Tutorial 3:

Projetar uma Uma Unidade Lógica Aritmética (ULA)

Versão 1.0

Sidney Lima, 2008.

Uma Unidade Lógica Aritmética (ULA) é um circuito combinacional que realiza as principais operações lógicas e aritméticas em um par de operando de n-bit (ex. A [6:0] e B [6:0]). As operações realizadas por uma ULA são controladas por um conjunto de entradas de seleção de funções. Neste tutorial será desenvolvida uma ULA de 2-bits, com duas entradas de seleção de funções: Entradas Seleção S1 e S0. As funções realizadas pela ULA são soma subtração, multiplicação e reset. Os módulos estão descrito em VHDL.

1. COMPONENTES DA ULA.

A construção de uma ULA básica foi subdividida em módulos menores: somador, subtrator, multiplicador, reset, mux4:1. A solução proposta para o este tutorial segue no diagrama abaixo que compõem a ULA.

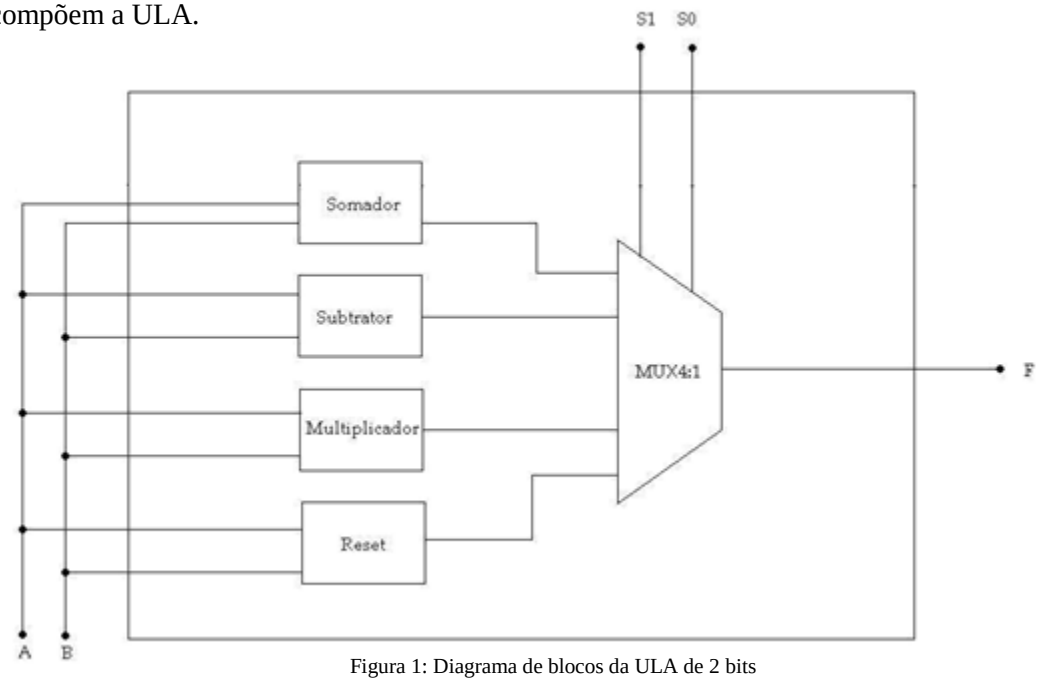

Figura 1: Diagrama de blocos da ULA de 2 bits

2.1 SOMADOR

Neste módulo é explicitado o módulo básico somador denominado FA na figura 2.a. que são empregados para somar bits individualmente. Quando há uma extensão do problema para a adição de duas palavras de n-bits, utiliza-se uma unidade de somadores em paralelo que permite a soma dos bits de cada coluna e conectamos o bit de vai um cada FA. A figura 2.b faz uma adaptação entre a forma comum de se operar a adição e a notação gráfica adotada pela figura 2.a.

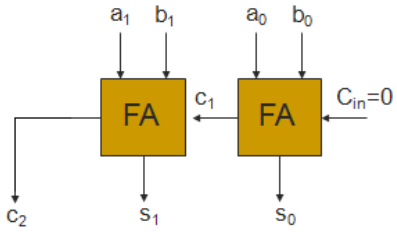

Figura 2.a: Representação gráfica do somador

$$\begin{array}{rr} & c_2 c_1 c_0 \\ & a_1 a_0 \\ + & b_1 b_0 \\ \hline & s_2 s_1 s_0 \end{array}$$

Figura 2.b: Esquema do somador de 2-bits

O código em VHDL estão descritos abaixo, lembrando que a saída do módulo chamada é de 5 bits com o objetivo de padronizar as saídas e corresponde ao somador representado na figura

2.a. Já o módulo somador diz respeito a caixa FA, ou seja, uma caixa preta para somar bits individualmente.

-- Método Somador:

```
entity somador is
PORT (A, B, Cin : in BIT;
            S, Cout : out BIT
            );
end somador;

architecture corpo of somador is
begin
S       <= A xor (B xor Cin);
Cout    <= (A and B) or ((A xor B) and Cin);
end corpo;
```

-- Método Chamada:

```
entity chamada is
PORT (A : in BIT_VECTOR(1 downto 0);
        B : in BIT_VECTOR(1 downto 0);
        S : out BIT_VECTOR(4 downto 0)
);
end chamada;
architecture corpo of chamada is
component somador
PORT (A, B, Cin : in BIT;
            S, Cout: out BIT);
end component;


signal C : BIT_VECTOR(4 downto 0);
begin
C(0) <= '0';
A1 : somador port map (A(0), B(0), C(0), S(0), C(1));
A2 : somador port map (A(1), B(1), C(1), S(1), C(2));
S(2) <= C(2);
S(4 downto 3) <= "00";


end corpo;
```

2.2 SUBTRADOR

Um pouco mais complicado do que a adição devido à necessidade de apresentar o conceito de “borrow” no sistema binário. Quando subtraímos um numero menor de um outro maior (ou iguais), regras são:

$$0-0=0$$
$$1-0=1$$
$$1-1=0$$

Porém, precisa de uma regra para o caso (0-1). Isto é normalmente escrito como 0-1=1 (usando um borrow de 1). Ao fazer uma operação aritmética, precisamos decidir como representar números negativos. Geralmente em sistemas digitais, números negativos são representados em complemento de dois. Isto tem várias vantagens em relação à representação sinal e magnitude como a facilidade de realizar operações de adição e subtração de números

negativos e positivos. Além disto, o número zero tem uma representação única em complemento de dois. O complemento de dois de um número N de n-bit é definido como,

$$2n - N = (2n - 1 - N) + 1.$$

A última representação fornece uma forma fácil de encontrar o complemento de dois: pegue o complemento do número e adicione 1. Por exemplo, para representar o número -5, nós calculamos o complemento de dois de 5 (0101) da seguinte forma,

$$\begin{array}{rl} 5\ (0\ 1\ 0\ 1) \rightarrow & 1\ 0\ 1\ 0\ \text{(complemento bit a bit)} \\ & +\quad 1 \\ \hline & 1\ 0\ 1\ 1\ \text{(complemento de dois)} \end{array}$$

A seguir, estão os códigos em VHDL do módulo subtrador e seus componentes. Vale ressaltar que no momento que entra as duas palavras no módulo subtrador, um módulo chamado **comparador** (um módulo interno) ordena as palavras em ordem decrescente**,** além de enviar um sinal de controle para o **avaliador**. O módulo **complemento2** é responsável pela aplicação do complemento de 2 na segunda palavra caso a primeira entrada seja maior que a segunda, ou no resultado da operação caso esta seja maior que aquela. O módulo avaliador escolhe a saída do **subtrador** através do sinal seletor enviado pelo módulo **comparador**.

-- Método Subtrador:

```
entity subtrador is
PORT (A : in BIT_VECTOR(1 downto 0);
        B : in BIT_VECTOR(1 downto 0);
        S : out BIT_VECTOR(4 downto 0)
);
end subtrador;
architecture corpo of subtrador is

component chamadasub
PORT (A, B : in BIT_VECTOR(1 downto 0);
      S : out BIT_VECTOR(4 downto 0));
end component;

component comparador
PORT (A,B : in bit_vector (1 downto 0);
      resp : out bit;
      U,V : out bit_vector (1 downto 0));
end component;

component complemento2
PORT (A : in BIT_VECTOR(1 downto 0);
      S : out BIT_VECTOR(1 downto 0));
end component;

signal
resp : BIT,
AUX1 : BIT_VECTOR(1 downto 0),
AUX2 : BIT_VECTOR(1 downto 0),
AUX3 : BIT_VECTOR(1 downto 0),
AUX4 : BIT_VECTOR(4 downto 0);

begin

CO: comparador port map (A, B, resp, AUX1, AUX2);
C:  complemento2 port map (AUX2, AUX3);
```

```
SO: chamadasub port map (AUX1, AUX3, AUX4);
MET: avaliador port map (AUX4, resp, S);
end corpo;
```

-- Método chamadasub:

```
entity chamadasub is
PORT (A : in BIT_VECTOR(1 downto 0);
        B : in BIT_VECTOR(1 downto 0);
        S : out BIT_VECTOR(4 downto 0)
);
end chamadasub;

architecture corpo of chamadasub is
component somador
PORT (A, B, Cin : in BIT;
            S, Cout: out BIT);
end component;

signal C : BIT_VECTOR(2 downto 0);
begin
C(0) <= '0';
A1 : somador port map (A(0), B(0), C(0), S(0), C(1));
A2 : somador port map (A(1), B(1), C(1), S(1), C(2));
S(4 downto 2) <= "000";
end corpo;
```

-- Método comparador:

```
entity comparador is
      port (a,b: in bit_vector (1 downto 0);
              resp: out bit;
              u,v: out bit_vector (1 downto 0));
end comparador;

architecture corpo of comparador is
begin
resp <= '0' when (a>=b) else
             '1';
u <= a when (a>=b) else
       b;
v <= b when (a>=b) else
       a;
end corpo;
```

-- Método complemento2:

```
entity complemento2 is
      port (A: in bit_vector (1 downto 0);
              S: out bit_vector (1 downto 0));
end complemento2;
architecture corpo of complemento2 is
component chamadasub
      port (a, b: in bit_vector (1 downto 0);
              q: out bit_vector (1 downto 0));
end component;
```

```
Signal Abarrado, AUX: bit_vector (1 downto 0);
begin
Abarrado <= not a;
AUX <= 01;
S1: chamadasub port map (Abarrado, AUX, q);
end corpo;
```

-- Método avaliador:

```
entity avaliador is
      port (A: in bit_vector (1 downto 0);
              resp: in bit;
              S: out bit_vector (4 downto 0));
end avaliador;
architecture corpo of mux2 is

component complemento2
PORT (A : in BIT_VECTOR(1 downto 0);
      S : out BIT_VECTOR(1 downto 0));
end component;

begin
C: complemento2 port map (A, B);
S <= A when (resp ='0') else
       B;
end corpo;
```

2.3 MULTIPLICADOR

Na multiplicação de duas palavras binárias são utilizadas uma combinação de operações and e do módulo somador descrito na seção 2.1. A figura 3 dão uma representação gráfica sobre a função multiplicador da ULA. O código em VHDL está descrito abaixo da figura.

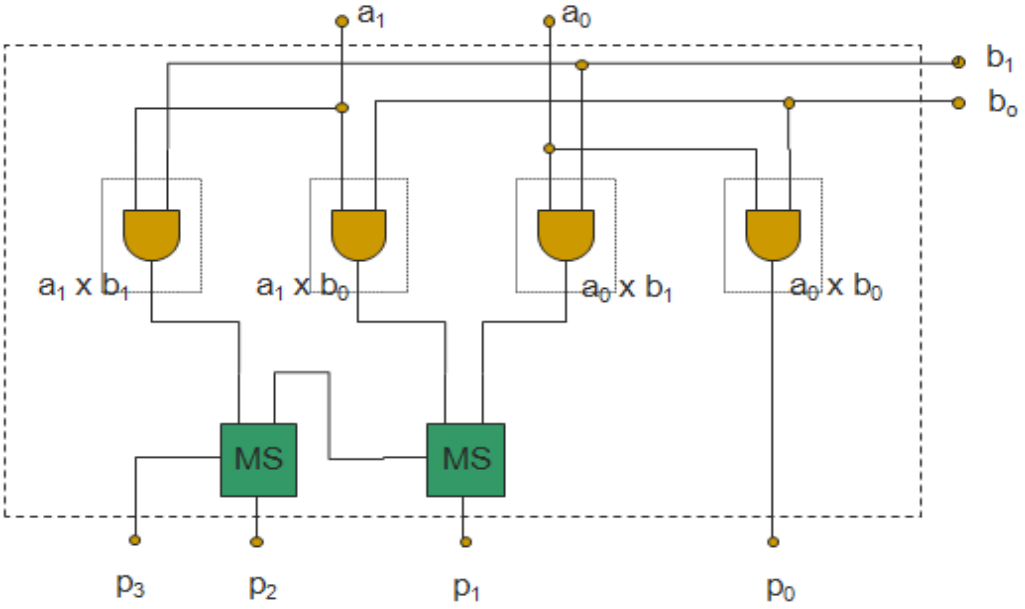

Figura 3: Representação gráfica do multiplicador

```
entity multiplicador is
      port (A,B: in bit_vector (1 downto 0);
               S: out bit_vector (4 downto 0));
end multiplicador;

architecture corpo of multiplicador is

component somador
      port (A, B, Cin: in bit;
               S, Cout: out bit);
end component;

Signal x1,x2,x3,x4,c1,c2: bit;
begin
C0 <= '0';
S(0) <= A(0) and B(0);
x1 <= A(0) and B(1);
x2 <= A(1) and B(0);
x3 <= A(1) and B(1);
X4 <= "00";
S1: somador port map (x1,x2,c0,S(1),c1);
S2: somador port map (x3,x4,c1,S(2),c2);
S(3) <= c2;
S(4) <= "0";
end corpo;
```

2.4 RESET

O módulo **reset** tem como finalidade zerar o display. Sendo o módulo mais simples, seu código em VHDL é descrito a seguir.

```
entity reset is
      port (q: out bit_vector (4 downto 0));
end reset;
architecture corpo of reset is
begin
q <= "00000";
end corpo;
```

2.5 CONTROLE SAÍDA DA ULA

Através da palavra de seleção S1 e S0, que é enviada a um módulo **mux4:1,** o **mux4:1** recebe como entrada as saídas dos módulos somador, subtrator, multiplicador e reset. A medida que a palavra de seleção varie (“00”,”01”,”10”,”11”) o **mux4:1** escolhe uma das entrada que será conectada a saída da ULA.

```
entity mux4 is
      port (a,b,c,d: in bit_vector (4 downto 0);
               s: in bit_vector (1 downto 0);
               f: out bit_vector (4 downto 0));
end mux4;
architecture logica of mux4 is
begin
f <= a when (s="00") else
       b when (s="01") else
       c when (s="10") else
       d;
end logica;
```