Lucrarea 4

Modelarea circuitelor secvenţiale

Această lucrare prezintă modul de descriere a semnalului de ceas şi a numărătoarelor în Verilog/VHDL, precum şi metode de semnalizare a depăşirii la numărare.

4.1 Scopul lucrării

- Modelarea semnalului de ceas.
- Modelarea unui numărător.
- Semnalizarea depăşirii la numărare.

4.2 Modelarea semnalului de ceas

La realizarea unui mediu de testare pentru un sistem secvenţial trebuie să se modeleze şi un semnal de ceas. Din punct de vedere al integrării în mediul de testare, semnalul de ceas poate fi produs de un modul separat sau poate fi parte din entitatea de nivel înalt.

- Semnal de ceas modelat în VHDL ca modul independent:

```
ENTITY ck_gen IS
  GENERIC (per : time := 10 ns);
  PORT (ck : BUFFER std_logic);
END ck_gen;

ARCHITECTURE ck_gen OF ck_gen IS
BEGIN
  ck <= NOT ck AFTER (per/2);
END ck_gen;
```

- Semnal de ceas modelat în VHDL ca un proces din cadrul unui modul de test:

```
ARCHITECTURE ck_gen_test OF ck_gen_test IS
  SIGNAL ck: Bit;
```

```
  ...
BEGIN
  ...
  PROCESS
  BEGIN
    ck <= '0', '1' AFTER 10 ns;
    WAIT FOR 20 ns;
  END PROCESS;
  ...
END ck_gen_test;
```

- Semnal de ceas modelat în Verilog cu perioadă parametrizabilă:

```
‘timescale 1ns/1ns

module ck_gen (ck);
parameter per = 10;
output ck;
reg ck;

initial ck <= 0;

always #per ck <= ~ ck;
endmodule
```

Instanţierea acestor module parametrizabile în cadrul unui modul de test, se poate face cu transmiterea altor valori pentru parametrii modulului.

Instanţierea şi transmiterea parametriilor modulului generator de ceas, în VHDL:

```
ARCHITECTURE ck_gen_test OF ck_gen_test IS
  SIGNAL s : std_logic;

  COMPONENT ck_gen
     GENERIC (per : time);
     PORT (ck : BUFFER std_logic);
  END COMPONENT;

  FOR DUT : ck_gen USE ENTITY WORK.ck_gen(ck_gen);

BEGIN
  DUT : ck_gen GENERIC MAP (20 ns)
               PORT MAP (s);
  ...
END test;
```

Instanţierea şi transmiterea parametriilor modulului generator de ceas, în Verilog:

```
module test_ck ();
```

```
wire ck1;

ck_gen #20 DUT (.ck(ck1));
endmodule
```

În ambele exemple s-a instanţiat un modul generator de semnal de ceas cu o perioadă de 20 ns transmisă ca parametru. În cazul în care nu se transmite nici o valoare pentru perioada semnalului de ceas, se va considera parametrul implicit de 10 ns, aşa cum este specificat în descrierea modulelor.

4.3 Numărătoare

Exemplul următor prezintă descrierea unui circuit numărător cu reset sincron.

- Varianta VHDL a numărătorului este prezentată în continuare:

```
ENTITY counter IS
  GENERIC (width : integer := 4);
  PORT (ck, rst_n : IN std_logic;
        counter : OUT std_logic_vector (width-1 DOWNTO 0);
        ovf : OUT std_logic);
END counter;

ARCHITECTURE counter OF counter IS
  SIGNAL s : std_logic_vector (width-1 DOWNTO 0);
BEGIN
  PROCESS (ck)
  BEGIN
    IF (ck = '1') AND (ck'event) THEN
        IF (rst_n = '0') THEN
           s <= (OTHERS => '0');  --initializare cu         ELSE
           s <= s + 1;
        END IF;
     END IF;
  END PROCESS;
  counter <= s;
END counter;
```

- Varianta Verilog a aceluiaşi numărător este prezentată în continuare:

```
‘timescale 1ns/1ns

module num (ck, rst_n, counter);
parameter width = 4;
input ck, rst_n;
output [width-1:0] counter;
reg [width-1:0] counter;
```

```
    always @(posedge ck)
    begin
      if (!rst_n)
         counter <= 0;
      else
         counter <= counter + 1;
    end
    endmodule
```

Pentru semnalizarea depăşirii la numărăre se determină cazul în care circuitul a trecut peste limita superioară a intervalului de numărare. Acest lucru este posibil prin realizarea unei funcţii SAU-NU între toţi biţii numărătorului, aşa cum se exemplifică în figura 4.1.

Codul VHDL corespunzător numărătorului cu semnalizarea depăşirii este:

```
ARCHITECTURE counter_1 OF counter IS
  SIGNAL s : std_logic_vector (width-1 DOWNTO 0);
BEGIN
  PROCESS (ck)
      VARIABLE v : std_logic;
  BEGIN
    IF (ck = '1') AND (ck'event) THEN
        IF (rst_n = '0') THEN
           s <= (OTHERS => '0');
        ELSE
           s <= s + 1;
           v := '0';
           FOR i IN width-1 DOWNTO 0 LOOP
               v := v OR s(i);
           END LOOP;
           v := NOT v;
        END IF;
    END IF;
    ovf <= v;
  END PROCESS;
  counter <= s;
END counter_1;
```

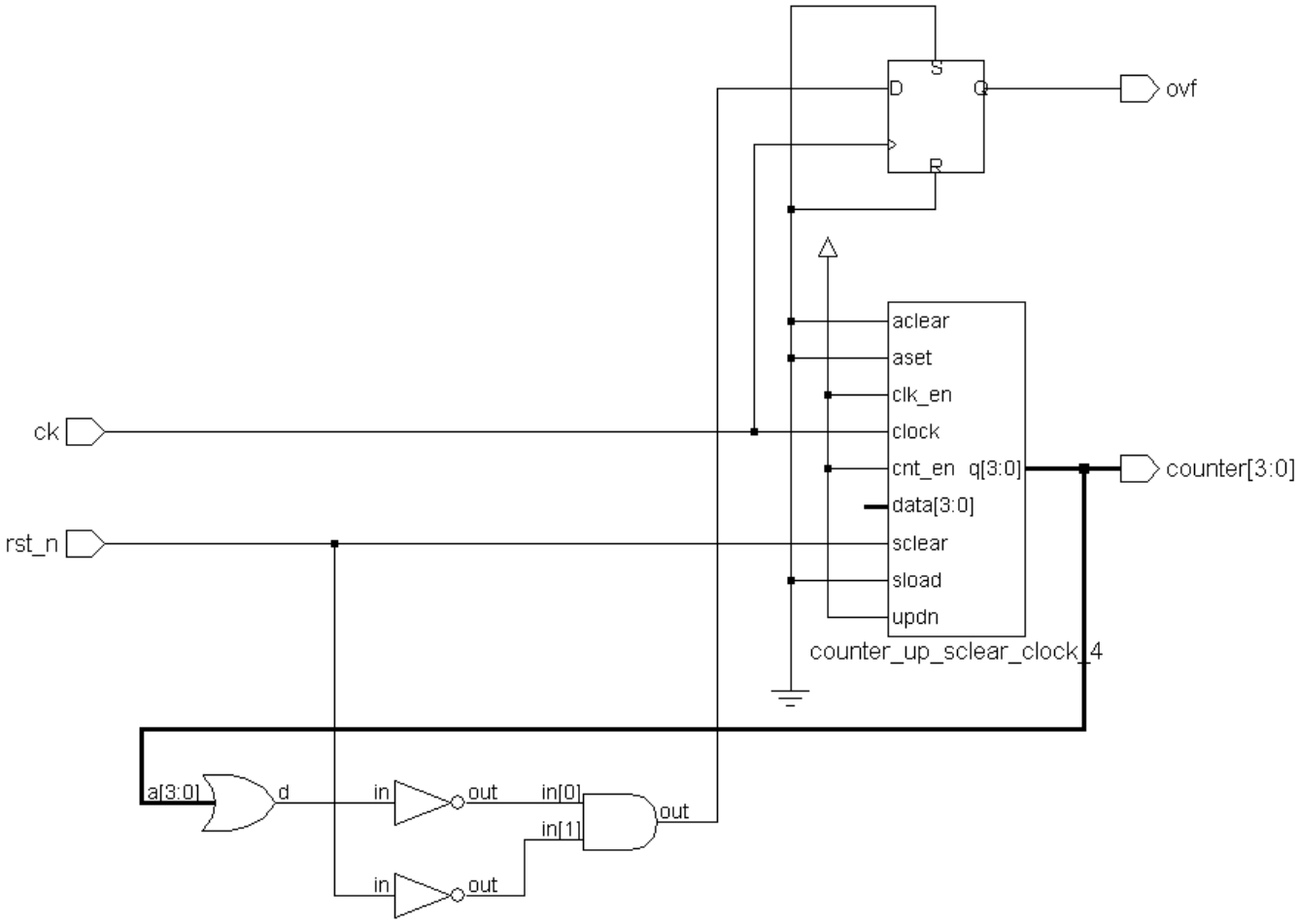

Figura 4.1: Numărător cu semnalizarea depăşirii sintetizat cu *Leonardo*.

Varianta Verilog a aceluiaşi modul este prezentată în continuare:

```
always @(posedge Ck)
begin
  counter <= counter + 1;
  ovf = ~|counter;
end
```

De remarcat că pentru un număr mic de biţi (4 sau 8) circuitul combinaţional fololsit pentru detecţia depăşirii nu are un timp de întârziere foarte mare. Pentru un număr mai mare de biţi (32 sau 64) timpul de propagare prin circuitul combinaţional creşte în mod considerabil (datorită numărului mare de intrări ale porţii SAU-NU).

O soluţie independentă de dimensiunea numărătorului este prezentată în figura 4.2.

Alternativa prezentată constă în compararea valorii curente şi a celei întârziate cu un tact a celui mai semnificativ bit al numărătorului. În momentul detectării succesiunii 1→0 se semnalează depăşire. Implementarea acestei soluţii este prezentată în continuare:

- VHDL

```
ARCHITECTURE counter_2 OF counter IS
```

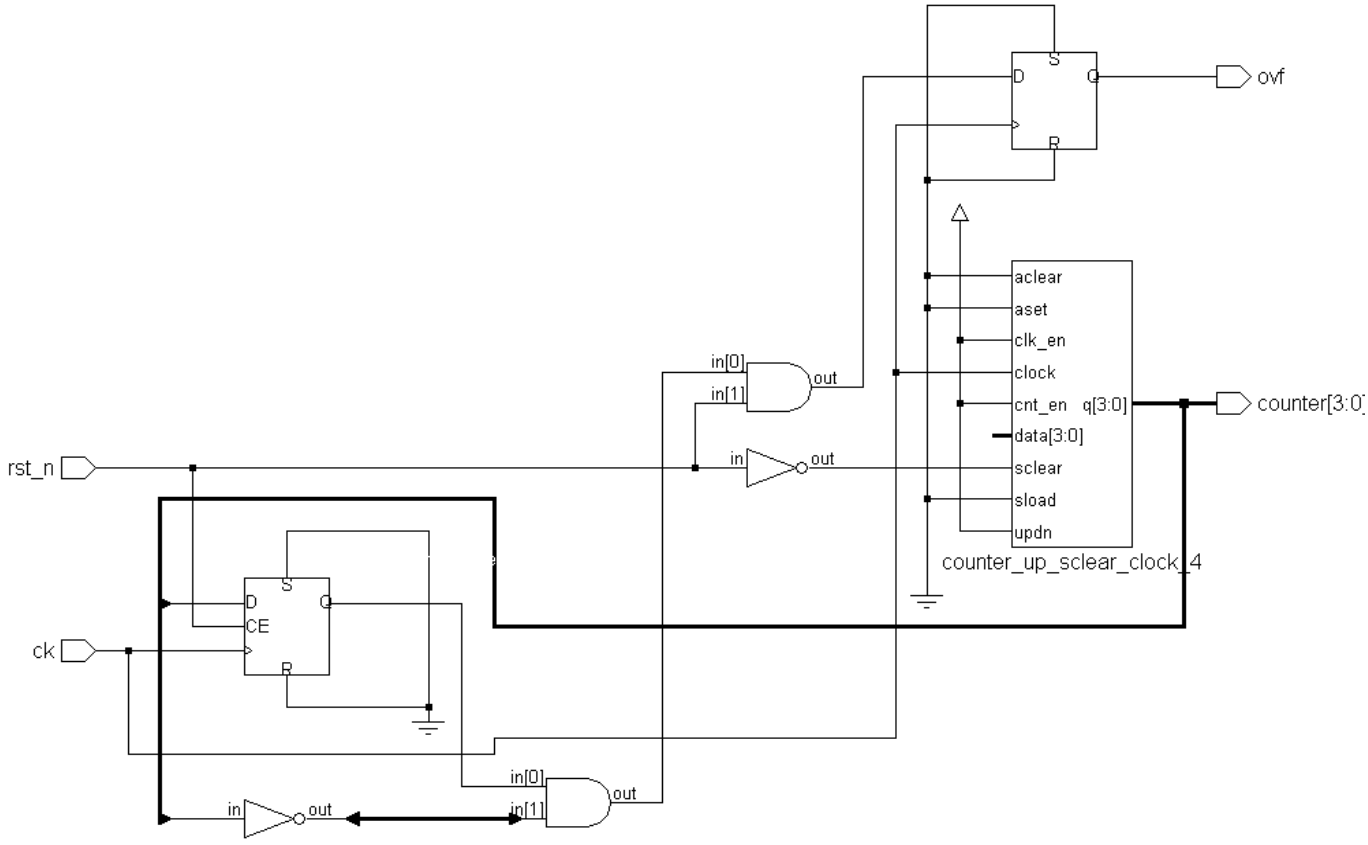

Figura 4.2: Numărător cu semnalizarea depăşirii independent de numărul de biţi.

```
  SIGNAL counter_msb_del : std_logic;
  SIGNAL s : std_logic_vector (width-1 DOWNTO 0);
BEGIN
  PROCESS (ck)
  BEGIN
    IF (ck = '1') AND (ck'event) THEN
        IF (rst_n = '0') THEN
           s <= (OTHERS => '0');
        ELSE
           s <= s + 1;
           counter_msb_del <= s(width-1);
        END IF;
        Ovf <= counter_msb_del AND NOT s(width-1);
     END IF;
  END PROCESS;
  counter <= s;
END counter_2;
```

- Verilog

```
always @(posedge Ck)
begin
  counter <= counter + 1;
  counter_msb_del <= counter[width-1];
```

```
  Ovf <= counter_msb_del & ~counter[width-1];
end
```

4.4 Simularea numărătoarelor.

1. Verificaţi existenţa directorului *C:\Home* şi a dreptului de scriere în acesta. Ştergeţi eventualele fişiere şi subdirectoare aflate în acest director. Aceste acţiuni se pot executa din sistemul de operare, folosind Windows Explorer. Copiaţi fişierele Verilog (fişiere cu extensia ".v") din directorul *\asd_hdl\exemple* în directorul *C:\home*. Acest director va fi director de proiect în *ModelSim*. Lansaţi în execuţie *ModelSim*.

2. Simulaţi circuitul numărător. Pentru simulare trebuie să realizaţi modulului de testbench, precum şi modulul de test aferente numărătorului. Pentru aceasta folosiţi fragmentele de cod prezentate în lucrare. Schema modulului de test este prezentată în figura 4.3.

   Descrierea circuitului numărător se găseşte în fişierul **num.v**.

3. Îmbunătăţiţi modelul de numărător prezentat anterior pentru a avea facilităţi de:

   - *reset* asincron;
   - *load* sincron;
   - *ce - Count Enable*;
   - numărare în sens crescător şi descrescător.

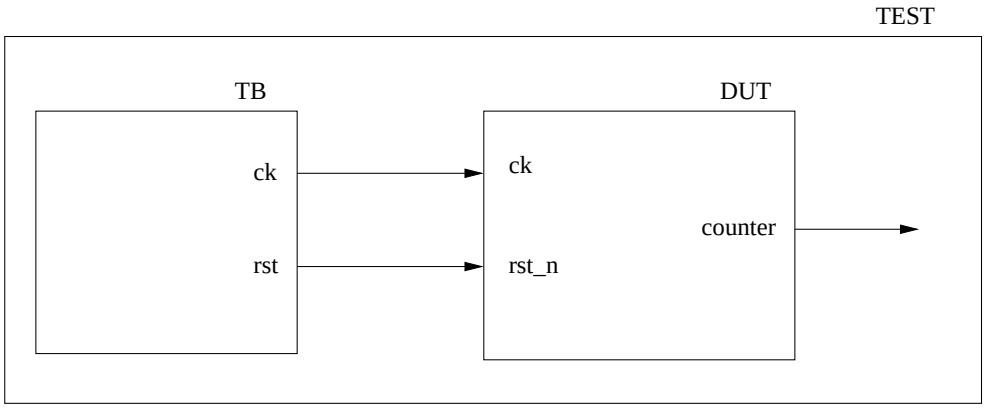

Figura 4.3: Structura modulului de test pentru numărător.

4.5 Exerciţii

1. Modelaţi un numărător în cod GRAY.

2. Se consideră interfaţa unui numărător pe 8 biţi:

```
module counter (clock, data_in, load, ce, UpNotDown,
                count_out, carry_out, borrow_out, parity_out);
output [7:0] count_out;
```

```
output       carry_out, borrow_out, parity_out;
input  [7:0] data_in;
input        clock, load, ce, UpNotDown;

reg [7:0] count_out;
reg       carry_out, borrow_out, parity_out;
// Inseraţi modulul vostru aici
endmodule
```

Funcţiile numărătorului sunt descrise în tabelul 4.1

Numărătorul este activ pe frontul pozitiv.

- *parity_out* semnalizează paritatea pară a celor 8 biţi de ieşire.
- *borrow_out* semnalizează o depăşire în cazul numărării în sens descrescător.
- *carry_out* semnalizează o depăşire în cazul numărării în sens crescător.

Modelaţi Verilog/VHDL modulul descris şi realizaţi testarea acestuia.

ce	*load*	*UpNotDown*	*Funcţie*
0	x	x	Păstrează starea
1	1	x	Încarcă *data_in*
1	0	0	Numără în sens descrescător
1	0	1	Numără în sens crescător

Tabelul 4.1: Funcţiile numărătorului.

3. Simulaţi şi testaţi următorul model analizând modul în care se face iniţializarea. Sintetizaţi modulul. Explicaţi rezultatele obţinute.

```
module dff (D, Q, Clk, Rst);

parameter width = 1,
reset_value = 0;

input [width - 1 : 0] D;
input Clk;
input Rst;
output [width - 1 : 0] Q;

reg [width - 1 : 0] Q;

initial Q = {width{1x}};

always @ ( posedge Clk or negedge Rst )
if ( Rst == 0 )
Q <= #1 reset_value;
else
```

```
Q <= #1 D;
endmodule
```