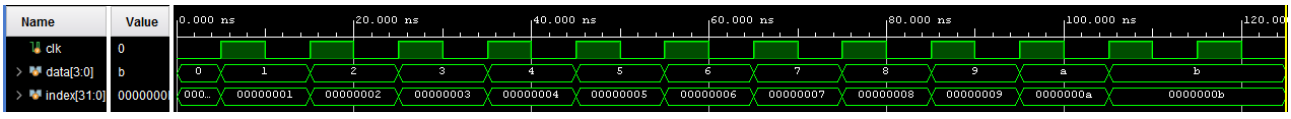
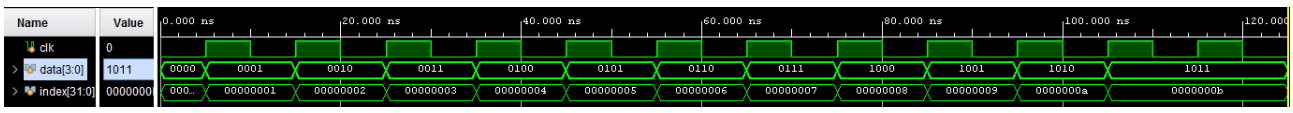
Laborator 1 - 03.03.2022

Rezolvarea exercițiului 2 din:
https://wiki.dcae.pub.ro/index.php/CID_aplicatii_1_:_Generare_de_forme_de_unda

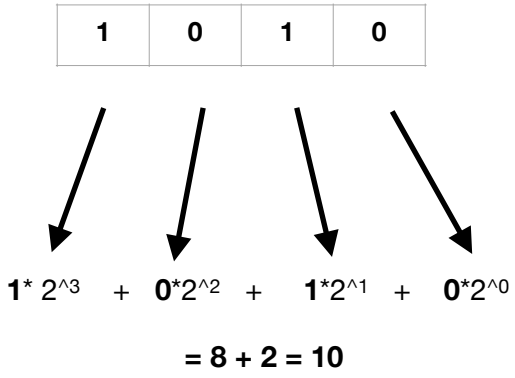
```
`timescale 1ns / 1ps

module data_clock_alligned();    //modul de test, nu avem lista de porturi
                                 //semnalele sunt generate intern

reg clk;          //semnalul de ceas
reg [3:0] data;   //semnalul de 4 biti pe care il vom alinia la frontul ceasului

//generarea semnalului de ceas
initial begin
    data <= 0;

    clk = 0;
    forever #5 clk = ~clk;
end
//

integer index;

//generarea semnalului aliniat la frontul ceasului
initial begin
    for(index = 0; index < 11; index = index + 1) begin //repeta de 10 ori
        @(posedge clk) begin //la fiecare front de ceas
            data <= data + 1; //incrementeaza semnalul data
        end
    end
    #20 $stop();
end
//

endmodule
```



Observați în rezultatul simulării faptul că variabila index are dimensiunea de 32 de biți, corespunzătoare tipului de date *integer*. Variabila *data* are dimensiunea de 4 biți și se incrementează la fiecare front de ceas. În figură formatul pentru reprezentare este hexazecimal. Pentru a-l schimba în format binar: right click pe forma de undă -> radix -> binary cu rezultatul de mai jos:



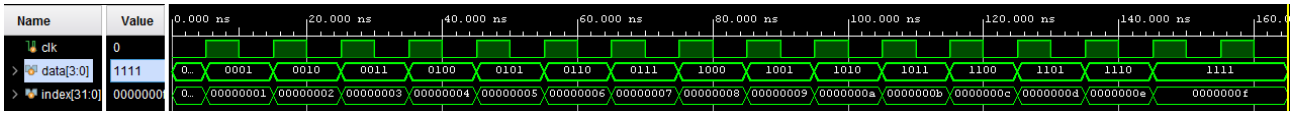
Cel mai mare număr care poate fi scris pe 4 biți este 2^4 - 1 = 15, a cărui reprezentare în binar este 1111. Ne reamintim conversia de la format binar la format zecimal cu exemplul de mai jos:



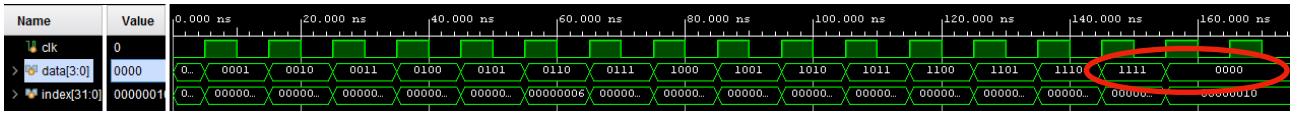
Vom modifica bucla for astfel încât să se încheie dupa 15 iterații:

```
for (index = 0; index < 15; index = index + 1)
```

cu rezultatul din figura de mai jos:



Acum variabila data a ajuns la valoarea maximă care poate fi reprezentată pe 4 biti. Mai facem o modificare astfel încât bucla for să se încheie după 16 iterații, iar rezultatul este prezentat în figura de mai jos:



Acum putem observa că la a 16-a iterație *data* devine 0. Am declarat această variabilă cu dimensiunea de 4 biți, o nouă incrementare ar însemna ca valoarea acesteia să fie **1 0000** (adică 16, reprezentat în binar). Simulatorul ne arată doar cei 4 biți, dimensiunea pe care a fost declarată variabila.

Laborator 2 - 10.03.2022

https://wiki.dcae.pub.ro/index.php/CID_aplicatii_2_:_Instantiere_si_porti_logice
https://wiki.dcae.pub.ro/index.php/Applications_2

1. Adder

Fișierul adder.v este prezentat în figura de mai jos și reprezintă o descrierea comportamentală a unui sumator pe 1 bit.

```
module adder(
    //lista porturilor
    output [1:0] c, //iesirea este pe 2 biti (suma a doua variabile de 1 bit)
    input a,
    input b
    );

//descrierea modulului
assign c = a + b;

endmodule
```

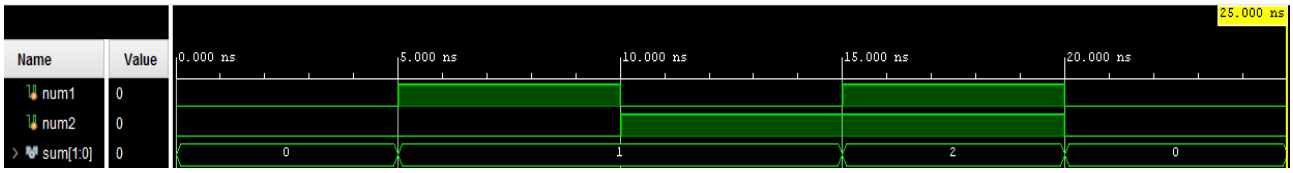
Pentru verificarea acestuia vom scrie un modul de test cu denumirea adder_tb (**tb** - test bench) prezentat și în figura de mai jos.

```
module adder_tb(); //modul de test, nu are interfata, semnalele sunt generate intern

reg num1;
reg num2;
wire [1:0] sum;

adder adder1(  //instantierea modulului adder (numele instantei = adder1)
    .a(num1), //portul_modulului(nume_fir_la_care_se_leaga_in_testbench)
    .b(num2),
    .c(sum)
    );

//generarea stimulilor
initial begin
    num1 = 0;
    num2 = 0;
 #5 num1 = 1;
 #5 num1 = 0;
    num2 = 1;
 #5 num1 = 1;
 #5 num1 = 0;
    num2 = 0;
 #5 $stop();
end

endmodule
```

Instanțierea modulului *adder* în modulul de test a fost realizată cu sintaxa:

```
nume_modul nume_instanta_a_modulului (
      .semnal_interfata_instanta(semnalul_la_care_va_fi_conectat)
      )
```

Rezultatul simulării este prezentat în figura de mai jos.

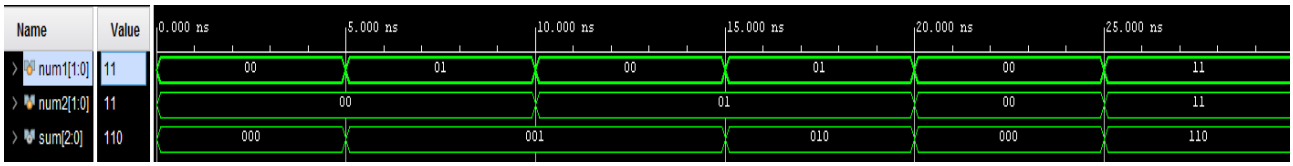


Să modificăm variabilele astfel încât să aibă dimenisunea de 2 biți. Suma celor două numere va fi pe 3 biți. Se vor face modificările necesare în fișierele *adder.v* și *adder_tb.v* pentru dimensiunea variabilelor.

```
`timescale 1ns / 1ps

module adder(
    //lista porturilor
    output [2:0] c,
    input  [1:0] a,
    input  [1:0] b
    );

//descrierea modulului
assign c = a + b;

endmodule
```

```
`timescale 1ns / 1ps
module adder_tb();

reg  [1:0] num1;
reg  [1:0] num2;
wire [2:0] sum;

adder adder1(
    .a(num1),
    .b(num2),
    .c(sum)
    );

//generarea stimulilor
initial begin
    num1 = 0;
    num2 = 0;
 #5 num1 = 1;
 #5 num1 = 0;
    num2 = 1;
 #5 num1 = 1;
 #5 num1 = 0;
    num2 = 0;
 #5 num1 = 3; //se poate scrie si 2'b11
    num2 = 3;
 #5 $stop();
end
```

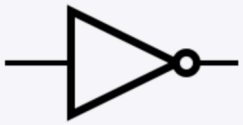
Rezultatul simulării este prezentat în figura de mai jos.



2. Poarta NAND formată din poartă AND și poartă NOT

AND

INTRARE		IEȘIRE
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Tip	Simbol	Tabel de adevăr
NOT		

INTRARE	IEȘIRE
A	NOT A
0	1
1	0

INTRARE		IEȘIRE
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Ieșirea porții NAND este **0** atunci când ambele intrări sunt **1**.
Pentru a descrie în limbajul Verilog poarta NAND putem folosi mai multe metode. Pentru început vom construi aceasta poartă dintr-o poarta AND și un inversor (NOT). Acest exercițiu servește la înțelegerea instanțierii modulelor si conectarea lor intr-un alt modul.

```
`timescale 1ns / 1ps

module and_gate(
    output out_and,
    input in1_and,
    input in2_and
    );

assign out_and = in1_and & in2_and;

endmodule
```

Modulul care descrie comportamental poarta AND

```
`timescale 1ns/1ps

module not_gate(
    output out_not,
    input  in_not
    );

assign out_not = ~in_not;

endmodule
```

Modulul care descrie comportamental poarta NOT

În continuare, aceste porți vor fi instanțiate într-un alt modul, numit *nand_gate* și se va face conexiunea între acestea (ieșirea modulului *and_gate* este intrare pentru modulul *not_gate*). În urma instanțierii vom observa cum este creată o ierarhie în care *nand_gate* este nivelul cel mai de sus (top level).

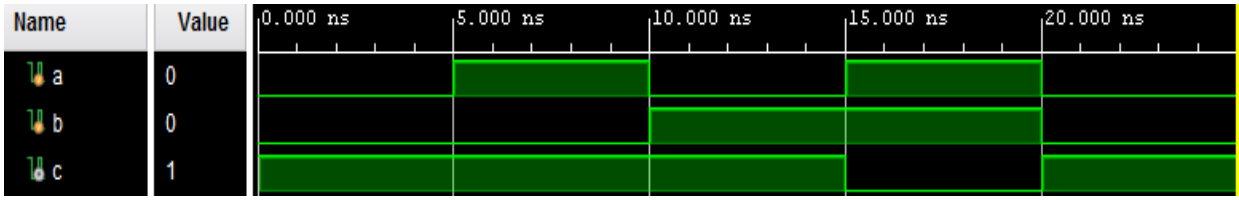
În figurile de mai jos este prezentat fișierul nand_gate.v împreună cu fișierul de test pentru acest modul și rezultatul simulării.

```
`timescale 1ns/1ps

module nand_gate(
    output out_nand,
    input in1_nand,
    input in2_nand
    );

and_gate and_gate1(
    .out_and(w0),
    .in1_and(in1_nand),
    .in2_and(in2_nand)
    );

not_gate not_gate1(
    .out_not(out_nand),
    .in_not(w0)
    );

endmodule
```

Modulul care descrie structural poarta NAND

```
`timescale 1ns / 1ps

module nand_gate_tb();

reg a,b;
wire c;

nand_gate nand_gate1(
    .out_nand(c),
    .in1_nand(a),
    .in2_nand(b)
    );

initial begin
    a = 0;
    b = 0;
 #5 a = 1;
 #5 a = 0;
    b = 1;
 #5 a = 1;
 #5 a = 0;
    b = 0;
 #5 $stop();

end
endmodule
```

Modulul de test



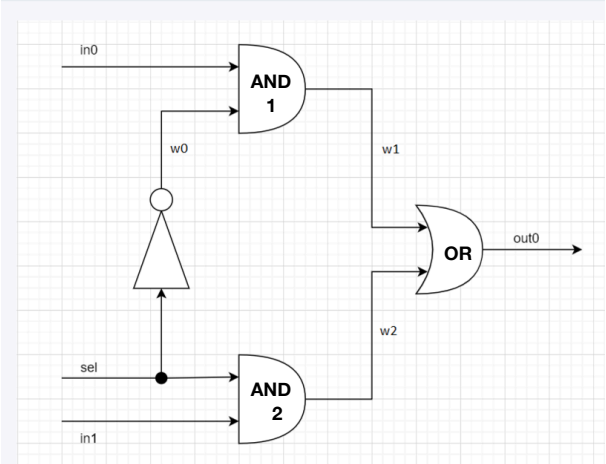
Observăm ca ieșirea, semnalul *c*, este **0** numai atunci când cele două intrări sunt **1**, așadar este îndeplinită funcția NAND.

3. Multiplexor

Circuitul din figură implementează în hardware funcția de selecție a unei intrări.

- Ieșirea unei porți **AND** este **1** atunci când **ambele** intrări sunt **1**.
- Ieșirea unei porți **OR** este **1** atunci când **oricare** dintre intrări este **1**.

Cu aceste observații putem construi tabelul de adevăr pentru multiplexorul din imagine.



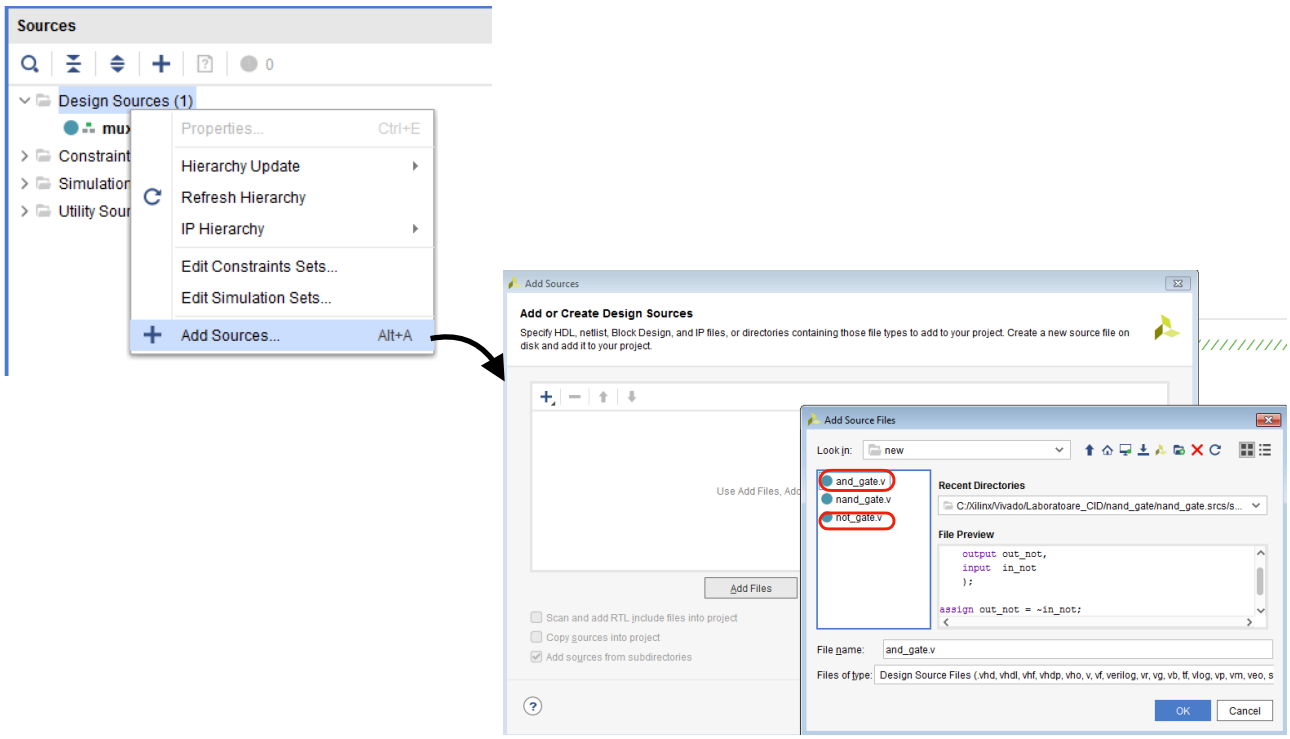
Tabelul de adevăr

in1	in0	sel	w0	w1	w2	out0
0	0	0	1	0	0	0
0	1	0	1	1	0	1
1	0	0	1	0	0	0
1	1	0	1	1	0	1
0	0	1	0	0	0	0
0	1	1	0	0	0	0
1	0	1	0	0	1	1
1	1	1	0	0	1	1

Vom descrie acest circuit prin mai multe metode. Pentru început vom utiliza o descriere structurală. Pentru porțile AND și NOT ne putem folosi de modulele scrise anterior. Vom crea un modul de top numit *mux.*

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////
3
4  module mux(
5      output out0,
6      input in0,
7      input in1,
8      input sel
9      );
10
11 //descrierea circuitului
12
13 //
14
15 endmodule
```

Ca să adăugăm fișiere din alte proiecte urmărim procedura din figurile de mai jos. Calea către fișierele sursă este *nume_proiect/nume_proiect.srcs/sources_1/new/.* În cazul de față adaugăm fișierele *and_gate.v* și *not_gate.v* din proiectul *nand_gate.*

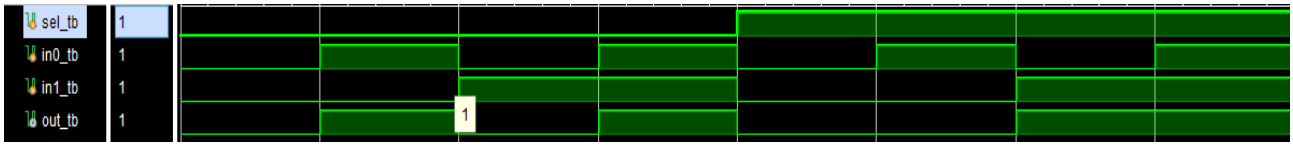


Creăm și un modul care să descrie poarta logică OR. (assign out_or = in1_or | in2_or) și instanțiem toate aceste porți în modulul de top, pentru care scriem și un modul de test.

```
`timescale 1ns / 1ps

module mux(
    output out0,
    input in0,
    input in1,
    input sel
    );

wire w0,w1,w2;

not_gate not_gate1(
    .in_not(sel),
    .out_not(w0)
    );

and_gate and_gate1(
    .in1_and(in0),
    .in2_and(w0),
    .out_and(w1)
    );

and_gate and_gate2(
    .in1_and(sel),
    .in2_and(in1),
    .out_and(w2)
    );

or_gate or_gate1(
    .in1_or(w1),
    .in2_or(w2),
    .out_or(out0)
    );

endmodule
```

```
`timescale 1ns / 1ps

module mux_tb();

reg in0_tb, in1_tb, sel_tb;
wire out_tb;

mux_behavioural mux1(
    .in0(in0_tb),
    .in1(in1_tb),
    .sel(sel_tb),
    .out0(out_tb)
    );

initial begin
    sel_tb = 0;
    in0_tb = 0;
    in1_tb = 0;
 #5 in0_tb = 1;
 #5 in0_tb = 0;
    in1_tb = 1;
 #5 in0_tb = 1;
 #5 sel_tb = 1;
    in0_tb = 0;
    in1_tb = 0;
 #5 in0_tb = 1;
 #5 in0_tb = 0;
    in1_tb = 1;
 #5 in0_tb = 1;
 #5 $stop();
end

endmodule
```

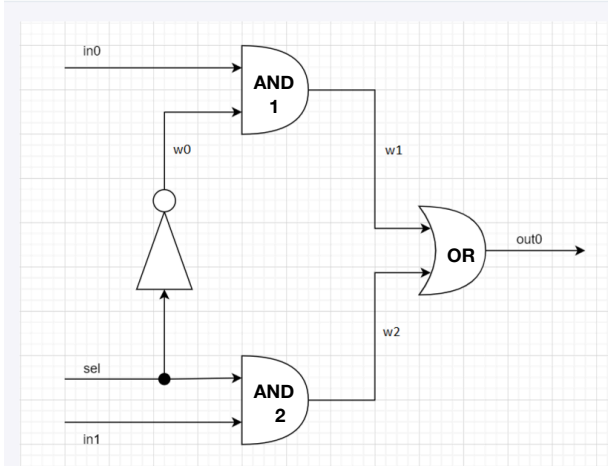
În figura de mai jos este prezentat rezultatul simulării. Observăm că atunci când *sel_tb = 0*, ieșirea *out_tb* ia valoarea *in0_tb*, iar atunci când *sel_tb = 1* ieșirea ia valoarea *in1_tb* conform cu tabelul de adevăr prezentat mai sus.



O altă descrierea structurală a multiplexorului poate fi realizată cu ajutorul unor primitive (porți logice care există deja descrise în limbaj). In instanțierea acestora, ieșirea se pune prima, urmata de intrări.

```
`timescale 1ns / 1ps

module mux_primitive(
    output out0,
    input in0,
    input in1,
    input sel
    );

and and_1(w1, in0, w0);
and and_2(w2, sel, in1);
not not_1(w0, sel);
or or_1(out0, w1, w2);

endmodule
```

Pentru a descrie comportamental modulul este necesară exprimarea ieșirii ca funcție de intrări. Pentru asta, abordarea constă în a porni de la ieșire si a merge pas cu pas către intrări, asa cum este exemplificat mai jos



pas1: out0 = ?
pas2: out0 = w1 | w2
pas3: out0 = w1 | (?)
pas4: out0 = w1 | (sel & in1)
pas5: out0 = (?) | (sel & in1)
pas6: out0 = (in0 & w0) | (sel & in1)
pas7: out0 = (in0 & (?)) | (sel & in1)
pas8: out0 = (in0 & (~sel)) | (sel & in1)

Obținem astfel modulul din figura de mai jos

```
`timescale 1ns / 1ps

module mux_behavioural(
    output out0,
    input in0,
    input in1,
    input sel
    );

assign out0 = (in0 & (~sel)) | (in1 & sel);

endmodule
```

Utilizând același modul de test vom putea observa că toate aceste forme de descriere conduc la același rezultat (prezentat anterior).