Przykłady do wykładu „Języki modelowania i symulacji”

dr inż. Bogdan Pankiewicz

Gdańsk, listopad 2011 – grudzień 2015

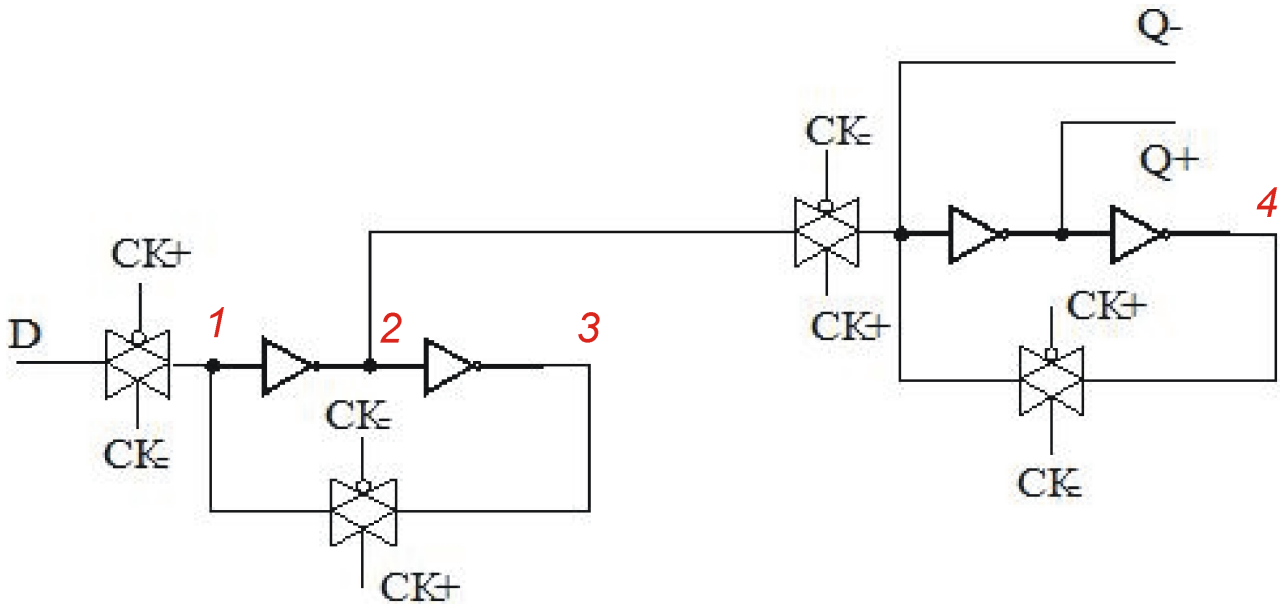
Część I - PSPICE

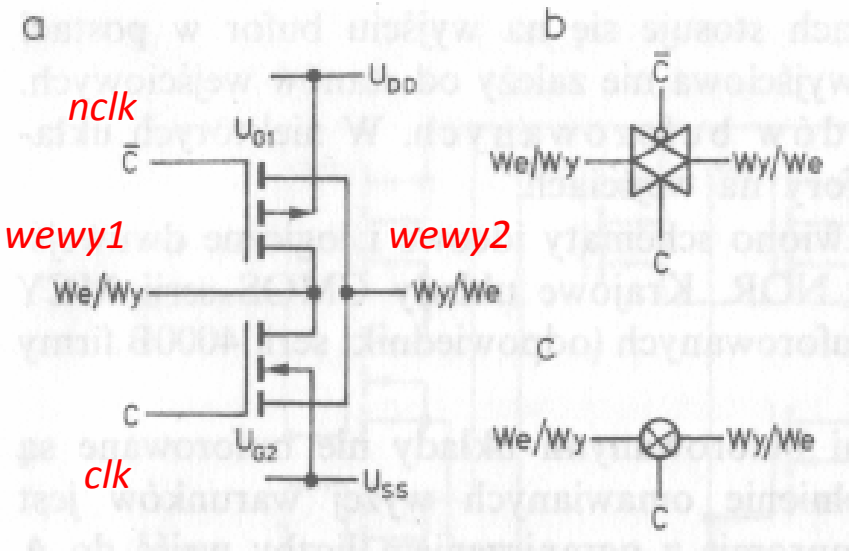
Przykład I - przerzutnik D

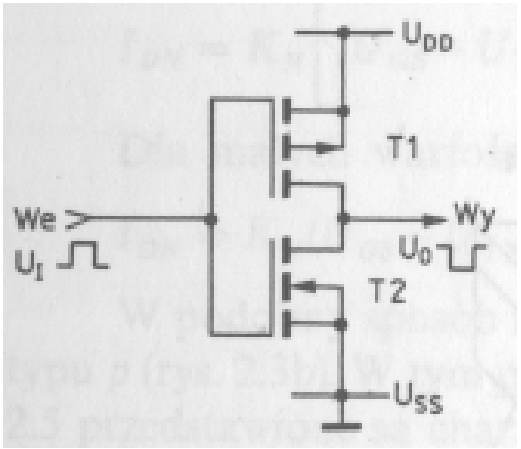
Czynności do wykonania:

- Wprowadzenie opisu bramki transmisyjnej.
- Opis invertera.
- Opis przerzutnika.
- Dodanie analiz.

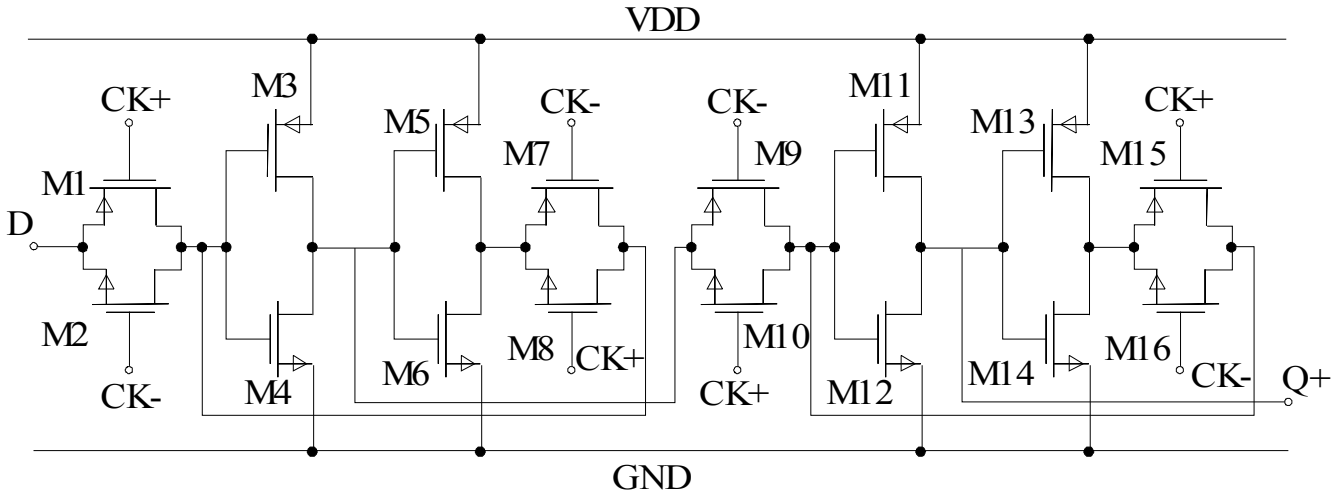
Schemat przerzutnika:







Schemat „płaski”



```
przerzutnik D
.sub b_tran wewy1 wewy2 clk nclk optional: Udd=vdd Uss=0
Mp1 wewy1 nclk wewy2 Udd pfet w=2.7u l=0.6u
Mn1 wewy1 clk wewy2 Uss nfet w=0.9u l=0.6u
.ends

.sub b_not we wy optional: Udd=vdd Uss=0
Mp1 wy we Udd Udd pfet w=2.7u l=0.6u
Mn1 wy we Uss Uss nfet w=0.9u l=0.6u
.ends

.sub DFF D Q+ Q- CK+
Xclk CK+ CK- b_not

Xt1 D 1 CK- CK+ b_tran
Xt2 1 3  CK+ CK- b_tran
Xt3 2 Q-  CK+ CK- b_tran
Xt4 Q- 4  CK- CK+ b_tran

Xn1 1 2  b_not
Xn2 2 3 b_not
Xn3 Q- Q+  b_not
Xn4 Q+ 4  b_not
.ends

Vdd vdd 0 3.3
XDFF D Q+ Q- clk DFF

Vclk clk 0 dc 0 pulse(0 3.3 5n 1p 1p 5n 10n)
Vd   D 0 dc 0 pulse(0 3.3 12.5n 1p 1p 15n 30n )

.tran .01n 120n
.MC 20 tran V([Q+])  RISE_EDGE(1.65)  LIST OUTPUT ALL
.inc "ami_c5.lib"
.probe
.end
```

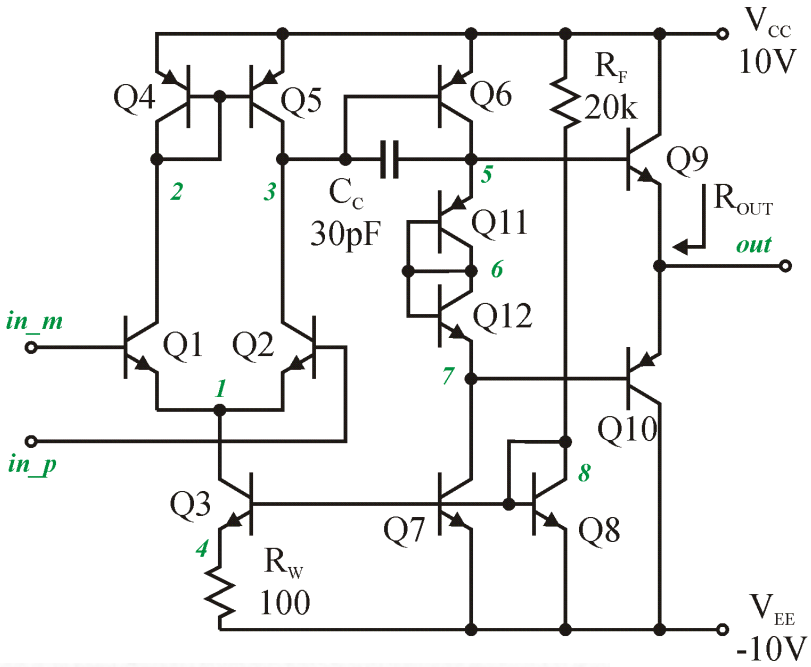
Przykład II – środkowoprzepustowy filtr Tow-Thomas

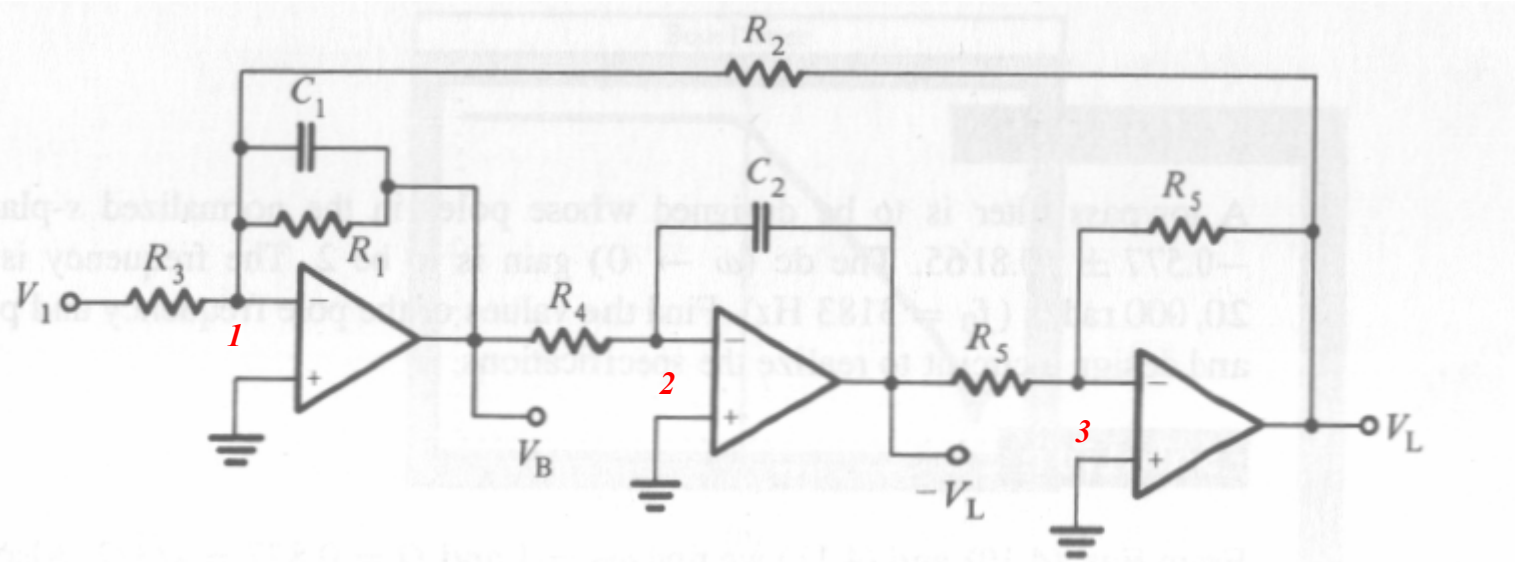
Zadania do wykonania:

- Wprowadzenie opisu wzmacniacza operacyjnego.
- Wprowadzenie opisu filtru.
- Analiza AC – znalezienie częstotliwości środkowej.
- Analiza czasowa dla podania przebiegu sinusoidalnego o częstotliwości środkowej i znalezienie amplitudy dla której jeszcze nie występuje obcięcie sygnału wyjściowego (z dokładnością do 100mV).
- Sprawdzenie wartości THD przebiegu prostokątnego na wejściu oraz na wyjściu filtru o identycznej częstotliwości środkowej i amplitudzie dwukrotnie niższej niż znalezionej w poprzednim punkcie. Należy rozwinąć 20 harmonicznych.
- Sprawdzenie zmian częstotliwości środkowej dla 5% tolerancji rezystancji i 10% pojemności. Zmiany wartości poszczególnych elementów powinny być niezależne od siebie. Należy wykonać 100 losowań.
- Wartość szumu w 3dB paśmie na wyjściu i odniesioną do wejścia.
- Dane elementów:
- R1=100k,
- R2=10k, R3=10k, R4=10k, R5a=10k, R5b=10k,
- C1=10n, C2=10n

Wzmacniacz i filtr - schematy

Modele tranzystorów bipolarnych należy pobrać ze strony laboratorium do przedmiotu JMiS





Odpowiedzi

- fo=1,5912kHz
- amp=0,92V
- THD=45,7% (48,3% teoretyczna dla nieskończonej liczby harmonicznych) i 1,35%
- 1,41-1,79kHz
- Pasmo 1,514–1.6726k, Voutsk=3,89uV, Vinsk=438,8nV

```
II order Tow-Thomas biquad
X1 0 1 Vb OPAMP
X2 0 2 mVl OPAMP
X3 0 3 Vl OPAMP
R1 1 Vb 100k
R2 1 Vl my_res 10k
R3 V1 1 my_res 10k
R4 Vb 2 my_res 10k
R5a mVl 3 my_res 10k
R5b 3 Vl my_res 10k
C1 1 Vb my_cap 10nF
C2 2 mVl my_cap 10nF
.model my_res res(r=1 dev=5%)
.model my_cap cap(c=1 dev=10%)
.param amp=0.92
*.step param amp .9 1 .01
.param fo=1.5912k
*.ac dec 50 10 10meg
.ac lin 1000 1k 2k
*.noise v([vb]) Vin 1
.mc 100 ac v([Vb]) MAX list output all
*.tran 1u 20m 0 1u
*.four 1.5912k 20 v([v1]) v([vb])
Xideal v1 out biquad_bp_id params: fo={fo} Q=10 Ho=10
**** Ideal BP biquad ****
.subckt biquad_bp_id in out params: fo=1 Q=1 Ho=1
E1 out 0 laplace {V(in)}={Ho*(s*fo*2*pi/Q)/(s*s+fo*2*pi/Q*s+2*pi*fo*2*pi*fo)}
.param pi=3.141593
.ends
*************************
*** Input signal
Vin v1 0 dc 0 ac 1 pulse({-amp/2} {amp/2} {1/fo/2} 1p 1p
+ {1/fo/2} {1/fo} ) ;sin(0 {amp} 1.5916k)
*** Power supply
Vee vee 0 -10V
Vcc vcc 0 10V
**** Operational Amplifier subcircuit description ****
.sub OPAMP in_p in_m out optional: vcc=vcc vee=vee
Q1 2 in_m 1 qnpn
Q2 3 in_p 1 qnpn
Q3 1 8 4 qnpn
Q4 2 2 vcc qpnp
Q5 3 2 vcc qpnp
Q6 5 3 vcc qpnp
Q7 7 8 vee qnpn
Q8 8 8 vee qnpn
Q9 vcc 5 out qnpn
Q10 vee 7 out qpnp
Q11 6 6 5 qpnp
Q12 6 6 7 qnpn
Cc 3 5 30p
Rw 4 vee 100
Rf vcc 8 20k
.ends
*****************************************
.lib modele8.lib
.probe
.end
```

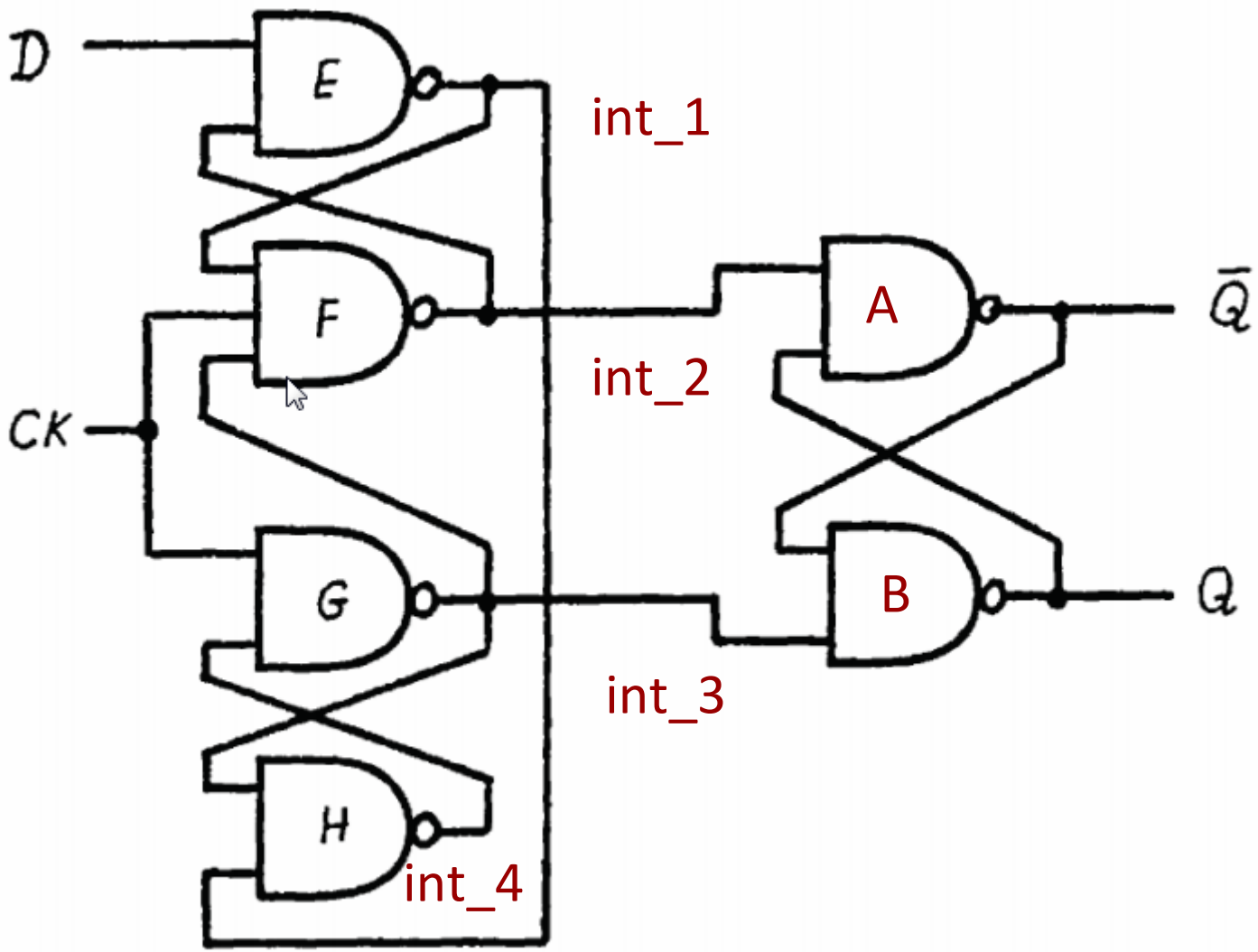
Część II
Verilog

Przykład 1 – przerzutnik D wyzwalany narastającym zboczem zegara składający się wyłącznie z bramek NAND

Zadania do wykonania

- znalezienie schematu i oznaczenie sygnałów wewnętrznych,
- opis modułu w języku Verilog,
- przygotowanie testbemch’a,
- wykonanie symulacji funkcjonalnej.

Schemat przerzutnika D z bramek NAND



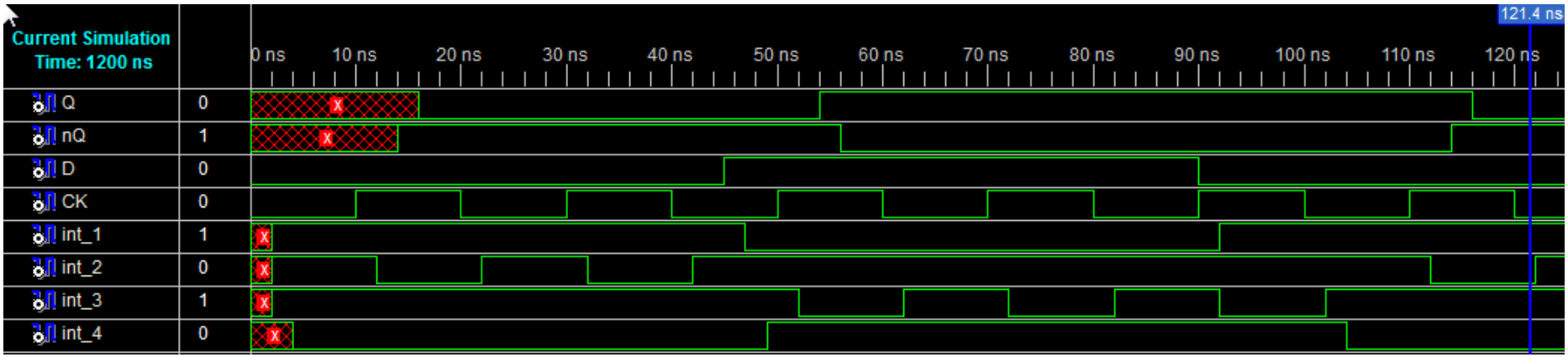
Kod Verilog przerzutnika

```
`timescale 1ns / 1ps
module d_ff_nands(D, CK, Q, nQ );

        input D, CK;
        output Q, nQ;

        wire int_1, int_2, int_3, int_4;

        nand #2 A(nQ,int_2,Q);
        nand #2 B(Q, nQ, int_3);
        nand #2 E(int_1,D,int_2);
        nand #2 F(int_2,int_1,CK,int_3);
        nand #2 G(int_3,CK, int_4);
        nand #2 H(int_4,int_3,int_1);

endmodule
```

Testbench

```
`timescale 1ns / 1ps
module tb;
        // Inputs
        reg D;
        reg CK;
        // Outputs
        wire Q;
        wire nQ;
        // Instantiate the Unit Under Test (UUT)
        d_ff_nands uut (
                .D(D),
                .CK(CK),
                .Q(Q),
                .nQ(nQ)
        );
initial
begin
        // Initialize Inputs
        D = 0;
        CK = 0;
        #200;
        $stop;
end
always
begin
        #10;
        CK = !CK;
end
always
begin
        #45;
        D = !D;
end
endmodule
```

Symulacja


Current Simulation Time: 1200 ns
121.4 ns
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
Q
0
nQ
1
D
0
CK
0
int_1
1
int_2
0
int_3
1
int_4
0

Przykład 2
Analiza dzielnika

Należy opisać w języku Verilog dzielnik z asynchronicznym resetem, wejściem zezwalającym i kierunku (wewnętrznego sygnału), podział przez 1000 zadany parametrem, wyjście powinno mieć wypełnienie 50% .

Zadanie domowe: wskaż 2 błędy w przedstawionym opisie i popraw je. Są to błędy polegające na niezgodności zadanej funkcjonalności z podanym kodem Verilog. Sam kod jest poprawny pod względem składni Verilog.

Ogólna składnia „always”

```
always @(posedge clk or posedge rst)
if (rst)
  begin

... tutaj następuje obsługa resetu...

  end
else
  begin

... tutaj następuje obsługa zbocza zegara...

  end
```

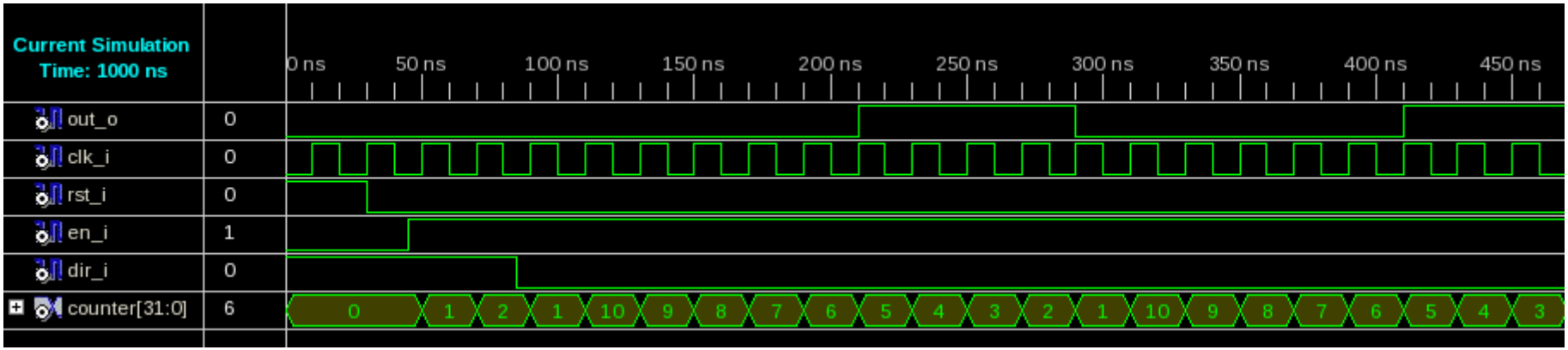
```
`timescale 1ns / 1ps
module counter(clk_i,rst_i,en_i,dir_i,out_o);
input clk_i,rst_i,en_i,dir_i;              // deklaracja kierunku portów
output out_o;                              // deklaracja sygnałów rejestrowych
reg out_o;
integer counter;                           //deklaracja sygnału wewnętrznego
parameter DivRatio=1000;                   // deklaracja parametru

always @(posedge clk_i or posedge rst_i)  //właściwy opis działania
if (rst_i)
  // obsługa części asynchronicznej bloku always
  begin
      counter=0;
      out_o=1'b0;
  end
else
```

```
// obsługa części synchronicznej bloku always
  begin
     if (en_i==1'b1)
        begin
           if (dir_i==1'b1)                    //zliczanie - sygnał „counter”
              if (counter==DivRatio)
                 counter=1;
              else
                 counter=counter+1;
           else
              if (counter==1)
                 counter=DivRatio;
              else
                 counter=counter-1;
                                               // obsługa sygnału wyjściowego "out_o"
           if (counter==1)
              out_o=1'b0;
           if (counter==DivRatio/2)
              out_o=1'b1;
        end
  end
endmodule
```

```
`timescale 1ns / 1ps
module tb;
        reg clk_i;
        reg rst_i;
        reg en_i;
        reg dir_i;
        wire out_o;

        // wstawienie badanego modułu
        counter uut (
                .clk_i(clk_i),
                .rst_i(rst_i),
                .en_i(en_i),
                .dir_i(dir_i),
                .out_o(out_o)
        );
        defparam uut.DivRatio=10;
```

```
//generacja sygnałów
        initial
                begin
                        clk_i = 0;
                        rst_i = 1;
                        en_i = 0;
                        dir_i = 1;
                        #30;
                        rst_i=0;
                        #15 en_i=1;
                        #40 dir_i=0;
                end
        // generacja zegara
        always
                #10 clk_i =~clk_i;

endmodule
```

Wynik symulacji



Przykład 3 zmodyfikowany dzielnik z przykładu 1

- Kod licznika z przykładu 1 został lekko zmodyfikowany,
- Zadanie domowe: po co wprowadzono funkcję CLogB2?

```
`timescale 1ns / 1ps
module counter(clk_i,rst_i,en_i,dir_i,out_o);

// deklaracja kierunku portów
input clk_i,rst_i,en_i,dir_i;
output out_o;
// deklaracja sygnałów rejestrowych
reg out_o;

//deklaracja sygnałów wewnętrznych
parameter DivRatio=1000;
reg [CLogB2(DivRatio-1)-1:0] counter;
//ceil of the log base 2
function integer CLogB2;
        input [31:0] Depth;
        integer i;
        begin
                i = Depth;
                for(CLogB2 = 0; i > 0; CLogB2 = CLogB2 + 1)
                        i = i >> 1;
        end
endfunction
always @(posedge clk_i or posedge rst_i)
if (rst_i)
  // obsługa części asynchronicznej bloku always
  begin
      counter=0;
                out_o=1'b0;
  end
else
```

```
begin// obsługa części synchronicznej bloku always
                if (en_i==1'b1)
                        begin
                                // obsługa licznika wewnętrznego -
                        if (dir_i==1'b1)
                                        if (counter==DivRatio-1)
                                                counter=0;
                                        else
                                                counter=counter+1;
                                else
                                        if (counter==0)
                                                counter=DivRatio-1;
                                        else
                                                counter=counter-1;

                                if (counter==0)
                                        out_o=1'b0;
                                if (counter==DivRatio/2)
                                        out_o=1'b1;
                        end
  end
endmodule
```

Przykład 4 – licznik w kodzie Graya

- Należy zrealizować opis 4-ro bitowego licznika w kodzie Graya,
- Licznik ma być synchroniczny z asynchronicznym resetem
- Wejscia clk_i oraz rst_i, wyjscie gray_o[3:0]

4-ro bitowy kod Gray-a

0000
0001
0011
0010
0110
0111
0101
0100

1100
1101
1111
1110
1010
1011
1001
1000

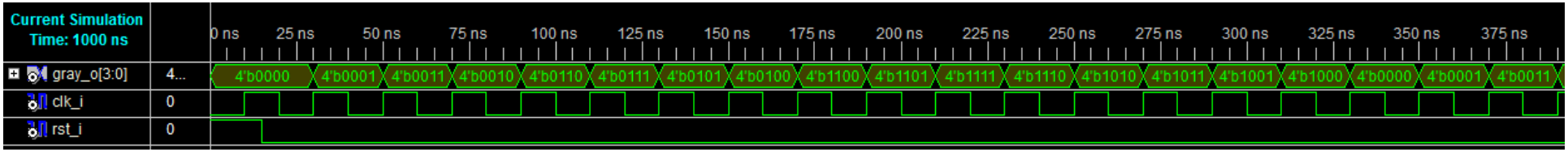
Licznik – opis Verilog

```
`timescale 1ns / 1ps
module gray(clk_i,rst_i,gray_o);
        input clk_i, rst_i;
        output reg [3:0] gray_o;
//synchronus circuit with async reset
always @(posedge clk_i or posedge rst_i)
        if (rst_i)
                gray_o = 4'b0;
        else
                begin
                        case (gray_o)
                                4'b0000: gray_o = 4'b0001;
                                4'b0001: gray_o = 4'b0011;
                                4'b0011: gray_o = 4'b0010;
                                4'b0010: gray_o = 4'b0110;
                                4'b0110: gray_o = 4'b0111;
                                4'b0111: gray_o = 4'b0101;
                                4'b0101: gray_o = 4'b0100;
                                4'b0100: gray_o = 4'b1100;
                                4'b1100: gray_o = 4'b1101;
                                4'b1101: gray_o = 4'b1111;
                                4'b1111: gray_o = 4'b1110;
                                4'b1110: gray_o = 4'b1010;
                                4'b1010: gray_o = 4'b1011;
                                4'b1011: gray_o = 4'b1001;
                                4'b1001: gray_o = 4'b1000;
                                4'b1000: gray_o = 4'b0000;
                                default: gray_o = 4'b0;
                        endcase
                end

endmodule
```

TESTBENCH:

```
`timescale 1ns / 1ps
module tb_gray;

        // Inputs
        reg clk_i;
        reg rst_i;

        // Outputs
        wire [3:0] gray_o;

        // Instantiate the Unit Under Test (UUT)
        gray uut (
                .clk_i(clk_i),
                .rst_i(rst_i),
                .gray_o(gray_o)
        );

        initial
        begin
                // Initialize Inputs
                clk_i = 0;
                rst_i = 1;
                #15
                rst_i = 0;
                // Wait 100 ns for global reset to finish
                #100;

                forever
                        #10 clk_i = ! clk_i;

        end

endmodule
```

Gray - symulacja



Przykład 5 – eliminacja drgań zestyków przycisku

- Należy opisać układ, który wyeliminuje drgania zestyków przycisku,
- Począwszy od naciśnięcia/puszczenia przycisku układ ma na czas 100ms zamrozić zmianę sygnału na wyjściu (sygnał clk_o),
- Dodatkowo wejście sw_i ma wybierać wyjście sygnału „oczyszczonego” lub bezpośrednio z wejścia btn_i,
- Wejścia zegara i resetu asynchronicznego: clk_i, rst_i.

Debouncer - kod

```
`timescale 1ns / 1ps
module debouncer(clk_i,rst_i,btn_i,sw_i,clk_o);
                input clk_i,rst_i,btn_i,sw_i;
                output clk_o;
parameter frequency = 50e6;                                     //in Hertzs
parameter debounce_time = 100;              // in miliseconds

integer counter;

reg [2:0] state;
// 000 - before button pressing
// 001 - just pressed, debounce time not passed
// 010 - pressed and debounce time passed
// 011 - just released, debounce time not passed
// 100 - released and debounce time passed

reg clk_int;
assign clk_o = (sw_i)? btn_i:clk_int;

always @(posedge clk_i or posedge rst_i)
                if (rst_i)
                                begin
                                                counter = 0;
                                                state = 3'b0;
                                                clk_int = 1'b0;
                                end
```

Kod – c.d.

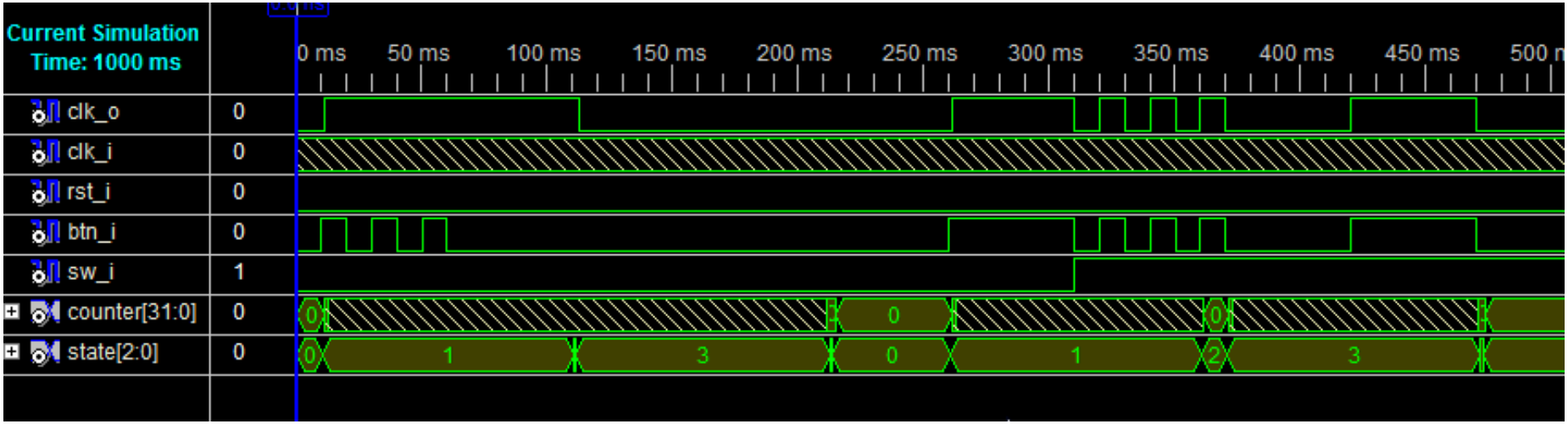
```
else
  begin
    case (state)
                3'b000:         begin
                                        counter = 0;
                                        if (btn_i)
                                        state = 3'b001;
                                end
                3'b001:         begin
                                        clk_int = 1'b1;
                                        if (counter == frequency/1000*debounce_time-1))
                                                state = 3'b010;
                                        else
                                                counter = ounter+1;
                                end
                3'b010:         begin
                                        counter = 0;
                                        if (!btn_i)
                                                state = 3'b011;
                                end
                3'b011:         begin
                                        clk_int = 1'b0;
                                        if (counter == frequency/1000*debounce_time))
                                                state = 3'b100;
                                        else
                                                counter = counter+1;
                                end
                3'b100:         state = 3'b000;
                default state = 0;
    endcase;
  end
endmodule
```

TESTBENCH:

```
  `timescale 1us / 1ps
module tb_debouncer;
                // Inputs
                reg clk_i, rst_i, btn_i, reg sw_i;
                // Outputs
                wire clk_o;
                // Instantiate the Unit Under Test (UUT)
                debouncer uut (.clk_i(clk_i), .rst_i(rst_i), .btn_i(btn_i), .sw_i(sw_i), .clk_o(clk_o));
                defparam uut.frequency = 1e3;

parameter bouncing_period = 10000;
                initial begin
                                // Initialize Inputs
                                clk_i = 0;
                                rst_i = 1;
                                btn_i = 0;
                                sw_i = 0;
                                #100
                                rst_i = 0;
                                #(bouncing_period)
                                btn_i=1;
                                #(bouncing_period)
                                btn_i=0;
                                #(bouncing_period)
                                btn_i=1;
                                #(bouncing_period*1)
                                btn_i=0;
                                #(bouncing_period)
                                btn_i=1;
                                #(bouncing_period)
                                btn_i=0;
                                #(bouncing_period*15)
```

```
#(bouncing_period*5)
                                btn_i=1;
                                #(bouncing_period*5)
                                btn_i=0;

                                sw_i = 1;

                                #(bouncing_period)
                                btn_i=1;
                                #(bouncing_period)
                                btn_i=0;
                                #(bouncing_period)
                                btn_i=1;
                                #(bouncing_period*1)
                                btn_i=0;
                                #(bouncing_period)
                                btn_i=1;
                                #(bouncing_period)
                                btn_i=0;
                                #(bouncing_period*5)
                                btn_i=1;
                                #(bouncing_period*5)
                                btn_i=0;


                        end
always
                        #500 clk_i = !clk_i;

endmodule
```

Debouncer - symulacja



Przykład 6 - nadajnik RS323

- Układ ma nadawać znak ‘A’ w kodzie ASCII na linię wyjściową TXD_o.
- Wejścia: clk_i (zegar, 50MHz), en_i (wejście zezwalające) i rst_i (reset asynchroniczny).
- Układ należy podzielić na 3 moduły: główny (main_module) spinający dzielnik częstotliwości (divider) oraz nadajnik (tx_rs232).
- Należy wykonać opis Verilog, symulację i sprawdzenie działania na płytce Spartan 3.

Dzielnik częstotliwości

```
1  `timescale 1ns / 1ps
2  module divider(clk_i,rst_i,en_i,clk_o);
3
4  // deklaracja kierunku portów
5  input clk_i,rst_i,en_i;
6  output reg clk_o;
7
8  //deklaracja sygnałów wewnętrznych
9  parameter DivRatio=10;
10 reg [CLogB2(DivRatio-1)-1:0] counter;
11 //ceil of the log base 2
12 function integer CLogB2;
13    input [31:0] Depth;
14    integer i;
15    begin
16       i = Depth;
17       for(CLogB2 = 0; i > 0; CLogB2 = CLogB2 + 1)
18          i = i >> 1;
19    end
20 endfunction
21 always @(posedge clk_i or posedge rst_i)
22 if (rst_i)
23   // obsługa części asynchronicznej bloku always
24   begin
25       counter=0;
26       clk_o=1'b0;
27   end
28 else
29    // obsługa części synchronicznej bloku always
30    begin
31       if (en_i==1'b1)
32          begin
33             if (counter==DivRatio-1)
34                counter=0;
35             else
36                counter=counter+1;
37             // wystawienie sygnału zewnętrznego
38             if (counter==0)
39                clk_o=1'b0;
40             if (counter==DivRatio/2)
41                clk_o=1'b1;
42          end
43    end
44 endmodule
45
```

Nadajnik RS232

```
1  `timescale 1ns / 1ps
2  module tx_rs232(clk_i,rst_i,en_i,data_i,TXD_o);
3     input clk_i, rst_i, en_i;
4     input [7:0] data_i;
5     output reg TXD_o;
6
7     reg [3:0] state;
8     // 0 - waiting for en_i
9     // 1 - start bit generation
10    // 2..9 - data_i trannsmision
11    // 10 - stop bit generation
12    // other - unused
13
14 always @(posedge clk_i or posedge rst_i)
15 if (rst_i)
16   // obsługa części asynchronicznej bloku always
17   begin
18       TXD_o=1'b1;
19       state = 0;
20   end
21 else
```

```
22    // obsługa części synchronicznej bloku always
23    begin
24       case (state)
25          0         :  if (en_i)
26                          state = 1;
27
28          1         :  begin
29                          state = 2;
30                          TXD_o = 0;
31                       end
32          2,3,4,5,6,7,8,9
33                    :  begin
34                          TXD_o = data_i[state-2];
35                          state = state+1;
36                       end
37          10        :  begin
38                          TXD_o = 1;
39                          state = 0;
40                       end
41          default   :  begin
42                          state = 0;
43                          TXD_o =1;
44                       end
45       endcase
46    end
47 endmodule
```

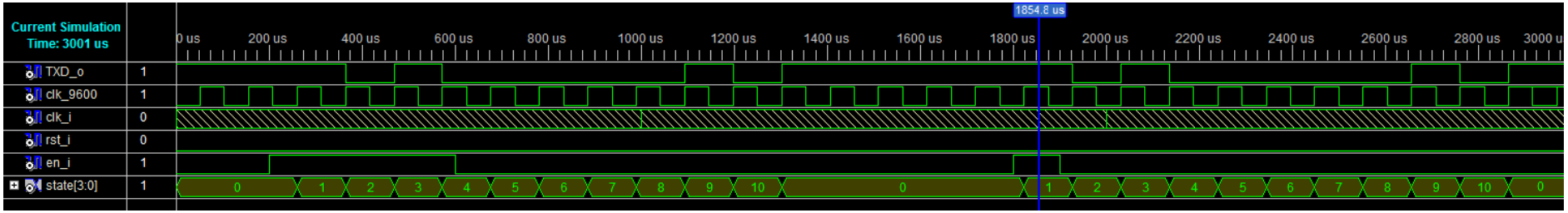
Moduł łączący bloki dzielnika i nadajnika oraz testbench

```
1  `timescale 1ns / 1ps
2  module main_module(clk_i,rst_i,en_i,TXD_o);
3     input clk_i, rst_i, en_i;
4     output TXD_o;
5
6  parameter bps=9600;
7  parameter freq=50_000_000;
8  parameter integer div_ratio = freq/bps;
9
10 wire clk_9600;
11
12 divider #(div_ratio) d_1(clk_i,rst_i,1,clk_9600);
13 tx_rs232 tx_1(clk_9600,rst_i,en_i,"A",TXD_o);
14
15 endmodule
```

```
1  `timescale 1ns / 1ps
2  module tb_main;
3     // Inputs
4     reg clk_i;
5     reg rst_i;
6     reg en_i;
7     // Outputs
8     wire TXD_o;
9
10    // Instantiate the Unit Under Test (UUT)
11    main_module uut (
12       .clk_i(clk_i),
13       .rst_i(rst_i),
14       .en_i(en_i),
15       .TXD_o(TXD_o)
16    );
17
18    initial begin
19       // Initialize Inputs
20       clk_i = 0;
21       rst_i = 1;
22       en_i = 0;
23       // Wait 100 ns for global reset to finish
24       #100;
25       rst_i=0;
26       #200e3
27       en_i=1;
28       #400e3
29       en_i=0;
30       #1200e3
31       en_i=1;
32       #100e3
33       en_i=0;
34    end
35
36    always
37       #10 clk_i = !clk_i;
38
39 endmodule
```

Plik UCF na płytkę Spartan 3 i symulacja funkcjonalna

```
1  # Clock:
2  NET "clk_i" LOC = "T9" ; # 50 MHz clock
3  # Push-buttons:
4  NET "rst_i" LOC = "L14" ; # pressed high BTN3
5  # RS232:
6  NET "TXD_o" LOC = "R13" ; # RS 232 TXD
7  #slider
8  NET "en_i" LOC = "K13" ; # SW7
```



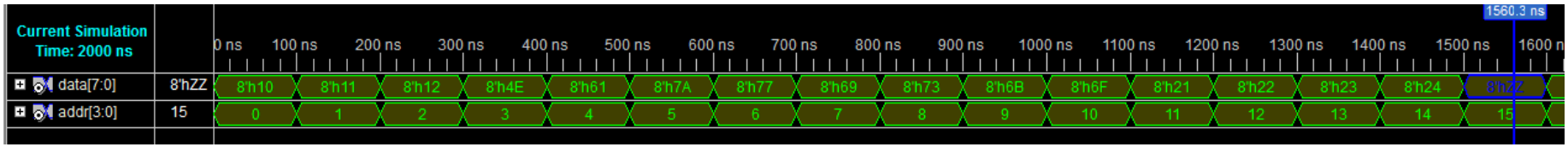
Przykład 7 – prosta pamięć asynchroniczna ROM

- Przykład pamięci o 16 słowach 8-io bitowych.
- Pamięć jest asynchroniczna – reaguje natychmiast po podaniu adresu na jej wejście.
- Należy zauważyć, że zawartość pod adresem 15 nie jest ustalona.

Kod Verilog pamięci ROM oraz jej testbench i symulacja

```verilog
`timescale 1ns / 1ps
module rom(addr,data);
   input [3:0] addr;
   output [7:0] data;

// array declaration
wire [7:0] memory [0:15];
   // assigning vaules to the memory slots
   assign memory[0]  = 8'h10;
   assign memory[1]  = 8'h11;
   assign memory[2]  = 8'h12;
   assign memory[3]  = "N";
   assign memory[4]  = "a";
   assign memory[5]  = "z";
   assign memory[6]  = "w";
   assign memory[7]  = "i";
   assign memory[8]  = "s";
   assign memory[9]  = "k";
   assign memory[10] = "o";
   assign memory[11] = 8'h21;
   assign memory[12] = 8'h22;
   assign memory[13] = 8'h23;
   assign memory[14] = 8'h24;

// output generation
assign data = memory[addr];

endmodule
```

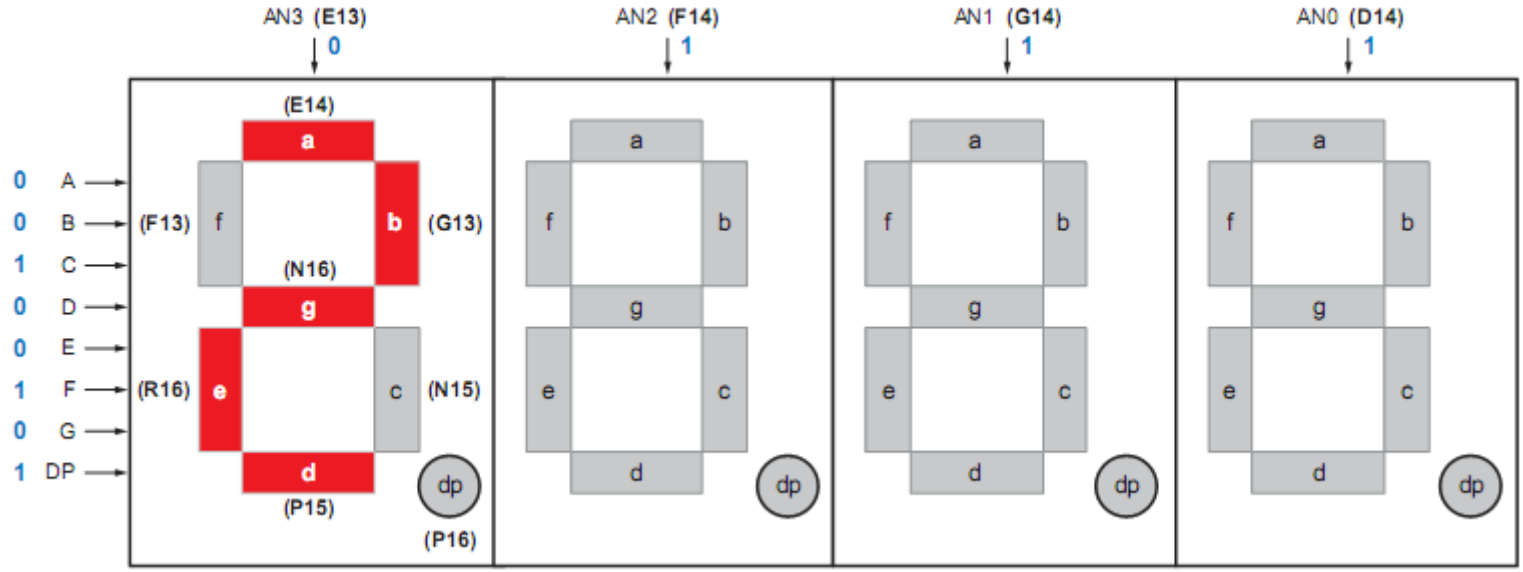
```verilog
`timescale 1ns / 1ps

module tb_rom;

   // Inputs
   reg [3:0] addr;

   // Outputs
   wire [7:0] data;

   // Instantiate the Unit Under Test (UUT)
   rom uut (
      .addr(addr),
      .data(data)
   );

   initial
      begin
         // Initialize Inputs
         addr = 0;
      end

   always
      begin
         #100;
         addr = addr+1;
      end

endmodule
```

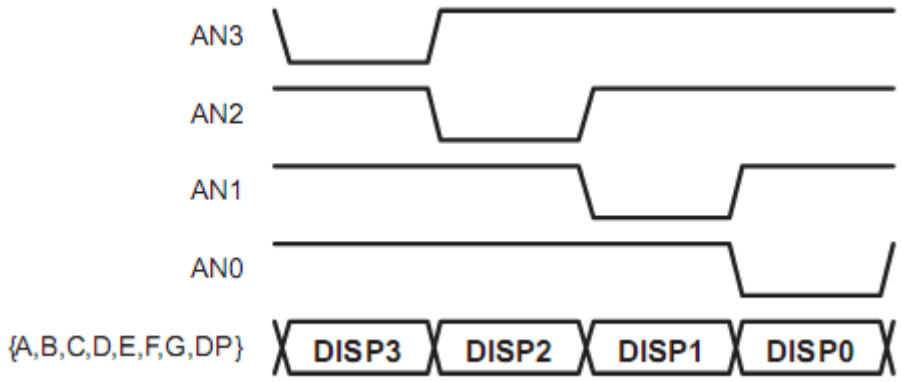


Przykład 8
Odbiornik RS232 i sterownik LED

Układ realizuje odbiór znaków w standardzie RS232 z prędkością transmisji 9600bps, 1 bitem stopu bez bitu parzystości a następnie wyświetla odebrany kod na 4 pozycyjnym wyświetlaczu 7 segmentowym z multipleksowaniem (dana pozycja aktywowana jest zerem, również segmenty aktywowane są zerem). Sygnały zewnętrzne: zegar wejściowy (clk_i) o częstotliwości 50MHz, asynchroniczny reset (rst_i), wejście danych (RXD_i), wyjście wspólnych anod aktywowanych zerem (4bity, led7_an_o) oraz wyjście segmentów wyświetlacza (8bitów, led7_seg_o).

Sterowanie wyświetlaczem:





Moduł główny

```
`timescale 1ns / 1ps
module rs_led_div (clk_i,rst_i,RXD_i,led7_an_o,led7_seg_o);

   parameter DivRatio = 50_000_000/9600/3;
   input clk_i, rst_i, RXD_i;
   output [3:0] led7_an_o;
   output [7:0] led7_seg_o;

   // internal signals
   wire clk_rs,clk_led;
   wire [7:0] data;

   // dividers
   counter #(DivRatio) div_rs232 (clk_i,rst_i,1'b1,1'b1,clk_rs);
   counter #(DivRatio*3*9600/1_000) div_leds  (clk_i,rst_i,1'b1,1'b1,clk_led);

   //RS 232 receiver
   rs_rcv receiver(clk_rs, rst_i, RXD_i, data);

   //LED driver
   led_disp led_driver (
            clk_led,    //clk_i
            rst_i,      //rst_i
            4'b0,       //digit0
            data[3:0],  //digit1
            data[7:4],  //digit2
            4'b0,       //digit3
            1'b0,       //dp0
            1'b0,       //dp1
            1'b0,       //dp2
            1'b0,       //dp3
            1'b0,       //on0
            1'b1,       //on1
            1'b1,       //on2
            1'b0,       //on3
            led7_seg_o, //led7_seg_o
            led7_an_o); //led7_an_o

endmodule
```

Dzielnik jest wzięty z przykładu nr 2

```
`timescale 1ns / 1ps
module counter(clk_i,rst_i,en_i,dir_i,out_o);

   // deklaracja kierunku portów
   input clk_i,rst_i,en_i,dir_i;
   output out_o;
   // deklaracja sygnałów rejestrowych
   reg out_o;

   //deklaracja sygnałów wewnętrznych
   parameter DivRatio=1000;
   reg [CLogB2(DivRatio-1)-1:0] counter;
   //ceil of the log base 2
   function integer CLogB2;
      input [31:0] Depth;
      integer i;
      begin
         i = Depth;
         for(CLogB2 = 0; i > 0; CLogB2 = CLogB2 + 1
            i = i >> 1;
      end
   endfunction
```

```
   always @(posedge clk_i or posedge rst_i)
      if (rst_i)  // obsługa części asynchronicznej bloku always
         begin
            counter=0;
            out_o=1'b0;
         end
      else
         // obsługa części synchronicznej bloku always
         if (en_i==1'b1)
            begin
               // obsługa licznika wewnętrznego
               if (dir_i==1'b1)
                  if (counter==DivRatio-1)
                     counter=0;
                  else
                     counter=counter+1;
               else
                  if (counter==0)
                     counter=DivRatio-1;
                  else
                     counter=counter-1;
               // ustawianie wyjścia
               if (counter==0)
                  out_o=1'b0;
               if (counter==DivRatio/2)
                  out_o=1'b1;
            end
endmodule
```

Odbiornik RS232

```
`timescale 1ns / 1ps
module rs_rcv (clk_i, rst_i, RXD_i, data_o);
   input clk_i, rst_i, RXD_i;
   output reg [7:0] data_o;

   reg [4:0] counter;
   reg [7:0] data_tmp;

   always @(posedge clk_i or posedge rst_i)
      if (rst_i)
         begin
            counter  = 5'b0;
            data_o   = 8'b0;
            data_tmp = 8'b0;
         end
      else
         begin
            case (counter)
               5'd0:  if (!RXD_i) counter = 5'd1;
               5'd1:  if (!RXD_i) counter = 5'd2;
               5'd28: counter=5'd0;
               default: counter = counter+1;
            endcase

            case (counter)
               5'd4: data_tmp[0] = RXD_i;
               5'd7: data_tmp[1] = RXD_i;
               5'd10: data_tmp[2] = RXD_i;
               5'd13: data_tmp[3] = RXD_i;
               5'd16: data_tmp[4] = RXD_i;
               5'd19: data_tmp[5] = RXD_i;
               5'd22: data_tmp[6] = RXD_i;
               5'd25: data_tmp[7] = RXD_i;
               5'd28: if (RXD_i) data_o = data_tmp;
               //default:
            endcase
         end
endmodule
```

Sterownik wyświetlacza

```
`timescale 1ns / 1ps
module led_disp(clk_i,rst_i,digit0,digit1,digit2,digit3,dp0,dp1,dp2,dp3,on0,on1,on2,on3,
                led7_seg_o,led7_an_o);
   input clk_i,rst_i,dp0,dp1,dp2,dp3,on0,on1,on2,on3;
   input [3:0] digit0,digit1,digit2,digit3;
   output reg [7:0] led7_seg_o;
   output reg [3:0] led7_an_o;

   // internal counter
   reg [1:0] counter;
   reg [3:0] digit;
   reg dp;

   always @(posedge clk_i or posedge rst_i)
      if (rst_i)
         begin
            counter = 2'd0;
            dp = 0;
            digit = 4'd0;
            led7_an_o = 4'd0;
         end
      else
         begin
            if (counter == 2'b11)
               counter = 2'b0;
            else
               counter = counter+1;
```

```
            case (counter)
               2'd0: begin
                        if (on0)
                           led7_an_o = 4'b1110;
                        else
                           led7_an_o = 4'b1111;
                           digit = digit0;
                           dp = dp0;
                     end
               2'd1: begin
                        if (on1)
                           led7_an_o = 4'b1101;
                        else
                           led7_an_o = 4'b1111;
                           digit = digit1;
                           dp = dp1;
                     end
               2'd2: begin
                        if (on2)
                           led7_an_o = 4'b1011;
                        else
                           led7_an_o = 4'b1111;
                           digit = digit2;
                           dp = dp2;
                     end
               2'd3: begin
                        if (on3)
                           led7_an_o = 4'b0111;
                        else
                           led7_an_o = 4'b1111;
                           digit = digit3;
                           dp = dp3;
                     end
            endcase
         end
```

Sterownik wyświetlacza c.d.

```
   // combinatorial decoding
   always @(digit or dp)
      begin
         led7_seg_o[0] = !dp;
         case (digit)
            4'd0: led7_seg_o[7:1] = 7'b0000001; // 0
            4'd1: led7_seg_o[7:1] = 7'b1001111; //1
            4'd2: led7_seg_o[7:1] = 7'b0010010; //2
            4'd3: led7_seg_o[7:1] = 7'b0000110; //3
            4'd4: led7_seg_o[7:1] = 7'b1001100; //4
            4'd5: led7_seg_o[7:1] = 7'b0100100; //5
            4'd6: led7_seg_o[7:1] = 7'b0100000; //6
            4'd7: led7_seg_o[7:1] = 7'b0001111; //7
            4'd8: led7_seg_o[7:1] = 7'b0000000; //8
            4'd9: led7_seg_o[7:1] = 7'b0000100; //9
            4'd10: led7_seg_o[7:1] = 7'b0001000; //10
            4'd11: led7_seg_o[7:1] = 7'b1100000; //11
            4'd12: led7_seg_o[7:1] = 7'b0110001; //12
            4'd13: led7_seg_o[7:1] = 7'b1000010; //13
            4'd14: led7_seg_o[7:1] = 7'b0110000; //14
            4'd15: led7_seg_o[7:1] = 7'b0111000; //15
            default: led7_seg_o[7:1] = 7'b1111110;
         endcase
      end
endmodule
```

Testbench

```
`timescale 1ns / 1ps
module tb;
        // Inputs
        reg clk_i;
        reg rst_i;
        reg RXD_i;
        // Outputs
        wire [3:0] led7_an_o;
        wire [7:0] led7_seg_o;
        // Instantiate the Unit Under Test (UUT)
        rs_led_div uut (
                .clk_i(clk_i),
                .rst_i(rst_i),
                .RXD_i(RXD_i),
                .led7_an_o(led7_an_o),
                .led7_seg_o(led7_seg_o));

initial begin
        // Initialize Inputs
        clk_i = 0;
        rst_i = 1;
        RXD_i = 1;
        #100 rst_i = 0;
        #1000000;
        RXD_i = 0;      //START
        #104000;
        RXD_i = 1;      //D0
        #104000
        RXD_i = 0;      //D1
        #104000
        RXD_i = 1;      //D2
        #104000
        RXD_i = 0;      //D3
        #104000
        RXD_i = 1;      //D4
        #104000
        RXD_i = 0;      //D5
        #104000
        RXD_i = 1;      //D6
        #104000
        RXD_i = 0;      //D7
        #104000
        RXD_i = 1;      //STOP
        #104000
        RXD_i = 1;      //STOP2
        #104000;

        // Add stimulus here

end
always
        #10 clk_i = ! clk_i;

endmodule
```

Część III
VHDL

Przykład 1
Prosty dzielnik częstotliwości

- układ synchroniczny z asynchronicznym resetem,
- wejścia: **`clk_i`** (zegar 50MHz), **`rst_i`** (reset asynchroniczny),
- wyjście: **`clk_o`**.
- układ dzieli częstotliwość sygnału **`clk_i`** przez 50_000_000 a wynik podziału podaje na wyjście **`clk_o`**,
- współczynnik podziału zdefiniowano za pomocą **`generic`**.

Ogólna składnia procesu synchronicznego z asynchronicznym resetem

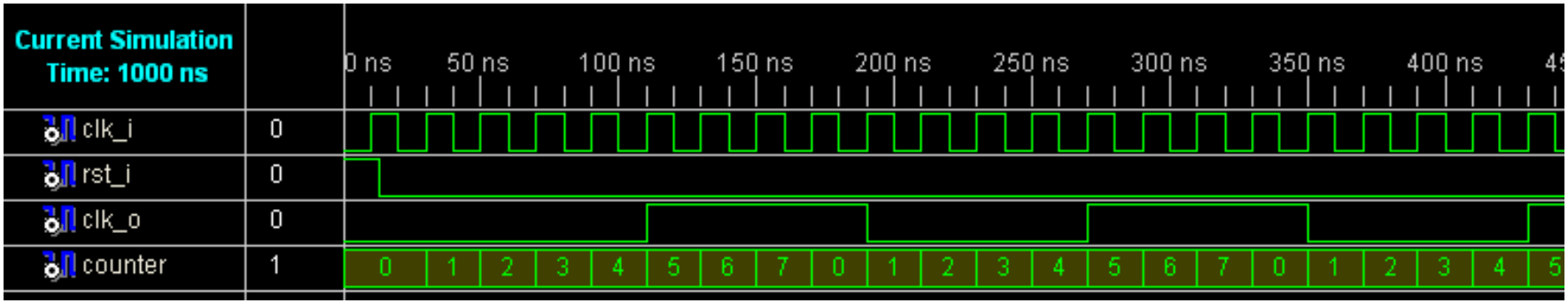
```
process (clk,rst)
begin
  if rst='1' then
        ... tutaj następuje obsługa resetu...
  elsif clk'event and clk='1' then
        ... tutaj następuje obsługa zbocza zegara...
  end if;
end process;
```

Dzielnik i testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity div is
    generic (div_ratio: integer:=50_000_000);
    port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           clk_o : out  STD_LOGIC);
end div;

architecture Behavioral of div is

signal counter : integer range 0 to div_ratio-1;

begin

process(clk_i,rst_i)
begin
   if rst_i='1' then
      clk_o <= '0';
      counter <= 0;
   elsif Rising_Edge(clk_i) then
      if counter = div_ratio-1 then
         counter <= 0;
      else
         counter <= counter + 1;
      end if;
      if counter = div_ratio/2 then
         clk_o <='1';
      end if;
      if counter = 0 then
         clk_o <= '0';
      end if;
   end if;
end process;

end Behavioral;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_div IS
END tb_div;
 ARCHITECTURE behavior OF tb_div IS

   COMPONENT div
   generic (div_ratio: integer:=50_000_000);
   port ( clk_i : in  STD_LOGIC;
          rst_i : in  STD_LOGIC;
          clk_o : out  STD_LOGIC);
   END COMPONENT;

   signal clk_i : std_logic := '0';
   signal rst_i : std_logic := '0';
   signal clk_o : std_logic := '0';

   constant clk_i_period : time := 20 ns;

BEGIN
   -- wstawienie komponentu
   uut: div
      generic map (div_ratio => 8)
      PORT MAP (
          clk_i => clk_i,
          rst_i => rst_i,
          clk_o => clk_o
        );
   -- generacja zegara
   clk_i_process :process
   begin
      clk_i <= '0';
      wait for clk_i_period/2;
      clk_i <= '1';
      wait for clk_i_period/2;
   end process;
   -- generacja resetu
   rst_i <= '1', '0' after 13 ns;
END;
```

Symulacja



Plik UCF dla płytki Spartan

```
# Push-button:
NET "rst_i" LOC = "L14" ; # pressed high BTN3
#50MHz clock:
NET "clk_i" LOC = "T9" ; # 50 MHz clock
# LED:
NET "clk_o" LOC = "K12" ; # high on
```

Przykład 2 – licznik w kodzie Graya

- Należy zrealizować opis 4-ro bitowego licznika w kodzie Graya,
- Licznik ma być synchroniczny z asynchronicznym resetem
- Wejścia clk_i oraz rst_i, wyjście gray_o[3:0]

4-ro bitowy kod Gray-a

0000	**1100**
0001	**1101**
0011	**1111**
0010	**1110**
0110	**1010**
0111	**1011**
0101	**1001**
0100	**1000**

Zamiana z kodu dwójkowego:

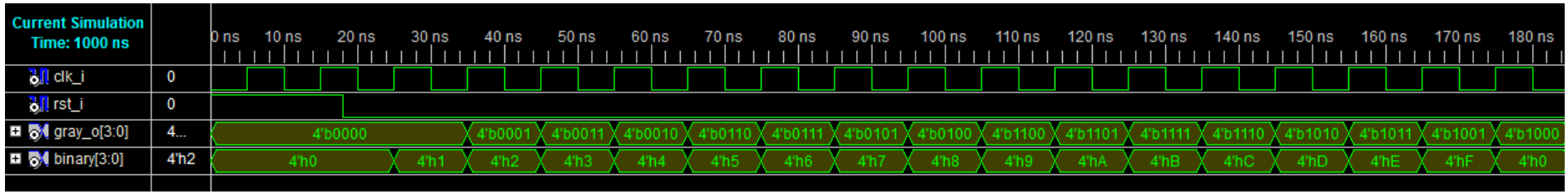
Gray <= binary xor binary/2

Kod licznika Graya

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity gray is
    Port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           gray_o : out  STD_LOGIC_VECTOR (3 downto 0));
end gray;

architecture Behavioral of gray is
   signal binary : std_logic_vector(3 downto 0);
begin

process(clk_i,rst_i)
   --variable binary : std_logic_vector(3 downto 0);
begin
   if rst_i = '1' then
      gray_o <= "0000";
      binary <= (others => '0');
   elsif Rising_Edge(clk_i) then
      binary <= binary + 1;
      gray_o <= binary xor ('0' & binary(3 downto 1));
   end if;
end process;

end Behavioral;
```

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY tb IS
END tb;

ARCHITECTURE behavior OF tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT gray
    PORT(
         clk_i : IN  std_logic;
         rst_i : IN  std_logic;
         gray_o : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;

   --Inputs
   signal clk_i : std_logic := '0';
   signal rst_i : std_logic := '0';

   --Outputs
   signal gray_o : std_logic_vector(3 downto 0);

   -- Clock period definitions
   constant clk_i_period : time := 10 ns;

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: gray PORT MAP (
          clk_i => clk_i,
          rst_i => rst_i,
          gray_o => gray_o
        );

   -- Clock process definitions
   clk_i_process :process
   begin
      clk_i <= '0';
      wait for clk_i_period/2;
      clk_i <= '1';
      wait for clk_i_period/2;
   end process;

   rst_i <= '1', '0' after 18 ns;

END;
```

Symulacja


Current Simulation Time: 1000 ns
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns
clk_i
0
rst_i
0
gray_o[3:0]
4...
4'b0000
4'b0001
4'b0011
4'b0010
4'b0110
4'b0111
4'b0101
4'b0100
4'b1100
4'b1101
4'b1111
4'b1110
4'b1010
4'b1011
4'b1001
4'b1000
binary[3:0]
4'h2
4'h0
4'h1
4'h2
4'h3
4'h4
4'h5
4'h6
4'h7
4'h8
4'h9
4'hA
4'hB
4'hC
4'hD
4'hE
4'hF
4'h0

Przykład 3
Prosty licznik binarny 8 bitowy

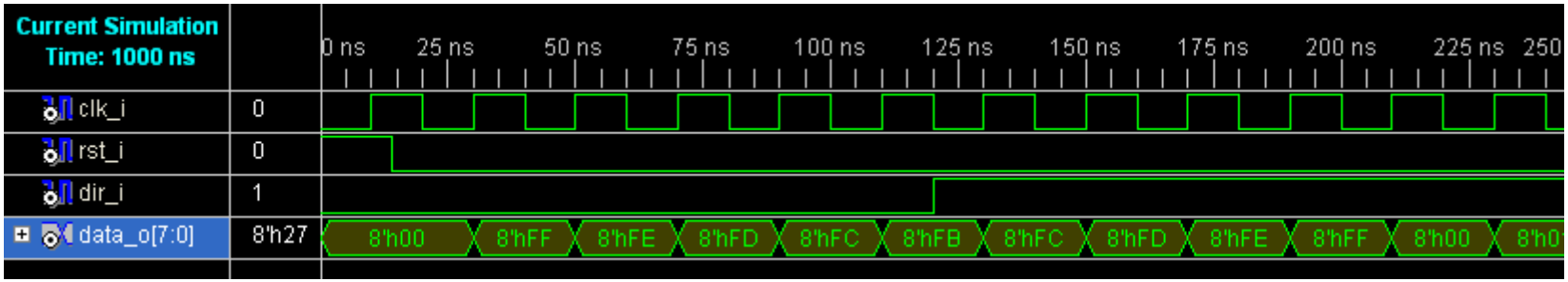
- układ synchroniczny z asynchronicznym resetem,
- wejścia: **`clk_i`** (zegar 50MHz), **`rst_i`** (reset asynchroniczny) oraz **`dir_i`** (kierunek zliczania),
- wyjście: **`data_o`** (ośmiobitowe).

Licznik i testbench

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity count is
    port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           dir_i : in  STD_LOGIC;
           data_o : out  STD_LOGIC_VECTOR(7 downto 0));
end count;

architecture Behavioral of count is

signal counter : STD_LOGIC_VECTOR(7 downto 0);

begin

data_o <= counter;
process(clk_i,rst_i)
begin
   if rst_i='1' then
      counter <= (others =>'0');
   elsif Rising_Edge(clk_i) then
      if dir_i='1' then
         counter <= counter + 1;
      else
         counter <= counter - 1;
      end if;
   end if;
end process;

end Behavioral;
```

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;

ENTITY tb_count IS
END tb_count;

ARCHITECTURE behavior OF tb_count IS
    COMPONENT count
    PORT(
         clk_i : IN  std_logic;
         rst_i : IN  std_logic;
         dir_i : IN  std_logic;
         data_o : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;

   signal clk_i : std_logic := '0';
   signal rst_i : std_logic := '0';
   signal dir_i : std_logic := '0';
   signal data_o : std_logic_vector(7 downto 0);
   constant clk_i_period : time := 20 ns;

BEGIN
   uut: count PORT MAP (
          clk_i => clk_i,
          rst_i => rst_i,
          dir_i => dir_i,
          data_o => data_o
        );
   clk_i_process :process
   begin
      clk_i <= '0';
      wait for clk_i_period/2;
      clk_i <= '1';
      wait for clk_i_period/2;
   end process;

rst_i <= '1', '0' after 14 ns;
dir_i <= '0', '1' after 120 ns;
END;
```

Symulacja


Current Simulation Time: 1000 ns
0 ns
25 ns
50 ns
75 ns
100 ns
125 ns
150 ns
175 ns
200 ns
225 ns
250
clk_i
0
rst_i
0
dir_i
1
data_o[7:0]
8'h27
8'h00
8'hFF
8'hFE
8'hFD
8'hFC
8'hFB
8'hFC
8'hFD
8'hFE
8'hFF
8'h00

Przykład 4 – nadajnik RS232

- Układ ma nadawać znak ‘B’ w kodzie ASCII na linię wyjściową TXD_o.
- Wejścia: clk_i (zegar, 50MHz), en_i (wejście zezwalające) i rst_i (reset asynchroniczny).
- Układ należy podzielić na 3 moduły: główny (main_module) spinający dzielnik częstotliwości (divider) oraz nadajnik (tx_rs232).
- Należy wykonać opis VHDL, symulację i sprawdzenie działania na płytce Spartan 3.

Dzielnik częstotliwości

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity divider is
7      generic(div_factor:integer:=10);
8      Port ( clk_i : in  STD_LOGIC;
9             rst_i : in  STD_LOGIC;
10            clk_o : out  STD_LOGIC);
11 end divider;
12
13 architecture Behavioral of divider is
14    signal counter: integer range 0 to div_factor-1;
15 begin
16    process(clk_i,rst_i)
17    begin
18       if rst_i='1' then
19          clk_o <= '0';
20          counter <= 0;
21       elsif Rising_Edge(clk_i) then
22          if counter = div_factor-1 then
23             counter <= 0;
24          else
25             counter <= counter +1;
26          end if;
27          if counter = 0 then
28             clk_o <= '0';
29          end if;
30          if counter = div_factor/2 then
31             clk_o <= '1';
32          end if;
33       end if;
34    end process;
35
36
37 end Behavioral;
```

Nadajnik RS232

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tx_rs232 is
    Port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           en_i : in  STD_LOGIC;
           data_i : in  STD_LOGIC_VECTOR (7 downto 0);
           TXD_o : out  STD_LOGIC);
end tx_rs232;

architecture Behavioral of tx_rs232 is

   type state_t is (waiting_for_en_i, start_bit,
               b0,b1,b2,b3,b4,b5,b6,b7, stop_bit);
   signal state:state_t;

begin
   process(clk_i,rst_i)
   begin
      if rst_i = '1' then
         state <= waiting_for_en_i;
         TXD_o <= '1';
      elsif Rising_Edge(clk_i) then
         case state is
            when waiting_for_en_i =>
                     if en_i = '1' then
                        state <= start_bit;
                     end if;
            when start_bit =>
                     state <= b0;
                     TXD_o <= '0';
            when b0 =>
                     state <= b1;
                     TXD_o <= data_i(0);
            when b1 =>
                     state <= b2;
                     TXD_o <= data_i(1);
            when b2 =>
                     state <= b3;
                     TXD_o <= data_i(2);
            when b3 =>
                     state <= b4;
                     TXD_o <= data_i(3);
            when b4 =>
                     state <= b5;
                     TXD_o <= data_i(4);
            when b5 =>
                     state <= b6;
                     TXD_o <= data_i(5);
            when b6 =>
                     state <= b7;
                     TXD_o <= data_i(6);
            when b7 =>
                     state <= stop_bit;
                     TXD_o <= data_i(7);
            when stop_bit =>
                     state <= waiting_for_en_i;
                     TXD_o <= '1';
         end case;
      end if;
   end process;
end Behavioral;
```

Moduł łączący bloki dzielnika i nadajnika

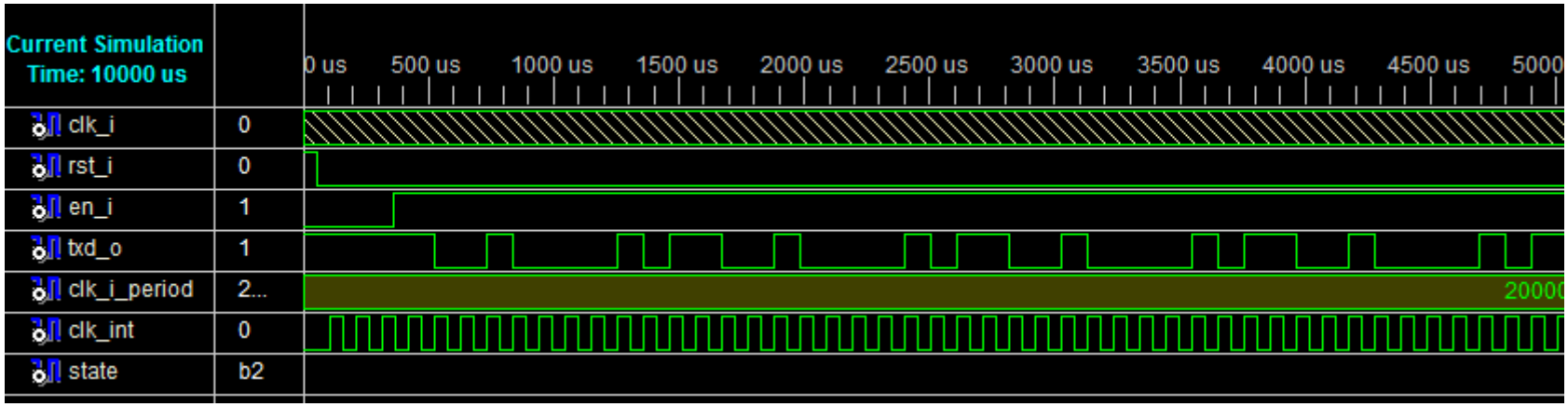
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity main_module is
7      Port ( clk_i : in  STD_LOGIC;
8             rst_i : in  STD_LOGIC;
9             en_i : in  STD_LOGIC;
10            TXD_o : out  STD_LOGIC);
11 end main_module;
12
13 architecture Behavioral of main_module is
14
15 component divider is
16     generic(div_factor:integer:=10);
17     Port ( clk_i : in  STD_LOGIC;
18            rst_i : in  STD_LOGIC;
19            clk_o : out  STD_LOGIC);
20 end component;
21
22 component tx_rs232 is
23     Port ( clk_i : in  STD_LOGIC;
24            rst_i : in  STD_LOGIC;
25            en_i : in  STD_LOGIC;
26            data_i : in  STD_LOGIC_VECTOR (7 downto 0);
27            TXD_o : out  STD_LOGIC);
28 end component;
29
30 signal clk_int:std_logic;
31 constant data:std_logic_vector(7 downto 0):=x"42";
32
33
34 begin
35
36 u1: divider
37    generic map(div_factor => 50_000_000/9600)
38    port map(clk_i,rst_i,clk_int);
39
40 u2: tx_rs232
41    port map(clk_int,rst_i,en_i,data,TXD_o);
42
43 end Behavioral;
```

Testbench
i
UCF

```
# Clock:
NET "clk_i" LOC = "T9" ; # 50 MHz clock
# Push-buttons:
NET "rst_i" LOC = "L14" ; # pressed high BTN3
# RS232:
NET "TXD_o" LOC = "R13" ; # RS 232 TXD
#slider
NET "en_i" LOC = "K13" ; # SW7
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
ENTITY tb_tx IS
END tb_tx;
ARCHITECTURE behavior OF tb_tx IS
    COMPONENT main_module
    PORT(
         clk_i : IN  std_logic;
         rst_i : IN  std_logic;
         en_i : IN  std_logic;
         TXD_o : OUT  std_logic
        );
    END COMPONENT;

   --Inputs
   signal clk_i : std_logic := '0';
   signal rst_i : std_logic := '0';
   signal en_i : std_logic := '0';

   --Outputs
   signal TXD_o : std_logic;
   constant clk_i_period : time := 20 ns;

BEGIN
   uut: main_module PORT MAP (
          clk_i => clk_i,
          rst_i => rst_i,
          en_i => en_i,
          TXD_o => TXD_o
        );

   clk_i_process :process
   begin
      clk_i <= '0';
      wait for clk_i_period/2;
      clk_i <= '1';
      wait for clk_i_period/2;
   end process;

   rst_i <= '1', '0' after 58 us;
   en_i <= '0', '1' after 360 us;

END;
```

Symulacja


Current Simulation Time: 10000 us
0 us
500 us
1000 us
1500 us
2000 us
2500 us
3000 us
3500 us
4000 us
4500 us
5000
clk_i
0
rst_i
0
en_i
1
txd_o
1
clk_i_period
2...
20000
clk_int
0
state
b2

Przykłady – 5, modele pamięci

- Przykłady są spakowane i dostępne na stronie przedmiotu: http://www.ue.eti.pg.gda.pl/~bpa/jmis/mems.zip
- Zawarte są 4 typy pamięci: asynchroniczna ROM, synchroniczna ROM, asynchroniczna RAM, synchroniczna RAM
- Wraz z modelami pamięci dołączone są pliki testbenchy

Przykład 6
Dzielnik + licznik + dekoder 7seg

- układ synchroniczny z asynchronicznym resetem,
- wejścia: **`clk_i`** (zegar 50MHz), **`rst_i`** (reset asynchroniczny) oraz **`dir_i`** (kierunek zliczania),
- wyjście: **`leds_o`** (ośmiobitowe, diody LED), **`led7_seg_o`** (8 bitów segmenty multipleksowanego LED aktywne zerem) oraz **`led7_an_o`** (4 bitowe anody multipleksowanego LED aktywne zerem),
- układ składa się z dzielnika i licznika z poprzednich przykładów przy czym wyjście dzielnika stanowi zegar licznika,
- wyjście licznika dołączone jest do diod LED (wyjście **`leds_o`**),
- dodatkowo 4 najmniej znaczące bity licznika są dekodowane i sterują 1 segmentem wyświetlacza LED.

Kod VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ex_3 is
    generic (div_ratio: integer:=50_000_000);
    port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           dir_i : in  STD_LOGIC;
           leds_o : out  STD_LOGIC_VECTOR (7 downto 0);
           led7_seg_o : out  STD_LOGIC_VECTOR (7 downto 0);
           led7_an_o : out  STD_LOGIC_VECTOR (3 downto 0)
           );
end ex_3;

architecture Behavioral of ex_3 is
-- deklaracje komponentów
component div
    generic (div_ratio: integer:=50_000_000);
    port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           clk_o : out  STD_LOGIC);
end component;
component count
    port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           dir_i : in  STD_LOGIC;
           data_o : out  STD_LOGIC_VECTOR(7 downto 0));
end component;
```

Kod VHDL c.d.

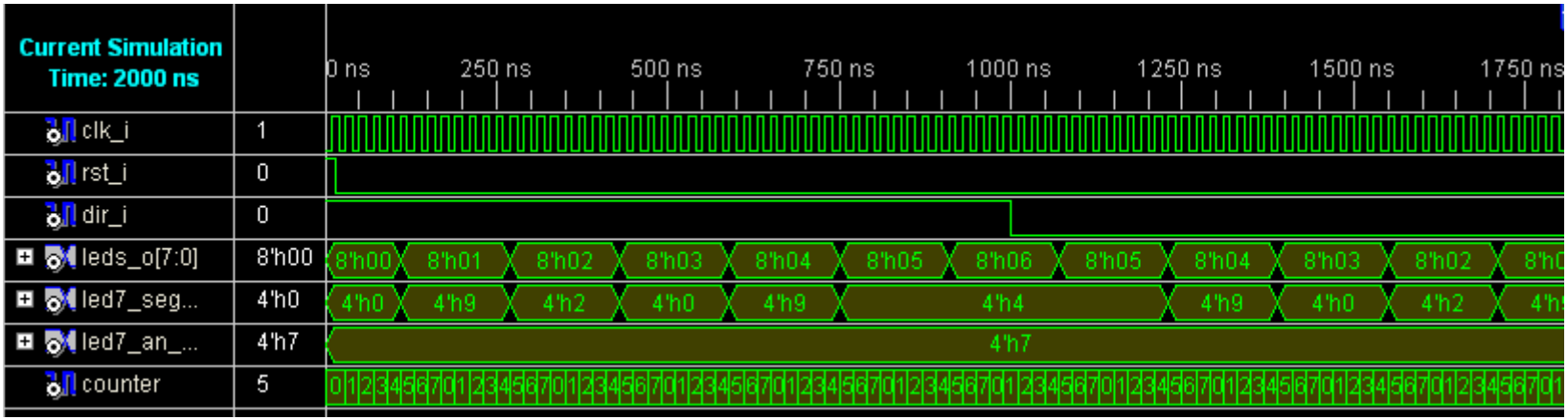
```
-- deklaracja funkcji dekodującej 7seg
function dekoder7seg(signal in_data: std_logic_vector(3 downto 0))
   return std_logic_vector is
   variable temp: std_logic_vector(7 downto 0);
begin
   -- temp:= (others =>'1');
   case in_data is
      when X"0"   => temp:="00000011";
      when X"1"   => temp:="10011111";
      when X"2"   => temp:="00100101";
      when X"3"   => temp:="00001101";
      when X"4"   => temp:="10011001";
      when X"5"   => temp:="01001001";
      when X"6"   => temp:="01000001";
      when X"7"   => temp:="00011111";
      when X"8"   => temp:="00000001";
      when X"9"   => temp:="00001001";
      when X"a"   => temp:="00010001";
      when X"b"   => temp:="11000001";
      when X"c"   => temp:="01100011";
      when X"d"   => temp:="10000101";
      when X"e"   => temp:="01100001";
      when X"f"   => temp:="01110001";
      when others => temp:="11111101";
   end case;
return (temp);
end function dekoder7seg;
-- deklaracje sygnałów wewnętrznych
signal clk_internal : STD_LOGIC;
signal data_internal : STD_LOGIC_VECTOR(7 downto 0);

begin
-- wstawienie komponentów
div_1: div
   generic map (div_ratio => div_ratio)
   port map(clk_o => clk_internal, rst_i => rst_i, clk_i => clk_i);
counter_1: count
   port map(clk_internal,rst_i,dir_i,data_internal);
-- dołączenie sygnału do wyjścia
leds_o<=data_internal;
-- ustawienie 3 segmentu wyświetlacza jako aktywnego a pozostałę wygaszone, aktywne zerem
led7_an_o <= (3=> '0', others => '1');
-- dekodowanie kombinacyjne (użycie funkcji) wyświetlacza 7 segmentowego, wyjścia aktywne zerem
led7_seg_o<=dekoder7seg(data_internal(3 downto 0));
end Behavioral;
```

Testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;

ENTITY tb_ex3 IS
END tb_ex3;

ARCHITECTURE behavior OF tb_ex3 IS

    COMPONENT ex_3
    generic (div_ratio: integer:=5_000_000);
    PORT(
         clk_i : IN  std_logic;
         rst_i : IN  std_logic;
         dir_i : IN  std_logic;
         leds_o : OUT  std_logic_vector(7 downto 0);
         led7_seg_o : OUT  std_logic_vector(7 downto 0);
         led7_an_o : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk_i : std_logic := '0';
   signal rst_i : std_logic := '0';
   signal dir_i : std_logic := '0';

   --Outputs
   signal leds_o : std_logic_vector(7 downto 0);
   signal led7_seg_o : std_logic_vector(7 downto 0);
   signal led7_an_o : std_logic_vector(3 downto 0);

   -- Clock period definitions
   constant clk_i_period : time := 20 ns;
```

```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: ex_3
   generic map (div_ratio => 8)
   PORT MAP (
          clk_i => clk_i,
          rst_i => rst_i,
          dir_i => dir_i,
          leds_o => leds_o,
          led7_seg_o => led7_seg_o,
          led7_an_o => led7_an_o
        );

   -- Clock process definitions
   clk_i_process :process
   begin
      clk_i <= '0';
      wait for clk_i_period/2;
      clk_i <= '1';
      wait for clk_i_period/2;
   end process;

rst_i<='1', '0' after 14 ns;
dir_i<= '1' , '0' after 1 us;

END;
```

Symulacja



Plik ucf na płytkę Spartan3

```
# Clock:
NET "clk_i" LOC = "T9" ; # 50 MHz clock
# Push-buttons:
NET "rst_i" LOC = "L14" ; # pressed high BTN3
# Direction:
NET "dir_i" LOC = "J13" ; # slider SW5
# Seven-segment LED display:
NET "led7_an_o<3>" LOC = "E13" ; # leftmost digit, active low
NET "led7_an_o<2>" LOC = "F14" ; # active low
NET "led7_an_o<1>" LOC = "G14" ; # active low
NET "led7_an_o<0>" LOC = "D14" ; # rightmost digit, active low
```

```
NET "led7_seg_o<7>" LOC = "E14" ; # segment 'a', active low
NET "led7_seg_o<6>" LOC = "G13" ; # segment 'b', active low
NET "led7_seg_o<5>" LOC = "N15" ; # segment 'c', active low
NET "led7_seg_o<4>" LOC = "P15" ; # segment 'd', active low
NET "led7_seg_o<3>" LOC = "R16" ; # segment 'e', active low
NET "led7_seg_o<2>" LOC = "F13" ; # segment 'f', active low
NET "led7_seg_o<1>" LOC = "N16" ; # segment 'g', active low
NET "led7_seg_o<0>" LOC = "P16" ; # segment 'dp', active low
# leds
NET "leds_o<7>" LOC = "P11" ;
NET "leds_o<6>" LOC = "P12" ;
NET "leds_o<5>" LOC = "N12" ;
NET "leds_o<4>" LOC = "P13" ;
NET "leds_o<3>" LOC = "N14" ;
NET "leds_o<2>" LOC = "L12" ;
NET "leds_o<1>" LOC = "P14" ;
NET "leds_o<0>" LOC = "K12" ;
```