

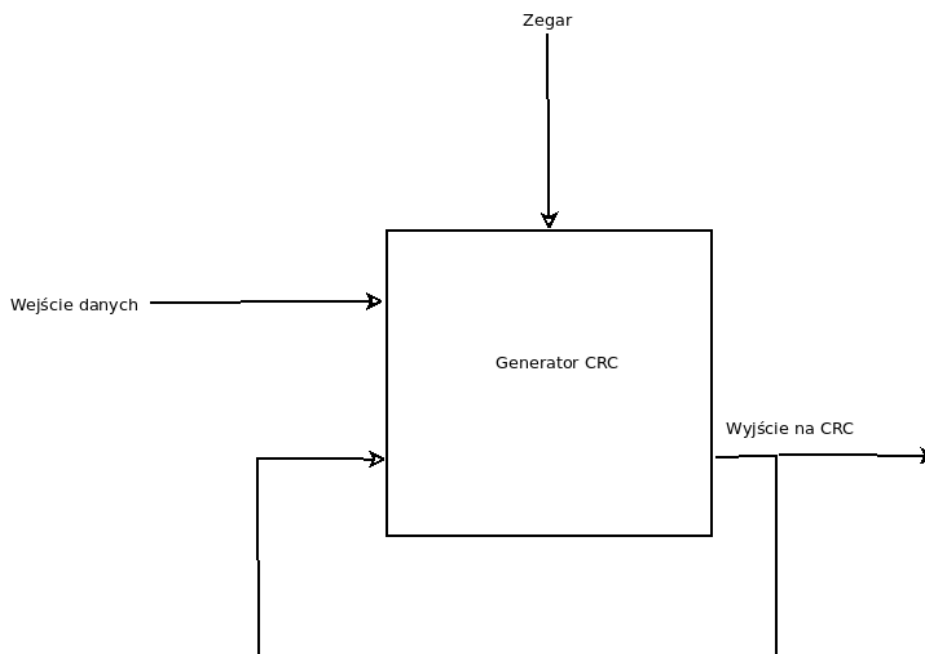
Lista 4 (VHDL) - rozwiązania

Błażej Wróbel, 250070, 3. rok, W11, informatyka

Zadanie 1.

A. Jak wygląda schemat blokowy generatora CRC-8 ?

Generator CRC ma wejścia na sygnał zegara i dane (dla których musi obliczyć sumę kontrolną) oraz wyjście na obliczoną sumę kontrolną. Po analizie kodu pakietu do generowania nowych sum kontrolnych można stwierdzić, że do obliczenia nowej wartości CRC generator korzysta z poprzedniej wartości obliczonej sumy. Zatem wartość na wyjściu CRC, musi być również przekierowana na wejście generatora. Z powyższego rozumowania wynika, że schemat blokowy tego generatora może wyglądać następująco:



B. Jak jest realizowana pamięć ROM ?

W tej architekturze pamięć ROM ma 8 bitowe komórki, które adresuje za pomocą 3 bitów (zatem możemy zaadresować maksymalnie 8 komórek). "Pamiętanie" jest realizowane za pomocą 8 elementowej tablicy 8 bitowych wektorów logicznych. Odczyt z pamięci (to jest pamięć ROM, zatem możemy tylko z niej czytać) odbywa się poprzez przekierowanie na wyjście pamięci wektora logicznego znajdującego się pod żądanym adresem (czyli pod konkretnym indeksem w tablicy wektorów logicznych)

C. Co jest przechowywane w tej pamięci ?

W pamięci ROM pod n-tym adresem jest przechowywana n-ta suma CRC (w pierwszym przypadku ta suma jest liczona dla wejściowego wektora **a0** (heksadecymalnie), a w drugim dla wejściowego wektora **66** (heksadecymalnie) i 'starego' CRC, które jest 8 elementowym wektorem zer). Przykład:

Zadanie 2.

A. Co realizują linie 68-72 ?

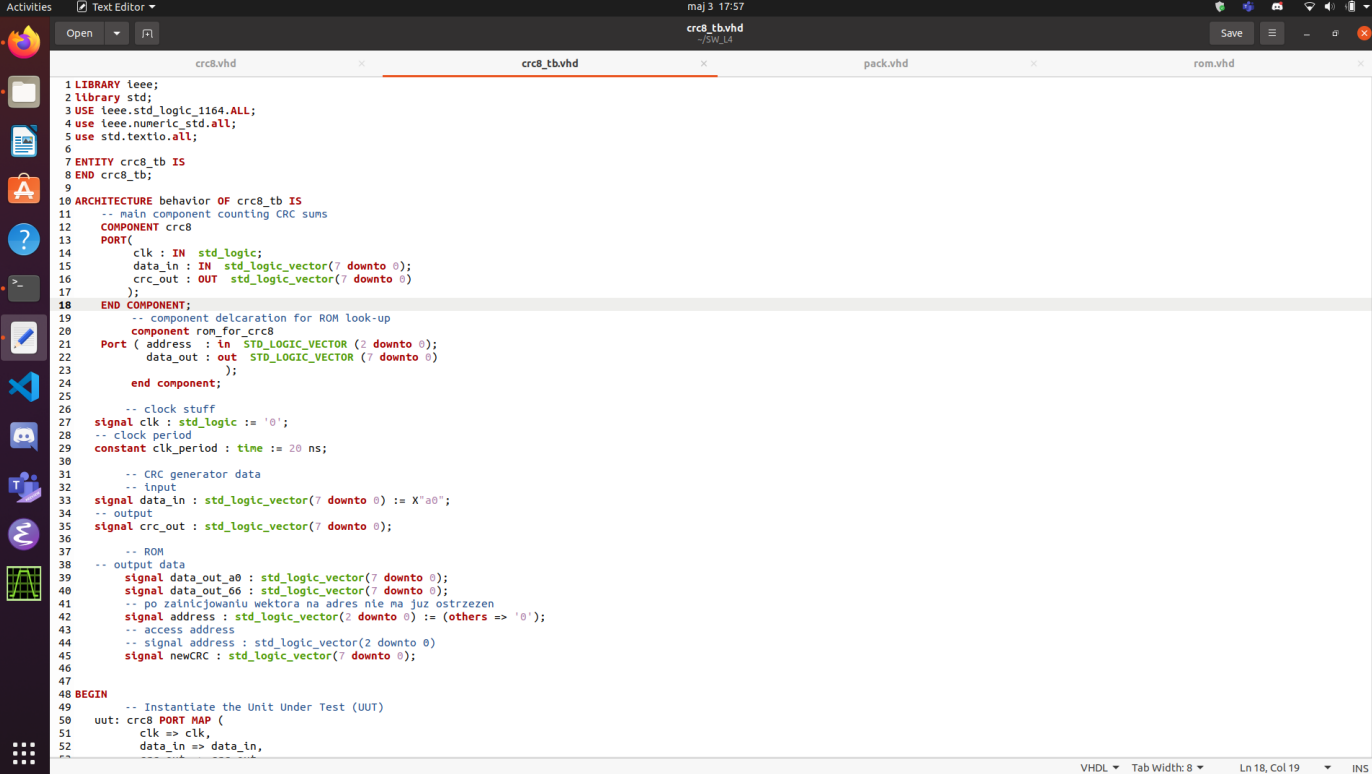
Układ, który testujemy składa się z dwóch komponentów : generatora sum CRC oraz pamięci ROM. Generator CRC do wygenerowania nowej sumy potrzebuje wartości poprzedniej sumy, która jest zapisana w pamięci ROM. Zatem te linie odpowiadają za zsynchronizowanie tych komponentów - generator CRC czeka, aż pamięć "odda" wartość potrzebnej sumy.

B. Co jest przyczyną generowanych przez testbench ostrzeżeń ?

Przyczyną ostrzeżeń generowanych przez testbench było to, że wektor do trzymania adresu nie był zainicjowany żadną wartością, więc mogły być w nim wartości typu undefined. VHDL nie potrafi zrzutować takich wartości na wartości logiczne (które mają być w wektorze typu unsigned), więc zgłasza ostrzeżenie. Rozwiązaniem jest zainicjowanie wektora do trzymania adresu (np. samymi zerami).

Zadanie 3.

Kod testu znajduje się w pliku `crc8_tb.vhd`. Oto kod:



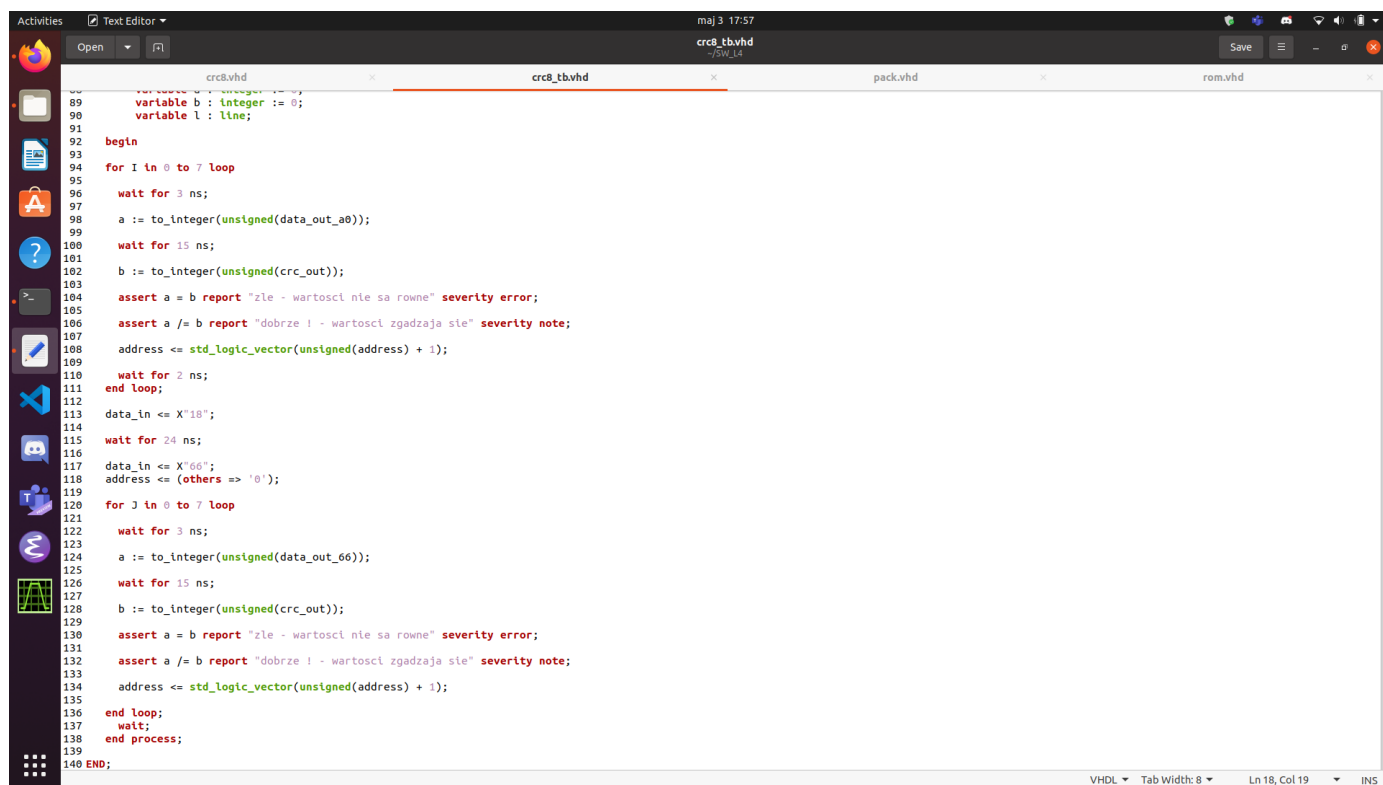
```
1 LIBRARY ieee;
2 library std;
3 USE ieee.std_logic_1164.ALL;
4 use ieee.numeric_std.all;
5 use std.textio.all;
6
7 ENTITY crc8_tb IS
8 END crc8_tb;
9
10 ARCHITECTURE behavior OF crc8_tb IS
11 -- main component counting CRC sums
12 COMPONENT crc8
13 PORT(
14     clk : IN std_logic;
15     data_in : IN std_logic_vector(7 downto 0);
16     crc_out : OUT std_logic_vector(7 downto 0)
17 );
18 END COMPONENT;
19 -- component declaration for ROM look-up
20 component rom_for_crc8
21 Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
22       data_out : out STD_LOGIC_VECTOR (7 downto 0)
23 );
24 end component;
25
26 -- clock stuff
27 signal clk : std_logic := '0';
28 -- clock period
29 constant clk_period : time := 20 ns;
30
31 -- CRC generator data
32 -- input
33 signal data_in : std_logic_vector(7 downto 0) := X"a0";
34 -- output
35 signal crc_out : std_logic_vector(7 downto 0);
36
37 -- ROM
38 -- output data
39 signal data_out_a0 : std_logic_vector(7 downto 0);
40 signal data_out_66 : std_logic_vector(7 downto 0);
41 -- po zainicjowaniu wektora na adres nie ma już ostrzeżeń
42 signal address : std_logic_vector(2 downto 0) := (others => '0');
43 -- access address
44 -- signal address : std_logic_vector(2 downto 0)
45 signal newCRC : std_logic_vector(7 downto 0);
46
47 BEGIN
48 -- Instantiate the Unit Under Test (UUT)
49 uut: crc8 PORT MAP (
50     clk => clk,
51     data_in => data_in,
52     --
```

```
Activities Text Editor maj 3 17:57
Open crc8.vhdl crc8_tb.vhdl pack.vhdl rom.vhdl
Save

38 -- Output data
39 signal data_out_a0 : std_logic_vector(7 downto 0);
40 signal data_out_66 : std_logic_vector(7 downto 0);
41 -- po zainicjowaniu wektora na adres nie ma juz ostrzezen
42 signal address : std_logic_vector(2 downto 0) := (others => '0');
43 -- access address
44 -- signal address : std_logic_vector(2 downto 0);
45 signal newCRC : std_logic_vector(7 downto 0);
46
47 BEGIN
48 -- Instantiate the Unit Under Test (UUT)
49 uut: crc8 PORT MAP (
50     clk => clk,
51     data_in => data_in,
52     crc_out => crc_out
53 );
54
55 -- Instance of ROM lookup for constant X"a0" input
56 rom_a0 : entity work.rom_for_crc8(const_a0)
57 port map (
58     address => address,
59     data_out => data_out_a0
60 );
61
62 -- Instance of ROM lookup for constant X"66" input
63 rom_66 : entity work.rom_for_crc8(const_66)
64 port map (
65     address => address,
66     data_out => data_out_66
67 );
68
69 -- Clock process definitions
70 clk_process : process
71     variable wait_done : natural := 0;
72 begin
73     if wait_done = 0
74     then
75         wait for clk_period * 0.2;
76         wait_done := 1;
77     end if;
78     clk <= '1';
79     wait for clk_period/2;
80     clk <= '0';
81     wait for clk_period/2;
82 end process;
83
84 -- Stimulus process
85 stim_proc: process
86     variable a : integer := 0;
87     variable b : integer := 0;
88     variable l : line;
89 end process;
```

```
Activities Text Editor maj 3 17:57
Open crc8.vhdl crc8_tb.vhdl pack.vhdl rom.vhdl
Save

70 clk_process : process
71     variable wait_done : natural := 0;
72 begin
73     if wait_done = 0
74     then
75         wait for clk_period * 0.2;
76         wait_done := 1;
77     end if;
78     clk <= '1';
79     wait for clk_period/2;
80     clk <= '0';
81     wait for clk_period/2;
82 end process;
83
84 -- Stimulus process
85 stim_proc: process
86     variable a : integer := 0;
87     variable b : integer := 0;
88     variable l : line;
89 begin
90     for I in 0 to 7 loop
91         wait for 3 ns;
92         a := to_integer(unsigned(data_out_a0));
93         wait for 15 ns;
94         b := to_integer(unsigned(crc_out));
95         assert a = b report "zle - wartosci nie sa rowne" severity error;
96         assert a /= b report "dobrze ! - wartosci zgadzaja sie" severity note;
97         address <= std_logic_vector(unsigned(address) + 1);
98         wait for 2 ns;
99     end loop;
100     data_in <= X"10";
101     wait for 24 ns;
102     data_in <= X"66";
103     address <= (others => '0');
104     for J in 0 to 7 loop
105         wait for 3 ns;
106     end loop;
```



```
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
END;
```

W innych plikach kod jest niezmieniony. Do powyższego testu dodałem dwie pętle **for**, które wykonują się 8 razy (bo tyle mamy adresów w pamięci ROM (adres jest pamiętany w 3 bitowym wektorze)). Pierwsza pętla testuje komponent, który jako wartość początkową (w wektorze na dane) przyjmuje **A0** (szesnastkowo), a druga testuje komponent przyjmujący jako wartość początkową **66** (szesnastkowo). Obydwa testy działają następująco: najpierw pobieram do zmiennej wartość z odpowiedniego miejsca w pamięci, potem czekam 15 ns i odbieram do drugiej zmiennej wartość wyliczonego kodu CRC, następnie porównuje obie wartości (tutaj obie wartości są wektorami logicznymi, które mogę rzutować na liczby całkowite - dlatego w testach używam zmiennych całkowitych) - jeśli wartości zmiennych są sobie równe, to zwracam odpowiedź, że jest dobrze (wartości w ROM są uznawane jako poprawne odpowiedzi). W przeciwnym przypadku zwracam informację, że wartość kodu jest zła. Pomiędzy testami muszę wyzerować wartość CRC (jeśli tego nie zrobię, to wtedy do dalszych obliczeń będzie używana wartość CRC z poprzedniego testu, co spowoduje, że odpowiedzi będą błędne) - dlatego na wejście danych daję wartość **66** (szesnastkowo) (xor tych samych wartości zawsze daje zero). Potem czekam 24 ns (pełny okres zegara + czas synchronizacji pakietów) i zeruje adres oraz daję odpowiednią wartość początkową na wejście danych (znowu **66**). W każdej iteracji zwiększam wartość adresu w pamięci o 1 (wartości adresu są w przedziale od 0 do 7) - oczywiście, to się dzieje w obu testach.