Universidade Federal da Paraíba

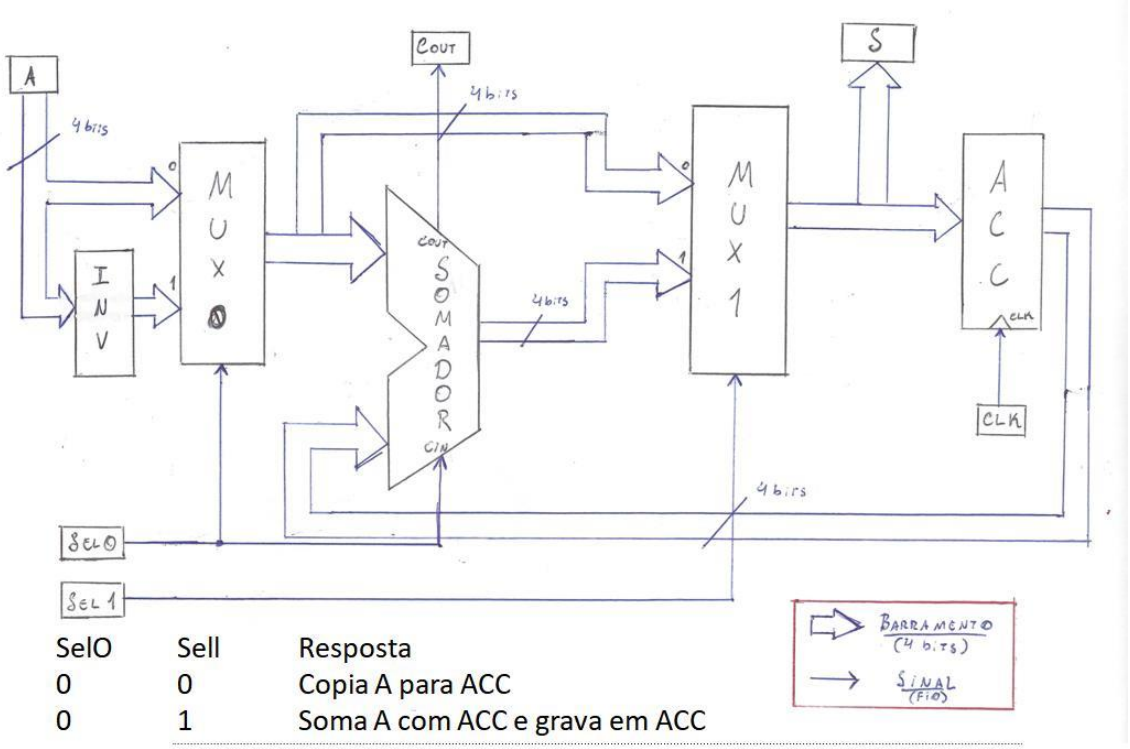
Drayton Corrêa Filho

Relatório da 4ª Avaliação

Projeto no Quartus II

João Pessoa - Paraíba
2017

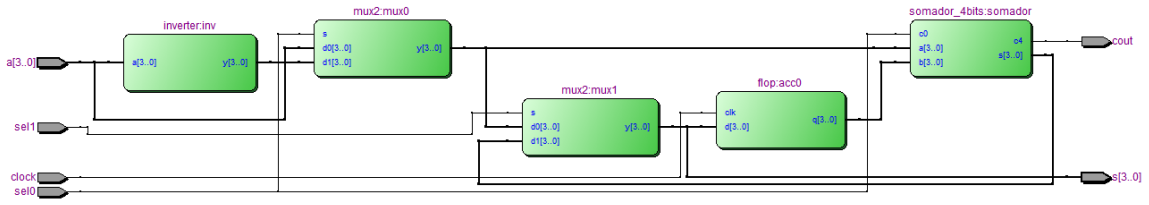
Como último projeto o professor requisitou que fosse feito um operador matemático que poderia somar, subtrair, inverter e acumular um determinado valor. Tal conjunto consistia de dois *multiplexers,* um inversor, um somador de 4 bits e um *flip-flop* que serviria como acumulador. A disposição que deveria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



SelO	Sell	Resposta
0	0	Copia A para ACC
0	1	Soma A com ACC e grava em ACC
1	0	Copia !A para ACC
1	1	Subtrai A de ACC e grava em ACC

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.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity add_sub_inv_acc is
   port(a: in STD_LOGIC_VECTOR(3 downto 0);
        sel0, sel1, clock: in STD_LOGIC;
        cout: out STD_LOGIC;
        s: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture struct of add_sub_inv_acc is
component somador_4bits
   port(c0: in STD_LOGIC;
        a, b: in STD_LOGIC_VECTOR(3 downto 0);
        pg, gg, c4: out STD_LOGIC;
        s: out STD_LOGIC_VECTOR(3 downto 0));
end component;

component mux2
   port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
        s: in STD_LOGIC;
        y: out STD_LOGIC_VECTOR(3 downto 0));
end component;

component inverter
   port(a: in  STD_LOGIC_VECTOR(3 downto 0);
        y: out STD_LOGIC_VECTOR(3 downto 0));
end component;

component flop
   port(clk: in  STD_LOGIC;
        d: in  STD_LOGIC_VECTOR(3 downto 0);
        q: out STD_LOGIC_VECTOR(3 downto 0));
end component;

signal ainv, amux0, amux1, yacc: STD_LOGIC_VECTOR(3 downto 0);
signal soma: STD_LOGIC_VECTOR(3 downto 0);
signal pg, gg: STD_LOGIC;

begin
   inv:  inverter port map(a, ainv);
   mux0: mux2 port map(a, ainv, sel0, amux0);
   mux1: mux2 port map(amux0, soma, sel1, amux1);
   somador: somador_4bits port map(sel0, amux0, yacc, pg, gg, cout, soma);
   acc0: flop port map(clock, amux1, yacc);
   s <= amux1;
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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main (void){
    int pg, gg;
    int sel0, sel1, i;
    int c0, bit0, bit1, bit2 , bit3;
    int a[4], b[4], acc[4], s[4];
    int ainv[4];
    int clock = 1;
    int clk = 0;
    int contador = 1;

    for(sel0 = 0; sel0 <= 1; sel0++){
        for(sel1 = 0; sel1 <= 1; sel1++){
            for(bit3 = 0; bit3 <= 1; bit3++){
                for(bit2 = 0; bit2 <= 1; bit2++){
                    for(bit1 = 0; bit1 <= 1; bit1++){
                        for(bit0 = 0; bit0 <= 1; bit0++){
                            int cout;

                            if(clock == 0){
                                clk = 0;
                            }

                            if(clock == 1){
                                clk = 1;
                            }

                            // Entrada do a de 4 bits
```

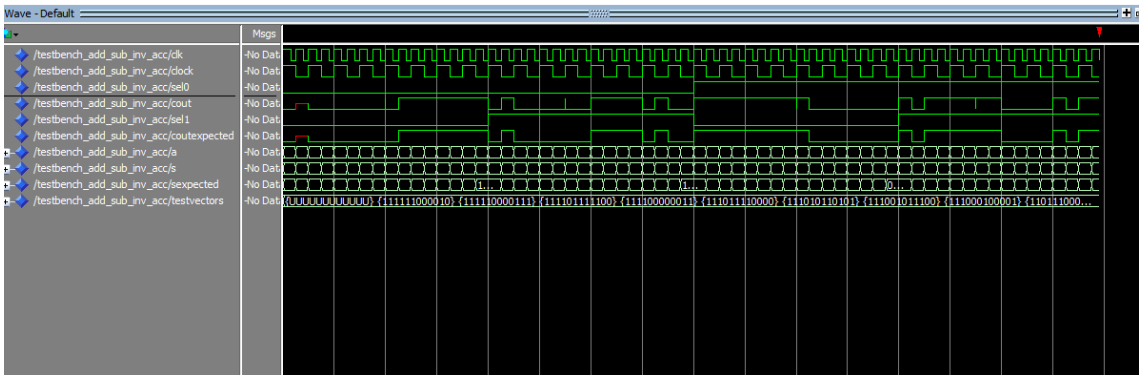
Feito isso, executei-o e copiei os vetores gerados da *prompt* para um arquivo *example.tv* que foi posto no diretório *simulation/modelsim* dentro da pasta do projeto. Um pequeno fragmento desses vetores é exibido ao lado.

```
0_0_0000_0000_0_1
0_0_0001_0001_U_0
0_0_0010_0010_0_1
0_0_0011_0011_0_0
0_0_0100_0100_0_1
0_0_0101_0101_0_0
0_0_0110_0110_0_1
0_0_0111_0111_0_0
0_0_1000_1000_0_1
0_0_1001_1001_1_0
0_0_1010_1010_1_1
0_0_1011_1011_1_0
0_0_1100_1100_1_1
0_0_1101_1101_1_0
0_0_1110_1110_1_1
0_0_1111_1111_1_0
0_1_0000_1111_0_1
0_1_0001_0000_1_0
0_1_0010_0010_0_1
0_1_0011_0011_0_0
0_1_0100_0111_0_1
0_1_0101_1000_0_0
0_1_0110_1110_0_1
0_1_0111_1111_0_0
0_1_1000_0111_1_1
0_1_1001_1000_1_0
0_1_1010_0010_1_1
0_1_1011_0011_1_0
0_1_1100_1111_0_1
0_1_1101_0000_1_0
```

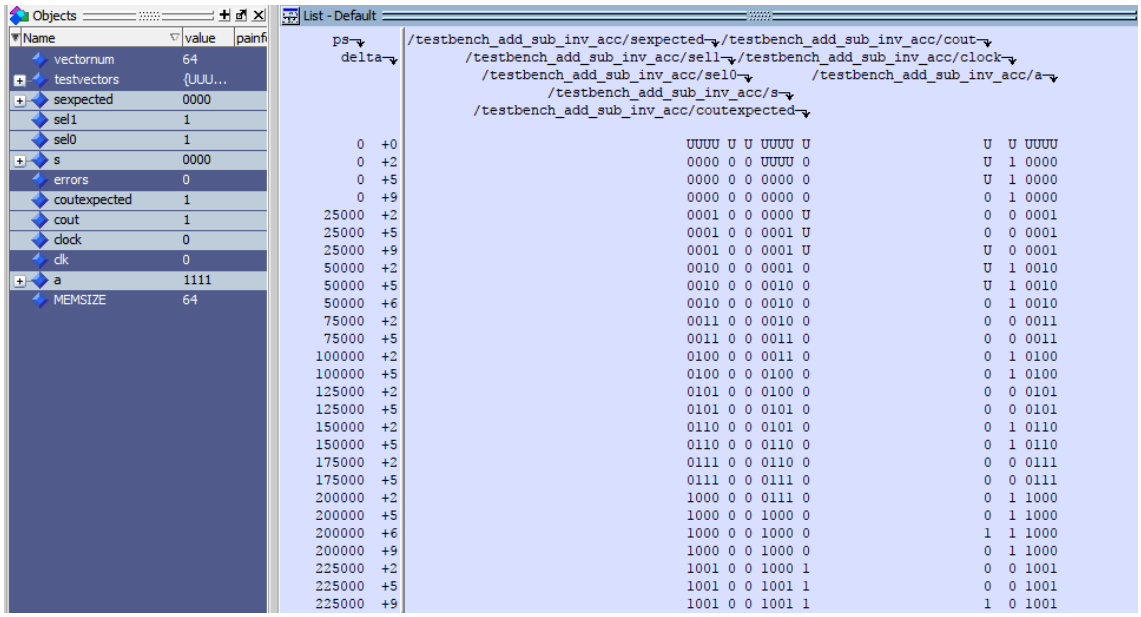
Em seguida, foi o momento de criar o *test bench* que aceitaria esses vetores e, para isso, eu copiei um de exemplos passados e o adaptei, um detalhe importante é que, assim como o *flop* e o *flopenr,* o *add_sub_inv_acc* precisou de um *clock* auxiliar dentro do *testbench* pois em seu interior foi 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.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use STD.TEXTIO.ALL ;

entity testbench_add_sub_inv_acc is
    -- no inputs or outputs
end;

architecture test of testbench_add_sub_inv_acc is
    component add_sub_inv_acc
    port(a: in STD_LOGIC_VECTOR(3 downto 0);
         sel0, sell, clock: in STD_LOGIC;
         cout: out STD_LOGIC;
         s: out STD_LOGIC_VECTOR(3 downto 0));
    end component;

signal clk, clock: STD_LOGIC;
signal sel0, sell, cout, coutexpected: STD_LOGIC;
signal a, s, sexpected: STD_LOGIC_VECTOR(3 downto 0);
constant MEMSIZE: integer := 64;
type tvarray is array (MEMSIZE downto 0) of STD_LOGIC_VECTOR (11 downto 0);
signal testvectors: tvarray;
shared variable vectornum, errors: integer;

begin
-- instantiate device under test
dut: add_sub_inv_acc port map (a, sel0, sell, clock, cout, s);
-- generate clock
process begin
    clk <= '1'; wait for 15 ns;
    clk <= '0'; wait for 10 ns;
```

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

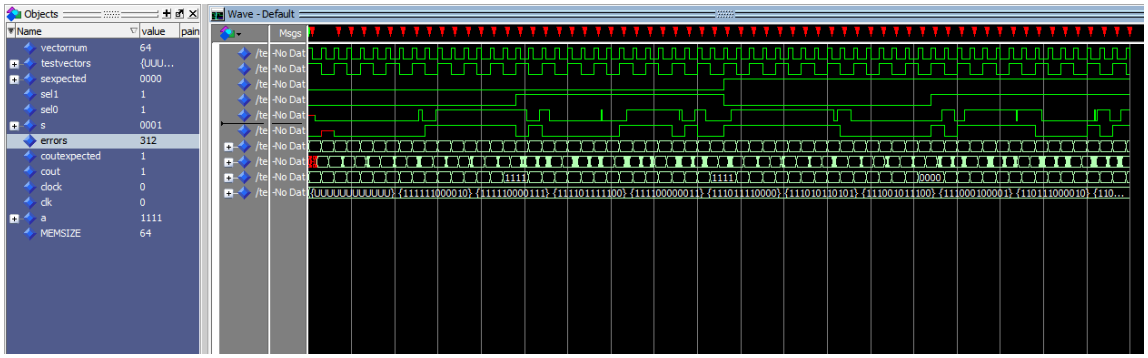


Em seguida, eu selecionei os *signals s, sexpected, sel0, sel1, cout, coutexpected, clock* e *a* e os exibi no formato de lista abaixo.

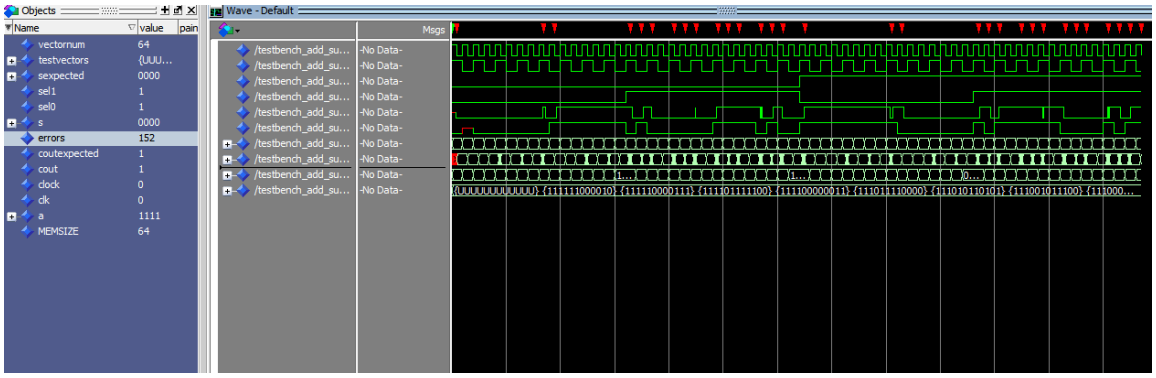


Como é possível observar, a simulação *RTL* não gerou quaisquer erros, então foi possível seguir para o próximo passo, que foi a simulação *gate level*. Mas antes, como sempre feito anteriormente, eu fui no *testbench* do projeto e mudei o tempo de espera de subida do *clk* de 15 ns para 5 ns. Após isso,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

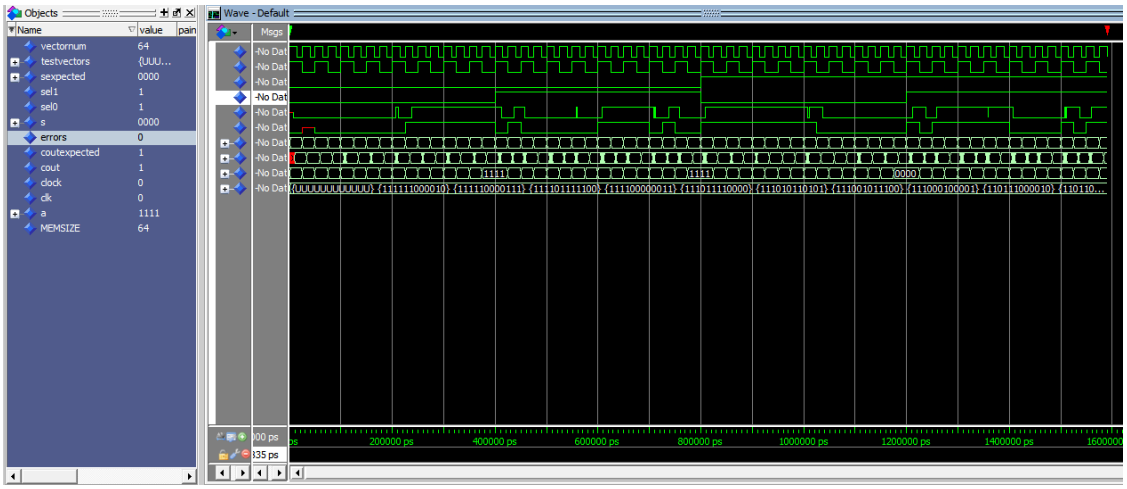
```
31  process begin
32      clk <= '1'; wait for 5 ns;
33      clk <= '0'; wait for 10 ns;
34  end process;
35  -- at start of test, load vectors
```



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.