chcemy wygenerować wszystkie możliwe kombinacje sygnałów wejściowych.



Wariant 1 (współbieżny)

```
IN_0 <=   '0', '1' after 100 ns, '0' after 200 ns, '1' after 300 ns, '0' after 400 ns,
              '1' after 500 ns, '0' after 600 ns, '1' after 700 ns, '0' after 800 ns;
IN_1 <=   '0', '1' after 200 ns, '0' after 400 ns, '1' after 600 ns, '0' after 800 ns;
IN_2 <=   '0', '1' after 400 ns, '0' after 800 ns;
```

Wariant 2 (jeden proces)

```
tb : process
  begin
    IN_2 <= '0';   IN_1 <= '0';   IN_0 <= '0';   wait for 100 ns;
    IN_2 <= '0';   IN_1 <= '0';   IN_0 <= '1';   wait for 100 ns;
    IN_2 <= '0';   IN_1 <= '1';   IN_0 <= '0';   wait for 100 ns;
    IN_2 <= '0';   IN_1 <= '1';   IN_0 <= '1';   wait for 100 ns;
    IN_2 <= '1';   IN_1 <= '0';   IN_0 <= '0';   wait for 100 ns;
    IN_2 <= '1';   IN_1 <= '0';   IN_0 <= '1';   wait for 100 ns;
    IN_2 <= '1';   IN_1 <= '1';   IN_0 <= '0';   wait for 100 ns;
    IN_2 <= '1';   IN_1 <= '1';   IN_0 <= '1';   wait for 100 ns;
    IN_2 <= '0';   IN_1 <= '0';   IN_0 <= '0';   -- opcja (końcowy stan wejść = 000)
    wait;                                        -- aby nie powtarzać sekwencji
  end process;
```

Wariant 3 (jeden proces, pominięte nadmiarowe przypisania)

```
tb : process
   begin
      IN_2 <= '0';   IN_1 <= '0';   IN_0 <= '0';   wait for 100 ns;
                                    IN_0 <= '1';   wait for 100 ns;
                     IN_1 <= '1';   IN_0 <= '0';   wait for 100 ns;
                                    IN_0 <= '1';   wait for 100 ns;
      IN_2 <= '1';   IN_1 <= '0';   IN_0 <= '0';   wait for 100 ns;
                                    IN_0 <= '1';   wait for 100 ns;
                     IN_1 <= '1';   IN_0 <= '0';   wait for 100 ns;
                                    IN_0 <= '1';   wait for 100 ns;
      IN_2 <= '0';   IN_1 <= '0';   IN_0 <= '0';   -- opcja (końcowy stan wejść = 000)
      wait;                                        -- aby nie powtarzać sekwencji
   end process;
```

W pokazanych powyżej przykładach pełna sekwencja testowa generowana jest jednokrotnie. W wariantach z procesami można uzyskać powtarzanie sekwencji w nieskończoność przez usuniecie instrukcji **wait** na końcu procesu.

Poniżej przedstawione są warianty z niezależnymi procesami dla wszystkich wejść z powtarzaniem sekwencji testowej w nieskończoność. W tym przypadku dodanie instrukcji **wait** na końcu procesów nie dałoby oczekiwanych sekwencji sygnałów testowych.

Wariant 4 (niezależne procesy dla wejść)

```
in_0_tb : process                    -- proces dla wejścia IN_0
   begin
      IN_0 <= '0';   wait for 100 ns;
      IN_0 <= '1';   wait for 100 ns;
   end process;
```

```
in_1_tb : process                    -- proces dla wejścia IN_1
   begin
      IN_1 <= '0';   wait for 200 ns;
      IN_1 <= '1';   wait for 200 ns;
   end process;
```

```
in_2_tb : process                    -- proces dla wejścia IN_2
   begin
      IN_2 <= '0';   wait for 400 ns;
      IN_2 <= '1';   wait for 400 ns;
   end process;
```

W wariancie z niezależnymi procesami można w razie potrzeby wygenerować sekwencję testową zawierającą zadaną liczbę zmian sygnałów. Przykład z użyciem pętli **for** przedstawiony jest poniżej.

Wariant 5 (niezależne procesy dla wejść, zadana liczba powtórzeń)

```
in_0_tb : process                          -- proces dla wejścia IN_0
   begin
      for i in 1 to 4 loop                 -- pętla określająca liczbę powtórzeń
         IN_0 <= '0';    wait for 100 ns;
         IN_0 <= '1';    wait for 100 ns;
      end loop;
      wait;                                -- aby nie powtarzać sekwencji
   end process;
```

```
in_1_tb : process                          -- proces dla wejścia IN_1
   begin
      for i in 1 to 2 loop                 -- pętla określająca liczbę powtórzeń
         IN_1 <= '0';    wait for 200 ns;
         IN_1 <= '1';    wait for 200 ns;
      end loop;
      wait;                                -- aby nie powtarzać sekwencji
   end process;
```

```
in_2_tb : process                          -- proces dla wejścia IN_2
   begin
      IN_2 <= '0';    wait for 400 ns;
      IN_2 <= '1';    wait for 400 ns;
      wait;                                -- aby nie powtarzać sekwencji
   end process;
```

W przypadku, gdy czasy stanu niskiego i wysokiego sygnałów testowych są jednakowe można ustawić stan początkowy sygnału a następnie cyklicznie zmieniać go na przeciwny. Wariant taki przedstawiono poniżej.

Wariant 6 (niezależne procesy dla wejść, zadana liczba powtórzeń, negacja sygnału)

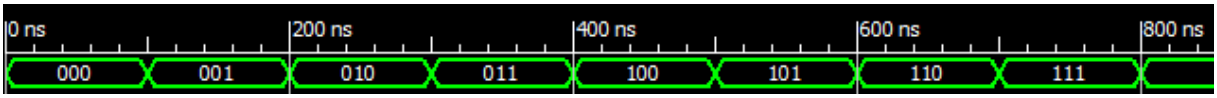
```
in_0_tb : process                    -- proces dla wejścia IN_0
   begin
      IN_0 <= '0';                   -- stan początkowy
      for i in 1 to 8 loop           -- pętla określająca liczbę powtórzeń
         wait for 100 ns;
         IN_0 <= not IN_0;           -- zmiana stanu na przeciwny
      end loop;
      wait;                          -- aby nie powtarzać sekwencji
   end process;
```

```
in_1_tb : process                    -- proces dla wejścia IN_1
   begin
      IN_1 <= '0';                   -- stan początkowy
      for i in 1 to 4 loop           -- pętla określająca liczbę powtórzeń
         wait for 200 ns;
         IN_1 <= not IN_1;           -- zmiana stanu na przeciwny
      end loop;
      wait;                          -- aby nie powtarzać sekwencji
   end process;
```

```
in_2_tb : process                    -- proces dla wejścia IN_2
   begin
      IN_2 <= '0';                   -- stan początkowy
      for i in 1 to 2 loop           -- pętla określająca liczbę powtórzeń
         wait for 400 ns;
         IN_2 <= not IN_2;           -- zmiana stanu na przeciwny
      end loop;
      wait;                          -- aby nie powtarzać sekwencji
   end process;
```

Przykład 2

Układ o trzech wejściach w postaci wektora IN_V (na schemacie użyta magistrala).



Wariant 1 (współbieżny)

```
IN_V <=    “000”,  “001” after 100 ns, “010” after 200 ns, “011” after 300 ns,
                   “100” after 400 ns, “101” after 500 ns, “110” after 600 ns,
                   “111” after 700 ns, “000” after 800 ns;
```

Wariant 2 (proces)

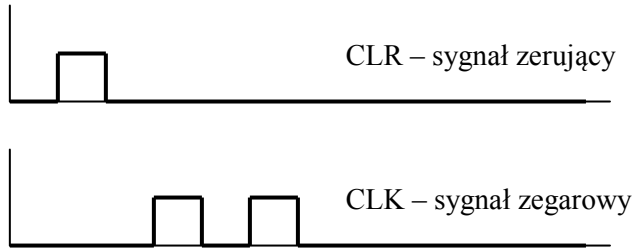
```
tb : process
   begin
      IN_V <= “000”;   wait for 100 ns;
      IN_V <= “001”;   wait for 100 ns;
      IN_V <= “010”;   wait for 100 ns;
      IN_V <= “011”;   wait for 100 ns;
      IN_V <= “100”;   wait for 100 ns;
      IN_V <= “101”;   wait for 100 ns;
      IN_V <= “110”;   wait for 100 ns;
      IN_V <= “111”;   wait for 100 ns;
      IN_V <= “000”;                    -- opcja
      wait;
   end process;
```

Wariant 3 (proces z pętlą)

```
tb : process
   begin
      for i in 0 to 7 loop              -- pętla określająca liczbę powtórzeń
         IN_V <= std_logic_vector(to_unsigned(i, 3));
         wait for 100 ns;
      end loop;
      IN_V <= “000”;                    -- opcja
      wait;
   end process;
```

Przykład 3

Dla licznika chcemy wygenerować sygnał zerujący (impuls o długości 50 ns, początek w punkcie 50 ns) oraz dwa impulsy zegarowe o okresie 50 + 50 ns, początek pierwszego 150 ns)



Wariant 1 (współbieżny)

```
CLR <= ‘0’, ‘1’ after 50 ns, ‘0’ after 100 ns;
CLK <= ‘0’, ‘1’ after 150 ns, ‘0’ after 200 ns, ‘1’ after 250 ns, ‘0’ after 300 ns;
```

Wariant 2 (jeden proces)

```
tb_clr_clk : process                    -- proces dla sygnałów CLR i CLK
   begin
      CLR <= ‘0’;
      CLK <= ‘0’;    wait for 50 ns;
      CLR <= ‘1’;    wait for 50 ns;
      CLR <= ‘0’;    wait for 50 ns;
      CLK <= ‘1’;    wait for 50 ns;
      CLK <= ‘0’;    wait for 50 ns;
      CLK <= ‘1’;    wait for 50 ns;
      CLK <= ‘0’;
      wait;                             -- aby nie powtarzać sekwencji
   end process;
```

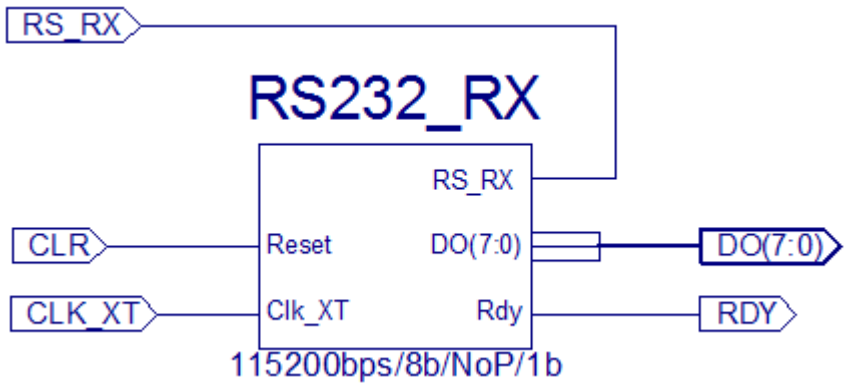
Wariant 3 (niezależne procesy dla CLR i CLK)

```
tb_clr : process                        -- proces dla sygnału CLR
   begin
      CLR <= '0';
      wait for 50 ns;                   -- czas rozpoczęcia zerowania
      CLR <= '1';
      wait for 50 ns;                   -- czas sygnału zerującego
      CLR <= '0';
      wait;                             -- aby nie powtarzać sekwencji
   end process;
```

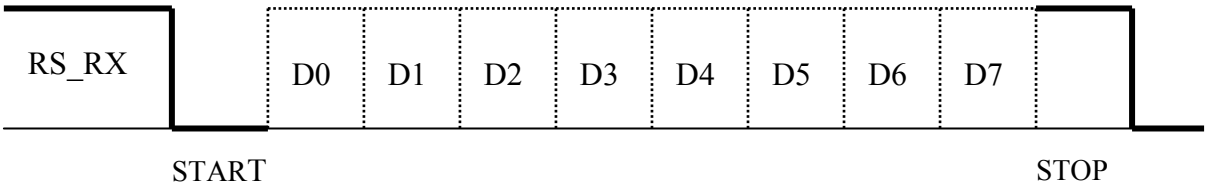
```
tb_clk : process                        -- proces dla sygnału CLK
   begin
      CLK <= '0';
      wait for 150 ns;                  -- czas do pierwszego zbocza narastającego
      for i in 1 to 2 loop              -- pętla określająca liczbę impulsów
         CLK <= '1';
         wait for 50 ns;                -- czas poziomu wysokiego impulsu
         CLK <= '0';
         wait for 50 ns;                -- czas poziomu niskiego impulsu
      end loop;
      wait;                             -- aby nie powtarzać sekwencji
   end process;
```

Symulacja odbioru danych RS232

Symulacja jest możliwa również przy testowaniu układów zawierających układ odbiornika RS (moduł RS232_RX). Załóżmy, że układ testowy zawiera moduł RS232_RX podłączony jak na poniższym schemacie:



Aby przeprowadzić symulację takiego układu konieczne będzie wygenerowanie na linii odbiornika RS_RX przebiegów testowych, odzwierciedlających dane odbierane w postaci szeregowej:



Odbiór bajtu rozpoczyna się po zmianie stanu linii RS_RX z 1 (stan pasywny) na 0 (bit START = 0). Po bicie startu odbierane są kolejne bity danych (8 bitów, w kolejności od najmłodszego do najstarszego) oraz bit STOP = 1. Po bicie stopu może od razu pojawić się bit startu kolejnego bajtu, ale linia odbiornika może również utrzymywać się w stanie pasywnym przez dowolnie długi czas (brak transmisji danych).

Symulacja odbioru kilku bajtów wymaga wygenerowania odpowiednich przebiegów na trzech liniach:

CLR - reset odbiornika (jednorazowo)
CLK_XT - zegar sterujący pracą modułu RS232_RX (przebieg okresowy)
RS_RX - przebiegi symulujące odbierane dane

Aby symulować odbiór danych trzeba znać powiązanie częstotliwości taktującej moduł RS232_RX z prędkością transmisji (liczbę taktów zegara przypadającą na jeden bit). W układzie docelowym (na płytce ZL-9572) mamy:

CLK_XT = 1.8432 MHz (okres zegara = 542 534 ps)
prędkość transmisji szeregowej BAUD = 115 200 bps

Wynika z tego, że liczba okresów zegara przypadająca na jeden bit jest równa:

BIT_CLOCKS = 1 843 200 Hz / 115 200 bps = 16

Poniżej podane zostaną wskazówki jak zmodyfikować wygenerowany automatycznie wzorzec modułu testowego (VHDL Test Bench). Dla przejrzystości podane poniżej parametry zostaną zdefiniowane jako stałe:

```
constant BIT_CLOCKS      : positive := 16;
constant CLK_XT_PERIOD   : time := 542534 ps;
constant BIT_PERIOD      : time := BIT_CLOCKS * CLK_XT_PERIOD;
constant SIM_CLOCKS      : positive := 810;
```

Stała SIM_CLOCKS określa długość symulacji w cyklach zegarowych i powinna być dostosowana do liczby symulowanych bajtów. Każdy bajt zawiera 10 bitów (8 bitów danych oraz bity startu i stopu). Zakładając, że pomiędzy odbieranymi bajtami nie ma dodatkowych opóźnień czas odbioru *n* bajtów jest równy:

SIM_CLOCKS = *n* * 10 * BIT_CLOCKS = *n* * 160

Wartość tę trzeba zwiększyć o czas wymagany na reset układu i ewentualną zwłokę przed rozpoczęciem symulacji odbioru danych (przeliczone na okresy zegara).

Aby symulacja odbioru kilku różnych bajtów była prosta i wygodna zdefiniowana została procedura *sim_rx_byte*. Jako argumenty procedura przyjmuje wartość odbieranego bajtu oraz linię, na której powinien być wystawiony w postaci szeregowej podany bajt.

```
-------------------------------------------------------------------------------------------------
-- Simulate received byte: RX_BYTE - received byte, RX_LINE - receiver line
-------------------------------------------------------------------------------------------------
procedure sim_rx_byte (constant RX_BYTE : in STD_LOGIC_VECTOR;
                       signal RX_LINE : out STD_LOGIC) is
   begin
      RX_LINE <= '0';                  -- START bit
      wait for BIT_PERIOD;
      for i in RX_BYTE'reverse_range loop
         RX_LINE <= RX_BYTE(i);
         wait for BIT_PERIOD;          -- DATA bit i = 0 .. 7
      end loop;
      RX_LINE <= '1';
      wait for BIT_PERIOD;             -- STOP bit
   end procedure sim_rx_byte;
```

Definicje stałych oraz procedury należy umieścić w bloku architektury (po definicji COMPONENT i sygnałów a przed blokiem BEGIN ... END. Wewnątrz bloku powinny być umieszczone definicje procesu *tb_clk* (symulacja CLK_XT) oraz głównego procesu *tb_rx*.

```
tb_clk : process
   begin
      CLK_XT <= '0';
      wait for 100 ns;
      for i in 1 to SIM_CLOCKS loop
         CLK_XT <= '0';
         wait for CLK_XT_PERIOD/2;
         CLK_XT <= '1';
         wait for CLK_XT_PERIOD/2;
      end loop;
      wait;                            -- aby nie powtarzać sekwencji
   end process;
```

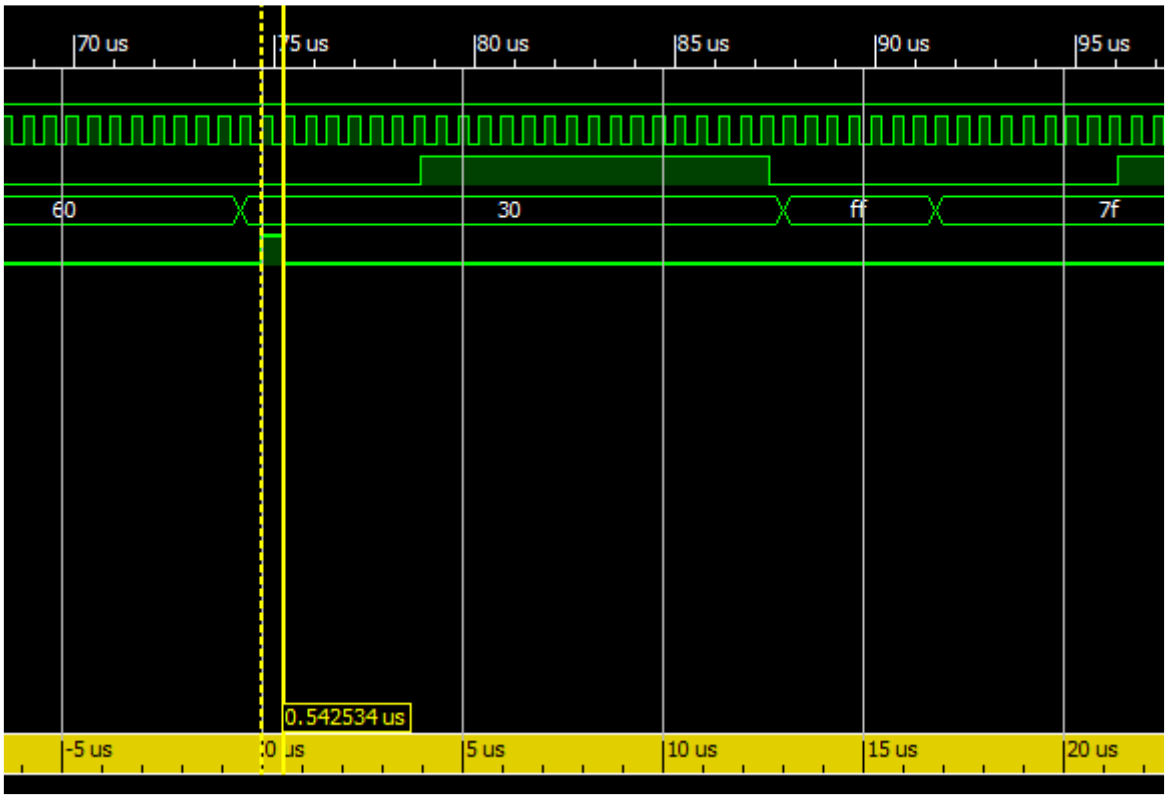
```
tb_rx : process
    begin
        RS_RX <= '1';                   -- RS_RX w stanie pasywnym
        CLR <= '1';
        wait for 50 ns;                 -- czas sygnału zerującego
        CLR <= '0';
        wait for 500 ns;
        sim_rx_byte(x"30", RS_RX);      -- '0'      30h
        sim_rx_byte(x"41", RS_RX);      -- 'A'      41h
        sim_rx_byte(x"31", RS_RX);      -- '1'      31h
        sim_rx_byte(x"52", RS_RX);      -- 'R'      52h
        sim_rx_byte(x"39", RS_RX);      -- '9'      39h
        wait;                           -- aby nie powtarzać sekwencji
    end process;
```

Poniżej pokazano dwa fragmenty wyników symulacji, kolejność sygnałów od góry:

CLR CLK_XT RS_RX DO RDY

Pierwszy fragment pokazuje końcowy etap odbioru pierwszego bajtu (30h). Na wykresie widać, że w trakcie odbioru dane na magistrali DO zmieniają się. Można zauważyć, że impuls RDY pojawia się jeszcze przed rozpoczęciem bitu stopu, z niewielkim przesunięciem w stosunku do środka najstarszego bitu danych.

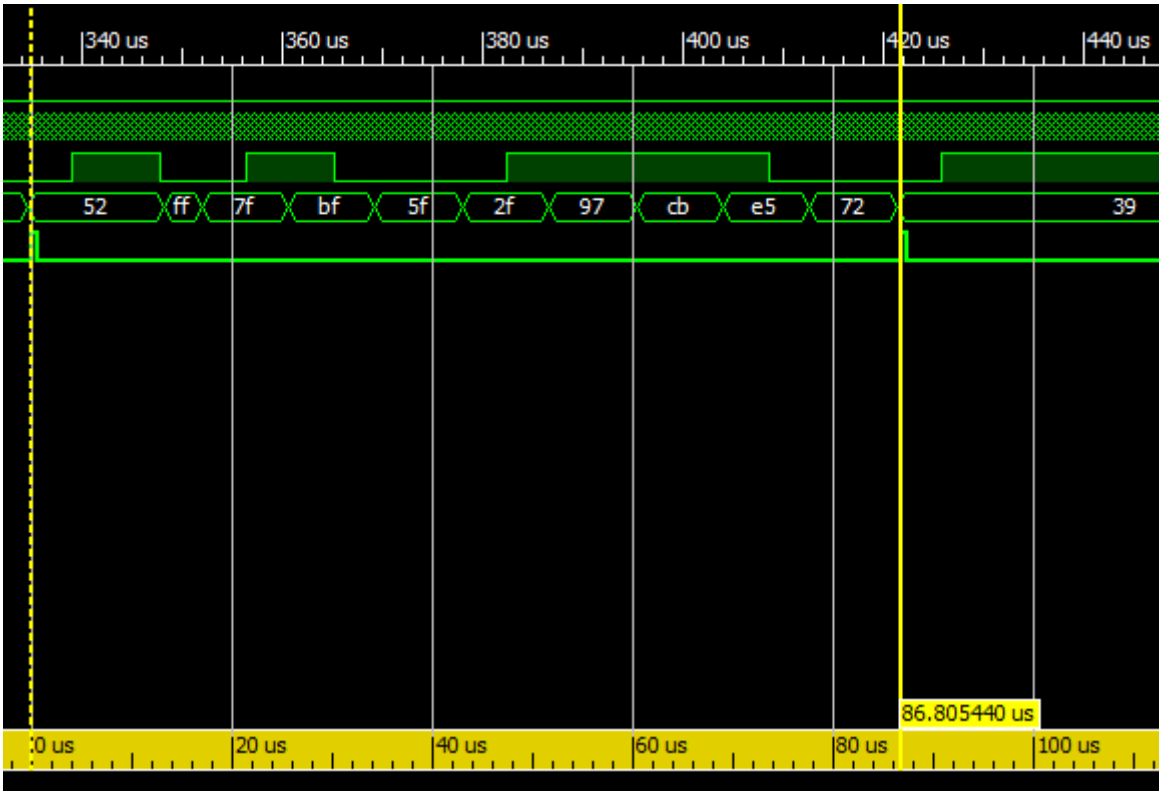
Widać również, że dane na magistrali DO (30h) utrzymują się aż do początku transmisji kolejnego bajtu (na początku bitu startu zmieniają się na FFh). Zmierzona długość impulsu RDY, zgodnie z oczekiwaniem, jest równa jednemu okresowi zegara CLK_XT.



Drugi fragment wyników symulacji ilustruje końcowy etap odbioru przedostatniego bajtu (52h) i pełny cykl odbioru ostatniego bajtu (39h). Pierwsza jedynka na linii RS_RX to bit stopu bajtu 52h, druga to bit D0 wartości 39h a trzecia jedynka o długości trzech bitów reprezentuje bity D3 - D4 - D5 bajtu 39h.

Na wykresie zmierzony został czas pomiędzy kolejnymi impulsami RDY, czyli całkowity czas transmisji jednego bajtu. Jest on zgodny z oczekiwanym na podstawie teoretycznych obliczeń:

10 * BITS_CLOCKS * CLK_XT_PERIOD = 10 * 16 * 0.542534 µs = 86.80544 µs



Warto zauważyć, że moduł RS232_RX dostępny jest tylko w postaci skompilowanej (pliki .ngc i .sym) dlatego opisane symulacje można przeprowadzić tylko w wariancie *Post-Fit*. Przy próbie symulacji typu *Behavioral* uzyskuje się stany niezdefiniowane na wyjściach modułu.