Języki Opisu Sprzętu

Prowadzący: dr inż. Andrzej Skoczeń
Współrzędne: D-10 222, tel. w. 28-72,
e-mail: skoczen@fis.agh.edu.pl

Wykład 6

2018

22 październik 2018

- **Zadania systemowe**
- **Konstrukcja symulatora**
- **Układ synchroniczny**

http://www.fis.agh.edu.pl/~skoczen/hdl

Zadania systemowe

Wywołuje się je w postaci: `$<słowo_kluczowe>`

Zatrzymywanie symulacji

```
$stop;
```

Zawiesza symulację i wprowadza tryb interaktywny np. w celu analizy sygnałów w projekcie.

Zakańczanie symulacji

```
$finish;
```

Kończy symulację.

Zadania systemowe

Wyświetlanie informacji

```
$display(p1, p2, ..., pn);     $write(p1, p2, ..., pn);
```

`p1, p2, ..., pn` **– zmienne, wyrażenia, ciągi znaków ujęte w cudzysłów.**

Różnica między `$display` **a** `$write` **polega na tym, że pierwszy na końcu ciągu znaków domyślnie wstawia koniec linii a drugi nie. Dla wyświetlenia kilku ciągów znaków w jednej linii używamy** `$write`**.**

Monitorowanie informacji

```
$monitor(p1, p2, ..., pn);
```

`p1, p2, ..., pn` **– zmienne, nazwy sygnałów, ciągi znaków ujęte w cudzysłów.**

W sposób ciągły monitoruje wartości zmiennych i sygnałów i wyświetla ich wartości w momentach zmiany choć jednego z parametrów. Zadanie to wystarczy wywołać tylko raz. Drugie wystąpienie zadania `$monitor` **powoduje deaktywację poprzedniego.**

Dwa pomocnicze zadania systemowe:

`$monitoron;` **aktywacja zadania** `$monitor` **(domyślne)**

`$monitoroff;` **deaktywacja zadania** `$monitor`

Zadania systemowe

Formatowanie ciągów znakowych:

`%h` lub `%H`	heksadecymalnie
`%d` lub `%D`	dziesiątkowo
`%o` lub `%O`	oktalnie
`%b` lub `%B`	binarnie
`%c` lub `%C`	znak ASCII
`%v` lub `%V`	moc sygnału w węźle
`%m` lub `%M`	nazwy hierarchiczne
`%s` lub `%S`	stringi
`%t` lub `%T`	bieżący czas
`%f` lub `%F`	liczby rzeczywiste w sposób dziesiętny
`%e` lub `%E`	liczby rzeczywiste w sposób wykładniczy
`%g` lub `%G`	liczby rzeczywiste w sposób dziesiętny lub wykładniczy, tak aby było krócej

Zadania systemowe

$display

```
module disp;
  initial begin
     $display("\\\t%%\n\"\123");
  end
endmodule
```

```
ncsim> run
\        %
"S
```

```
module printval;
  reg [11:0] r1;
  initial begin
    r1 = 12'd10;
    $display("Printing with maximum size - :%d: :%h:",r1,r1 );
    $display("Printing with minimum size - :%0d: :%0h:",r1,r1 );
  end
endmodule
```

```
ncsim> run
Printing with maximum size - :  10: :00a:
Printing with minimum size - :10: :a:
```

Zadania systemowe

$display

```
module disp;
  reg [31:0] rval;
  pulldown (pd);
  initial begin
      rval = 32'd101;
      $display("rval = %h hex %d decimal",rval,rval);
      $display("rval = %o octal %b binary",rval,rval);
      $display("rval has %c ascii character value",rval);
      $display("pd strength value is %v",pd);
      $display("current scope is %m");
      $display("%s is ascii value for 101",101);
      $display("simulation time is %t", $time);
    end
endmodule
```

```
ncsim> run
rval = 00000065 hex 101 decimal
rval = 00000000145 octal 00000000000000000000000001100101 binary
rval has e ascii character value
pd strength value is StX
current scope is disp
   e is ascii value for 101
simulation time is 0
```

Zadania systemowe

$monitor

```
`timescale 1ns / 1ns
module top;
reg clock, reset;
initial begin
  clock = 1'b0;
  forever
    #5 clock = ~clock;
end
```

```
initial begin
  reset = 1'b0;
  #1 reset = 1'b1;
  #3 reset = 1'b0;
end
```

```
//monitorowanie wartości czasu  i sygnałów zegara i resetu
initial
  $monitor($time, "ns  Wartosci sygnałow clock = %b reset = %b",
                                                clock, reset);
initial #20 $finish;
endmodule
```

```
0ns  Wartosci sygnałow clock = 0 reset = 0
1ns  Wartosci sygnałow clock = 0 reset = 1
4ns  Wartosci sygnałow clock = 0 reset = 0
5ns  Wartosci sygnałow clock = 1 reset = 0
10ns  Wartosci sygnałow clock = 0 reset = 0
15ns  Wartosci sygnałow clock = 1 reset = 0
```

Funkcja systemowa `$random`

Generacja liczb losowych jest konieczna dla tworzenia losowych wymuszeń w modułach testujących. Służy do tego funkcja systemowa:

```
$random(<seed>)
```

Zwraca ono 32-bitową losową liczbę całkowitą ze znakiem. Standard nie określa algorytmu.

Parametr `seed` (posiew) sprawia, że różne strumienie losowe są generowane dla różnych jego wartości. Może być zadeklarowany jako: `integer`, `reg`, lub `time`.
Przypisanie wartości do zmiennej `seed` musi być wykonane przed wywołaniem.

```
$random % b
```

Daje liczby losowe z zakresu [(-b+1): (b-1)].

```
reg [23:0] rand;
rand = $random % 60;
```

Generacja liczb losowych z zakresu od -59 do 59.

```
reg [23:0] rand;
rand = {$random} % 60;
```

Nawiasy klamrowe oznaczają utworzenie modułu wylosowanej wartości co ogranicza zakres do wartości dodatnich tzn. od 0 do 59.

```
reg serData;
serData = {$random} % 2;
```

Generacja losowego strumienia zer i jedynek.

Funkcja systemowa `$bitstoreal`

Generacja rzeczywistych liczb losowych w terminach znaku `sgn`, wykładnika `exp` i mantysy `man`. Potrzebne jest użycie funkcji systemowej `$bitstoreal`.

```
module Tb();
reg  sgn;
reg [10:0] exp;
reg [51:0] man;
real r_num;
    initial begin
        repeat(5) begin
            sgn = $random;
            exp = $random;
            man = $random;
            r_num = $bitstoreal({sgn,exp,man});
            $display("r_num = %e",r_num);
        end
    end
endmodule
```

```
r_num = 3.649952e+193
r_num = -1.414950e-73
r_num = -3.910319e-149
r_num = -4.280878e-196
r_num = -4.327791e+273
```

Zadanie systemowe `$readmemb, $readmemh`

Zadania do inicjowania pamięci zawartością pliku.

`$readmemb` – wczytuje zmienne binarne

`$readmemh` – wczytuje zmienne heksadecymalne

```
module test;
reg [7:0] memory [0:7];
integer i;
initial begin
  $readmemb("init.dat", memory);
  for(i=0; i<8; i=i+1)
    $display("memory[%0d]=%b",i, memory[i]);
end
endmodule
```

Plik `init.dat`:

```
@002
11111111 01010101
00000000 10101010
@006
1111zzzz 00001111
```

```
memory[0]= xxxxxxxx
memory[1]= xxxxxxxx
memory[2]= 11111111
memory[3]= 01010101
memory[4]=00000000
memory[5]= 10101010
memory[6]= 1111zzzz
memory[7]= 00001111
```

```
$readmemb("<nazwa_pliku>",
      <zmienna_pam>,
      <start_adres>,
      <finish_adres>);
```

Zadanie systemowe

```
initial
  if ($test$plusargs("test01"))
    $readmemh("test01.dat",stim_mem);
  if ($test$plusargs("test02"))
    $readmemh("test02.dat",stim_mem);
  if ($test$plusargs("test03"))
    $readmemh("test03.dat", stim_mem);
  if ($test$plusargs("test04"))
    $readmemh("test04.dat", stim_mem);
```

Incisive®, Cadence®

```
% ncsim snapshot_name +test01 &
% ncsim snapshot_name +test02 &
% ncsim snapshot_name +test03 &
% ncsim snapshot_name +test04 &
```

ISim, Xilinx®

```
% isim snapshot_name -plusargs test01 &
% isim snapshot_name -plusargs test02 &
% isim snapshot_name -plusargs test03 &
% isim snapshot_name -plusargs test04 &
```

Zadanie systemowe

```
module rom #(parameter NW=16) (input clk, read, input [3:0] addr,
    output reg [7:0] oneWord);

reg [7:0] ROmem [0:NW-1];

initial $readmemb("rom.txt1",ROmem);

always @(posedge clk)
   if(read)
      oneWord <= ROmem[addr];

endmodule
```

Synteza pamięci stałej

ISim, Xilinx®

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <rom>.
    Related source file is "romPrg.v".
WARNING:Xst:1781 - Signal <ROmem> is used but never assigned. Tied to default value.
    Found 16x8-bit ROM for signal <$varindex0000> created at line 12.
    Found 8-bit register for signal <oneWord>.
    Summary:
        inferred   1 ROM(s).
        inferred   8 D-type flip-flop(s).
Unit <rom> synthesized.
```

```
=========================================================================
*                     Advanced HDL Synthesis                            *
=========================================================================


Synthesizing (advanced) Unit <rom>.
INFO:Xst:3044 - The ROM <Mrom__varindex0000> will be implemented as a read-only BLOCK RAM, absorbing the register: <oneWord>.
INFO:Xst:3225 - The RAM <Mrom__varindex0000> will be implemented as BLOCK RAM
```

Zadanie systemowe

Synteza pamięci stałej

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
Top Level Output File Name         : rom
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : Yes

Design Statistics
# IOs                              : 14

Cell Usage :
# BELS                             : 1
#      GND                         : 1
# RAMS                             : 1
#      RAMB16BWE                   : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 13
#      IBUF                        : 5
#      OBUF                        : 8
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s700anfgg484-4

 Number of Slices:                        0  out of   5888     0%
 Number of IOs:                          14
 Number of bonded IOBs:                  14  out of    372     3%
 Number of BRAMs:                         1  out of     20     5%
 Number of GCLKs:                         1  out of     24     4%
```

ISim, Xilinx®

Zadania obsługi plików

Obsługa plików poprzez zadania systemowe:

- Otwarcie pliku: `<uchwyt_pliku>=$fopen("<nazwa_pliku>",tryb);`

`<uchwyt_pliku>` - 32-bitowa wartość z MSB = 1;

Deskryptory predefiniowane:

STDIN	32'h8000_000
STDOUT	32'h8000_001
STDERR	32'h8000_002

- Pisanie do pliku: `$fdisplay(<uchwyt_pliku>, p1, p2, ..., pn);`
  `$fmonitor(<uchwyt_pliku>, p1, p2, ..., pn);`

`p1, p2, ..., pn` – zmienne, sygnały lub łańcuchy znakowe w cudzysłowie tak samo jak dla zwykłych zadań `$display`, `$monitor`

`tryb` – sposób otwarcia pliku: `r` – odczyt, `w` – zapis, `a` – kontynuacja.

- Odczyt z pliku: `$fgetc`, `$ungetc`, `$fgets`, `$fscanf`, `$fread`
- Zamknięcie pliku: `$fclose(<uchwyt_pliku>);`

Zadania obsługi plików - odczyt

Zadanie	Opis
`$fgetc`	Czyta jedne znak
`$ungetc`	Wstawia znak `c` spowrotem do bufora określonego deskryptorem pliki `fd`
`$fgets`	Czyta jedną linię
`$fscanf`	Czyta formatowane dane
`$fread`	Czyta dane binarne

Zadanie obsługi plików - odczyt

```
module str();
integer fptr, c;
reg [8*12:1] nazwa_plik, buffer;
initial nazwa_plik = "datafile.txt";

initial begin
  fptr = $fopen(nazwa_plik,"r");
  if( !fptr ) $stop;
  $display("Otwarto plik %s do czytania o deskryptorze %b",
                                        nazwa_plik, fptr);
  c = 1;
  while ( c != 0 ) begin
        c = $fgets(buffer, fptr);
        $write(" number=%0d data:%s", c, buffer);
  end
  $fclose(fptr);
end

endmodule
```

```
abababababa
xxxxbbbbcccc
ccccdddd
mmmmmmmmmmmmm
```

Bufor o długosci 12 bajtów

```
Otwarto plik datafile.txt do czytania o deskryptorze 10000000000000000000000000000011
 number=12 Data:abababababa
 number=12 Data:xxxxbbbbcccc number=1 Data:
 number=9 Data:   ccccdddd
 number=12 Data:mmmmmmmmmmmm umber=2 Data:          m
 number=0 Data:          m
```

Napotkano koniec pliku, więc bufor `buffer` nie został zaktualizowany.

Zadanie obsługi plików - zapis

```
module strout();
reg [255:0] v;
reg [7:0] xb;
integer fptr;
initial begin
  xb = {8{1'b1}};
  v = {32{xb}};
  fptr = $fopen("output_data.txt", "w");
  if(!fptr) begin
        $display("Blad !");
        $stop;
  end
  $display("Otwarto plik output_data.txt o deskryptorze %b", fptr);
  $fwrite(fptr, "Value of v is: %b", v);
  $fwrite(fptr, "Some string.");
  $fclose(fptr);
end
endmodule
```

```
Value of v is:
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111
Some string.
```

Wynik symulacji:

```
Otwarto plik output_data.txt o deskryptorze 10000000000000000000000000000011
```

Zadanie obsługi plików

Wielokanałowy deskryptor:

- służy tylko do zapisu
- powstaje w nieobecności argumentu `tryb`
- ma MSB=0 i LSB=0
- zawiera tylko jeden bit ustawiony na 1.

Zapis do wielu plików równocześnie jest możliwy poprzez alternatywę deskryptorów wielokanałowych.

Zapis wielokanałowy

```
module mcdTry;
integer mcd1, mcd2, cluster, all;
initial begin
        mcd1 = $fopen("plik1.dat");
        if(mcd1 == 0)
                $display("Blad otwarcia pliku !");
        mcd2 = $fopen("plik2.dat");
        if(mcd2 == 0)
                $display("Blad otwarcia pliku !");
        cluster = mcd1 | mcd2;
        //uzupelnij o stdout
        all = cluster | 1'b1;
        $display("mcd1\t=%b\nmcd2\t=%b\ncluster\t=%b\nall\t=%b",
               mcd1,mcd2,cluster,all);
        $fdisplay(all,"Czas: %0d ns", $time);
        #10 $fdisplay(cluster,"XXXXXXXX", $time);
        forever #10 $fdisplay(mcd2, "tylko do plik2.txt");
end
initial #100 $finish;
endmodule
```

Zapis wielokanałowy

Wyniki z symulacji kodu na poprzednim slajdzie:

plik2.txt

```
Czas: 0 ns
XXXXXXXX                    10
tylko do plik2.txt
tylko do plik2.txt
tylko do plik2.txt
tylko do plik2.txt
tylko do plik2.txt
tylko do plik2.txt
tylko do plik2.txt
tylko do plik2.txt
```

plik1.txt

```
Czas: 0 ns
XXXXXXXX                    10
```

stdout

```
ncsim> run
mcd1    =00000000000000000000000000000010
mcd2    =00000000000000000000000000000100
cluster =00000000000000000000000000000110
all     =00000000000000000000000000000111
Czas: 0 ns
Simulation complete via $finish(1) at time 100 NS + 0
./mcd.v:18 initial #100 $finish;
```

Typy symulatorów cyfrowych

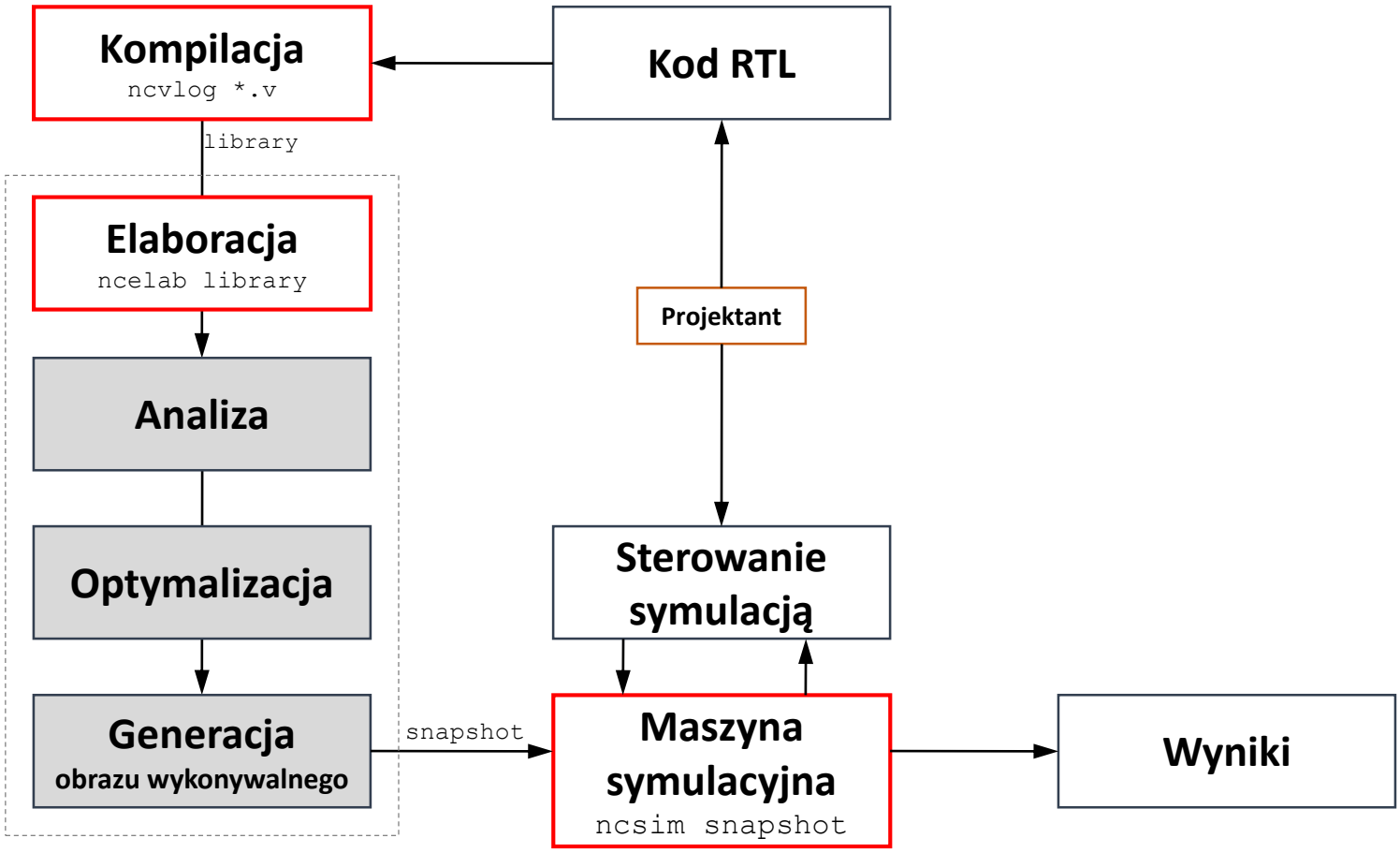
Pod względem konstrukcji rozróżniamy:

- Interpretery (*interpreted simulators*) – wczytywane są kolejne komendy HDL-a, budowane struktury danych w pamięci i bezpośrednio wykonywane

- Kompilowane (*compiled code simulators*) – projekt w HDL-u konwertowany jest do równoważnego kodu w C, który następnie kompilowany jest standardowym kompilatorem, a otrzymany obraz wykonywalny jest następnie uruchamiany

- Natywne (*native code simulators*) – projekt w HDL-u konwertowany jest bezpośrednio do równoważnego obrazu wykonywalnego dla konkretnej platformy sprzętowej komputera, a następnie otrzymany obraz ten jest uruchamiany

Architektura symulatora natywnego



Kompilacja – sprawdzenie syntaktyki kodu i utworzenie wewnętrznych składników dla następnej fazy,
Elaboracja – dołączenie wewnętrznych składników, rozwinięcie pętli, ekspansja pętli `generate`, ustawianie parametrów, przesłanie argumentów do zadań i funkcji ... ,
Generacja – utworzenie kodu, który będzie użyty przez maszynę symulacyjną.

Algorytmy symulacji

Trzy typy algorytmu symulacji:

- Wszystkie elementy są procesowane cały czas (*oblivious-driven*) – odpowiedni dla symulatorów analogowych,
- Element projektu (bramka lub blok kodu) jest procesowany tylko w momentach zdarzeń (*event-driven*), którymi są zmiany wartości sygnałów lub zmiennych,
- Obliczana jest tylko odpowiedź ustalona układu w chwilach wyznaczonych przez aktywne zbocza zegara (*cycle-based*),

Układ synchroniczny

Charakteryzuje się:

- jednym sygnałem zegarowym (*master clock*) i
- jednym sygnałem ustawiania/kasowania (*master set/reset*),

które sterują wszystkimi elementami sekwencyjnymi w projekcie.

Za najbardziej bezpieczne podejście w dziedzinie zapewnienia właściwych relacji czasowych w układzie scalonym uważa się metodologię układu synchronicznego.

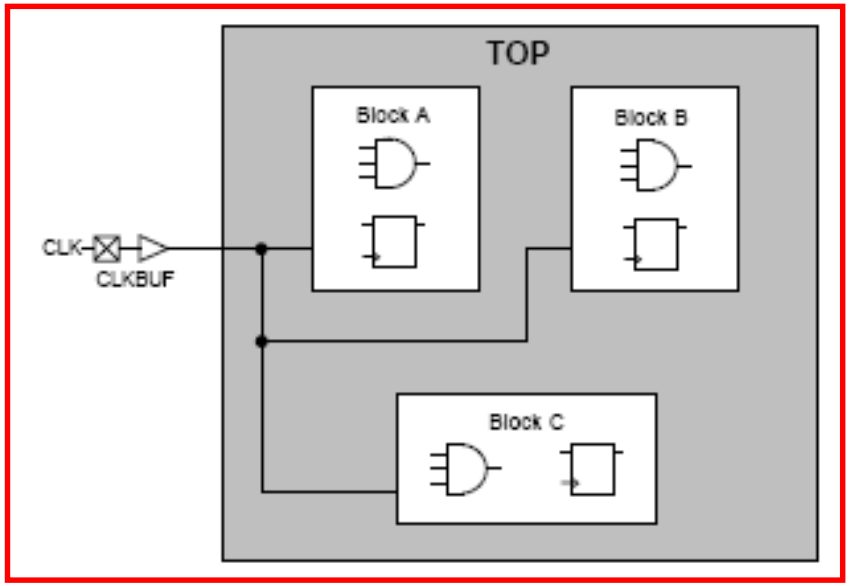
Podejście w pełni synchroniczne może wydłużać wstępną fazę projektu (*up-front design time*). Jednak osiągamy znaczną oszczędność w czasie weryfikacji.

Całkowicie synchroniczny projekt może używać więcej zasobów ale:

- Układ będzie bardziej stabilny,
- Krótszy będzie czas debugowania,
- Skuteczna będzie statyczna analiza czasowa,
- Łatwość przeniesienia na inną technologię.

Reguły projektu synchronicznego

- Wszystkie przerzutniki w ścieżce danych powinny być bezpośrednio połączone do jednego wspólnego globalnego bufora zegara,
- Wszystkie przerzutniki powinny być wyzwalane na tym samym zboczu zegara,
- Nie mogą istnieć przerzutniki taktowane sygnałem innym niż zegar,
- Jeśli potrzeba więcej niż jeden zegar i dane wędrują pomiędzy domenami różnych zegarów, to muszą być resynchronizowane za pomocą kilku przerzutników gdy przechodzą z jednej domeny do drugiej.




```
always @(posedge nie_zegar)
```