

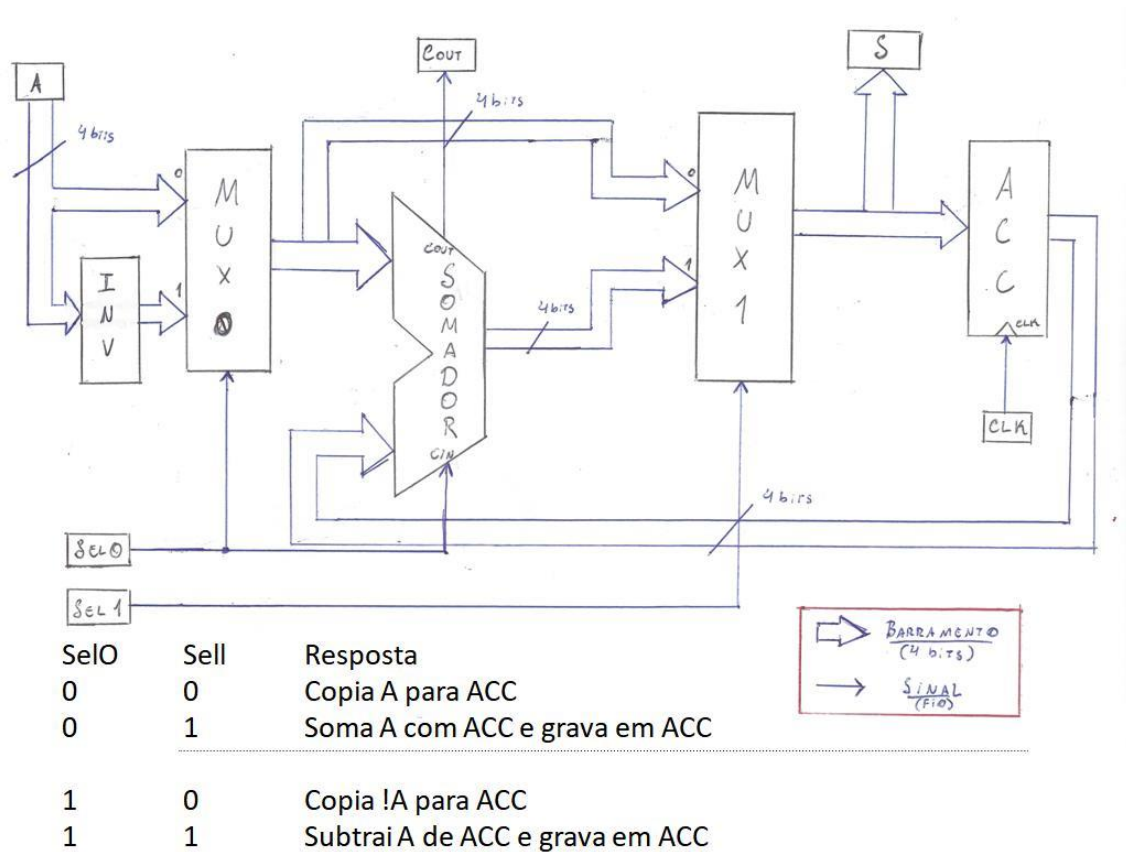
Universidade Federal da Paraíba

Drayton Corrêa Filho

**Relatório da 4ª Avaliação**  
Projeto no Quartus II

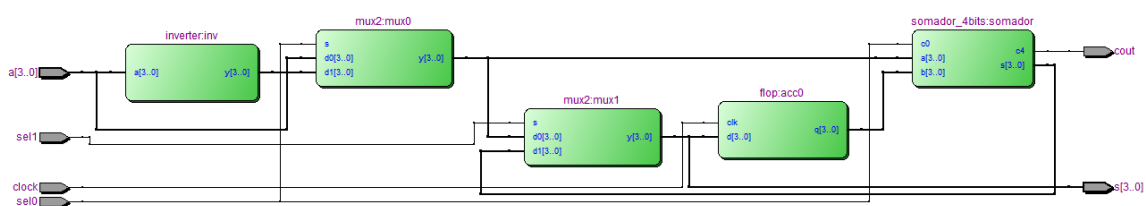
João Pessoa - Paraíba  
2017

Como último projeto o professor requisitou que fosse feito um operador matemático que poderia somar, subtrair, inverter e acumular um determinado valor. Tal conjunto consistia de dois *multiplexers*, um inversor, um somador de 4 bits e um *flip-flop* que serviria como acumulador. A disposição que deveria ser organizada tais componentes foi fornecida através da imagem a seguir.



Para executar isso no *Quartus* foi necessário primeiramente buscar os arquivos *VHDL* de antigos exemplos os quais serviriam como *components* dessa aplicação e coloca-los no diretório em que seria posto o *VHDL* de mais alto nível desse projeto.

Feito isso, criei o *top-level* nomeando-o de *add\_sub\_inv\_acc* e declarei no interior de sua *architecture* os *components* para logo após chamá-los de tal forma a que fosse construído uma composição parecida à do desenho acima, utilizando, é claro, de sinais auxiliares que serviram de forma similar a fios ligando os blocos, o código do *top-level* é mostrado na próxima página e a sua visualização *Register-Transfer Level* (feita indo em *Tools*, *Netlist Viewers* e *RTL Viewer*) é exibida abaixo.



```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity add_sub_inv_acc is
5  port(a: in STD_LOGIC_VECTOR(3 downto 0);
6       sel0, sel1, clock: in STD_LOGIC;
7       cout: out STD_LOGIC;
8       s: out STD_LOGIC_VECTOR(3 downto 0));
9  end;
10
11 architecture struct of add_sub_inv_acc is
12 component somador_4bits
13 port(c0: in STD_LOGIC;
14      a, b: in STD_LOGIC_VECTOR(3 downto 0);
15      pg, gg, c4: out STD_LOGIC;
16      s: out STD_LOGIC_VECTOR(3 downto 0));
17 end component;
18
19 component mux2
20 port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
21      s: in STD_LOGIC;
22      y: out STD_LOGIC_VECTOR(3 downto 0));
23 end component;
24
25 component inverter
26 port(a: in STD_LOGIC_VECTOR(3 downto 0);
27      y: out STD_LOGIC_VECTOR(3 downto 0));
28 end component;
29
30 component flop
31 port(clk: in STD_LOGIC;
32      d: in STD_LOGIC_VECTOR(3 downto 0);
33      q: out STD_LOGIC_VECTOR(3 downto 0));
34 end component;
35
36 signal ainv, amux0, amux1, yacc: STD_LOGIC_VECTOR(3 downto 0);
37 signal soma: STD_LOGIC_VECTOR(3 downto 0);
38 signal pg, gg: STD_LOGIC;
39
40 begin
41     inv: inverter port map(a, ainv);
42     mux0: mux2 port map(a, ainv, sel0, amux0);
43     mux1: mux2 port map(amux0, soma, sel1, amux1);
44     somador: somador_4bits port map(sel0, amux0, yacc, pg, gg, cout, soma);
45     acc0: flop port map(clock, amux1, yacc);
46     s <= amux1;
47 end;

```

Em seguida, foi o momento de criar os vetores que seriam usados para testar o código acima e, para isso, eu confeccionei um programa em C que basicamente executava da mesma forma que o *add\_sub\_inv\_acc*, mas com uma sequência de entrada preestabelecida.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  int main (void){
6      int pg, gg;
7      int sel0, sel1, i;
8      int c0, bit0, bit1, bit2, bit3;
9      int a[4], b[4], acc[4], s[4];
10     int ainv[4];
11     int clock = 1;
12     int clk = 0;
13     int contador = 1;
14
15     for(sel0 = 0; sel0 <= 1; sel0++){
16         for(sel1 = 0; sel1 <= 1; sel1++){
17             for(bit3 = 0; bit3 <= 1; bit3++){
18                 for(bit2 = 0; bit2 <= 1; bit2++){
19                     for(bit1 = 0; bit1 <= 1; bit1++){
20                         for(bit0 = 0; bit0 <= 1; bit0++){
21                             int cout;
22
23                             if(clock == 0){
24                                 clk = 0;
25                             }
26
27                             if(clock == 1){
28                                 clk = 1;
29                             }
30
31                             // Exatidão do a de 4 bits

```

Feito isso, executei-o e copiei os vetores gerados da *prompt* para um arquivo *example.tv* que foi posto no diretório *simulation/modelsim* dentro da pasta do projeto. Um pequeno fragmento desses vetores é exibido ao lado.

Em seguida, foi o momento de criar o *test bench* que aceitaria esses vetores e, para isso, eu copiei um de exemplos passados e o adaptei, um detalhe importante é que, assim como o *flop* e o *flopenr*, o *add\_sub\_inv\_acc* precisou de um *clock* auxiliar dentro do *testbench* pois em seu interior foi usado um *flip-flop*, já que esse circuito tem uma memória e os valores posteriores de cada linha de teste dependem dos anteriores guardados no acumulador.

```

1  0_0_0000_0000_0_1
2  0_0_0001_0001_0_0
3  0_0_0010_0010_0_1
4  0_0_0011_0011_0_0
5  0_0_0100_0100_0_1
6  0_0_0101_0101_0_0
7  0_0_0110_0110_0_1
8  0_0_0111_0111_0_0
9  0_0_1000_1000_0_1
10 0_0_1001_1001_1_0
11 0_0_1010_1010_1_1
12 0_0_1011_1011_1_0
13 0_0_1100_1100_1_1
14 0_0_1101_1101_1_0
15 0_0_1110_1110_1_1
16 0_0_1111_1111_1_0
17 0_1_0000_1111_0_1
18 0_1_0001_0000_1_0
19 0_1_0010_0010_0_1
20 0_1_0011_0011_0_0
21 0_1_0100_0111_0_1
22 0_1_0101_1000_0_0
23 0_1_0110_1110_0_1
24 0_1_0111_1111_0_0
25 0_1_1000_0111_1_1
26 0_1_1001_1000_1_0
27 0_1_1010_0010_1_1
28 0_1_1011_0011_1_0
29 0_1_1100_1111_0_1
30 0_1_1101_0000_1_0

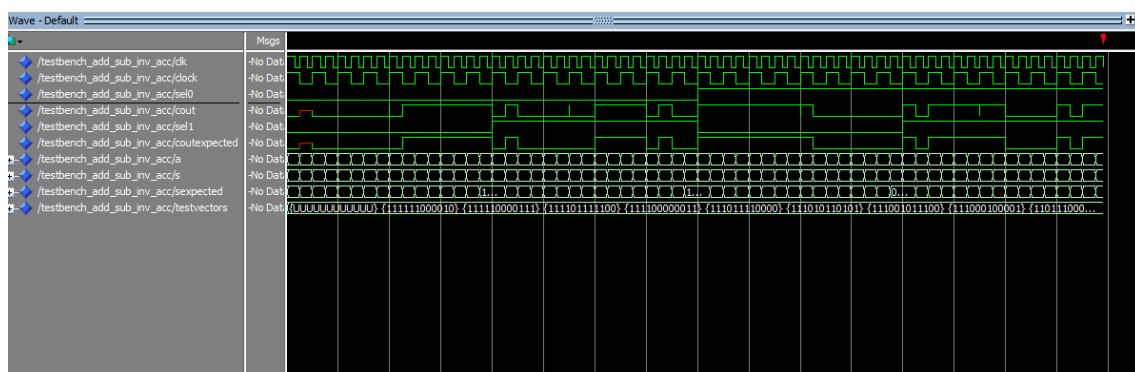
```

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  use IEEE.STD_LOGIC_unsigned.ALL;
5  use STD.TEXTIO.ALL ;
6
7  entity testbench_add_sub_inv_acc is
8      -- no inputs or outputs
9  end;
10
11 architecture test of testbench_add_sub_inv_acc is
12     component add_sub_inv_acc
13     port(a: in STD_LOGIC_VECTOR(3 downto 0);
14         sel0, sel1, clock: in STD_LOGIC;
15         cout: out STD_LOGIC;
16         s: out STD_LOGIC_VECTOR(3 downto 0));
17     end component;
18
19     signal clk, clock: STD_LOGIC;
20     signal sel0, sel1, cout, coutexpected: STD_LOGIC;
21     signal a, s, sexpected: STD_LOGIC_VECTOR(3 downto 0);
22     constant MEMSIZE: integer := 64;
23     type tarray is array (MEMSIZE downto 0) of STD_LOGIC_VECTOR (11 downto 0);
24     signal testvectors: tarray;
25     shared variable vectornum, errors: integer;
26
27     begin
28         -- instantiate device under test
29         dut: add_sub_inv_acc port map (a, sel0, sel1, clock, cout, s);
30         -- generate clock
31     process begin
32         clk <= '1'; wait for 15 ns;
33         clk <= '0'; wait for 10 ns;
34     end process;
35
36     -- generate test vectors
37     process
38     begin
39         vectornum := 0;
40         while vectornum < MEMSIZE
41         loop
42             a <= testvectors(vectornum);
43             sel0 <= '0';
44             sel1 <= '0';
45             clock <= '1';
46             wait for 15 ns;
47             clock <= '0';
48             wait for 10 ns;
49             coutexpected <= '0';
50             sexpected <= a;
51             vectornum := vectornum + 1;
52         end loop;
53     end process;
54
55     -- generate test results
56     process
57     begin
58         while vectornum < MEMSIZE
59         loop
60             cout <= coutexpected;
61             s <= sexpected;
62             vectornum := vectornum + 1;
63         end loop;
64     end process;
65
66     -- generate test report
67     process
68     begin
69         errors := 0;
70         while vectornum < MEMSIZE
71         loop
72             if cout <= coutexpected and s <= sexpected
73             then
74                 errors := errors + 1;
75             end if;
76             vectornum := vectornum + 1;
77         end loop;
78         report "Test Results: " & errors & " errors out of " & MEMSIZE & " tests.";
79     end process;
80
81 end architecture test;

```

A próxima etapa consistiu de efetivamente testar o projeto rodando a simulação *RTL* indo em *Tools, Run EDA Simulation Tool* e *EDA RTL Simulation*.



Em seguida, eu selecionei os *signals* *s*, *sexpected*, *sel0*, *sel1*, *cout*, *coutexpected*, *clock* e *a* e os exibi no formato de lista abaixo.

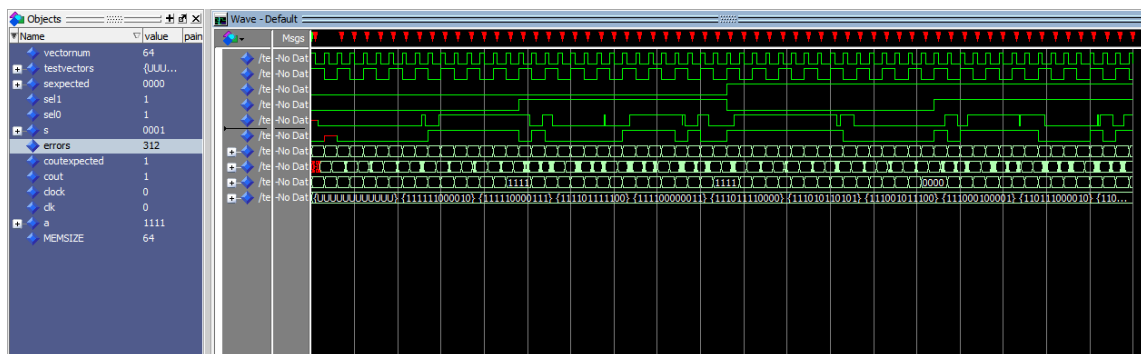
Time	/testbench_add_sub_inv_acc/sexpected	/testbench_add_sub_inv_acc/cout	/testbench_add_sub_inv_acc/sel1	/testbench_add_sub_inv_acc/clock	/testbench_add_sub_inv_acc/sel0	/testbench_add_sub_inv_acc/s	/testbench_add_sub_inv_acc/coutexpected
0 +0	UUUU	U	UUUU	U	U	UUUU	U
0 +2	0000	0	0000	0	U	1 0000	U
0 +5	0000	0	0000	0	U	1 0000	U
0 +9	0000	0	0000	0	0	1 0000	0
25000 +2	0001	0	0000	U	0	0 0001	0
25000 +5	0001	0	0001	U	0	0 0001	0
25000 +9	0001	0	0001	U	U	0 0001	U
50000 +2	0010	0	0001	0	U	1 0010	U
50000 +5	0010	0	0010	0	U	1 0010	U
50000 +6	0010	0	0010	0	0	1 0010	0
75000 +2	0011	0	0010	0	0	0 0011	0
75000 +5	0011	0	0011	0	0	0 0011	0
100000 +2	0100	0	0011	0	0	1 0100	0
100000 +5	0100	0	0100	0	0	1 0100	0
125000 +2	0101	0	0100	0	0	0 0101	0
125000 +5	0101	0	0101	0	0	0 0101	0
150000 +2	0110	0	0101	0	0	1 0110	0
150000 +5	0110	0	0110	0	0	1 0110	0
175000 +2	0111	0	0110	0	0	0 0111	0
175000 +5	0111	0	0111	0	0	0 0111	0
200000 +2	1000	0	0111	0	0	1 1000	0
200000 +5	1000	0	1000	0	0	1 1000	0
200000 +6	1000	0	1000	0	1	1 1000	1
200000 +9	1000	0	1000	0	0	1 1000	0
225000 +2	1001	0	1000	1	0	0 1001	0
225000 +5	1001	0	1001	1	0	0 1001	0
225000 +9	1001	0	1001	1	1	0 1001	1

Como é possível observar, a simulação *RTL* não gerou quaisquer erros, então foi possível seguir para o próximo passo, que foi a simulação *gate level*. Mas antes, como sempre feito anteriormente, eu fui no *testbench* do projeto e mudei o tempo de espera de subida do *clk* de 15 ns para 5 ns. Após isso, rodei a simulação.

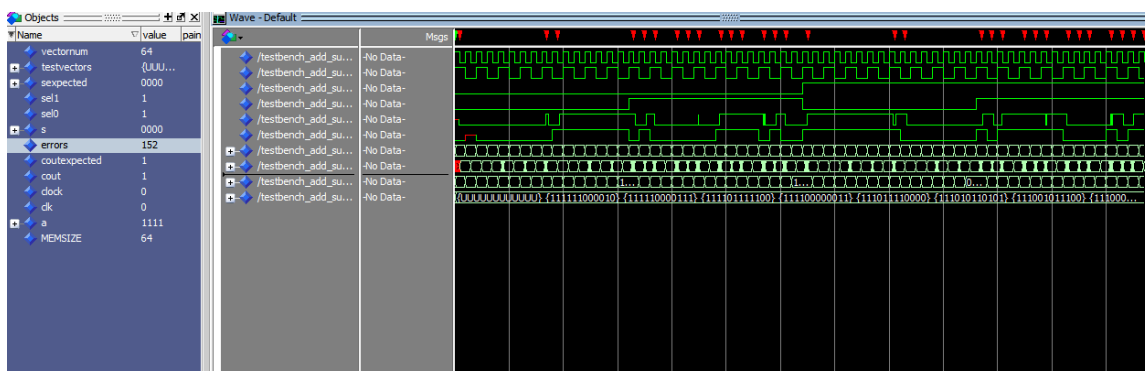
```

31 process begin
32     clk <= '1'; wait for 5 ns;
33     clk <= '0'; wait for 10 ns;
34 end process;
35 -- at start of test, load vectors

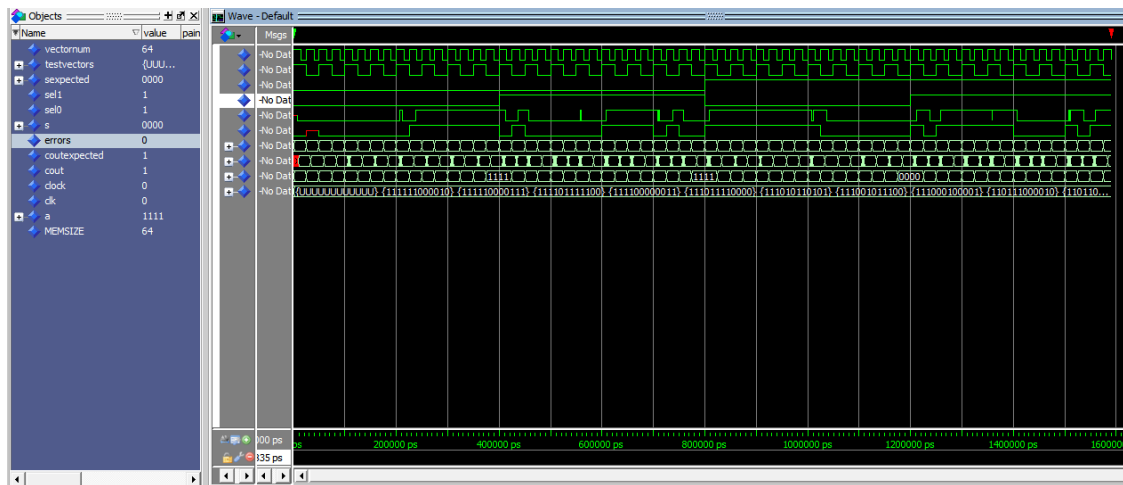
```



Com isso foram gerados 312 erros, logo eu subi o tempo de espera para 10 ns, o que produziu o seguinte resultado.



O número de erros decaiu para 152, mas não chegou a 0, então foi preciso subir o tempo de espera para 15 ns.



O que finalmente fez o número de erros chegar em 0, sendo possível concluir que 15 ns é o tempo de espera mínimo para que não sejam gerados erros.