UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO

Relatório de Projeto Grau A:
Unidade Lógica Aritmética

Aline Nunes de Souza
Davi de Souza Leão Schmitz

São Leopoldo
2021

Aline Nunes de Souza
Davi de Souza Leão Schmitz

Relatório de Projeto Grau A:
Unidade Lógica Aritmética

Trabalho apresentado para a disciplina Prototipação Digital, pelo Curso de Engenharia da Computação da Universidade do Vale do Rio dos Sinos – UNISINOS, ministrada pela professora Bruna Fernandes Flesch.

São Leopoldo
2021

SUMÁRIO

INTRODUÇÃO

O presente trabalho descreve as etapas para o desenvolvimento do circuito no *software Logisim* e implementação por meio da linguagem *VHSIC Hardware Description Language* (VHDL) no *software ISE* de uma Unidade Lógica Aritmética (ULA). Como operações, o circuito deve possibilitar realização de Adição, Subtração em complemento de dois, Operação AND entre dois dados de entrada. Para os dados de entrada, deve-se considerar que possuem 4 bits. Para fins de análise e comprovação de funcionamento, há também a descrição de cada operação por simulação no *ISIM*.

METODOLOGIA

Para desenvolver a ULA, primeiramente foi desenvolvida a tabela verdade de um somador completo (Full Adder). Para fins de simplificação, será considerado 2 bits no desenvolvimento da tabela e do karnaugh, mas o restante será considerado como 4 bits.

Tabela verdade

Tabela verdade do circuito somador completo

Na figura 1 foi representado a tabela verdade do circuito somador completo de 2 bits, resultando em (2^2) * 2 = 8 linhas. Originalmente seriam 32 linhas de tabela pois 2^(4) * 2.

Figura 1 - Tabela verdade do circuito somador completo

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fonte: os autores (2021).

Tabela verdade da porta AND

Na figura 2 foi representado a tabela verdade da porta AND para fins de exemplificação.

Figura 2 - Tabela verdade da Porta AND

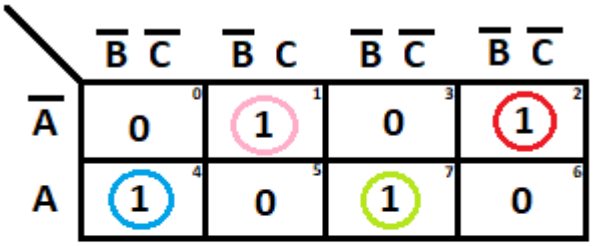
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Fonte: os autores (2021).

Expressões lógicas das saídas e simplificação com mapas de Karnaugh

Com base nas tabelas verdade anteriormente apresentadas, foi desenvolvido o karnaugh de 2 bits da entrada, conforme figura 3, e do *carry out* conforme figura 4.

Figura 3 - Karnaugh da saída

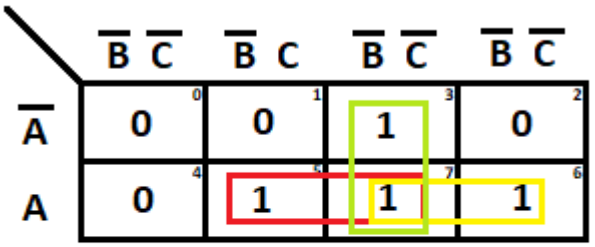


$$S = A\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C}$$

$$S = Cin \oplus (A \oplus B)$$

Fonte: os autores (2021).

Figura 4 - Karnaugh do Carry out



$$Cout = A\,Cin + B\,Cin + AB$$

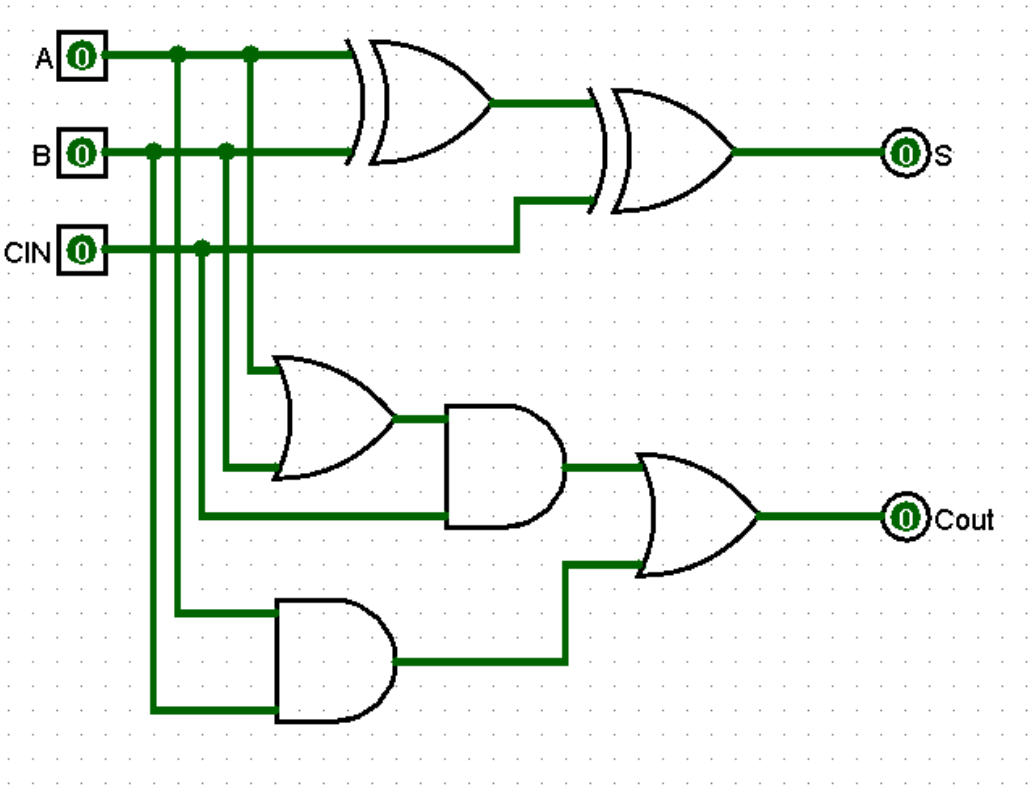
$$Cout = AB + Cin\,(A+B)$$

Fonte: os autores (2021).

Circuito Somador Completo

A partir da tabela verdade e expressões booleanas, foi possível construir um módulo somador com duas entradas de 1 bit, contendo ainda um bit de *carry in* e um de *carry out*. Este circuito é apresentado na Figura 5:

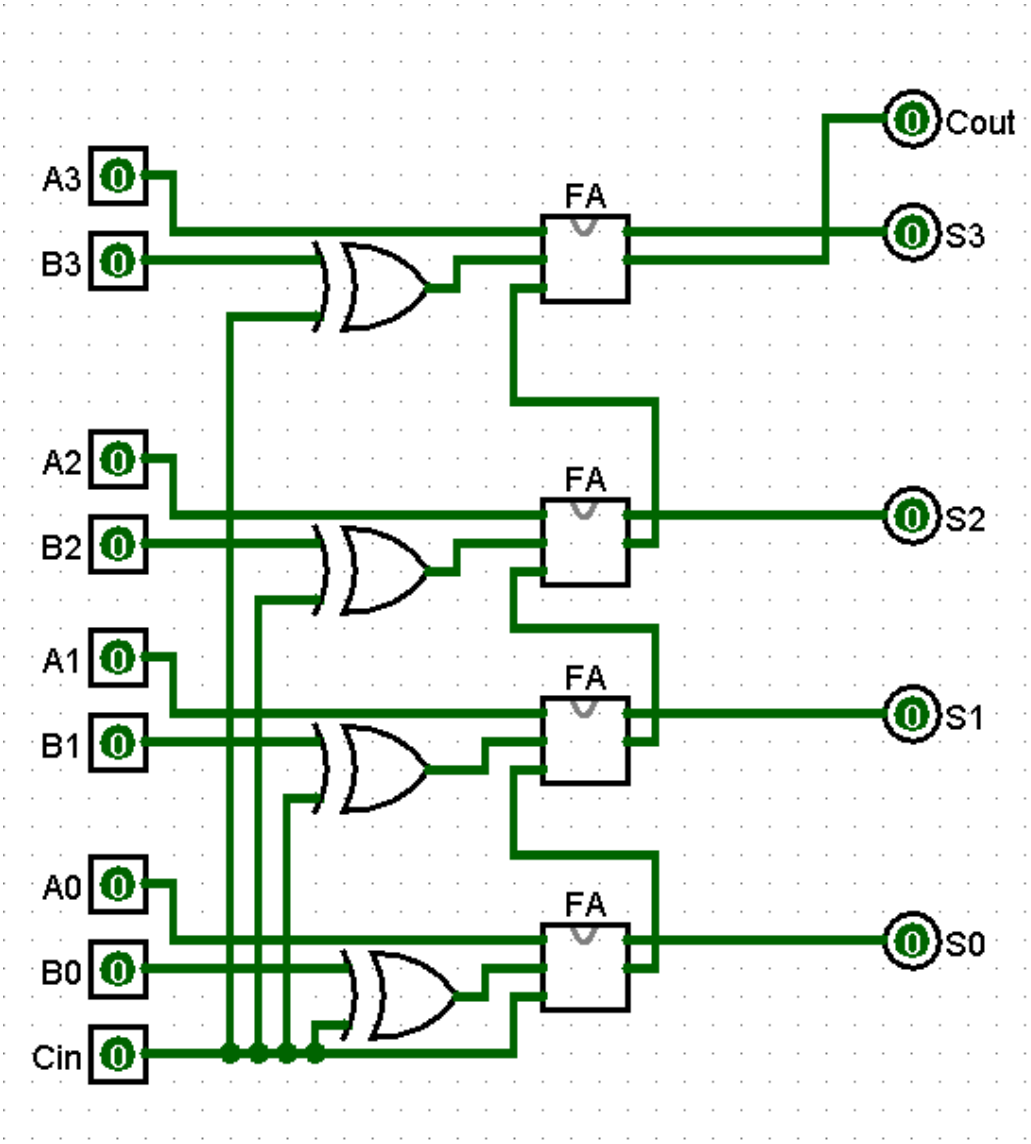
Figura 5 - Circuito somador para dados de 1 bit



Fonte: os autores (2021).

Para a construção do somador completo de 4 bits proposto, foram utilizados 4 módulos de 1 bit, conectados de forma a propagar de um módulo ao outro o bit de *carry out*. Desta forma a saída obtida é de um total de 5 bits, sendo 4 das saídas dos somadores e 1 o carry out do último módulo somador, neste caso representando o bit mais significativo. Este circuito pode ser observado na Figura 6.

Figura 6 - Circuito somador completo para dados de 4 bits



Fonte: os autores (2021).

As operações de subtração podem ser realizadas no mesmo circuito somador completo, através do uso do complemento de 2. Limitando-se a operações de subtração A - B, com A > B, podemos realizar somente uma soma de A com o complemento de 2 da entrada B. Este complemento de 2 será obtido invertendo-se todos os bits de B e somando 1 ao final.

Figura 7 - Tabela verdade da porta XOR

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

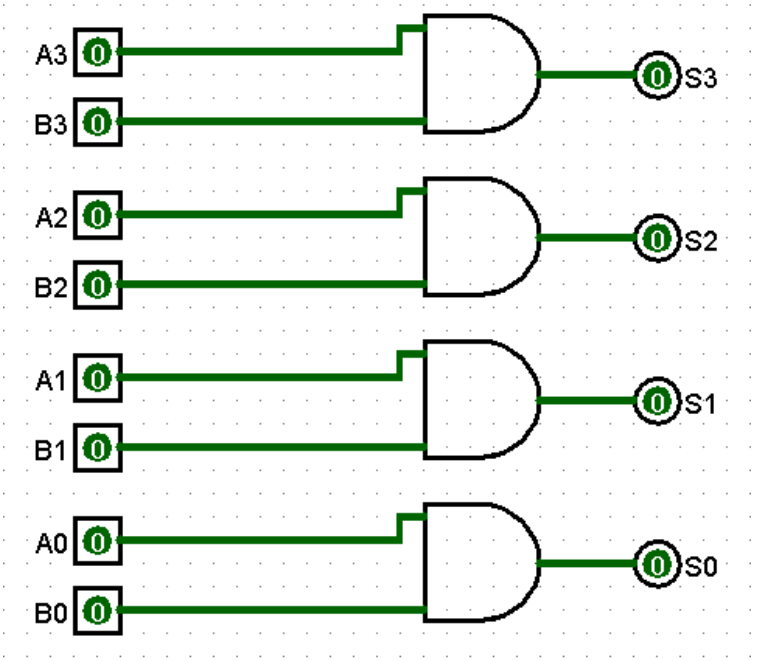
Fonte: os autores (2021).

No circuito da Figura 6 temos a entrada Cin como seletora entre operações de soma e subtração. Caso Cin seja 1, teremos 1 como umas das entradas das portas XOR, e B a outra. Neste caso, a saída da porta XOR será sempre o valor inverso de B (conforme tabela verdade da Figura 7). Além disso, o valor 1 de Cin serve como *carry in* do módulo somador do bit menos significativo, somando 1 ao resultado.

Circuito para operação AND

Outra funcionalidade proposta para a ULA foi a de AND bit a bit. Para isto foi desenvolvido o circuito apresentado na Figura 8, utilizando somente portas AND entre os bits de cada entrada.

Figura 8 - Operação AND bit a bit

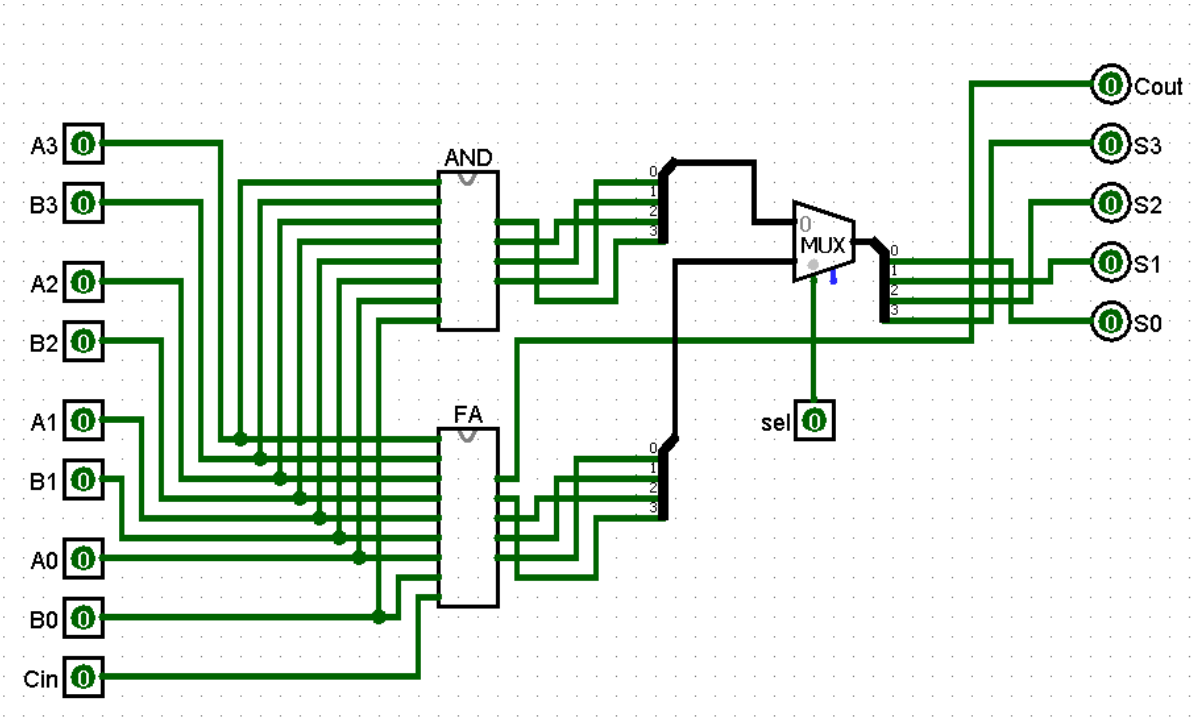


Fonte: os autores (2021).

Circuito completo da ULA

Unindo as funcionalidades apresentadas de soma, subtração e AND, chegamos ao circuito completo da ULA, apresentado na Figura 9.

Figura 9 - Circuito completo da ULA



Fonte: os autores (2021).

O circuito conta com duas entradas de 4 bits, uma entrada Cin para a seleção entre as operações de soma e subtração, e um MUX para controlar se a saída será a do somador ou a do bloco AND. Como saída temos 5 bits, sendo que o mais significativo deve ser desconsiderado em operações de subtração.

Implementação do VHDL

Para a implementação da ULA em VHDL, tendo o auxílio do circuito anteriormente desenvolvido, a primeira etapa foi definir as portas da entidade ULA. Foram utilizadas as entradas de dados A e B de 4 bits, C_in para a seleção entre as operações de soma e subtração, MuxSel de 1 bit e S sendo que S contém 4 bits de dados e mais 1 bit para comportar o *carry out*.

Figura 10 - Portas definidas no VHDL

```
entity ula is
   port (
      A           :  IN  std_logic_vector(3 downto 0);
      B           :  IN  std_logic_vector(3 downto 0);
      C_in        :  IN std_logic;
      MuxSel      :  IN std_logic;
      S           :  OUT std_logic_vector(4 downto 0)
  );
end ula;
```

Fonte: os autores (2021).

Foram utilizados sinais para representar as conexões do circuito durante as operações de soma, subtração e complemento de dois.

Figura 11 - Sinais definidos no VHDL

```
signal s_b0: std_logic :='0';
signal s_b1: std_logic :='0';
signal s_b2: std_logic :='0';
signal s_b3: std_logic :='0';

signal s_soma0: std_logic_vector(1 downto 0):="00"; --2
signal s_soma1: std_logic_vector(1 downto 0):="00";
signal s_soma2: std_logic_vector(1 downto 0):="00";
signal s_soma3: std_logic_vector(1 downto 0):="00";

signal s_soma: std_logic_vector(4 downto 0) := "00000";
signal s_and: std_logic_vector(4 downto 0) := "00000";
```

Fonte: os autores (2021).

As atribuições para fazer a lógica de soma, subtração com complemento de dois e AND bit a bit estão apresentadas na figura 12. Para implementar a lógica da porta AND bit a bit, é necessário concatenar um bit de ‘0’ pois o AND entre A e B sempre vai retornar 4 bits, e o contador é de 5 bits.

Figura 12 - Atribuições assíncronas no VHDL

```
s_b0 <= b(0) xor C_in;
s_b1 <= b(1) xor C_in;
s_b2 <= b(2) xor C_in;
s_b3 <= b(3) xor C_in;

s_soma0 <= ("0"&A(0 downto 0)) + ("0"&s_b0) + ("0"&C_in);
s_soma1 <= ("0"&A(1 downto 1)) + ("0"&s_b1) + ("0"&s_soma0(1));
s_soma2 <= ("0"&A(2 downto 2)) + ("0"&s_b2) + ("0"&s_soma1(1));
s_soma3 <= ("0"&A(3 downto 3)) + ("0"&s_b3) + ("0"&s_soma2(1));

s_soma(0) <= s_soma0(0);
s_soma(1) <= s_soma1(0);
s_soma(2) <= s_soma2(0);
s_soma(3) <= s_soma3(0);
s_soma(4) <= s_soma3(1) xor C_in;

s_and <= ("0"&(A and B));
```

Fonte: os autores (2021).

Foi utilizado um processo para monitorar a entrada de seleção do mux, conforme figura 13. Nessa etapa é realizado o controle da seleção entre bloco de Soma/Subtração ou AND. Se a entrada de seleção do MUX for 1, a ULA vai receber o valor da saída do bloco de AND. Caso for 0, receberá do bloco de Soma/Subtração. Como o nosso MUX possui somente dois bits, o processo nunca vai executar o laço de *else*, mas, caso seja necessário ter mais bits de seleção, é interessante a utilização de um valor default de saída.

Figura 13 - Processo definido no VHDL

```
process (MuxSel, s_soma, s_and)
begin
   if MuxSel = '0' then
      S <= s_soma;
   elsif MuxSel = '1' then
      S <= s_and;
   else
      S <= "00000";
   end if;
end process;
```

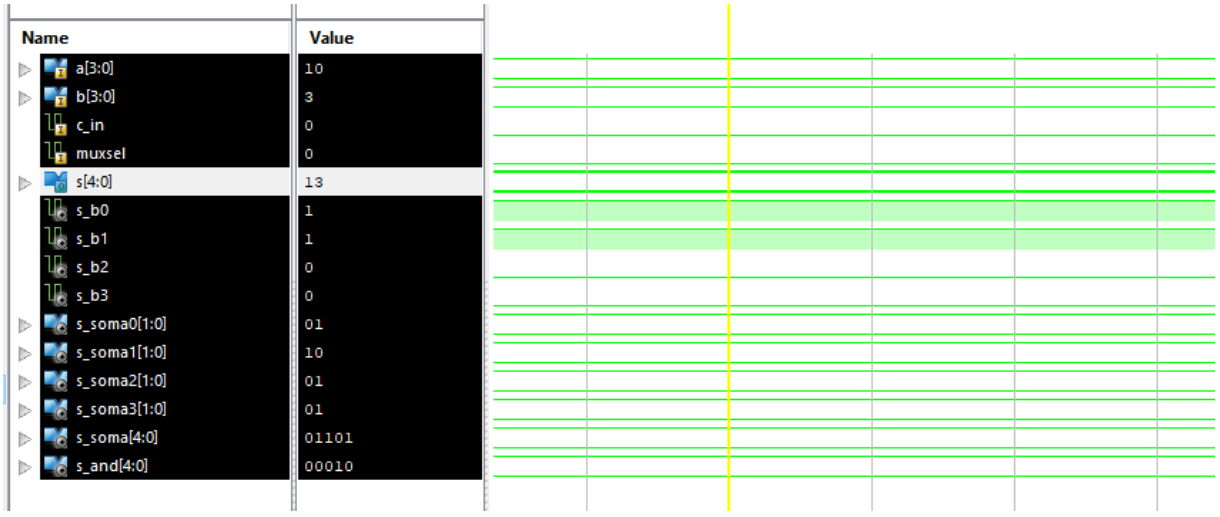
Fonte: os autores (2021).

RESULTADOS

Foram realizadas simulações do circuito no *software ISIM* para analisar e comprovar o funcionamento da ULA para as operações de soma, subtração e AND bit a bit. Para simular os valores de entrada, é importante ressaltar que a condição de funcionamento do circuito é que o valor de A sempre seja maior que B. Além disso, os valores das entradas sempre foram setados em binário e depois exibição do valor foi trocado para *unsigned decimal*.

Para realizar os testes com operações de soma, é necessário definir a entrada Cin como 0 e MuxSel como 0. A figura 14 apresenta uma operação de soma com A valendo 10 e B valendo 3.

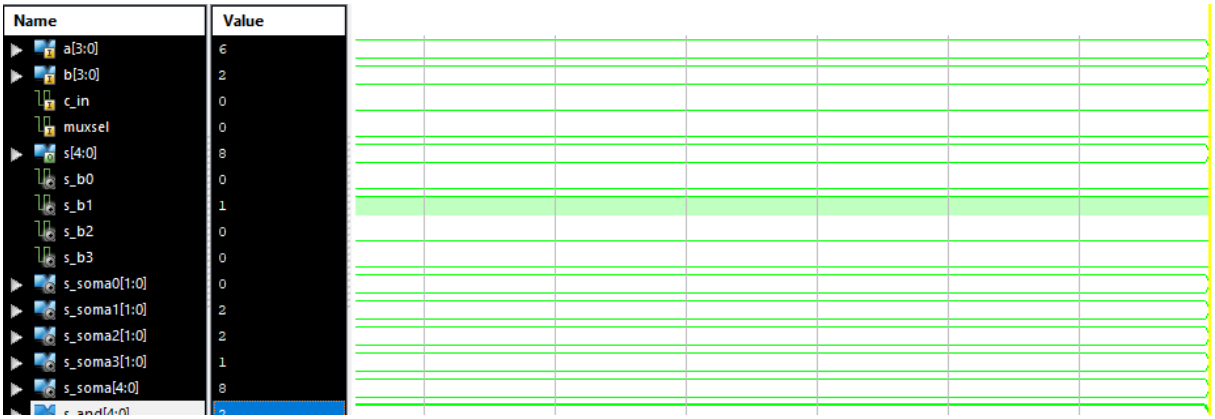
Figura 14 - Operação de Soma



Fonte: os autores (2021).

Na figura 15 foi setado 6 em binário, 0110, para o A e 2 em binário, 0010, para o B.

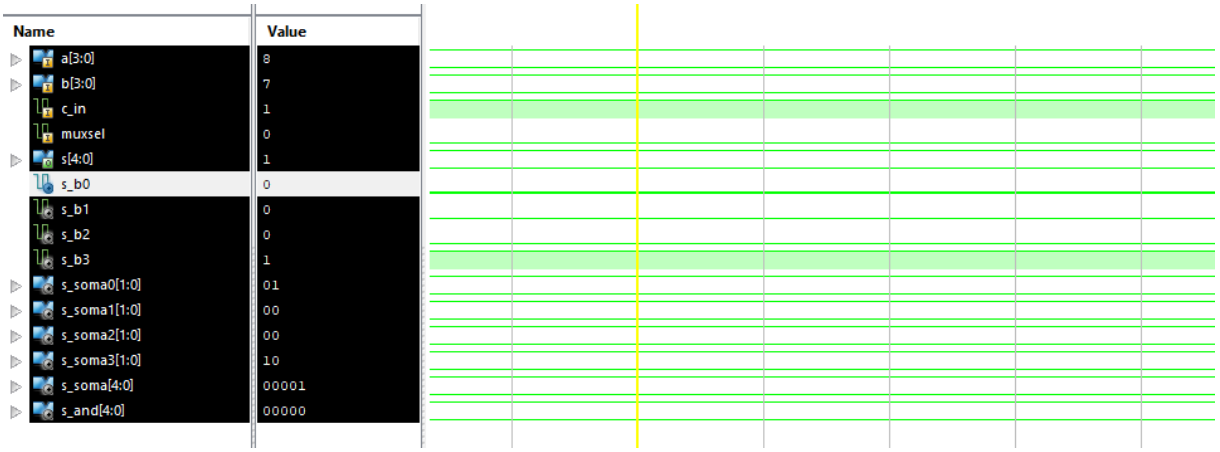
Figura 15 - Operação de Soma



Fonte: os autores (2021).

Para realizar os testes com operações de subtração, é necessário definir a entrada Cin como 1 e MuxSel como 0. A figura 16 apresenta uma operação de subtração com A valendo 8, sendo setado o valor em binário de 1000**,** e B valendo 7.

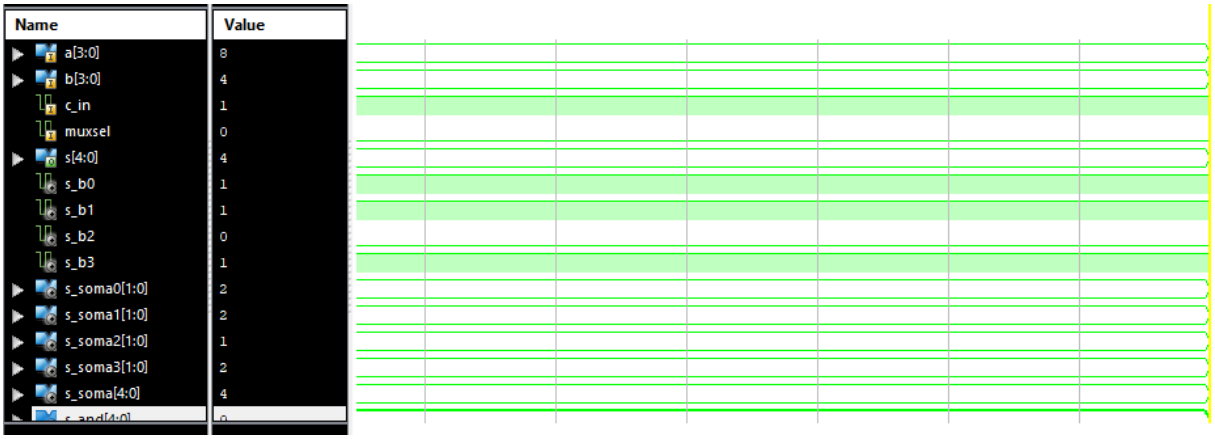
Figura 16 - Operação de Subtração



Fonte: os autores (2021).

Na figura 17 foi setado 8 em binário, 1000, para o A e 4 em binário, 0100, para o B.

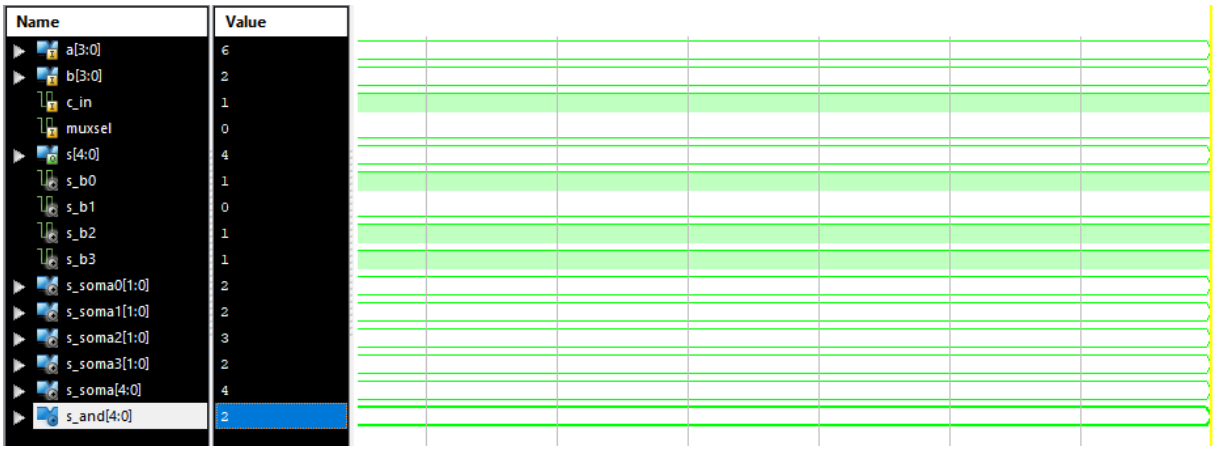
Figura 17 - Operação de Subtração



Fonte: os autores (2021).

Na figura 18 foi setado 6 em binário, 0110, para o A e 2 em binário, 0010, para o B.

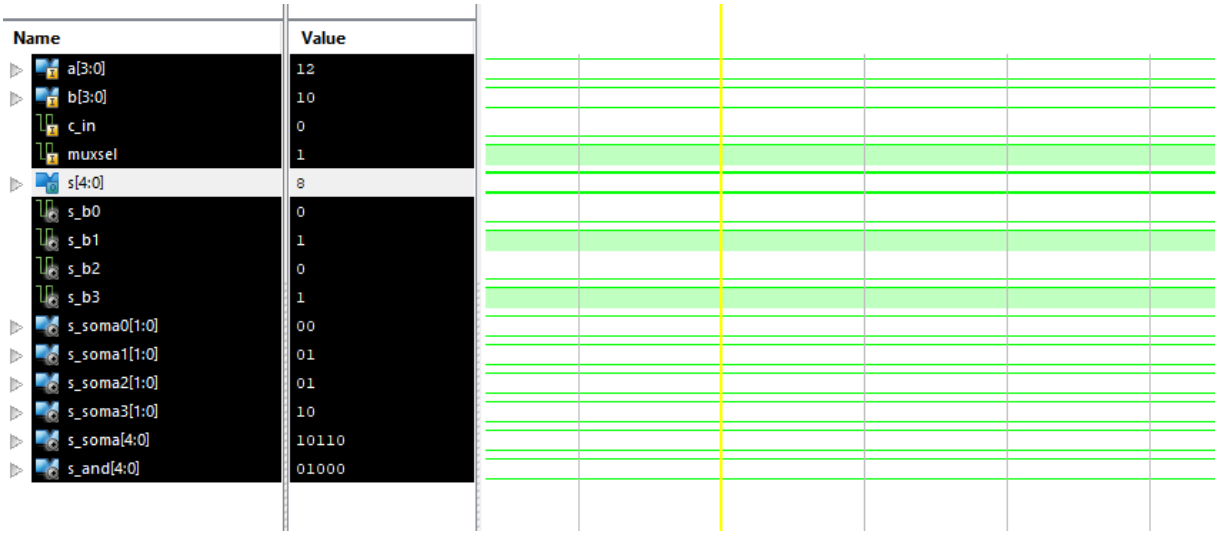
Figura 18 - Operação de Subtração



Fonte: os autores (2021).

Para realizar os testes com operações de AND bit a bit, é necessário definir a entrada Cin como 0 e MuxSel como 1. A figura 19 apresenta a operação com A valendo 12 e B valendo 10.

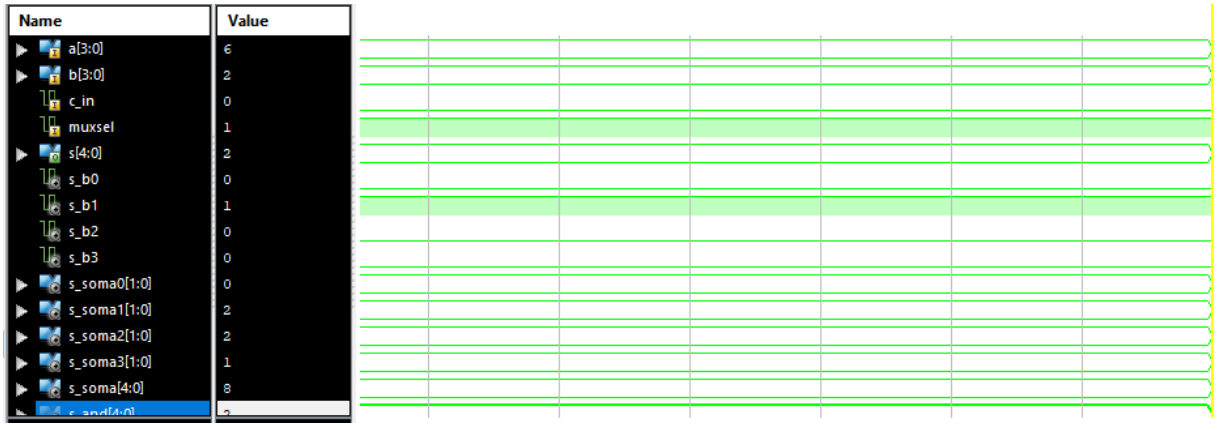
Figura 19 - Operação AND bit a bit



Fonte: os autores (2021).

Na figura 20 foi setado 6 em binário, 0110, para o A e 2 em binário, 0010, para o B.

Figura 20 - Operação de AND bit a bit



Fonte: os autores (2021).

CONCLUSÃO

Com a realização do presente trabalho, foi possível revisar conteúdos da disciplina de Sistemas Digitais e praticar o desenvolvimento em VHDL. A partir das tabelas verdades e expressões lógicas, conseguimos desenvolver o circuito lógico no Logisim que nos ajudou a construir o código. A parte que tivemos mais dificuldade foi relembrar a parte de álgebra booleana, operar com bits e conectar o bloco somador com o AND. Com as simulações no *software ISIM*, podemos ver o funcionamento de cada operação com diferentes entradas e comprovar seu funcionamento.