UNIVERSIDADE FEDERAL DA PARAÍBA
DEPARTAMENTO DE INFORMÁTICA
CONCEPÇÃO ESTRUTURADA
PROF: DR. ANTONIO CARLOS CAVALCANTI

SOMADOR ACUMULADOR

Edimar Bezerra da Silva Neto
Matricula 11409565

João Pessoa / 2018

1. Somador / Subtrator de 4bits com acumulador

O circuito consiste em uma entrada de 4 bits, que irá receber instruções que definirão o seu comportamento. Sendo subdividido em blocos:

- **Inversor :** que irá inverter a entrada.
- **2 Mux :** Um mux para escolher entre a entrada e a entrada invertida sendo controlado por uma entrada sel 0, e o segundo mux irá decidir quem irá para a saída e para o acumulador.
- **Somador :** executará a soma do número que sai do primeiro mux, o acumulador e o cin. A junção dessas instruções.
- **Acumulador :** Dependendo de um clock, ele irá acumular o valor da entrada.
  O circuito possui o seguinte comportamento lógico:

	SEL 0	SEL 1	Saida/ACC
Acumulador	0	0	A
Somador	0	1	A + ACC
Inversor	1	0	!A
Subtrator	1	1	!A + ACC

2. Golden Model em C

Iniciamos com o Modelo de Referência de Ouro, que trata-se de um modelo de referência em alto nível de um dado circuito capaz de descrever seu comportamento, utilizaremos linguagem C e produziremos um arquivo de teste .tv.

A seguir algumas funções utilizadas e seus vetores de saída:

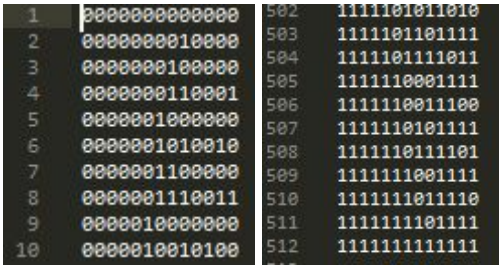
Inversor

```c
int inversor(int a){
    return a^15;
}
```

```
1  00001111
2  00011110
3  00101101
4  00111100
5  01001011
6  01011010
7  01101001
8  01111000
9  10000111
10 10010110
11 10100101
12 10110100
13 11000011
14 11010010
15 11100001
16 11110000
```

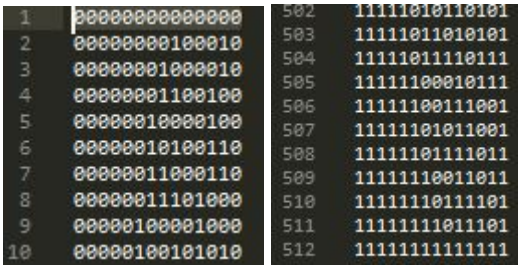
Mux

```
int mux2(int a ,int b,int sel){
    if (sel==0)
    {
        return a;
    }
    if (sel==1)
    {
        return b;
    }

    }
```



Somador

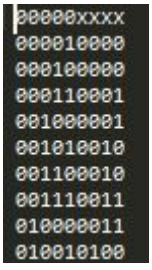
```
int sum (int a, int b, int cin){
  return (a+b+cin) & 0xF;
}
```



Acumulador

Inicializamos a saída com um x já que ele só vai receber o primeiro valor no clock= 1.

```
int acc(int a ,int clk){
    static int q =0;

    if (clk ==1){
        q = a;
        return q;
    }
    else{

        return q;
    }
}
```



Somador acumulador

Foi utilizados todos os blocos acima para o golden model, e utilizamos 2 arquivos de vetores, 1 com 128 casos, e outro vetor de testes criado para talvez uma melhor visualização:

```
000000000000
100000000000
000111111110
100111111111
010000011111
110000011111
010111100001
110111100000
001000000000
101000000000
001111111110
101111111101
011000011111
111000011111
011111100001
111111100010
```

3. Implementação no Quartus

A seguir trabalharemos no quartus com linguagem system verilog, mostrando cada arquivo module .sv, posteriormente mostrando seu RTL viewer.

Criaremos também seu arquivo testbench que testaremos os vetores gerados pelo nosso golden model, melhor visualizado nas simulações RTL e Gate Level utilizando modelsim. RTL é um nível de abstração que descreve o comportamento do circuito ou dispositivo (modelo comportamental) baseado no fluxo de sinais ou transferência de dados, o Gate Level que descreve a representação booleana real do circuito ou dispositivo. No código será contabilizado os casos de erros, fazendo comparações com as saídas esperadas no vetor e as saídas geradas na simulação.

Inversor

```
1  module inv(a , y);
2      input logic [3:0] a;
3      output logic [3:0] y;
4      assign y = ~a;
5  endmodule
```

RTL Viewer

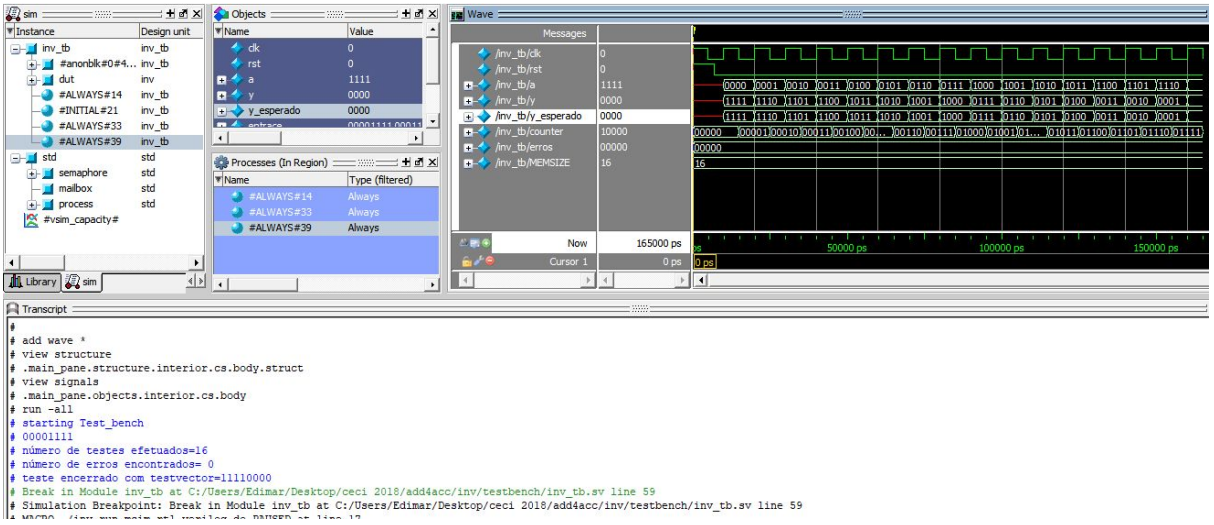
TestBench

```
`timescale 1ns / 100ps module inv_tb;

    logic clk, rst;
    logic [3:0] a, y, y_esperado;
    logic [7:0] entrace [0:15];
    logic [0:4] counter,erros;

const int MEMSIZE = 16;

        inv dut(
        .a(a),
        .y(y)
        );
    always begin
        clk = 1;
        #5;
        clk = 0;
        #5;
    end

    initial
      begin
      counter = 0;
      erros=0;
        $display("starting Test_bench");
        $readmemb("inv.tv",entrace);
        $display("%b",entrace[counter]);
        rst = 1;
        #7;
        rst = 0;
        end

 always @(posedge clk)
 if (~rst)
 begin // skip during reset
         a[3:0]= entrace[counter][7:4];
         y_esperado[3:0] = entrace[counter][3:0];
     end
```

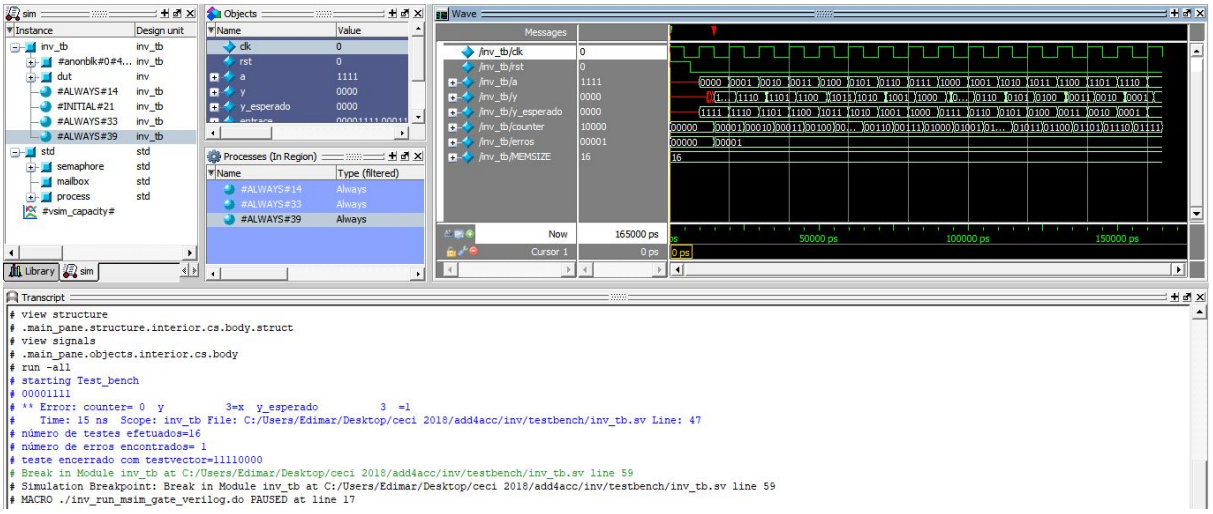
Trecho de Código para contar o numero de erros e de casos simulados.

```
always @(negedge clk)
if (~rst)
begin // skip during reset
        //assert (y === entrace[counter][0]) else $error("y");
        for(int k = 0; k < 4; k++)
        begin
  assert (y[k]=== y_esperado[k]) else
    begin //erros=erros+1;
          $error("counter=%d",counter,"  y%d",k,"=%b",y[k],"  y_esperado%d",k,"  =%b",y_esperado[k]);
          erros++;
    end;
  end
    counter = counter + 1;
    if (counter == (MEMSIZE) )
        begin
                        $display("número de testes efetuados=%d",counter);
                        $display("número de erros encontrados=%d",erros);
                        $display("teste encerrado com testvector=%b",entrace[counter-1]);


            $stop;
        end

    end


endmodule
```

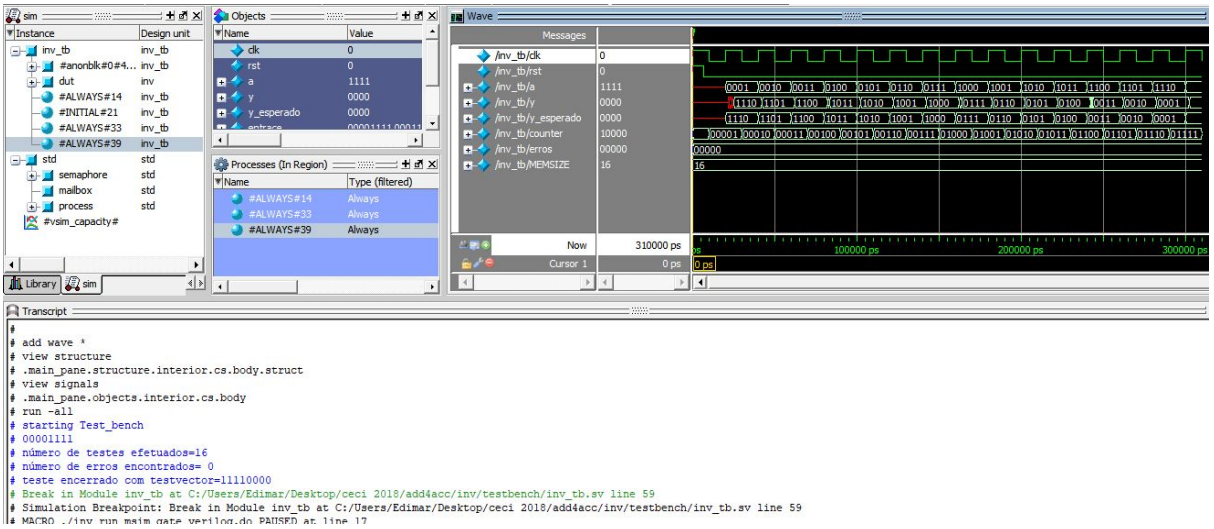
Simulação RTL Level



Simulação Gate Level



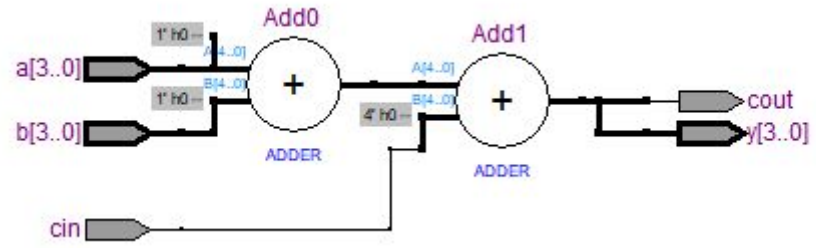
Utilizamos clk #10 para diminuir os erros apresentados nas simulações.



Somador

```
module sum(input [3:0] a,input [3:0] b,input cin, output [3:0] y,output cout);

assign {cout,y[3:0]}= a+b+cin;

endmodule
```

RTL Viewer

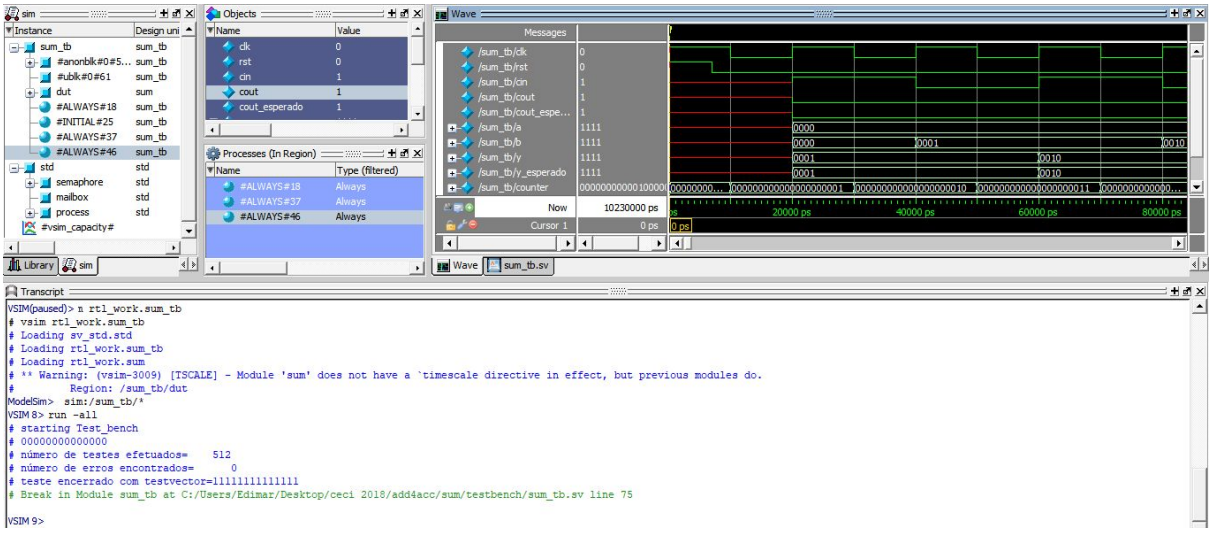


Test bench

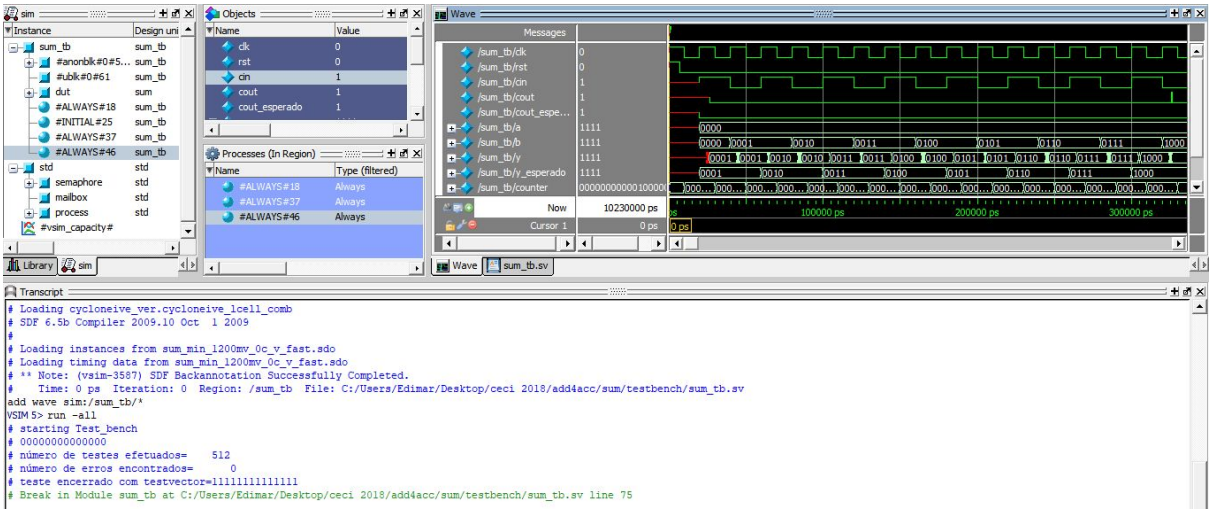
```
`timescale 1ns / 100ps module sum_tb;

    logic clk, rst, cin, cout, cout_esperado;
    logic [3:0] a, b, y, y_esperado;
    logic [13:0] entrace [0:511];

    logic [0:20] counter,erros;

const int MEMSIZE = 512;

        sum dut(
        .a(a),
        .b(b),
        .cin(cin),
        .y(y),
        .cout(cout)
        );
    always begin
        clk = 1;
        #10;
        clk = 0;
        #10;
    end


 initial
   begin
   counter = 0;
   erros=0;
     $display("starting Test_bench");
     $readmemb("sum.tv",entrace);
     $display("%b",entrace[counter]);
     rst = 1;
     #7;
     rst = 0;
     end

 always @(posedge clk)
 if (~rst)
 begin // skip during reset
        a[3:0]= entrace[counter][13:10];
        b[3:0]= entrace[counter][9:6];
        cin = entrace[counter][5];
        y_esperado[3:0]= entrace[counter][4:1];
        cout_esperado = entrace[counter][0];
     end
```

```
    always @(negedge clk)
    if (~rst)
    begin // skip during reset
            //assert (y === entrace[counter][0]) else $error("y");
            for(int k = 0; k < 4; k++)
            begin
              assert (y[k]=== y_esperado[k]) else
                begin //erros=erros+1;
                      $error("counter=%d",counter,"  y%d",k,"=%b",y[k],"  y_esperado%d",k,"  =%b",y_esperado[k]);
                      erros++;
                end;
            end
            begin
              assert (cout=== cout_esperado) else
                begin //erros=erros+1;
                      $error("counter=%d",counter,"=%b",cout,"  =%b",cout_esperado);
                      erros++;
                end;
            end
        counter = counter + 1;
        if (counter == (MEMSIZE) )
            begin
                        $display("número de testes efetuados=%d",counter);
                        $display("número de erros encontrados=%d",erros);
                        $display("teste encerrado com testvector=%b",entrace[counter-1]);
            $stop;
            end
        end
endmodule
```

Simulação RTL Level



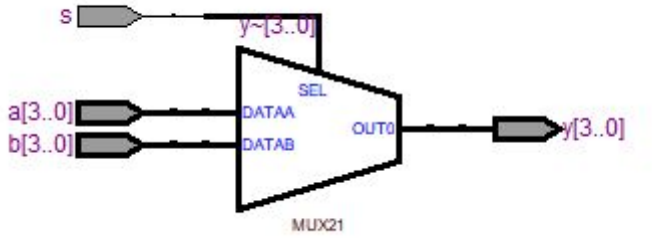
Simulação Gate Level



Mux

```
module mux2(a, b, s, y);
    input logic [3:0] a, b;
    output logic [3:0] y;
    input logic s;
    assign y = s ? b : a;
endmodule
```

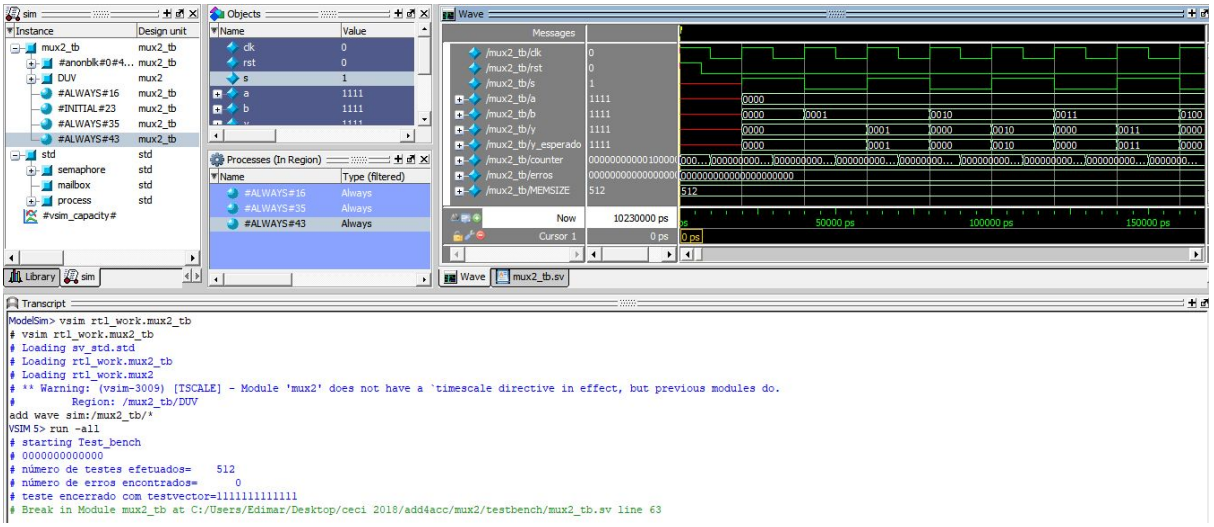
RTL Viewer



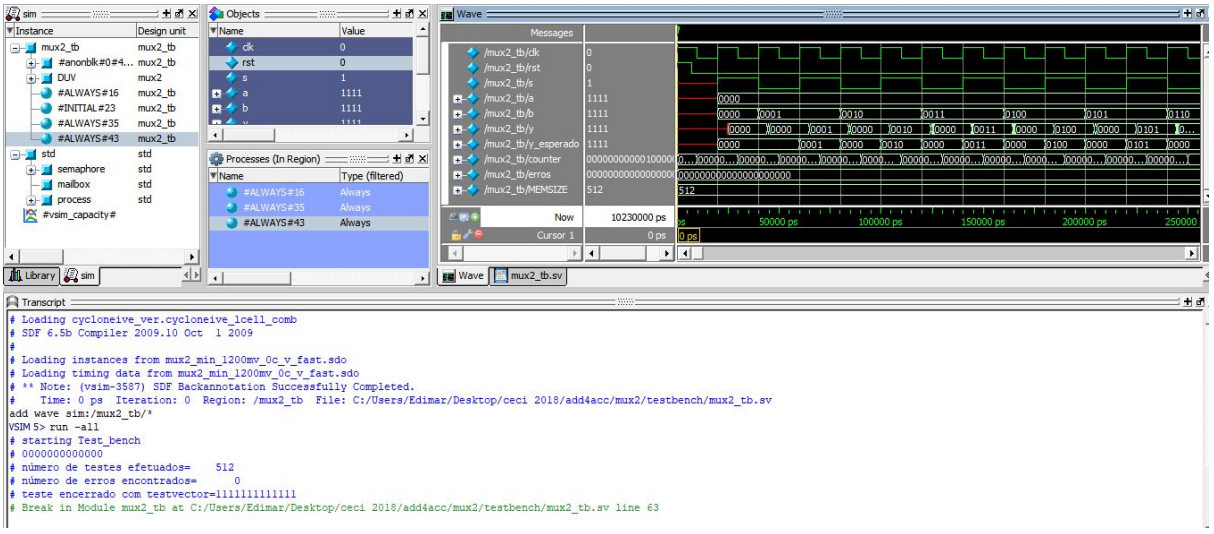
TestBench

```
`timescale 1ns / 100ps module mux2_tb;

    logic clk, rst;
    logic s;
    logic [3:0] a, b, y, y_esperado;
    logic [12:0] entrace [0:511];
    logic [0:20] counter,erros;

const int MEMSIZE = 512;
        mux2 DUV(
        .a(a),
        .b(b),
        .s(s),
        .y(y)
        );
    always begin
        clk = 1;
        #10;
        clk = 0;
        #10;
    end

   initial
     begin
     counter = 0;
     erros=0;
       $display("starting Test_bench");
       $readmemb("mux2.tv",entrace);
       $display("%b",entrace[counter]);
       rst = 1;
       #7;
       rst = 0;
       end

   always @(posedge clk)
   if (~rst)
   begin // skip during reset
           a[3:0]= entrace[counter][12:9];
           b[3:0]= entrace[counter][8:5];
           s = entrace[counter][4];
           y_esperado[3:0] = entrace[counter][3:0];
       end
```

Simulação RTL Level



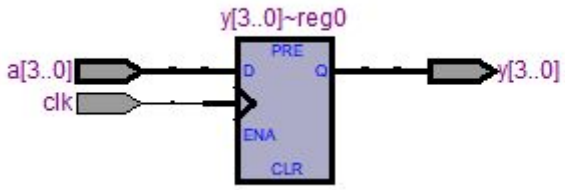
Simulação Gate Level



Acumulador

```
module acc(input logic [3:0] a, input logic clk, output logic [3:0] y);

always @(posedge clk)
 y <= a;
endmodule
```

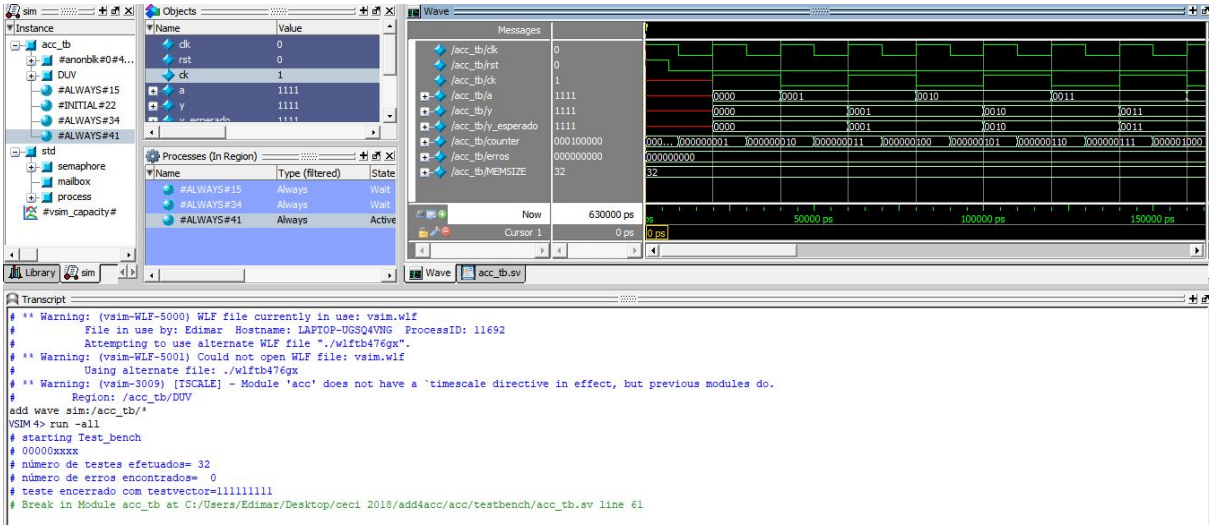
RTL VIEWER



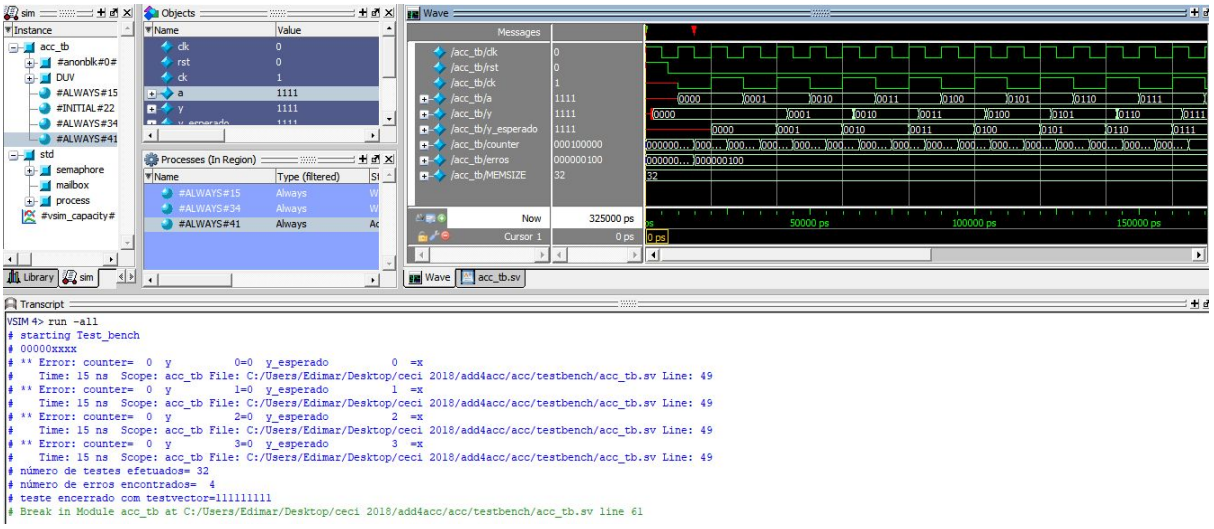
TestBench

```
`timescale 1ns / 100ps module acc_tb;

    logic clk, rst, ck;
    logic [3:0] a, y, y_esperado;
    logic [8:0] entrace [0:31];
    logic [0:8] counter,erros;

const int MEMSIZE = 32;

        acc DUV(
        .clk(ck),
        .a(a),
        .y(y)
        );
    always begin
        clk = 1;
        #10;
        clk = 0;
        #10;
    end

initial
  begin
  counter = 0;
  erros=0;
    $display("starting Test_bench");
    $readmemb("acc.tv",entrace);
    $display("%b",entrace[counter]);
    rst = 1;
    #7;
    rst = 0;
    end

always @(posedge clk)
if (~rst)
begin // skip during reset
        a[3:0]= entrace[counter][8:5];
        ck = entrace[counter][4];
        y_esperado[3:0] = entrace[counter][3:0];
    end
```

SIMULAÇÃO RTL



SIMULAÇÃO GATE LEVEL

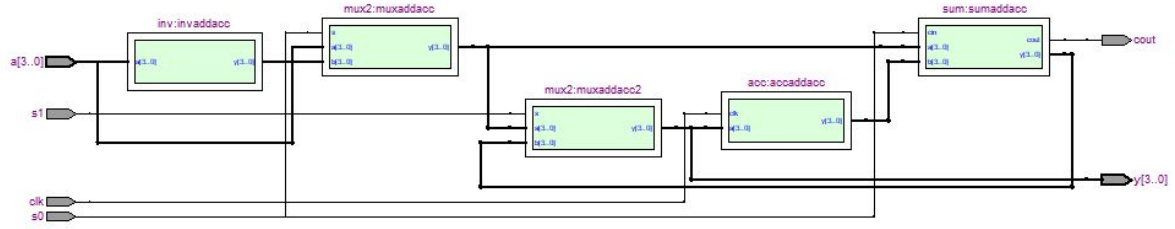


Somador Acumulador

Agora juntamos todos os blocos feitos anteriormente(inv, mux, sum , acc) e criamos um novo estrutural em verilog e um novo testbench para o nosso vetor de testes.

```
module addacc(input logic [3:0] a,
input logic clk,
input logic s0,s1,
output logic cout,
output logic [3:0] y);

logic [3:0] ainv, mux1, sum1, acc1;
    inv invaddacc(a, ainv);
    mux2 muxaddacc(a, ainv, s0, mux1);
    acc accaddacc(y, clk, acc1);
    sum sumaddacc(mux1, acc1, s0, sum1, cout);
    mux2 muxaddacc2(mux1, sum1, s1, y);

endmodule
```

RTL Viewer

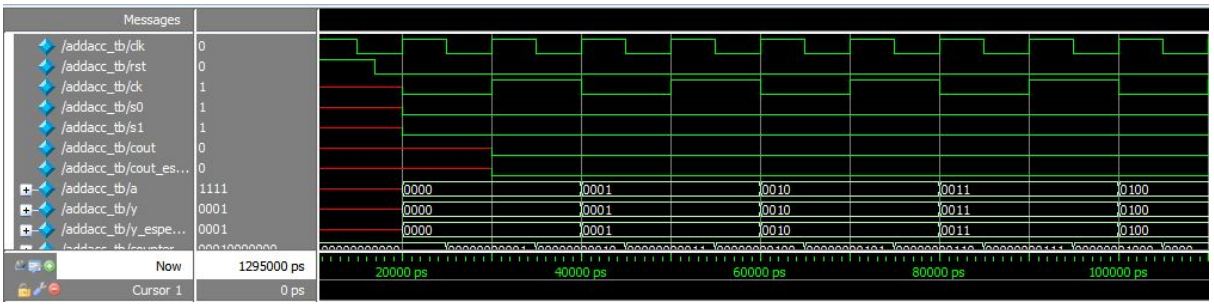


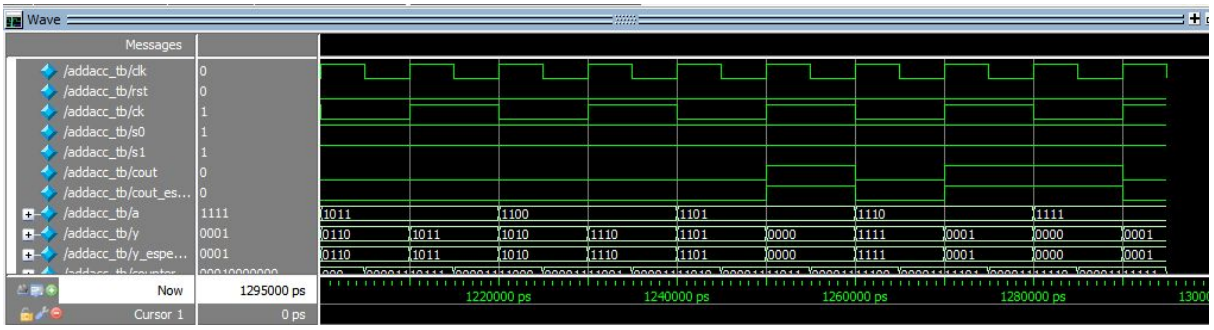
Test Bench

```
`timescale 1ns / 100ps module addacc_tb;

    logic clk, rst, ck;
    logic s0, s1, cout, cout_esperado;
    logic [3:0] a, y, y_esperado;
    logic [11:0] entrace [0:127];
    logic [0:10] counter,erros;

const int MEMSIZE = 128;

        addacc DUV(
        .a(a),
        .y(y),
        .clk(ck),
        .s0(s0),
        .s1(s1),
        .cout(cout)
        );
    always begin
        clk = 1;
        #10;
        clk = 0;
        #10;
    end
```

```
initial
  begin
  counter = 0;
  erros=0;
    $display("starting Test_bench");
    $readmemb("addacc.tv",entrace);
    $display("%b",entrace[counter]);
    rst = 1;
    #17;
    rst = 0;
    end

always @(posedge clk)
if (~rst)
begin // skip during reset
        ck = entrace[counter][11];
        s0 = entrace[counter][10];
        s1 = entrace[counter][9];
        a[3:0]= entrace[counter][8:5];
        y_esperado[3:0] = entrace[counter][4:1];
        cout_esperado = entrace[counter][0];
    end
```

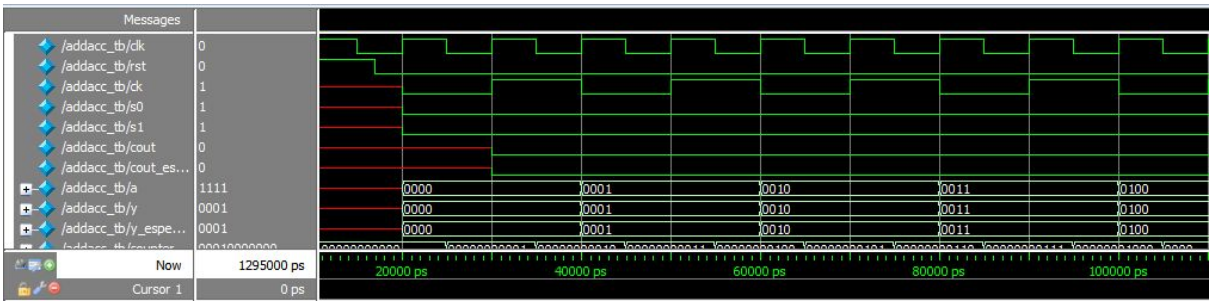
Simulação RTL



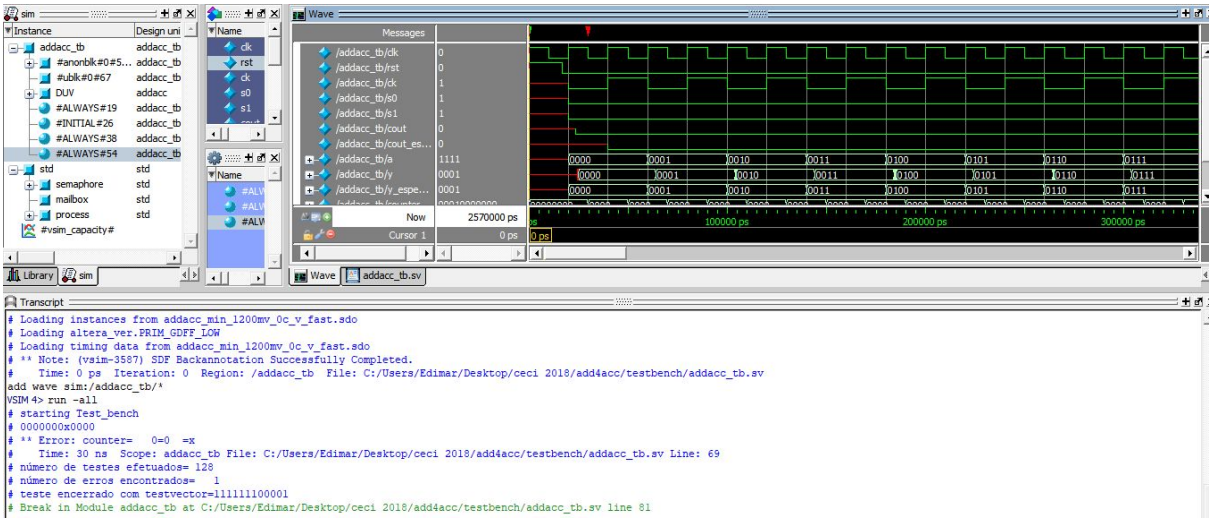


```
#         Region: /addacc_tb/DUV/muxaddacc
# Loading rtl_work.acc
# ** Warning: (vsim-3009) [TSCALE] - Module 'acc' does not have a `timescale directive in effect, but previous modules do.
#         Region: /addacc_tb/DUV/accaddacc
# Loading rtl_work.sum
# ** Warning: (vsim-3009) [TSCALE] - Module 'sum' does not have a `timescale directive in effect, but previous modules do.
#         Region: /addacc_tb/DUV/sumaddacc
add wave sim:/addacc_tb/*
VSIM 7> run -all
# starting Test_bench
# 0000000x0000
# número de testes efetuados= 128
# número de erros encontrados=   0
# teste encerrado com testvector=111111100001
# Break in Module addacc_tb at C:/Users/Edimar/Desktop/ceci 2018/add4acc/testbench/addacc_tb.sv line 81

VSIM 8>
```

Simulação Gate Level

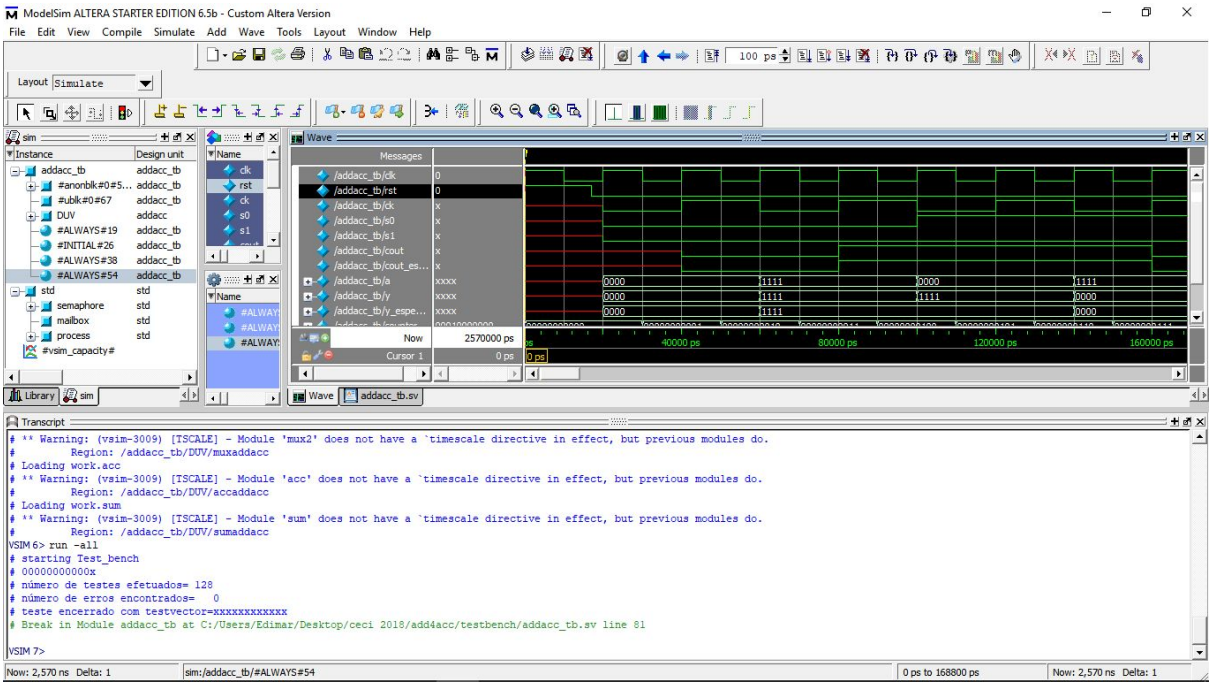


com clk #10



2º vetor de teste

Simulação RTL



Simulação Gate Level