UNIVERSIDADE DE BRASÍLIA

Adrianne Alves da Silva - 160047595
Ana Paula Gomes de Matos - 160023629

TURMA "F"
PRÁTICA DE ELETRÔNICA DIGITAL - 119466
Experimento 03: Circuitos somadores e subtratores

Brasília - DF
05 de maio de 2017

UNIVERSIDADE DE BRASÍLIA

PRÁTICA DE ELETRÔNICA DIGITAL - 119466

Relatório referente ao primeiro experimento da disciplina Prática de Eletrônica Digital, ministrada pela professora Lourdes Mattos Brasil, realizado no dia 07 de abril de 2017.

Alunas:

Adrianne Alves da Silva

Ana Paula Gomes de Matos

Brasília – DF

SUMÁRIO

1 INTRODUÇÃO

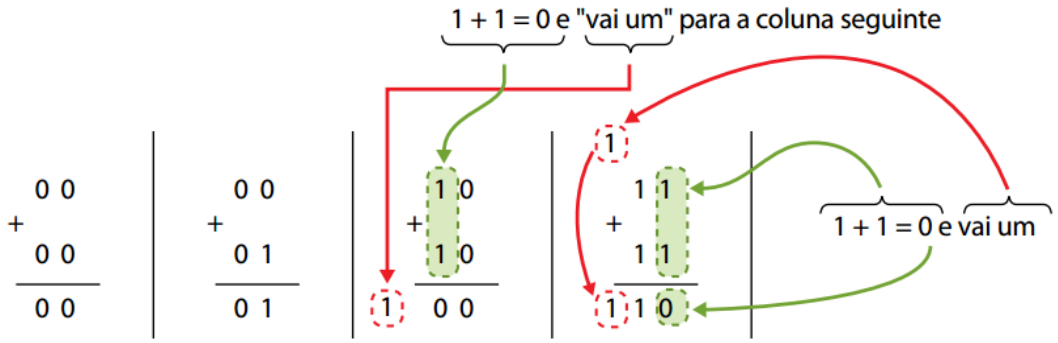
1.1 Circuitos aritméticos

As operações matemáticas são facilmente identificadas em diferentes situações cotidianas, seja para unir objetos diferentes em um mesmo recipiente (soma), retirá-los de lá (subtração), unir vários objetos iguais (multiplicação) ou até mesmo, separar os objetos em recipientes diferentes (divisão). Apesar de triviais, estes exemplos mostram de forma clara a importância das operações matemáticas e suas aplicações nas mais diversas atividades diárias.

Diante disso, é de se esperar que situações mais complexas exijam operações aritméticas mais longas, dessa maneira, não seria interessante realizar todos os cálculos necessários de forma mecânica, pois existem situações em que é humanamente impossível realizar as contas manualmente e é exatamente para isso que os **circuitos aritméticos** existem. Eles foram criados para executarem as operações citadas inicialmente e qualquer outra que possa ser implementada em um circuito combinatório (aqueles cujas saídas dependem exclusivamente dos valores de entrada). [2]

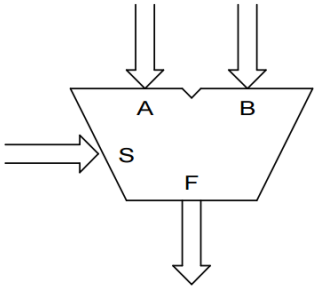
Dentro da arquitetura interna de um microprocessador existe a ULA (unidade lógica aritmética) que é onde são realizadas as operações lógicas e aritméticas [3]. Em geral, a soma, subtração e multiplicação de números binários são realizadas de forma semelhante à base decimal conhecida, contudo, vale lembrar que diferente da base decimal a expressão “vai um” é usada quando a soma (tratando-se do operador aritmético usual) é igual a 2 (dois), e não (10) dez, como expresso no exemplo a seguir:

Figura 01: Exemplo “vai um”



O arranjo logístico da ULA, de maneira simplificada, consiste em um circuito com entradas A e B que, de acordo com o sinal de controle S, fornece um resultado em F .

Figura 02: Esquema de uma ULA



Em geral, não existem circuitos que realizam a subtração nos microprocessadores e a motivação para isso é manter um circuito mais simples que possa realizar ambas as operações utilizando o mínimo de componentes. Nesse quesito, utiliza-se a adição para realização de uma operação de diferença, e isto é feito por meio da representação de números negativos em formas de sinal magnitude, complemento de 1 e complemento de dois.

1.1.1 Sinal-magnitude

Nessa representação utiliza-se os bits “0” e “1” para representar os sinais “-” (negativo) e “+” (positivo) respectivamente. Esses valores compõe o dígito mais à esquerda do número, anteriormente à sua magnitude (figura 03).

Figura 03: Representação por sinal-magnitude

$$+27_{10} = \underbrace{0}_{+}\ \underbrace{1\,1\,0\,1\,0\,1}_{\text{magnitude}}$$

$$-55_{10} = \underbrace{1}_{-}\ \underbrace{1\,1\,0\,1\,0\,0\,1}_{\text{magnitude}}$$

1.1.2 Complemento de um

O complemento de um é a representação de um número negativo binário através da inversão de todos os bits do mesmo, como é possível observar a seguir. É um método simples de representação que abre margem para uma segunda representação, o complemento de dois.

Figura 04: Representação por complemento de 1

$$101_{10} = \mathbf{01100101_2}$$
$$\text{-}101_{10} = \mathbf{10011010}_{\ 2}$$

1.1.3 Complemento de dois

Da mesma maneira que os demais métodos de representação de números negativos, o complemento de dois o faz. Sua lógica assemelha-se ao complemento de 1, com a variante de que é somado 1 ao número obtido com a primeira transformação.

Exemplo 01:

101_{10} = $\boldsymbol{01100101_2}$

i) **Inverte-se todos os bits**

$$10011010_2$$

ii) **Soma-se uma unidade**

$10011010_2 + 1 = 10011011_2 = -101_{10}$

Existem várias formas de representar um número negativo em binário, contudo, o complemento de dois é a forma mais utilizada nos sistemas modernos, pois permite realizar uma operação de subtração realizando, na verdade, uma adição. O exemplo a seguir mostra isso de forma mais clara:

Exemplo 02:

Base 10: (21 -19) = 2

Base 2:

(00010101) - (00010011) = (00010101 + 11101101)

Esse comportamento é facilmente observável na representação da figura 05 abaixo.

Figura 05: Operação de soma com complemento de 2

```
  1 1 1 1  1   1  ←  os "vai um"
  0 0 0 1  0 1 0 1
                  +
  1 1 1 0  1 1 0 1
 ------------------
1 0 0 0 0  0 0 1 0
```

Observa-se que o resultado final apresenta 9 bits, porém, o último número a esquerda não deve ser considerado, pois usou-se inicialmente apenas números com 8 bits, logo nota-se que o resultado final em binário é exatamente o mesmo que em decimal, ou seja, $(00000010)_2 = 2_{10}$.

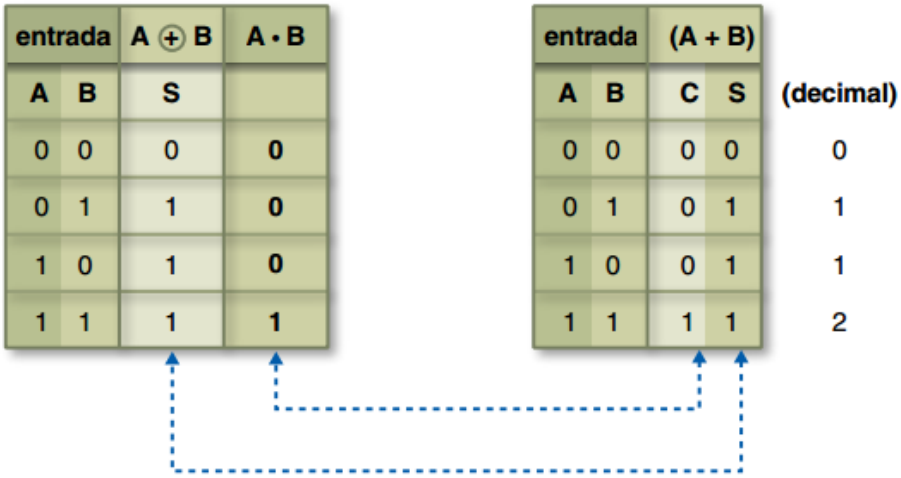
1.2 Circuito somador

As portas lógicas fundamentais (AND, OR, NOT) trazem muitas vezes o sentido de operação aritmética na lógica dos circuitos, entretanto, é preciso salientar que as **operações lógicas** não são equivalentes a **operações aritméticas**, mesmo que os sinais de representação utilizados sejam iguais. [2]

1.2.1 Meio Somador

O circuito meio somador traduz as operações simples de soma em binário, tal que ele aceita somente dois dígitos binários em suas entradas e produz um dígito da soma e um dígito do "vai um" ou *carry* [3]. É importante considerar que os circuitos em geral utilizam das portas lógicas fundamentais para descreverem a sua lógica, o mesmo acontece com o meio- somador. Observe as tabelas abaixo das portas XOR, AND e a da soma aritmética de dois binários:

Figura 06: comparação entre a porta XOR, AND e a soma aritmética de binários

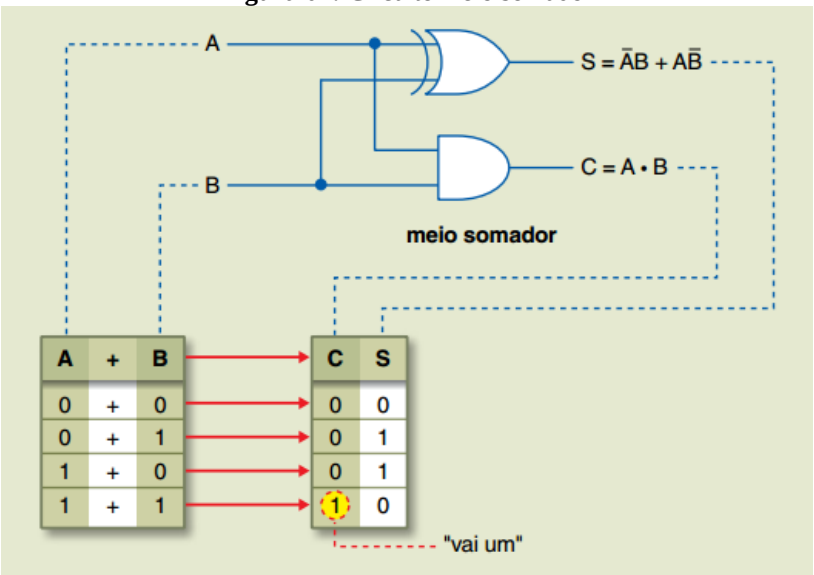


entrada		A ⊕ B	A • B
A	B	S	
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

entrada		(A + B)		
A	B	C	S	(decimal)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	2

Nota-se que por meio da porta XOR é possível obter os dígitos menos significativos da soma aritmética (A+B), já com a porta AND obtem-se o bit "vai um" da operação citada. Isso pode ser observado mais claramente na figura 03.

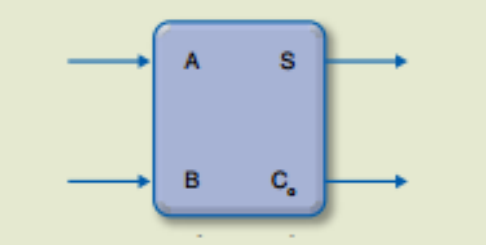
Figura 07: Circuito meio somador



Fonte: www.albertoferes.com.br

A representação em bloco do meio somador deixa ainda mais simples a relação de entradas e saídas de dados no circuito meio somador, isso porque são necessárias duas entradas e duas saídas apenas (figura 04).

Figura 08: Circuito meio somador em um único bloco



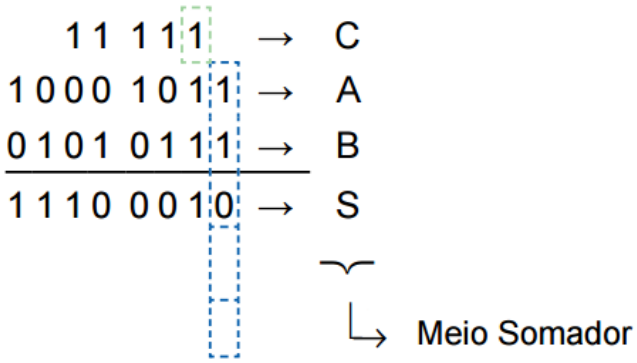
Fonte: www.albertoferes.com.br

O meio somador também é conhecido como *half adder*, e o dígito de transporte C, como *carry*. Vale salientar que não é possível somar mais de dois algarismos com o meio somador, para isto, deve-se usar o somador completo. [2]

1.2.2 Somador Completo

O somador completo, apesar de usar o conceito do meio somador, não pode ser substituído por ele. A exemplo disso, considere a soma de dois binários aleatórios (10001011) + (01010111) como representado a seguir:

Figura 09: Soma de dois números binários



Observa-se que os bits da coluna à direita e o "vai um" podem ser obtidos através do meio somador, apresentado anteriormente. Entretanto, nas demais colunas o meio somador não se aplica, pois existe a possibilidade de ocorrer 3 bits envolvidos na soma caso aconteça um "vai um" da soma anterior, diante disto, visto que o meio somador suporta apenas 2 entradas, faz-se necessário o uso de um circuito aritmético contendo 3 entradas e duas saídas (tabela 1).

Tabela 1: Tabela verdade do circuito somador completo

Entradas			Saídas	
A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Legenda:

A e B são os bits somados;
Ci é o *carry in*, "vai um" da coluna anterior - entrada no somador;
C_o é o *carry out*, "vai um" - saída do somador;
Ci é a saída do somador;
C_o é a entrada do somador seguinte.

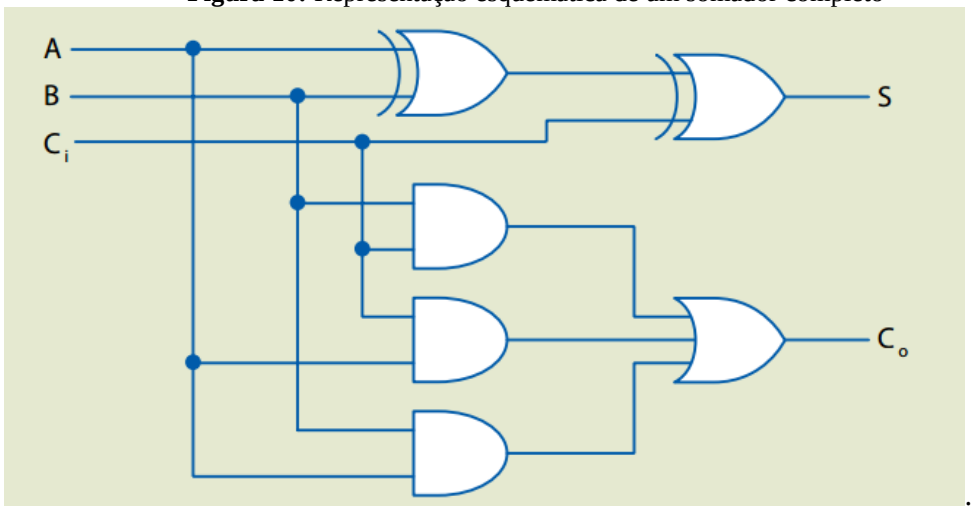
Analisando a tabela, é possível retirar as expressões lógicas que descrevem esse circuito, e além disso, simplificá-las por meio de métodos como o mapa de Karnaug , dessa forma, obtém-se o seguinte resultado :

$$S = A \oplus B \oplus C_i$$

$$C_o = A \cdot B + B \cdot C_i + A \cdot C_i$$

A sua representação esquemática é relativamente simples em comparação com o seu poder computacional ao ser conectado com outros tipos de circuitos (figura 10).

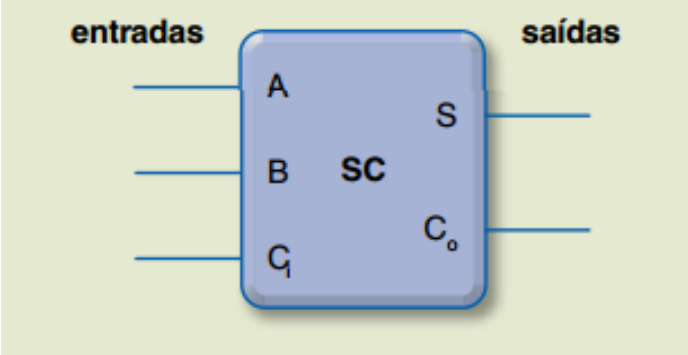
Figura 10: Representação esquemática de um somador completo



Fonte: www.albertoferes.com.br

De maneira ainda mais simplificada, em termos de suas entradas e saídas, pode ser representado em bloco, conforme a figura abaixo:

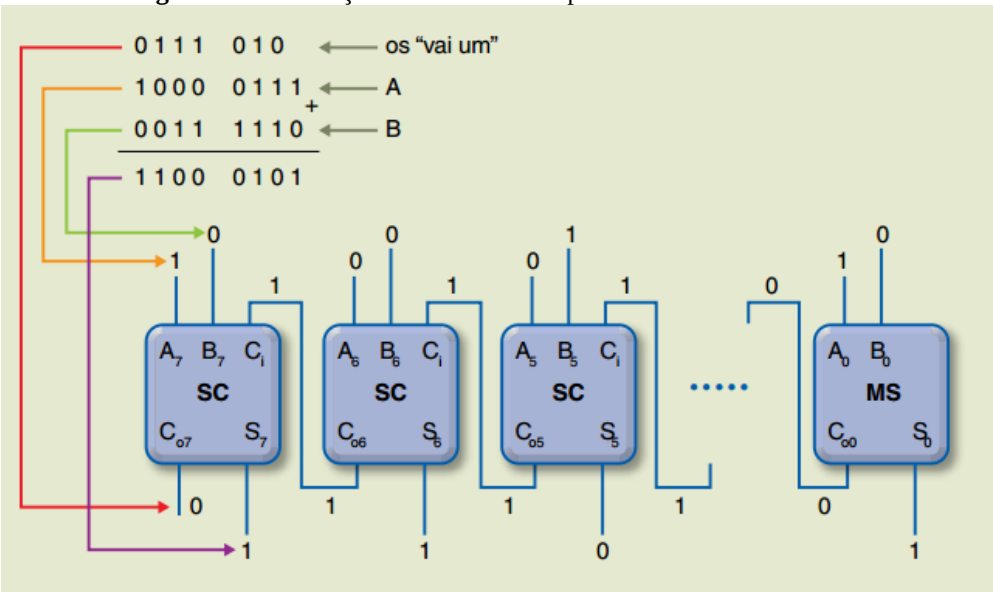
Figura 11: Somador completo simplificado



Fonte: www.albertoferes.com.br

A ilustração abaixo mostra um exemplo de soma binária A+B, sendo A = (1000 0111) e B = (0011 1110), ressaltando que o meio somador usado pode ser substituído por um somador completo sem prejuízo ao resultado, entretanto, o contrário não se aplica.

Figura 12: Associação do Somador completo com o meio somador

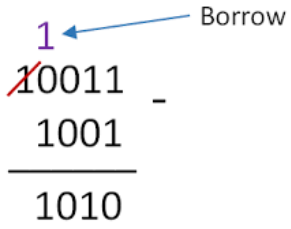


Fonte: www.albertoferes.com.br

1.3 Subtrator

O subtrator nada mais é que um circuito lógico que realiza operações de subtração, esse tipo de lógica necessita de três diferentes entradas, sendo duas para os número a serem subtraídos e uma para o empréstimo, pois a subtração em binário necessita dessa configuração[3].

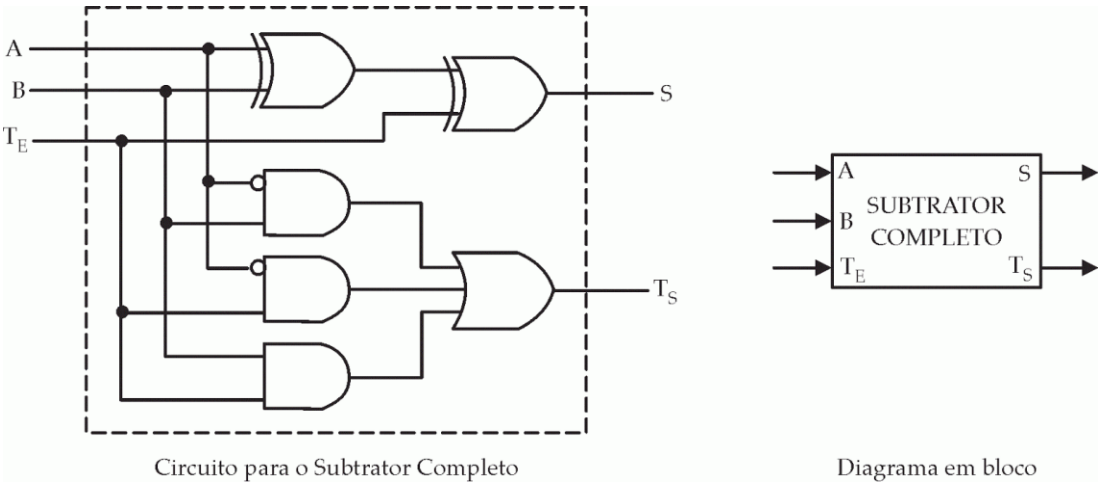
Figura 13: Operação de subtração em binário



Fonte: http://www.bosontreinamentos.com.br

O uso de um circuito subtrator encarece um projeto por haver a necessidade de ser implementado separadamente do somador, por esse motivo, é pouco utilizado. Para contornar essa ocasião, procura -se utilizar a representação numérica de complemento de 2, afim de transformar os números negativos e possibilitar uma soma equivalente. Dessa forma, a subtração é tratada apenas como um caso especial da adição (figura 14).

Figura 14: Representação em bloco e esquemático de um circuito subtrator completo

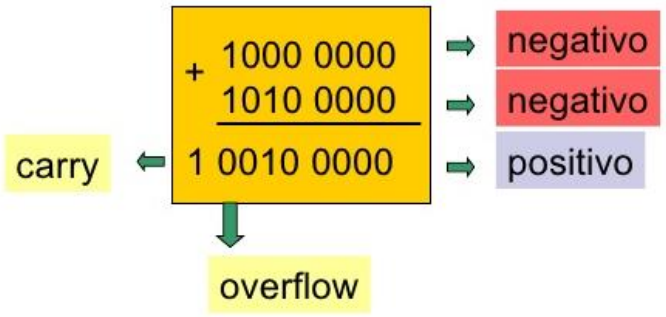


Fonte: http://ula--ula.blogspot.com.br

1.4 *Overflow* aritmético

Ao realizar uma operação aritmética entre números binários existe um tamanho fixo para os termos envolvidos, como por exemplo, 6 ou 8 bits , dessa maneira, há um limite máximo de representação para cada um deles.Nesse sentido, overflow aritmético nada mais é que um estouro dessa capacidade. Se o meu sistema suporta números de até 2 bits o maior número a ser representado é 3 (decimal) ou "11", logo, qualquer soma que exceda esse valor culmina em overflow[3].Esse acontecimento, previsto dessa maneira pode ser detectado durante a implementação do circuito quando após a soma os sinais magnitudes iguais, mudarem.

Figura 14: Exemplo de overflow



Nota-se na figura que a marca desse comportamento está na presença de carry no dígito mais à esquerda do número, o que denota que, tendo acabado as posições no vetor tamanho do número, excedeu um valor.

1.5 Objetivo

Mostrar ao aluno o funcionamento da aritmética binária e o processo de implementação de circuitos somadores.

1.6 Justificativa

É notório o avanço do meio digital atualmente. A cada dia que passa, surgem diferentes sistemas digitais substituindo algum processo mecânico, que apesar dos diversos debates filosóficos a respeito, não deixa de ser bom. O principal intuito do desenvolvimento de qualquer tecnologia nos dias de hoje gira em torno da minimização do esforço humano e redução do tempo gasto com determinada atividade, agilizando os processos. Um exemplo prático disso é tema deste relatório, os circuitos que realizam operações aritméticas, isso porque, a matemática está em absolutamente tudo, e pode apresentar um comportamento facilmente previsível, que resulta em cálculos simples, ou complexo que torna o processo de cálculo humanamente impossível.

Considerando os aspectos cotidianos que envolvem a todo tempo algum processo tecnológico que realiza cálculos aritméticos para cumprir as suas tarefas, como por exemplo, um computador, é de extrema importância o entendimento a respeito do funcionamento desses circuitos. O domínio desse tipo de conhecimento permite conhecer maneiras de se trabalhar de maneiras mais simples e eficientes com as operações aritméticas.

2 EXPERIMENTOS E DISCUSSÃO

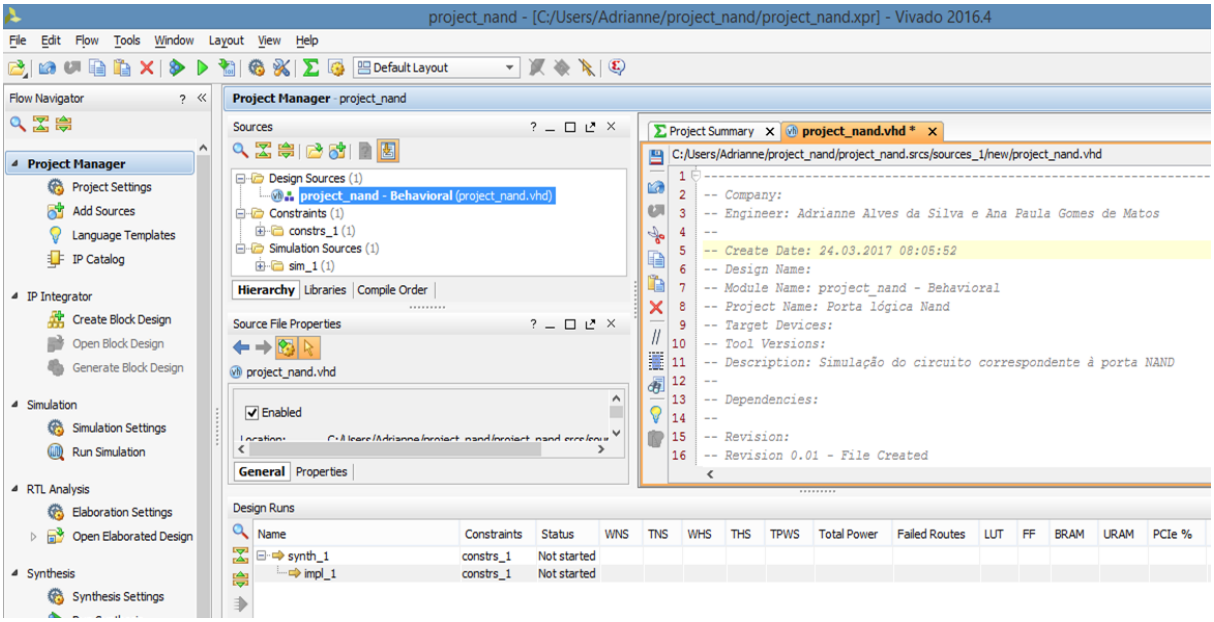
O software utilizado para codificar e simular os experimentos foi o VIVADO, versão 2013.4, rodando no sistema operacional Windows 8, em uma máquina pessoal. Parte do experimento foi realizado em sala, nos computadores da universidade.

2.1 Preparação do ambiente de simulação

O passo inicial para a realização do experimento foi criar um novo projeto no *software* Vivado. A configuração do mesmo foi realizada adicionando a placa Basys 3 ao projeto, por meio da aba "*add or create constraints*". Após esses passos foi necessário apenas criar o arquivo relativo ao *design*, adicionando o arquivo com o tipo vhdl, para que o código pudesse ser implementado nesta linguagem. O módulo foi então definido de acordo com o número de portas de entrada e saída do circuito referente à cada atividade solicitada, nomeadas as entradas, em sua maioria como A, B, va, vb , Cin,sel e Saída com S.

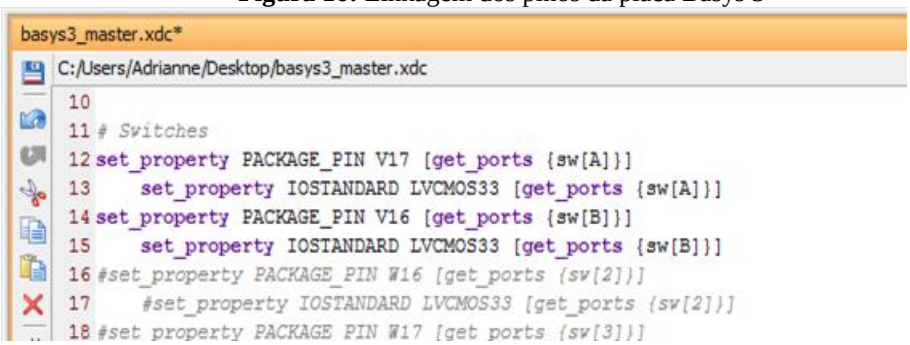
É necessário então ativar na aba de *project settings, bitstream* o arquivo bin_file que possibilita escrita binária sem uso do cabeçalho. Assim, obteve-se o ambiente demonstrado na figura a seguir.

Figura 15: Ambiente de trabalho



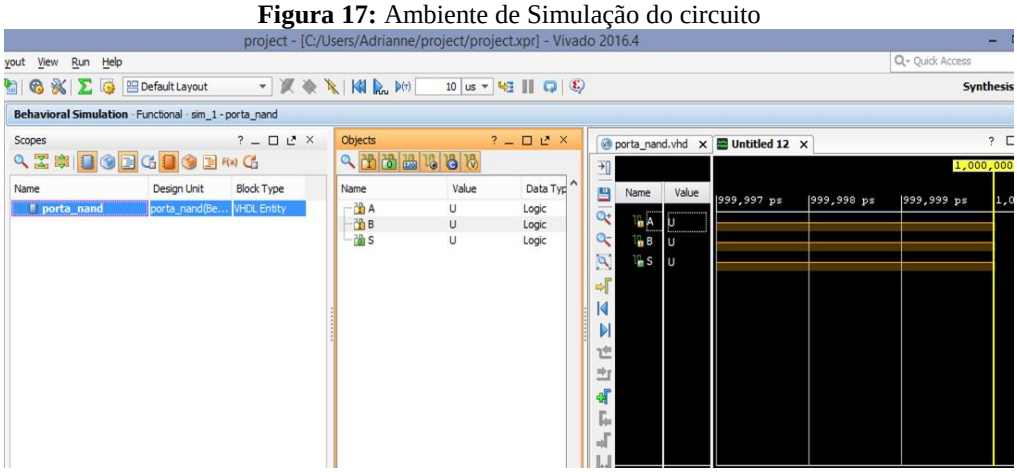
Sendo assim, foi possível abrir o arquivo vhdl definido e implementá-lo. Para cada atividade, forneceu-se ao arquivo o código e as expressões booleanas adequadas. Por sequência do passo-a-passo fornecido, foram linkados os pinos de entrada e saída da placa Basys com os dispositivos de entrada e saída implementadas. No arquivo basys3.xdc (**figura 8**), isso foi feito por meio da retirada do comentário das linhas referentes ao número de entradas e saídas presentes em cada atividade, além da adequada atribuição de cada uma das portas, segundo a nomenclatura adotada (A, B,va,vb,,S), essa atividade seria essencial para a verificação diretamente numa FPGA.

Figura 16: Linkagem dos pinos da placa Basys 3



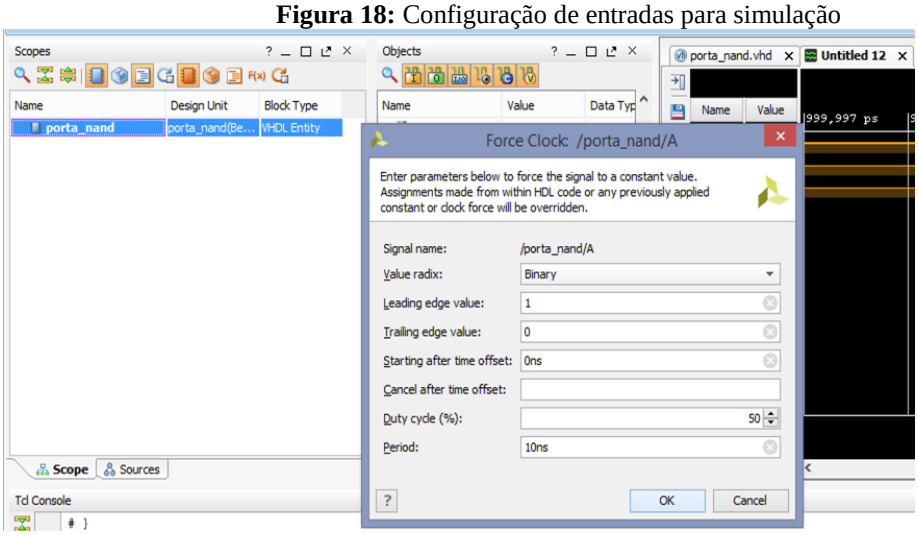
Após todos esses passos foi necessário sintetizar o circuito por meio do *Run Synthesis* , gerando um esquemático facilmente visualizado no *RTL Analysis* . Nesse ponto, é que tornou-se possível realizar a simulação do experimento, bastando clicar em *run simulation* que gera um ambiente de simulação (**figura 17**).

Figura 17: Ambiente de Simulação do circuito



Nesse passo, é possível configurar os valores referentes à cada entrada, assim como o tempo de mudança de sinal (0 ou 1) para a geração de ondas . Isso é feito clicando sobre cada entrada com o botão direito, na opção de *Force Clock*, conforme a figura a seguir.

Figura 18: Configuração de entradas para simulação



A primeira configuração realizada é a modificação do argumento *value* para o tipo binário informando entradas diferenciadas para os argumento *Leading edge value* e *Trailing edge value*, que seja 0 ou 1 . Define-se um período para as ondas, diferentes para cada entrada. É legal se o valor de uma for o dobro da outra, por exemplo. E então, basta executar a simulação.

2.2 Experimento

O experimento consistiu em três diferentes atividades envolvendo o multiplexador, que leva em consideração uma opção de SEL para determinar o que será feito, o complemento de 1 de um número, que como explicado anteriormente diz respeito à representação de um número negativo e circuitos somadores e subtratores.

2.2.1 Complemento de 1

O experimento consistiu em codificar e simular um circuito que realizasse o complemento de 1 de um número de 3 bits, possuindo ainda uma seletora, tal que se SEL fosse 0 deveria retornar apenas o número e se 1, o seu complemento. Esse problema é relativamente

simples, possui 4 entradas e 3 saídas pois cada número possui três bits . Assim, a tabela verdade obtida foi:

Tabela 2: Tabela verdade complemento de 1 com sel

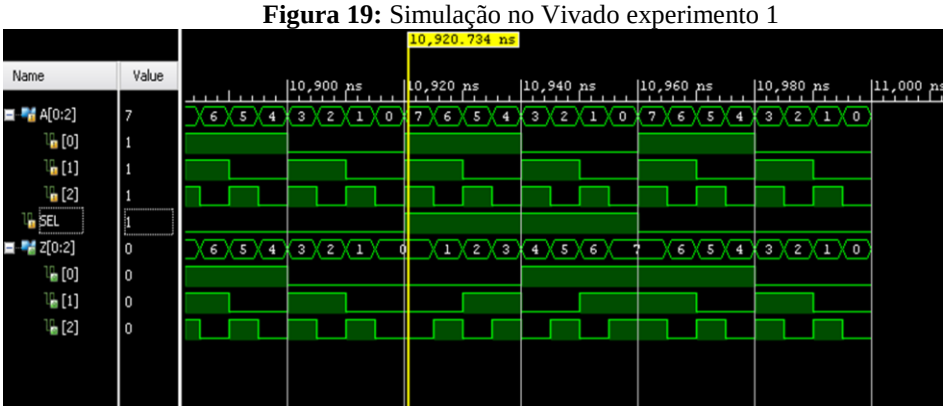
SEL	A	B	C	A	B	C
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	1
1	0	1	1	1	0	0
1	1	0	0	0	1	1
1	1	0	1	0	1	0
1	1	1	0	0	0	1
1	1	1	1	0	0	0

Dessa forma, bastava testar se o SEL era 0 ou 1 e caso 1 inverter todos os números, dessa maneira, obteve -se o seguinte código fonte:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity projeto_3_1 is
   Port ( A : in BIT_VECTOR (0 to 2);
          SEL : in BIT;
          S : out BIT_VECTOR (0 to 2));
end projeto_3_1;

architecture Behavioral of projeto_3_1 is

begin

process (A,SEL)
   begin
      if (SEL = '0') then S<= A;
      elsif(SEL = '1') then S<= not(A);
      end if;
   end process;

end Behavioral;
```

Após a sintetização e checagem do esquema gerado (ver no fim do arquivo), obteve-se uma onda coerente com a simulação.

Figura 19: Simulação no Vivado experimento 1



2.2.2 Detecção de Overflow

A proposta desse experimento consiste apenas na elaboração de um circuito voltado única e exclusivamente para a detecção de overflow, pensando em ser utilizado em conjunto com um circuito somador. Levando em consideração que o overflow pode ser observado considerando os bits de sinal, tal que se ambos forem iguais, ocorre overflow. Assim, a tabela - verdade obtida está expressa a seguir, considerando A e B como os dígitos de sinal de cada número e C o resultado da soma equivalente, já representado em complemento de 2.

Tabela 3: Tabela verdade detecção de overflow

A	B	C	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

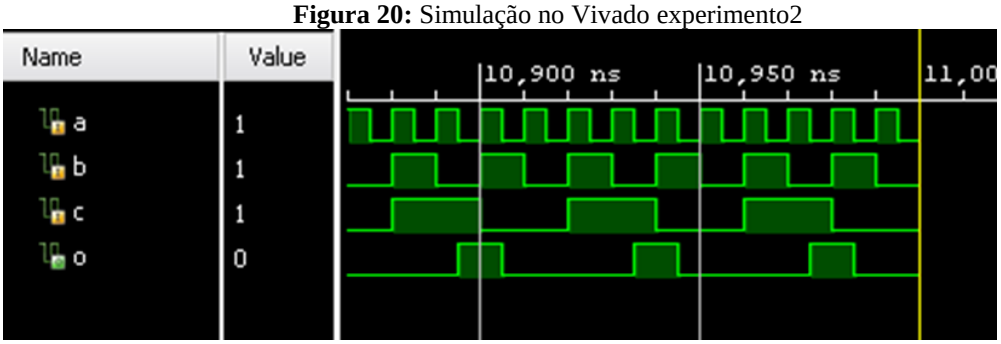
Têm-se que A e B são os bits de sinais dos dois números que estão sendo somados e acontece overflow quando, sendo A e B iguais, o resultado da soma (C) é diferente, assim, sinaliza-se overflow = ‘1’.Dessa tabela retira-se a expressão que deu origem ao código VHDL a seguir.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity project_3_2 is
    Port (
        a: in BIT;
        b : in BIT;
```

```
        c : in BIT;
        o : out BIT);
end project_3_2;

architecture Behavioral of project_3_2 is

begin

process (a,b,c)
   begin
      o <= (not(A) and not(b) and c) or (a and b and not(c));
   end process;

end Behavioral;
```

Esse comportamento de overflow, pode ser claramente observado nas ondas a seguir(figura 20), e o esquemático encontra-se ao fim desse documento.

Figura 20: Simulação no Vivado experimento2



2.2.3 Circuito Somador Subtrator

O circuito Somador e subtrator sugerido gira em torno da ideia de que esteja trabalhando com os números em complemento de 2, e dado o enunciado, parte-se do pressuposto de que A é sempre um número positivo. A decisão entre soma e subtração é dada através de uma sel ou seletora, tal que se esse valor for igual a 0 , qualifica a soma, se 1 , a subtração.

Em caso de subtração realiza-se a soma de A com o complemento de 2 de B. Sendo este um circuito um pouco maior, é possível fazê-lo em módulos, em uma programação estrutural, dessa forma, têm-se um código VHDL para cada microprocesso. É importante salientar que para a soma de cada 2 bits é necessário um circuito somador, por tanto, para 3 bits seriam necessários três somadores.

A implementação do somador, utilizando a biblioteca *IEEE.STD_LOGIC_ARITH.ALL* é mais simplificada, entretanto, se esse caminho não for tomado, podemos relacionar a soma binária entre dois números na seguinte tabela :

Tabela 4: Tabela verdade soma binária

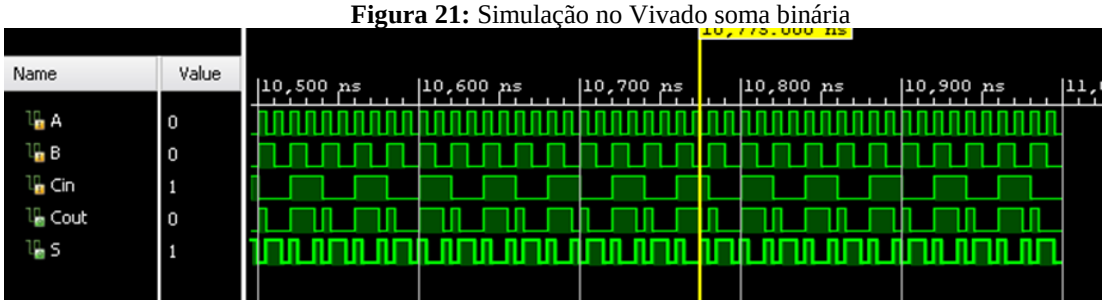
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Observamos dessa tabela que a soma entre dois bits é na verdade a XOR entre ele. Entretanto, na soma é necessário considerar o carry in, que seria o que chamamos de “vai um”, dessa maneira, seria a XOR entre A,B e Cin. Nesse sentido, foi construído o seguinte somador, sendo este um somador completo(tabela 5) :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture Behavioral of somador is

begin

S <= (A xor B xor Cin);
Cout <= (A and B) or (A and Cin) or (B and Cin);

end Behavioral;
```

As formas de onda se configuram como uma forma mais eficiente de visualização do funcionamento, por esse motivo, observa-se-a abaixo.

Figura 21: Simulação no Vivado soma binária



A tabela que representa um circuito somador completo é uma boa maneira de entender a origem das expressões utilizadas, dessa maneira, obteve-se :

Tabela 5: Tabela verdade soma binária

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

Como foi dito anteriormente, são somados dois a dois bits por vez, assim, seriam necessárias três estruturas semelhantes a essa para somar os três bits sugeridos pela questão, assim, numa classe em nível hierárquico maior, construímos essa estrutura segundo o código abaixo :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity somar3bits is
   Port ( va : in BIT_VECTOR (2 downto 0);
          vb : in BIT_VECTOR (2 downto 0);
          cin : in BIT;
          vs : out BIT_VECTOR (2 downto 0);
          overflow : out BIT);
end somar3bits;

architecture Behavioral of somar3bits is

-- Declarando os componentes necessários

component somador
   Port ( A : in BIT;
          B : in BIT;
          Cin : in BIT;
          Cout : out BIT;
          S : out BIT);
end component;

-- Como precisaremos somar 3 vezes, precisamos de 3 sinais

SIGNAL S1 : BIT;
SIGNAL S2 : BIT;
SIGNAL S3 : BIT;

begin
```

```
-- Relizando a conexão entre as portas

L1 : somador port map (A => va(0), B=> vb(0), Cin => cin , S => vs(0), Cout=> S1 );
L2 : somador port map (A => va(1), B=> vb(1), Cin => S1 , S=> vs(1), Cout=> S2 );
L3 : somador port map (A => va(2), B=> vb(2), Cin => S2 , S=> vs(2), Cout=> S3 );

end Behavioral;
```

É importante lembrar que para decidir a operação a ser feita foi utilizada uma seletora ou sel, cuja estrutura está apresentada a seguir :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity seletor is
   Port ( va : in BIT_VECTOR (2 downto 0);
          vb : in BIT_VECTOR (2 downto 0);
          sel : in BIT;
          vs : out BIT_VECTOR (2 downto 0));
end seletor;

architecture Behavioral of seletor is

begin

process (va, vb, sel)
   begin
      if (sel = '0') then vs<= va;
      elsif(sel = '1') then vs<= vb;
      end if;
   end process;

end Behavioral;
```

Assim, caso sel recebesse o valor 0, deveria realizar a soma, se recebesse o valor 1, deveria realizar a subtração. Pensando no uso do complemento de 2 para a subtração, foi implementado um circuito responsável por esse feito, cujo código encontra-se a seguir:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity inversor is
          Port ( va : in BIT_VECTOR (2 downto 0);
          vs : out BIT_VECTOR (2 downto 0));
end inversor;

architecture Behavioral of inversor is
```

```
begin

vs <= not(va) xor "001";


end Behavioral;
```

Todos esses módulos, na programação estruturada do VHDL necessita de uma classe ou circuito principal, por assim dizer, neste são feitas todas as conexões necessárias com os componentes produzidos. Dessa maneira, desenvolveu-se o seguinte código :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity somasubtracao is
        Port ( va : in BIT_VECTOR (2 downto 0);
        vb : in BIT_VECTOR (2 downto 0);
        sel : in BIT;
        vs : inout BIT_VECTOR (2 downto 0);
        overflow : out BIT);
end somasubtracao;

architecture Behavioral of somasubtracao is

component somar3bits
        Port ( va : in BIT_VECTOR (2 downto 0);
        vb : in BIT_VECTOR (2 downto 0);
        cin : in BIT;
        vs : out BIT_VECTOR (2 downto 0)
        );
end component;

component inversor
        Port ( va : in BIT_VECTOR (2 downto 0);
        vs : out BIT_VECTOR (2 downto 0));
end component;

component seletor
        Port ( va : in BIT_VECTOR (2 downto 0);
        vb : in BIT_VECTOR (2 downto 0);
        sel : in BIT;
        vs : out BIT_VECTOR (2 downto 0));
end component;

-- SINAIS

SIGNAL L1 : BIT_VECTOR (2 downto 0);
SIGNAL L2 : BIT_VECTOR (2 downto 0);
```

begin

inversora : inversor port map (va=>vb, vs => L1);
opcao : seletor port map(va => vb, vb => L1, sel => sel, vs => L2);
soma3 : somar3bits port map (va => va , vb=>L2, cin => sel , vs=>vs);

overflow <= (va(2) and not(vb(2)) and vs(2)) or (va(2) and vb(2) and not (vs(2)));

end Behavioral;

Por fim, após a devida configuração do ambiente de simulação , utilizando o *force clock* para introduzir os valores iniciais e o período da onda, sendo o último caracterizado por uma entrada possuir o dobro da outra, obteve-se o esquemático apresentado no fim deste documento, e as ondas a seguir.

Figura 21: Hierarquia do projeto

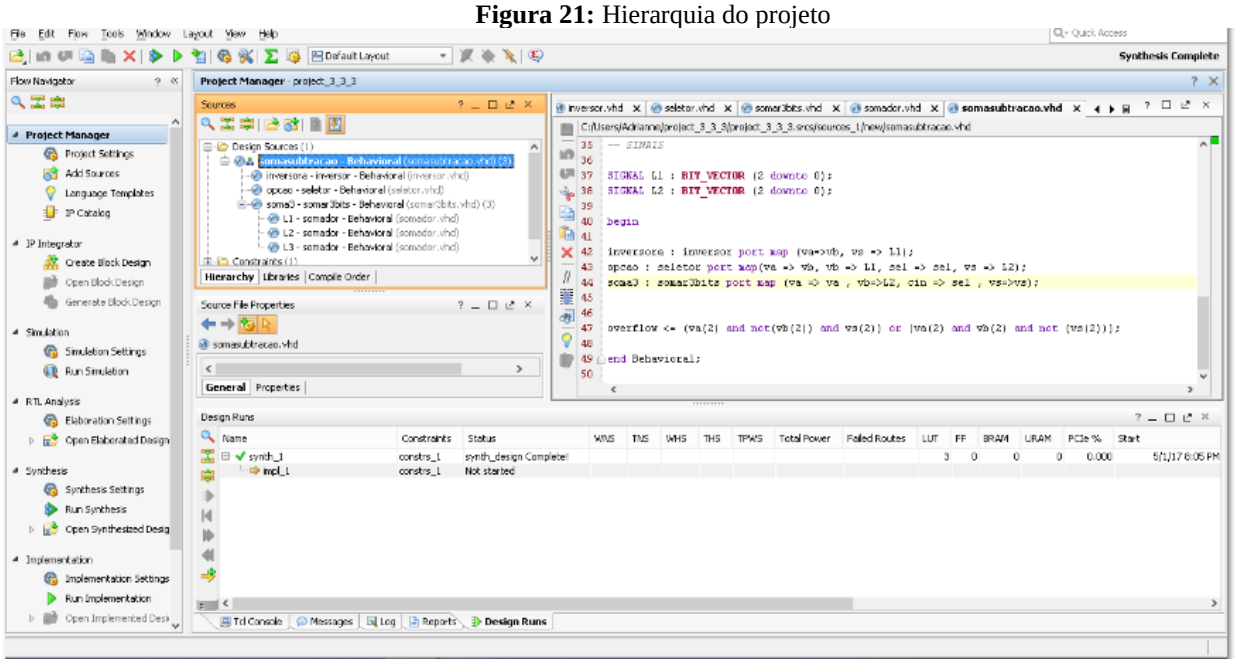
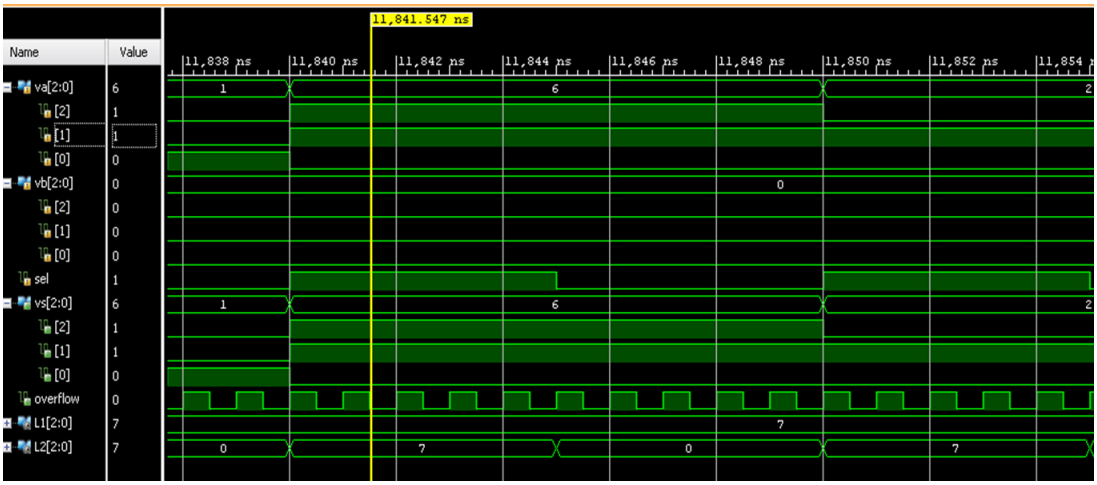


Figura 22: Ondas da simulação do circuito somador subtrator no software vivado



3 CONCLUSÕES

Diante do que foi apresentado anteriormente, nota-se que todas as atividades solicitadas no experimento foram executadas com resultados satisfatórios, ou seja, foi possível simular um circuito que permite realizar o complemento de 1 de um número com 3 bits conforme solicitado no projeto 1, foi ainda simulado o projeto 2 que se refere a um circuito capaz de detectar a condição de *overflow*, por fim, foi simulado o projeto 3 referente a um circuito somador/ subtrator de 3 bits codificado na forma de complemento de 2 e que também indica a condição de *overflow*. A maior dificuldade para a realização deste experimento foi o fato das aulas de laboratórios e as da teoria estarem desalinhadas fazendo com que os alunos que estão tendo o primeiro contato com a disciplina tenham mais dificuldade em entender o que é solicitado e como deve ser feito, requerendo assim mais tempo para a disciplina.

REFERÊNCIAS BIBLIOGRÁFICAS

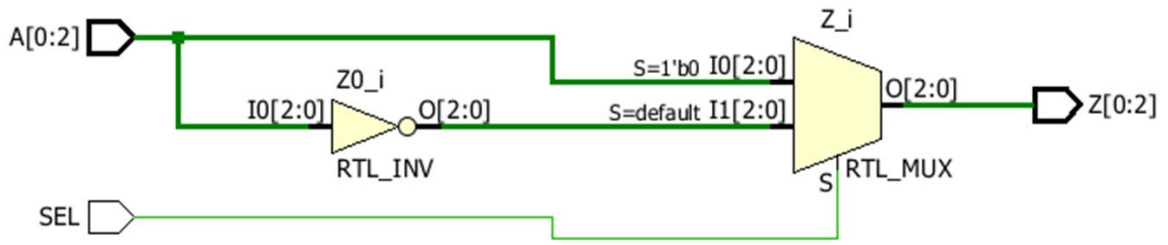
[1] FONSECA FILHO, Cléuzio. **História da computação: O Caminho do Pensamento e da Tecnologia**. EDIPUCRS, 2007. p. 56-58.

[2] AMARAL, Valder Moreira. **Eletrônica Digital**. São Paulo: Fundação Padre Anchieta, 2011. p. 31-33. 4 v.

[3] DUECK, Robert K. **Digital Design with CPLD applications and VHDL**. Granta Books, 2001.

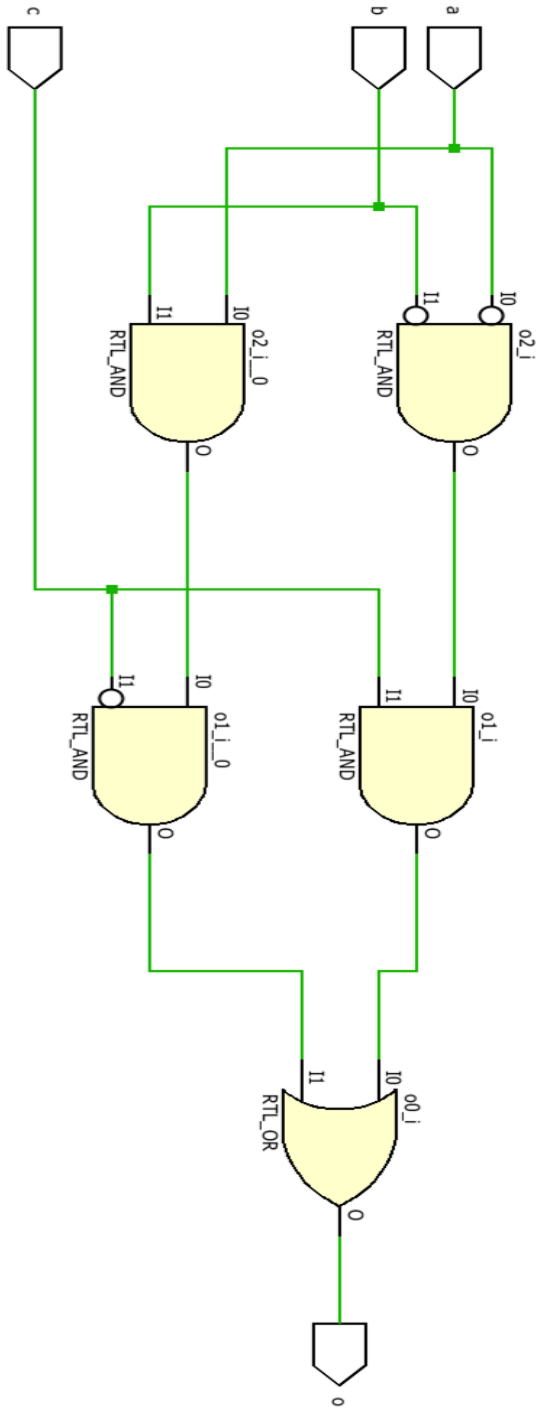
DIAGRAMAS ESQUEMÁTICOS

Figura 23: Complemento de 1 com a seletora



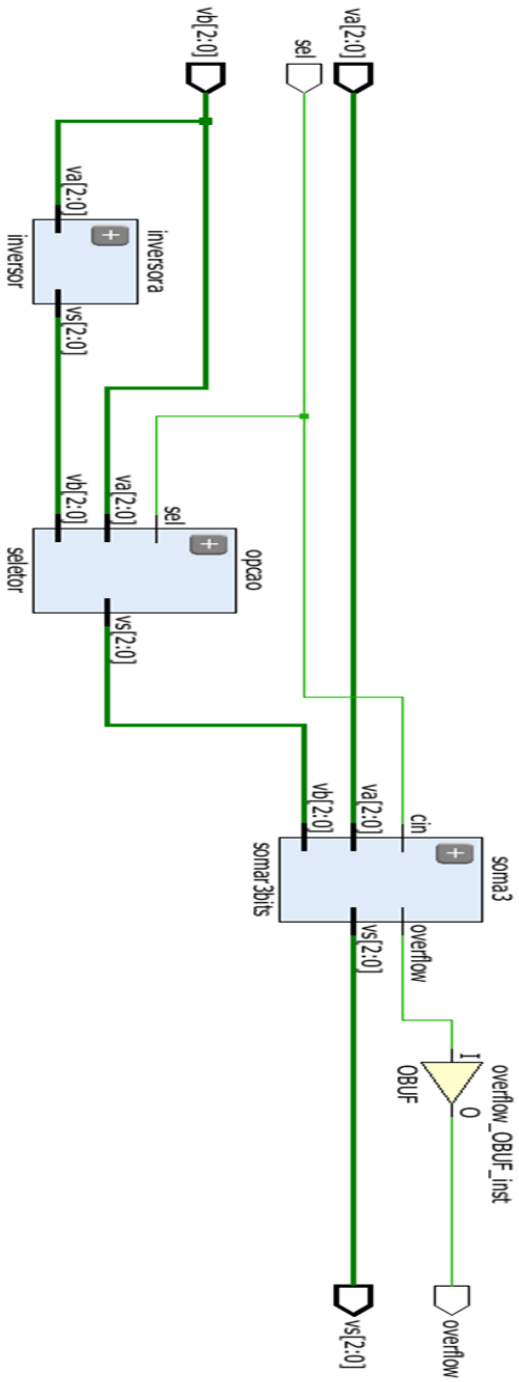
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Somadores Subtratores	1.0	19/04/2017	**Revisado por:**	Ana Paula Gomes de Matos

Figura 23: Detecção de Overflow



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Somadores Subtratores	1.0	19/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 24: Circuito somador subtrator



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Somadores Subtratores	1.0	19/04/2017	**Revisado por:**	Ana Paula Gomes de Matos