LABORATOR 2 -assign

Implementat, i urm˘atoarele funct, ii Booleene utilizˆand doar port, i NAND ˆın Verilog.

//HARTA KARNOUGH .V

module ex1c (

input a, b, c, d,

output f3

);

//write Verilog code here

Assign f3=

endmodule

module ex1c\_tb;

reg a, b, c, d;

wire f3;

ex1c ex1c\_i (.a(a), .b(b), .c(c), .d(d), .f3(f3));

integer k;

initial begin

$display("Time\ta\tb\tc\td\tabcd\_10\tf3");

$monitor("%0t\t%b\t%b\t%b\t%b\t%0d\t%b", $time, a, b, c, d, {a,b,c,d}, f3);

{a, b, c, d} = 0;

for (k = 1; k < 16; k = k + 1)

#10 {a, b, c, d} = k;

end

endmodule

//HARTA KARNOUGH RUN. TXT //do run.txt

????? trb schimbat nume daca schimb ce cum????

if [file exists "work"] {vdel -all}

vlib work

if [catch "vlog ex1c.v"] return

vsim -voptargs=+acc ex1c\_tb

run -all

quit -sim

Implementat, i convertorul Binary Coded Decimal dat prin tabelul de adev˘ar de mai jos. Construit, i codul Verilog al acestui modul.

////Basicly o intrare 4 biti si o iesire 4 biti, pt fiecare bit harta K

module ex3 (

input [3:0] i,

output [3:0] o

);

Assign o[3]=

Assign o[2]=

Assign o[1]=

Assign o[0]=

endmodule

module ex3\_tb;

reg [3:0] i;

wire [3:0] o;

ex3 ex3\_i (.i(i), .o(o));

integer k;

initial begin

$display("Time\ti\to");

$monitor("%0t\t%b\t%b", $time, i, o);

i = 0;

for (k = 1; k < 10; k = k + 1)

#10 i = k;

end

endmodule

LABORATOR 3- always

Construit, i un modul Verilog numit msd, utilizˆand blocuri always, avˆand o intrare i pe 5 bit, i s, i o ies, ire o pe 4 bit, i. Ies, irea o are valoarea celei mai semnificative cifre zecimale din reprezentarea num˘arului f˘ar˘a semn de la intrarea i. Exemplu: i = 23 → o = 2; i = 9 → o = 9.

module msd (

input [4:0] i,

output reg[3:0] o

);

//write Verilog code here

always @(\*) begin

if(i<10)

o=i;

else

o=i/10;

end

endmodule

module msd\_tb;

reg [4:0] i;

wire [3:0] o;

msd msd\_i (.i(i), .o(o));

integer k;

initial begin

$display("Time\ti\t\to");

$monitor("%0t\t%b(%2d)\t%b(%0d)", $time, i, i, o, o);

i = 0;

for (k = 1; k < 32; k = k + 1)

#10 i = k;

end

endmodule

Proiectat, i un modul numit div3, avˆand o intrare i, pe 4 bit, i si o ies, ire o, pe num˘arul minim de bit, i necesar, astfel ˆıncˆat la ies, ire s˘a fie furnizat cˆatul ˆımp˘art, irii num˘arului f˘ar˘a semn de la intrare la 3. Nu se va folosi operatorul de ˆımp˘art, ire ”/” iar solut, ia va folosi blocuri always. Exemplu: i = 10 → o = 3; i = 2 → o = 0.

module div3 (

input [3:0] i,

output reg[2:0][/\*replace with width of output o here\*/:0] o

);

//write Verilog code here

Reg aux[3:0];

always@(\*) begin

o=3’b000;

aux=i;

while(aux>2) begin

o=o+1;

aux=aux-3;

end

endmodule

module div3\_tb;

reg [3:0] i;

wire [/\*replace with width of output o here\*/:0] o;

div3 div3\_i (.i(i), .o(o));

integer k;

initial begin

$display("Time\ti\t\to");

$monitor("%0t\t%b(%2d)\t%b(%0d)", $time, i, i, o, o);

i = 0;

for (k = 1; k < 16; k = k + 1)

#10 i = k;

end

endmodule

Proiectat, I un dispozitiv de num˘arare a bit, ilor de 1 din reprezentarea num˘arului conectat la intrarea i. Modulul se va numi cnt1s, s, I are o intrare I, pe 6 bit, I s, I o ies, ire o pe num˘arul minim necesar de bit, i. Construit, I acest modul utilizˆand blocuri always.

Module cnt1s (

input [5:0] i,

output reg[/\*replace with width of output o here\*/2:0] o

);

//write Verilog code here

integer digit

always@(\*) begin

o=3’b000;

for(digit=0; digit<6;digit=digit+1)

o=o+i[digit];

end

endmodule

module cnt1s\_tb;

reg [5:0] i;

wire [/\*replace with correct width of output o here\*/:0] o;

cnt1s cnt1s\_i (.i(i), .o(o));

integer k;

initial begin

$display("Time\ti\t\to");

$monitor("%0t\t%b(%2d)\t%b(%0d)", $time, i, i, o, o);

i = 0;

for (k = 1; k < 64; k = k + 1)

#10 i = k;

end

endmodule

Construit, i un dispozitiv numit seq3b, avˆand o intrare i pe 4 bit, i s, i o ies, ire o pe 1 bit. Ies, irea va fi activ˘a daca ˆın num˘arul binar de la intrare exista o secvent,˘a de 3 bit, i consecutivi avˆand aceeas, i valoare. Exemplu: i = 14 → o = 1; i = 9 → o = 0.

module seq3b (

input [3:0] i,

output reg o

);

//write Verilog code here

always@(\*) begin

case(i)

0,1,7,8,14,15:o=1;

default:o=0;

endcase

end

endmodule

module seq3b\_tb;

reg [3:0] i;

wire o;

seq3b seq3b\_i (.i(i), .o(o));

integer k;

initial begin

$display("Time\ti\t\to");

$monitor("%0t\t%b(%2d)\t%b", $time, i, i, o);

i = 0;

for (k = 1; k < 16; k = k + 1)

#10 i = k;

end

endmodule

Proiectat, i un modul mul5bcd avˆand o intrare i pe 4 bit, i s, i dou˘a ies, iri d s, i u ambele pe 4 bit, i. La intrarea i se primes,te o cifr˘a BCD pe 4 bit, i iar modulul va furniza la ies, iri rezultatul ˆınmult, irii cifrei i cu cifra 5 ˆın BCD: ies, irea d va reprezenta cifra zecilor pentru rezultat iar ies, irea u va reprezenta cifra unit˘at, ilor pentru rezultat. Exemplu: i = 3(0011) → d = 1(0001), u = 5(0101); i = 9 → d = 4(0100), u = 5(0101).

module mul5bcd (

input [3:0] i,

output [3:0] d, u

);

//write Verilog code here

reg [5:0]aux;

always@(\*) begin

aux=5\*i;

d=aux/10;

u=aux%10;

end

endmodule

module mul5bcd\_tb;

reg [3:0] i;

wire [3:0] d, u;

mul5bcd mul5bcd\_i (.i(i), .d(d), .u(u));

integer k;

initial begin

$display("Time\ti\t\td\t\tu");

$monitor("%0t\t%b(%4d)\t%b(%4d)\t%b(%4d)", $time, i, i, d, d, u, u);

i = 0;

for (k = 1; k < 10; k = k + 1)

#10 i = k;

end

endmodule

Proiectati unitatea text2nibble, avˆand intrarea i pe 8 bit, i s, i ies, irea o pe 4 bit, i. Intrarea i primes,te un caracter ASCII. Dac˘a caracterul este cifr˘a zecimal˘a (’0’ la ’9’), furnizeaz˘a la ies, ire valoarea cifrei, altfel furnizeaz˘a valoarea 15.

module text2nibble (

input [7:0] i,

output reg[3:0] o

);

//write Verilog code here

always@(\*) begin

if(i>=48 && i<=57)

o=i-48;

else

o=15;

end

endmodule

module text2nibble\_tb;

reg [7:0] i;

wire [3:0] o;

text2nibble text2nibble\_i (.i(i), .o(o));

integer k;

initial begin

$display("Time\ti\ti\_chr\to");

$monitor("%0t\t%b\t%c\t%b(%d)", $time, i, i, o, o);

i = 0;

for (k = 1; k < 256; k = k + 1)

#10 i = k;

end

endmodule

Proiectati un registru pe 4 bit, i, numit r4b. Registrul are facilit˘at, i de ˆıncarcare paralel˘a a cont, inutului de la intrarea d pe 4 bit, i s, i de deplasare la dreapta a cont, inutului cu 1 bit, caz ˆın care valoarea bitului mai semnificativ este primit˘a de la intrarea pe un bit sh in. Registrul are intr˘arile de comand˘a, sincrone, ld - care activeaz˘a ˆıncarcarea paralel˘a s, i, repsectiv, sh - care declans,az˘a deplasarea la dreapta. Registrul va avea ies, irea q, pe 4 bit, i, reprezentˆand cont, inutul registrului. Pentru implementare modificat, i fis, ierul surs˘a r4b.v .

Not˘a: Dispozitivele secvent, iale sincrone, cum este s, i registrul din problem˘a, vor avea implicit intr˘arile de tact clk s, i o linie de init, ializare, asincron˘a, activ˘a la 0, rst b, except, ie cazul ˆın care sunt specificate alte semnale.

//asta nu e verificata

module r4b (

input clk, rst\_b, ld, sh, sh\_in,

input [3:0] d,

output reg [3:0] q

);

//write Verilog code here

alsways@(posedge clk, negedge rst\_b)

begin

if(rst\_b==0)

q<=0;

else

begin

if(ld)

q<=d;

if(sh)

begin

q>>1;

q[3]<=sh\_in;

end

end

end

endmodule

module r4b\_tb;

reg clk, rst\_b, ld, sh, sh\_in;

reg [3:0] d;

wire [3:0] q;

r4b r4b\_i (.clk(clk), .rst\_b(rst\_b), .ld(ld), .sh(sh), .sh\_in(sh\_in), .d(d), .q(q));

initial begin

{clk, rst\_b} = 0;

#5 rst\_b = 1;

#45 clk = 1;

repeat (40)

#50 clk = ~clk;

end

integer k, l;

initial begin

$display("Time\top\td\tsh\_in\tq");

{d, sh\_in} = 0; {ld, sh} = 0;

for (k = 0; k < 32; k = k + 1) begin

$display("%0t\t%s\t%b\t%b\t%b", $time, (ld) ? "LOAD" : (sh) ? "SHIFT" : "NO\_OP", d, sh\_in, q);

#100 l = $urandom; {d, sh\_in} = l[6:2]; {ld, sh} = l[1:0] % 3;

end

$display("%0t\t%s\t%b\t%b\t%b", $time, (ld) ? "LOAD" : (sh) ? "SHIFT" : "NO\_OP", d, sh\_in, q);

end

endmodule

LABORATOR 4- testbench SI MODULE IN MODULE?

Construit, i ˆın Verilog unitatea Full Adder Cell (FAC) (numit˘a fac). Modulul are 3 intr˘ari de 1-bit x, y s, i ci, precum s, i 2 ies, iri de 1-bit z s, i co. Scriet, i codul ˆıntr-un fis, ier numit fac.v

FAC

Construit, i un testbench pentru verificarea exhaustiv˘a a unit˘at, ii fac de mai sus, modul numit fac tb. Cont, inutul noului modul poate fi inclus ˆın acelas, i fis, ier fac.v, ˆımpreun˘a cu modulul fac. Construit, i un fis, ier script, numit run fac.txt, pentru compilare, lansare s, i rularea simul˘arii unit˘at, ii testbench fac tb

module FAC(

input x,y,ci,

output z, co

);

assign z=x^y^ci;

assign co=x&y | x&ci | y&ci;

endmodule

module FAC\_tb;

reg x,y,ci;

wire z,co;

FAC uut(.x(x), .y(y), .ci(ci), .z(z), .co(co));

integer i;

initial begin

{x,y,ci}=0;

for(i=;i<8;i=i+1)

begin

#10 {x,y,ci}=i;

#10

$display("%b %b", co,z)

end

end

endmodule

//trebuia scris instanctele uut in modul NU in testbench

Utilizˆand 2 instant,e fac, construit, i un sumator pe 2 bit, i, numit add2b, avˆand 2 intr˘ari a cˆate 2-bit, i x s, i y, o intrare de 1-bit ci, respectiv avˆand ies, irea o reprezentˆand suma pe 2 bit, i s, i transportul de ies, ire, co. Scriet, i codul ˆın fis, ierul add2b.v. Construit, i un testbench pentru verificarea exhaustiv˘a a implement˘arii, numit add2b tb s, i fis, ierul script asociat, run add2b.txt

GEN AM SI FISIERU DE DINAINTE DE LA FAC SI IN ASTA BAG ADD2B LA CARE FAC TESTBENCH siiiiii pun ambele si fac.v si add2b.v in run dar fisieru de testbench e add2b\_tb

module add2b(

input [1:0]x,y,

input cin,

output [1:0]z,

output cout

);

endmodule

module add2b\_tb;

reg [1:0]x,y;

reg cin;

wire [1:0]z;

wire cout;

wire tmp;

FAC inst1(.x(x[0], .y(y[0]), .ci(cin), .z(z[0]), .co(tmp));

FAC inst2(.x(x[1], .y(y[1], .ci(tmp), .z(z[1], .co(cout));

//de fapt instale astea trb puse in modul add2b

integer i;

initial begin

{x,y,cin}=0;

for(i=0;i<32;i=i+1)

begin

#10 {x,y,cin}=i;

#10

$display("%b %b", cout, z);

end

end

endmodule

Proiectat, i un modul cmp2b pentru compararea a 2 numere f˘ar˘a semn pe 2 bit, i s, i care 2 intr˘ari x s, i y pentru cele 2 numere de comparat, respectiv are 3 ies, iri a cˆate 1-bit: eq(egal), lt(mai mic decˆat) s, i gt(mai mare decˆat). Scriet, i codul ˆın fis, ierul cmp2b.v.

Si verificat ca merge 😊

module cmp2b(

input [1:0]x,y,

output reg eq, lt, gt

);

always@(\*)begin

{eq, gt, lt}=0;

if(x>y)

gt=1;

if(x==y)

eq=1;

if(x<y)

lt=1;

end

endmodule

module cmp2b\_tb;

reg [1:0]x,y;

wire eq, lt, gt;

cmp2b inst1(.x(x), .y(y), .eq(eq), .lt(lt), .gt(gt));

integer i;

initial begin

{x,y}=0;

for(i=0;i<16;i=i+1)

begin

#10 {x,y}=i;

#10

$display("%b %b %b", eq, lt, gt);

end

end

endmodule

Folosind unitatea cmp2b, construit, i un comparator pentru valori f˘ar˘a semn pe 4-bit, i, numit cmp4b, avˆand intr˘arile de 4-bit, i x s, i y, respectiv 3 ies, iri a cˆate 1-bit: eq, lt s, i gt. Scriet, i codul ˆın fis, ierul cmp4b.v. Construit, i un modul testbench pentru verificarea exghaustiv˘a a implement˘arii, numit cmp4b tb ˆımpreun˘a cu fis, ierul script asociat, numit run cmp4b.txt

module cmp2b(

input [1:0]a,b,

output reg eqal, lthan, gthan

);

always@(\*)begin

{eqal, gthan, lthan}=0;

if(a>b)

gthan=1;

if(a==b)

eqal=1;

if(a<b)

lthan=1;

end

endmodule

module cmp4b(

input [3:0]x, y,

output reg eq, lt,gt

);

wire eqq, ltt, gtt, eqqq, lttt, gttt;

cmp2b inst1(.a(x[3:2]), .b(y[3:2]), .eqal(eqq), .lthan(ltt), .gthan(gtt));

cmp2b inst2(.a(x[1:0]), .b(y[1:0]), .eqal(eqqq), .lthan(lttt), .gthan(gttt));

always@(\*)begin

if(eqq==1 && eqqq==1)

{eq,lt,gt}=3'b100;

else begin

if(eqq==0)begin

if(ltt==1)

{eq,lt,gt}=3'b010;

else

{eq,lt,gt}=3'b001;

end

else begin

if(lttt==1)

{eq,lt,gt}=3'b010;

else

{eq,lt,gt}=3'b001;

end

end

end

endmodule

Realizat, i un modul pentru adunarea a 2 numere ˆıntregi reprezentate ˆın codul C1, numit c1 add4b, care are 2 intr˘ari a cˆate 4-bit, i x s, i y ˆımpreun˘a cu intrarea de 1 bit ci, generˆand la ies, iri semnalele sum˘a z pe 4 bit, i (f˘ar˘a transport de ies, ire care va fi folosit ca end around carry). Construit, i un modul testbench pentru verificarea exghaustiv˘a a implement˘arii, numit c1add4b tb ˆımpreun˘a cu fis, ierul script asociat, numit run c1add4b.txt.

LABORATOR 5- MODULE PARAMETRIZATE SI TRI-STATE ??

Construit, i un multiplexor 4-la-1 parametrizat, numit mux 2s. Modulul va fi parametrizat prin lat, imea intr˘arilor de date s, i a ies, irii, avˆand urm˘atoarea interfat,˘a:

1 module mux 2s #( 2 pa ram e t e r w = 4 // wi d t h pa ram e t e r 3 ) ( 4 i n p u t [ w=1: 0] d0 , d1 , d2 , d3 , //4 da ta i n p u t s 5 i n p u t [ 1 : 0 ] s , // s e l e c t i o n i n p u t 6 o u t p u t [ w=1: 0] o // da ta o u t p u t 7 ) ;

Implementat, i multiplexorul folosind drivere tri-state.

TEST

1. Construiti un modul ce realizeaza conversia unel citre BCD in corespondentul el in formatul 2-din -5 /4210

Tabelul de adevar:

Zecimal BCD 2-din-5 74210

/\* Construiti un modul ce realizeaza conversia unei cifre BCD in corespondentul ei in formatul 2-din-5 74210. \*/

module bcd\_to\_2outof5(

input[3:0] bcd\_in,

output reg[4:0] \_2outof5);

/\* Write verilog code here \*/

always@(\*) begin

case(bcd\_in)

0: \_2outof5= 5'b11000;

1: \_2outof5= 5'b00011;

2: \_2outof5= 5'b00101;

3: \_2outof5= 5'b00110;

4: \_2outof5= 5'b01001;

5: \_2outof5= 5'b01010;

6: \_2outof5= 5'b01100;

7: \_2outof5= 5'b10001;

8: \_2outof5= 5'b10010;

9: \_2outof5= 5'b10100;

endcase

end

endmodule

module bcd\_to\_2outof5\_tb;

reg[3:0] bcd\_in;

reg[4:0] exp\_2outof5;

wire[4:0] act\_2outof5;

wire verdict;

bcd\_to\_2outof5 uut(.bcd\_in(bcd\_in), .\_2outof5(act\_2outof5));

integer tests\_total, tests\_passed, nota;

assign verdict = exp\_2outof5 === act\_2outof5;

initial begin

$display("bcd\_in\t\texpected\_2outof5\tactual\_2outof5\tPassed(1)/Failed(0)");

$monitor("%6b\t\t%16b\t%14b\t%18b", bcd\_in, exp\_2outof5, act\_2outof5, verdict);

tests\_total = 0;

tests\_passed = 0;

bcd\_in = 4'd0;

exp\_2outof5 = 5'b11000;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd1;

exp\_2outof5 = 5'b00011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd2;

exp\_2outof5 = 5'b00101;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd3;

exp\_2outof5 = 5'b00110;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd4;

exp\_2outof5 = 5'b01001;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd5;

exp\_2outof5 = 5'b01010;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd6;

exp\_2outof5 = 5'b01100;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd7;

exp\_2outof5 = 5'b10001;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd8;

exp\_2outof5 = 5'b10010;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bcd\_in = 4'd9;

exp\_2outof5 = 5'b10100;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

$display("Passed / Total: %2d / %2d", tests\_passed, tests\_total);

nota = tests\_passed \* 100 / tests\_total \* 2;

$display("Nota: %1d.%02d", nota / 100, nota % 100);

end

endmodule

2. Construiti un modul ce insumeaza 2 cifre BCD si returneaza rezultatul in 2 iesiri, fiecare cu cate 4 biti: cifra zecilor si cifra unitatilor.

Folositi ex2.v si run\_ex2.txt. Nota: Este necesar si fisierul ex2\_tests.dat, ce trebuie sa fie in acelasi folder cu ex2.v (2p)

/\* Construiti un modul ce insumeaza 2 cifre BCD si returneaza rezultatul in 2 iesiri, fiecare cu cate 4 biti: cifra zecilor si cifra unitatilor. \*/

module sum\_bcd(input[3:0] nr\_1, nr\_2,

output reg[3:0] out\_u, out\_d);

/\* Write verilog code here \*/

reg [4:0]s=0;

always@(\*)begin

s=nr\_1+nr\_2;

out\_u=s%10;

out\_d=s/10;

end

endmodule

module sum\_bcd\_tb;

reg[3:0] nr\_1, nr\_2;

wire[3:0] act\_out\_u, act\_out\_d;

reg[3:0] exp\_out\_u, exp\_out\_d;

wire verdict\_u, verdict\_d;

sum\_bcd uut(.nr\_1(nr\_1), .nr\_2(nr\_2), .out\_u(act\_out\_u), .out\_d(act\_out\_d));

integer tests\_total, tests\_passed, nota;

integer data\_file, random\_int;

assign verdict\_u = exp\_out\_u === act\_out\_u;

assign verdict\_d = exp\_out\_d === act\_out\_d;

initial begin

$display("nr\_1\tnr\_2\t\texpected\_out\_u\tactual\_out\_u\tPassed(1)/Failed(0)\texpected\_out\_d\tactual\_out\_d\tPassed(1)/Failed(0)");

$monitor("%4d\t%4d\t\t%14d\t%12d\t%18d\t%14d\t%12d\t%18d", nr\_1, nr\_2, exp\_out\_u, act\_out\_u, verdict\_u, exp\_out\_d, act\_out\_d, verdict\_d);

tests\_total = 0;

tests\_passed = 0;

data\_file = 0;

data\_file = $fopen("ex2\_tests.dat", "r");

if (data\_file == 0) begin

$display("Adauga fisierul ex2\_tests.dat in acelasi folder");

$finish;

end

while(!$feof(data\_file)) begin

random\_int = $fscanf(data\_file, "%d %d %d %d\n", nr\_1, nr\_2, exp\_out\_d, exp\_out\_u);

tests\_total = tests\_total + 2;

#1;

tests\_passed = tests\_passed + verdict\_d + verdict\_u;

end

$display("Passed / Total: %2d / %2d", tests\_passed, tests\_total);

nota = tests\_passed \* 100 / tests\_total \* 2;

$display("Nota: %1d.%02d", nota / 100, nota % 100);

end

endmodule

3. Sa se construiasca un registru de deplasare pe w biti (implicit 4) si valoare de reset rst val (implicit O) (parametrii). Modulul va avea functionalitati de shiftare (prin intrare sh) si incarcare sincrone (prin intrare (d), active la 1, si reset asincron (prin intrarea rst\_b), activ la 0.

Daca intrarea sh este activa, atunci modulul va face shiftare cu o pozitie la dreapta a valor inregistrate. Intrarea sh type specifica tipul shiftarii:

\* 0 - shiftare logica

\* 1 - shiftare aritmetica (valoarea inregistrata este considerata ca find in complement de 2)

Folositi ex3.v si run ex3.txt (2.5p)

/\* Sa se construiasca un registru de deplasare pe w biti (implicit 4) si valoare de reset rst\_val (implicit 0)(parametrii).

Modulul va avea functionalitati de shiftare (prin intrarea sh) si incarcare sincrone (prin intrare ld), active la 1, si reset asincron (prin intrarea rst\_b), activ la 0.

Daca intrarea sh este activa, atunci modulul va face shiftare cu o pozitie la dreapta a valorii inregistrate. Intrarea sh\_type specifica tipul shiftarii:

\* 0 - shiftare logica

\* 1 - shiftare aritmetica (valoarea inregistrata este considerata ca fiind in complement de 2)\*/

module shift\_rgst #(/\* define parameters \*/ parameter w=4, rst\_val=0 )

(input [/\*fill\*/w-1:0] d,

input clk, ld, rst\_b, sh, sh\_type,

output reg [/\*fill\*/w-1:0] q);

/\* Write Verilog code here \*/

//varianta BUBU

always@(posedge clk, negedge rst\_b) begin

if(!rst\_b)

q<=rst\_val;

else begin

if(ld==1)

q<=d;

if(sh==1)begin

if(sh\_type==0)begin

q<=q>>1; //shift logica

end

else

begin

q<=q>>1;

q[w-1]<=q[w-1];

//shift aritmetica

end

end

end

end

//varianta MIHAI

always@(posedge clk, negedge rst\_b) begin

if(!rst\_b)

q=rst\_val;

else begin

if(ld==1)

q=d;

if(sh==1)begin

if(sh\_type==0)begin

q=q>>1; //shiftare logica

end

else

begin

q=q>>1;

q[w-1]=q[w-2];

//shiftare aritmetica

end

end

end

end

endmodule

module shift\_rgst\_tb\_0(output reg[31:0] tests\_total, tests\_passed);

reg signed [3:0] d;

reg clk, ld, rst\_b, sh, sh\_type;

wire [3:0] act\_q;

reg [3:0] exp\_q;

wire verdict;

shift\_rgst uut (.d(d), .clk(clk), .ld(ld), .rst\_b(rst\_b), .sh(sh), .sh\_type(sh\_type), .q(act\_q));

assign verdict = exp\_q === act\_q;

initial begin

clk = 0;

repeat(20) #2 clk = ~clk;

end

initial begin

$display("TESTBENCH 0: Valori implicite ale parameterilor");

$display("Time\tclk\tld\trst\_b\tsh\tsh\_type\td\t\texpected\_q\tactual\_q\tPassed(1)/Failed(0)");

$monitor("%4t\t%3b\t%2b\t%5b\t%2b\t%7b\t%4b\t\t%10b\t%8b\t%18b", $time, clk, ld, rst\_b, sh, sh\_type, d, exp\_q, act\_q, verdict);

tests\_total = 0;

tests\_passed = 0;

ld = 0;

rst\_b = 1;

sh = 0;

sh\_type = 0;

exp\_q = 4'dX;

#1;

rst\_b = 0;

d = 4'd2;

exp\_q = 4'd0;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#1;

rst\_b = 1;

d = 4'd7;

ld = 1;

exp\_q = 4'd0;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#2;

rst\_b = 1;

d = 4'd7;

ld = 1;

exp\_q = 4'd7;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

ld = 0;

sh = 1;

sh\_type = 0;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

sh\_type = 0;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#2;

sh = 1;

sh\_type = 1;

exp\_q = 4'b0001;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 0;

ld = 1;

d = 4'b1100;

exp\_q = 4'b1100;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

ld = 0;

sh\_type = 1;

d = 4'b0011;

exp\_q = 4'b1110;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

ld = 0;

sh\_type = 0;

d = 4'b1100;

exp\_q = 4'b0111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 0;

ld = 1;

sh\_type = 1;

d = 4'b1111;

exp\_q = 4'b1111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 0;

d = 4'b0000;

exp\_q = 4'b1111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 1;

d = 4'b1010;

exp\_q = 4'b1111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 0;

d = 4'b0101;

exp\_q = 4'b0111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

sh\_type = 1;

d = 4'b1010;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh =0;

sh\_type = 0;

ld = 1;

d = 4'b1010;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#1;

$display("Passed / Total: %2d / %2d", tests\_passed, tests\_total);

end

endmodule

module shift\_rgst\_tb\_1(output reg[31:0] tests\_total, tests\_passed);

reg signed [3:0] d;

reg clk, ld, rst\_b, sh, sh\_type;

wire [3:0] act\_q;

reg [3:0] exp\_q;

wire verdict;

shift\_rgst #(.rst\_val(4'd5)) uut (.d(d), .clk(clk), .ld(ld), .rst\_b(rst\_b), .sh(sh), .sh\_type(sh\_type), .q(act\_q));

assign verdict = exp\_q === act\_q;

initial begin

#50;

clk = 0;

repeat(20) #2 clk = ~clk;

end

initial begin

#50;

$display("TESTBENCH 1: Modificat valoarea de reset");

$display("Time\tclk\tld\trst\_b\tsh\tsh\_type\td\t\texpected\_q\tactual\_q\tPassed(1)/Failed(0)");

$monitor("%4t\t%3b\t%2b\t%5b\t%2b\t%7b\t%4b\t\t%10b\t%8b\t%18b", $time, clk, ld, rst\_b, sh, sh\_type, d, exp\_q, act\_q, verdict);

tests\_total = 0;

tests\_passed = 0;

ld = 0;

rst\_b = 1;

sh = 0;

sh\_type = 0;

exp\_q = 4'dX;

#1;

rst\_b = 0;

d = 4'd2;

exp\_q = 4'd5;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#1;

rst\_b = 1;

d = 4'd7;

ld = 1;

exp\_q = 4'd5;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#2;

rst\_b = 1;

d = 4'd7;

ld = 1;

exp\_q = 4'd7;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

ld = 0;

sh = 1;

sh\_type = 0;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

sh\_type = 0;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#2;

sh = 1;

sh\_type = 1;

exp\_q = 4'b0001;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 0;

ld = 1;

d = 4'b1100;

exp\_q = 4'b1100;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

ld = 0;

sh\_type = 1;

d = 4'b0011;

exp\_q = 4'b1110;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

ld = 0;

sh\_type = 0;

d = 4'b1100;

exp\_q = 4'b0111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 0;

ld = 1;

sh\_type = 1;

d = 4'b1111;

exp\_q = 4'b1111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 0;

d = 4'b0000;

exp\_q = 4'b1111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 1;

d = 4'b1010;

exp\_q = 4'b1111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 0;

d = 4'b0101;

exp\_q = 4'b0111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

sh\_type = 1;

d = 4'b1010;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh =0;

sh\_type = 0;

ld = 1;

d = 4'b1010;

exp\_q = 4'b0011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#1;

$display("Passed / Total: %2d / %2d", tests\_passed, tests\_total);

end

endmodule

module shift\_rgst\_tb\_2(output reg[31:0] tests\_total, tests\_passed);

reg signed [7:0] d;

reg clk, ld, rst\_b, sh, sh\_type;

wire [7:0] act\_q;

reg [7:0] exp\_q;

wire verdict;

shift\_rgst #(.w(8)) uut (.d(d), .clk(clk), .ld(ld), .rst\_b(rst\_b), .sh(sh), .sh\_type(sh\_type), .q(act\_q));

assign verdict = exp\_q === act\_q;

initial begin

#100;

clk = 0;

repeat(20) #2 clk = ~clk;

end

initial begin

#100;

$display("TESTBENCH 2: Modificat numarul de biti");

$display("Time\tclk\tld\trst\_b\tsh\tsh\_type\td\t\texpected\_q\tactual\_q\tPassed(1)/Failed(0)");

$monitor("%4t\t%3b\t%2b\t%5b\t%2b\t%7b\t%4b\t\t%10b\t%8b\t%18b", $time, clk, ld, rst\_b, sh, sh\_type, d, exp\_q, act\_q, verdict);

tests\_total = 0;

tests\_passed = 0;

ld = 0;

rst\_b = 1;

sh = 0;

sh\_type = 0;

exp\_q = 8'dX;

#1;

rst\_b = 0;

d = 8'd2;

exp\_q = 8'd0;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#1;

rst\_b = 1;

d = 8'd7;

ld = 1;

exp\_q = 8'd0;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#2;

rst\_b = 1;

d = 8'd7;

ld = 1;

exp\_q = 8'd7;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

ld = 0;

sh = 1;

sh\_type = 0;

exp\_q = 8'b00000011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

sh\_type = 0;

exp\_q = 8'b00000011;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#2;

sh = 1;

sh\_type = 1;

exp\_q = 8'b00000001;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 0;

ld = 1;

d = 8'b11111100;

exp\_q = 8'b11111100;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

ld = 0;

sh\_type = 1;

d = 8'b11000011;

exp\_q = 8'b11111110;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

ld = 0;

sh\_type = 0;

d = 8'b00111100;

exp\_q = 8'b01111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 0;

ld = 1;

sh\_type = 1;

d = 8'b11111111;

exp\_q = 8'b11111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 0;

d = 8'b00000000;

exp\_q = 8'b11111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 1;

d = 8'b10101010;

exp\_q = 8'b11111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh = 1;

ld = 0;

sh\_type = 0;

d = 8'b01010101;

exp\_q = 8'b01111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#3;

sh = 1;

sh\_type = 1;

d = 8'b10101010;

exp\_q = 8'b00111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

sh =0;

sh\_type = 0;

ld = 1;

d = 8'b00001010;

exp\_q = 8'b00111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

#1;

$display("Passed / Total: %2d / %2d", tests\_passed, tests\_total);

end

endmodule

module shift\_rgst\_tb;

wire[31:0] tests\_total\_0, tests\_total\_1, tests\_total\_2, tests\_passed\_0, tests\_passed\_1, tests\_passed\_2;

integer nota;

shift\_rgst\_tb\_0 tb\_0(.tests\_total(tests\_total\_0), .tests\_passed(tests\_passed\_0));

shift\_rgst\_tb\_1 tb\_1(.tests\_total(tests\_total\_1), .tests\_passed(tests\_passed\_1));

shift\_rgst\_tb\_2 tb\_2(.tests\_total(tests\_total\_2), .tests\_passed(tests\_passed\_2));

wire[31:0] tests\_total\_tb;

wire[31:0] tests\_passed\_tb;

assign tests\_total\_tb = tests\_total\_0 + tests\_total\_1 + tests\_total\_2;

assign tests\_passed\_tb = tests\_passed\_0 + tests\_passed\_1 + tests\_passed\_2;

initial begin

#150;

$display("All Passed / All Total: %3d / %3d", tests\_passed\_tb, tests\_total\_tb);

nota = tests\_passed\_tb \* 250 / tests\_total\_tb;

$display("Nota: %1d.%02d", nota / 100, nota % 100);

end

endmodule

4. Proiectati un modul ce citeste bit cu bit pe front-ul crescator al semnalului de clk in maniera FIFO (primul bit primit este cel mai semnificativ bit. Odata Ce A Citit a biti, modulul ii va inregistra in iesirea byte. lesirea byte va ramane pe valoarea precedenta (initial O) cat modulul citeste biti si se va schimba doar cand modsivicititiatios.

biti. Intrarea flush a modulului (activa pe nivel 1) va provoca eliminarea tutoror bitilor cititi pana in momentul respectiv de modul si reluarea citirii a altor 8 biti (jesirea

/\* Proiectati un modul ce citeste bit cu bit pe front-ul crescator al semnalului de clk in maniera FIFO (primul bit primit este cel mai semnificativ bit).

Odata ce a citit 8 biti, modulul ii va inregistra in iesirea byte.

Iesirea byte va ramane pe valoarea precedenta (initial 0) cat modulul citeste biti si se va schimba doar cand modulul a citit alti 8 biti.

Intrarea flush a modulului (activa pe nivel 1) va provoca eliminarea tutoror bitilor cititi pana in momentul respectiv de modul si reluarea citirii a altor 8 biti

(iesirea byte inca isi pastreaza valoarea chiar si in cazul asta). \*/

module spi\_rx(

input bit, clk, flush,

output reg[7:0] byte

);

/\* Write Verilog code here \*/

reg [7:0] valoare=0;

integer contor=7;

initial byte=0;

always@(posedge clk)begin

if (flush==0)begin

if (contor>=0)begin

valoare[contor]=bit;

contor=contor-1;

end

if (contor<0) begin

byte=valoare;

contor=7;

valoare=0;

end

end

else begin

contor=7;

valoare=0;

end

end

endmodule

module spi\_rx\_tb;

reg bit, clk, flush;

wire[7:0] act\_byte;

reg[7:0] exp\_byte;

wire verdict;

spi\_rx uut(.bit(bit), .clk(clk), .flush(flush), .byte(act\_byte));

integer tests\_total, tests\_passed, nota;

assign verdict = exp\_byte === act\_byte;

initial begin

clk = 0;

repeat(80) #1 clk = ~clk;

end

initial begin

$display("Time\tclk\tflush\tbit\t\tactual\_byte\texpected\_byte\tPassed(1)/Failed(0)");

$monitor("%4t\t%3b\t%5b\t%3b\t\t%11b\t%9b\t%18d", $time, clk, flush, bit, act\_byte, exp\_byte, verdict);

tests\_total = 0;

tests\_passed = 0;

exp\_byte = 8'd0;

#1;

bit = 1;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'd0;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b11001010;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

flush = 1;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

flush = 0;

#1;

bit = 1;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#1;

tests\_passed = tests\_passed + verdict;

bit = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 0;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

exp\_byte = 8'b00111111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

bit = 1;

exp\_byte = 8'b10100111;

tests\_total = tests\_total + 1;

#2;

tests\_passed = tests\_passed + verdict;

$display("Passed / Total: %2d / %2d", tests\_passed, tests\_total);

nota = tests\_passed \* 100 / tests\_total \* 25;

$display("Nota: %1d.%03d", nota / 1000, nota % 1000);

end

endmodule