

**Universidade Federal de Campina Grande - UFCG**  
**Centro de Engenharia Elétrica e Informática - CEEI**  
**Departamento de Engenharia Elétrica - DEE**

**Nome:** Alysson Machado de Oliveira Barbosa

**Email:** alysson.barbosa@ee.ufcg.edu.br

**Disciplina:** Laboratório de Circuitos Lógicos

**Professora:** Fernanda Cecília Correia Lima Loureiro

**Experimento 05 - Aritmética Binária com HDL (verilog)**

**Objetivo Geral**

Este experimento consiste na realização de quatro partes específicas e possui como objetivo geral o estudo da aritmética binária, bem como do projeto e implementação dos circuitos lógicos que realizam essas operações, ou seja, soma e subtração, utilizando Verilog. Nesse experimento, será realizado a tarefa de quatro objetivos distintos:

- **Somador Binário de 4 Bits;**
- **Seletor de Função Igualdade/Complemento de 1;**
- **Detector de Estouro de Capacidade;**
- **Somador/Subtrator de 4 bits;**

## Objetivo 1

Especificação e implementação de um Somador Binário, com o projeto realizado utilizando a linguagem Verilog.

### Blocos Lógicos 1

Nesse experimento, foi realizado a implementação de um somador binário de 4 bits, conforme o diagrama lógico abaixo:

**Figura 01 - Bloco Lógico de um Somador de 4 bits**



Fonte: Imagem Autoral

Observando o bloco lógico acima, teremos as seguintes entradas e saídas:

- **A e B** = Números em binário com 4 bits;
- **Ci** = Peso inicial do “Carry” (para o somador, sempre deve ser 0);
- **S** = O resultado da soma de A e B;
- **Ts** = Armazena o “Carry” de cada um dos bits de S;

Para projetar o somador, devemos primeiro pensar em fragmentar a tarefa em pequenas partes. Desse modo, é necessário a implementação de um módulo menor que realizar a operação de soma bit a bit, conforme o diagrama lógico abaixo:

**Figura 02 - Bloco Lógico de um Somador Completo de 1 bit**



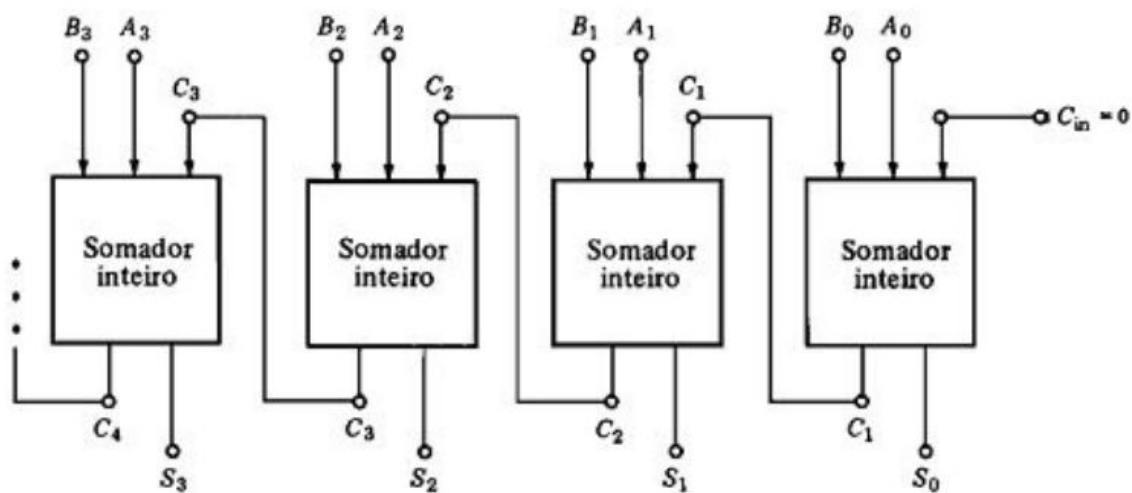
Fonte: Imagem Autoral

Cujas entradas e saída são análogas ao diagrama lógico para o somador de 4 bits, exceto pela quantidade de bits sendo operado:

- **A** e **B** = Números em binário com 1 bits;
- **C<sub>i</sub>** = Peso do “Carry”;
- **S** = O resultado da soma de A e B;
- **T<sub>s</sub>** = Armazena o “Carry” de S;

É necessário realizar a instanciação do módulo que realiza o Somador Completo de 1 bit, em cadeia, n quantidade de vezes para um projeto de Somador de N bits. No caso em análise, teremos que instanciá-lo 4 vezes para obter um Somador de 4 bits.

**Figura 03 - Bloco Lógico de um Somador de 4 bits a partir de Somadores Completos de 1 bit**



Fonte: Imagem Autoral

## Expressão Lógica 1

Para realizar a implementação deste circuito somador de 1 bit, devemos ter a seguinte tabela verdade:

**Figura 04 - Tabela Verdade de um Somador Completo de 1 bit**

A	B	Te	S	Ts
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: Imagem Autoral

Analisando a tabela verdade acima, é possível implementar o Mapa de Karnaugh para obter uma expressão lógica simplificada:

**Figura 05 - Mapa de Karnaugh para o Somador Completo de 1 bit**

**S**

A / B Te	0 0	0 1	1 1	1 0
0	0	1	0	1
1	1	0	1	0

**Ts**

A / B Te	0 0	0 1	1 1	1 0
0	0	0	1	0
1	0	1	1	1

Fonte: Imagem Autoral

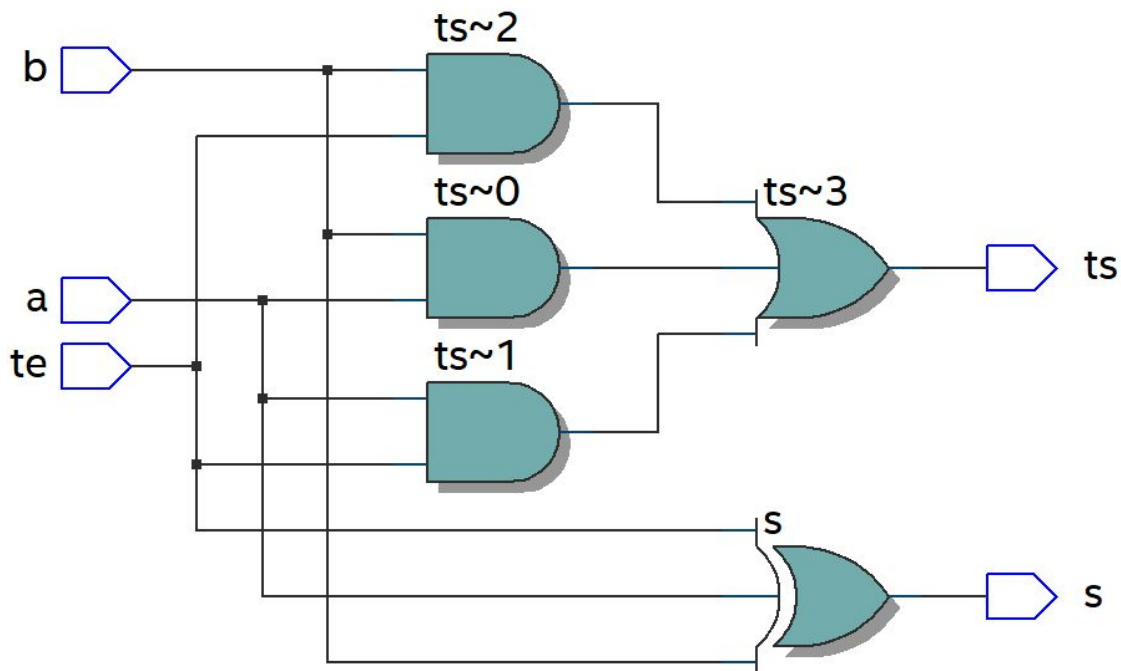
Realizando a simplificação do Mapa de Karnaugh acima, teremos as seguintes expressões lógicas simplificadas:

$$S = A \wedge B \wedge Te$$
$$Ts = ATe + BTe + AB$$

## Circuito 1

Abaixo foi definido o circuito responsável por implementar o Somador Completo de 1 bit:

**Figura 06 - Circuito do Somador Completo de 1 bit**



Fonte: Imagem Autoral

## Implementação em Verilog 1

Para implementar o Somador de 4 bits usando a linguagem de Descrição de Hardware Verilog, deveremos criar um módulo que realize o Somador Completo de 1 bit e, em seguida, colocá-lo em cadeia 4 vezes para implementar o módulo Somador de 4 bits. É importante frisar que a entrada “Ci” deve ser sempre 0 para que o circuito funcione corretamente.

**Figura 07 - Somador Completo de 1 bit em Verilog**

```
1 module somador_completo
2   (
3     input a, b, te,
4     output s, ts
5   );
6
7   assign s = (a ^ b) ^ te;
8   assign ts = (a & b) | (a & te) | (b & te);
9
10  endmodule
```

Fonte: Imagem Autoral

**Figura 08 - Somador de 4 bits em Verilog**

```
1 module somador_4bits
2   (
3     input [3:0]a, b,
4     input ci,
5     output [3:0]s, co
6   );
7
8   somador_completo sc1(a[0],b[0], ci , s[0], co[0]);
9   somador_completo sc2(a[1],b[1], co[0], s[1], co[1]);
10  somador_completo sc3(a[2],b[2], co[1], s[2], co[2]);
11  somador_completo sc4(a[3],b[3], co[2], s[3], co[3]);
12
13 endmodule
```

Fonte: Imagem Autoral

Para verificar o funcionamento dos circuitos implementados, foi criado um módulo de teste. Abaixo é possível analisar o código, o diagrama lógico completo e o diagrama de estado.

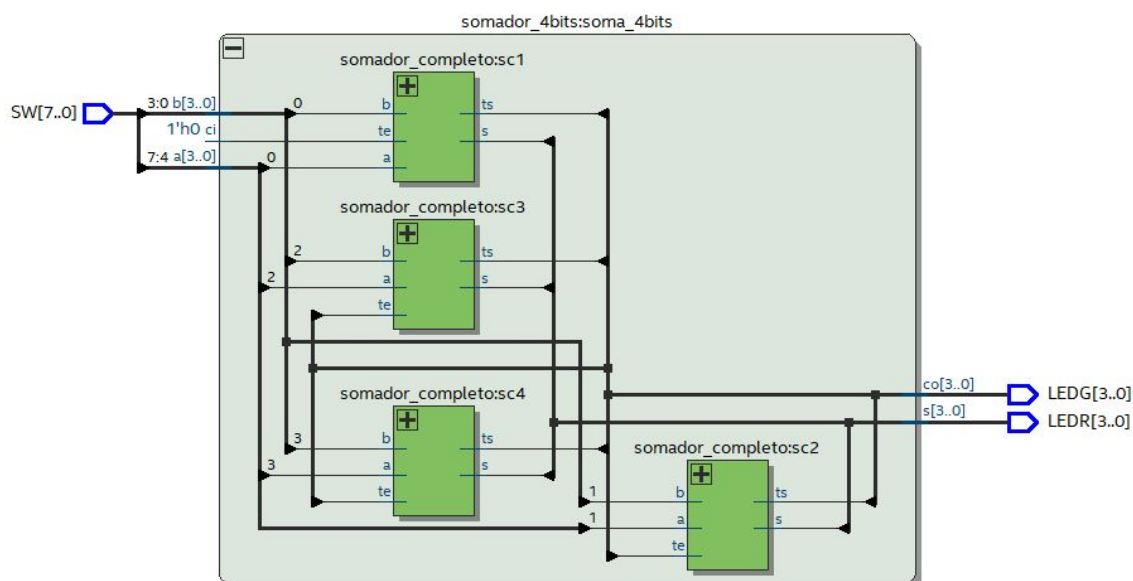
**Figura 09 - Implementação do Módulo de Teste**

```
1 module Mod_Testel
2   (
3     input [7:0]SW,
4     output [3:0]LEDR, LEDG
5   );
6
7   somador_4bits soma_4bits(SW[7:4], SW[3:0], 0, LEDR, LEDG);
8
9 endmodule
```

Fonte: Imagem Autoral

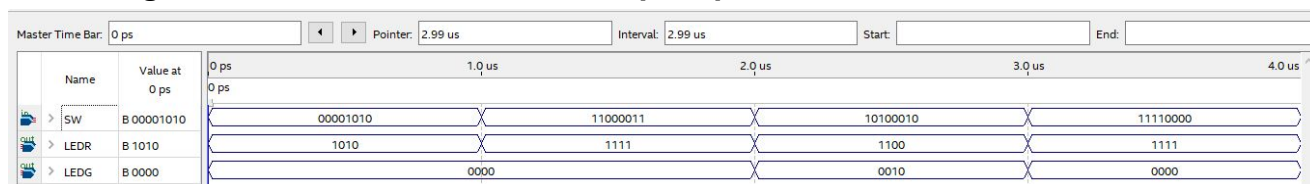
- **[7:0]SW** = definimos um vetor de 8 bits em que os primeiros 4 bits é relativo à parte A a ser somada e os 4 bits restantes é referente a parte B da soma;
- **[3:0]LEDR** = saída de 4 bits que mostra o resultado obtido da soma;
- **[3:0]LEDG** = saída que mostra os “Carrys” respectivo a cada bit;

**Figura 10 - Diagrama Lógico Completo do Somador de 4 Bits**



Fonte: Imagem Autoral

**Figura 11 - Visualização de 4 exemplos para o Somador de 4 bits**



Fonte: Imagem Autoral

O resultado obtido no diagrama de estado acima pode ser resumido na seguinte tabela verdade:

SW[7]	SW[6]	SW[5]	SW[4]	SW[3]	SW[2]	SW[1]	SW[0]	LEDR [3]	LEDR [2]	LEDR [1]	LEDR [0]	LEDG [3]	LEDG [2]	LEDG [1]	LEDG [0]
0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0
1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0
1	0	1	0	0	0	1	0	1	1	0	0	0	0	1	0
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0

## Objetivo 2

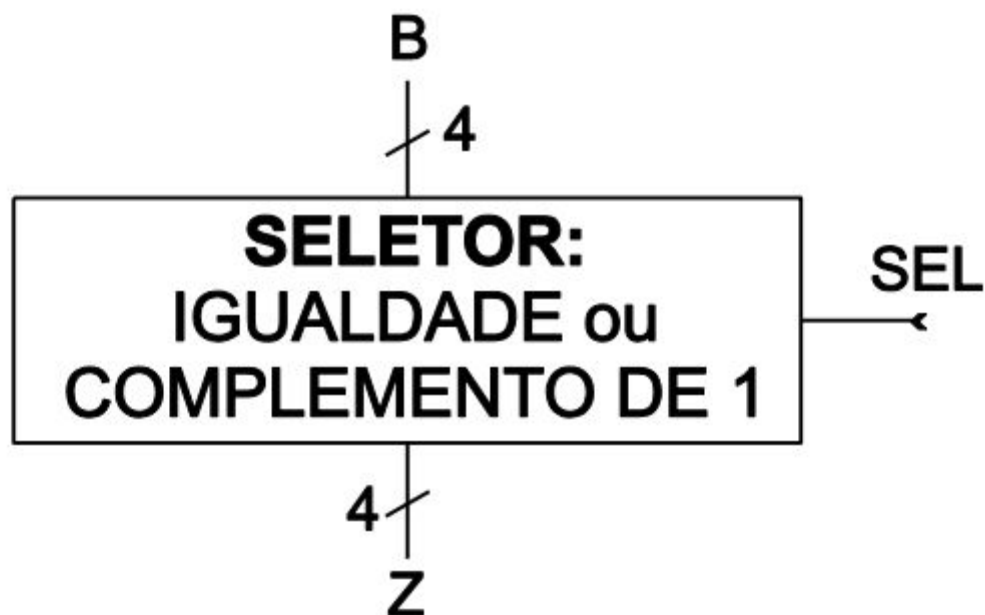
Será realizado a Especificação e implementação de um Seletor de função Igualdade/Complemento de 1, com o projeto realizado utilizando a linguagem Verilog. A função realizada por esse circuito depende do valor da entrada de controle de seleção SEL:

- Se  $SEL = 0$ , a função selecionada é a IGUALDADE e então  $Z = B$ ;
- Se  $SEL = 1$ , a função selecionada é o COMPLEMENTO DE 1 e então  $Z = C1(B)$ ;

## Bloco Lógico 2

Para esse circuito seletor de igualdade/complemento de 1, devemos ter o seguinte bloco lógico:

**Figura 12 - Bloco Lógico do Seletor de Igualdade/Complemento de 1**



Fonte: Imagem Autoral

Observando o bloco lógico acima, podemos analisar as seguintes entrada e saídas do circuito:

- **B** = Número em binário com 4 bits;
- **SEL** = chave seletora para Igualdade(0) ou Complemento de 1(1);
- **Z** = Saída em número binário com 4 bits;



## Expressão Lógica 2

Para o circuito Seletor de Igualdade/Complemento de 1, teremos a seguinte tabela verdade:

B[3]	B[2]	B[1]	B[0]	SEL	S[3]	S[2]	S[1]	S[0]
1	0	1	0	0	1	0	1	0
1	0	1	0	1	0	1	0	1

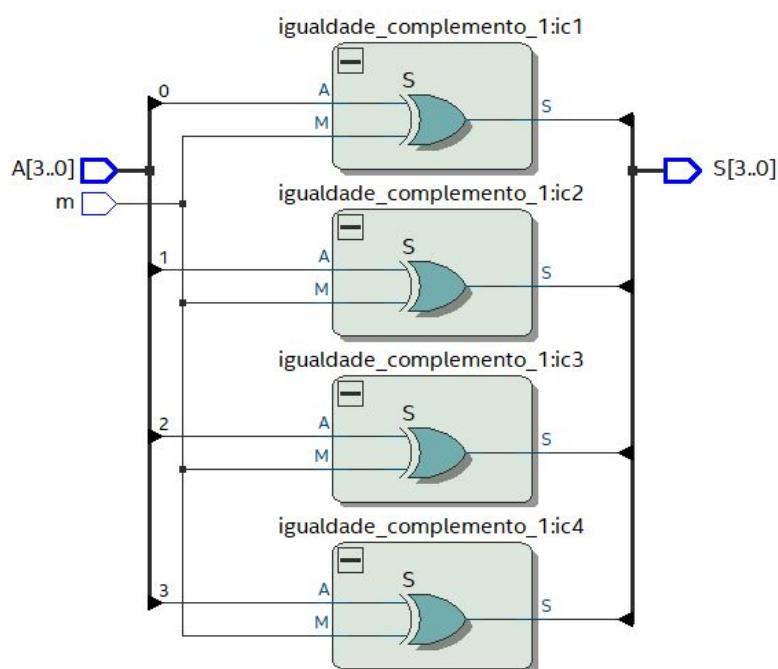
De modo que é possível usar a seguinte expressão lógica para representar tal circuito:

$$S = B \wedge SEL$$

## Circuito 2

Obtendo a expressão lógica, poderemos visualizar o circuito responsável por implementar tal ideia:

**Figura 13 - Circuito do Seletor de Igualdade/Complemento de 1**



Fonte: Imagem Autoral

## Implementação em Verilog 2

Observe abaixo o código necessário para implementar tal circuito usando a linguagem de descrição de hardware verilog:

**Figura 14 - Código em Verilog do Seletor com 1 bit**

```
1 module igualdade_complemento_1
2   (
3     input A, M,
4     output S
5   );
6
7   assign S = A ^ M;
8
9 endmodule
```

Fonte: Imagem Autoral

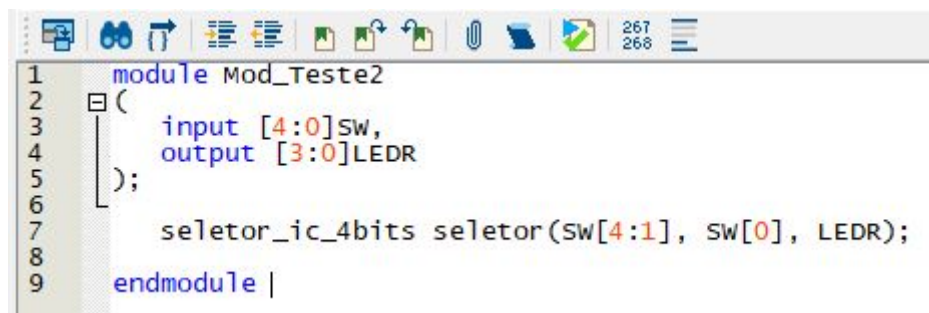
**Figura 15 - Código em Verilog do Seletor com 4 bits**

```
1 module seletor_ic_4bits
2   (
3     input [3:0]A,
4     input m,
5     output [3:0]s
6   );
7
8   igualdade_complemento_1 ic1(A[0], m, S[0]);
9   igualdade_complemento_1 ic2(A[1], m, S[1]);
10  igualdade_complemento_1 ic3(A[2], m, S[2]);
11  igualdade_complemento_1 ic4(A[3], m, S[3]);
12
13 endmodule
```

Fonte: Imagem Autoral

Para testar os módulos implementados, foi criado um outro módulo de teste, observe abaixo os códigos e o resultado obtido:

**Figura 16 - Módulo de teste para o Seletor de Igualdade/Complemento 1**



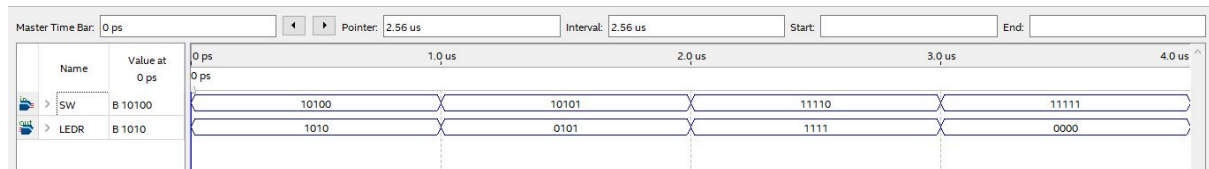
```
1 module Mod_Teste2
2   (
3     input [4:0]SW,
4     output [3:0]LEDR
5   );
6
7   seletor_ic_4bits seletor(SW[4:1], SW[0], LEDR);
8
9 endmodule
```

Fonte: Imagem Autoral

- **[4:0]SW** = 5 bits de entrada, em que o último bit é referente a chave (SEL) e os 4 primeiros bits (B) são os que serão propagados ou invertidos na saída, dependendo do valor da chave.
- **[3:0]LEDR** = 4 bits de saída;

Para o teste implementado, teremos os seguintes exemplos no diagrama de estado:

**Figura 17 - Diagrama de Estado para o Seletor**



Fonte: Imagem Autoral

SW[4]	SW[3]	SW[2]	SW[1]	SW[0]	LEDR [3]	LEDR [2]	LEDR [1]	LEDR [0]
1	0	1	0	0	1	0	1	0
1	1	1	1	0	1	1	1	1
1	0	1	0	1	0	1	0	1
1	1	1	1	1	0	0	0	0

### Objetivo 3

Especificação e implementação de um Detector de estouro de capacidade para ser usado com um somador de números com sinais, codificados em Complemento de 2, com o projeto realizado utilizando a linguagem Verilog.

### Bloco Lógico 3

Abaixo foi implementado o bloco lógico para um Detector de estouro.

**Figura 18 - Bloco Lógico de um Detector de estouro**



Fonte: Imagem Autoral

Em que teremos nos dados de entrada e dados de saída as seguintes representações:

- **S** = Bit mais significativo da soma de A e B;
- **A** = Bit mais significativo de A;
- **B** = Bit mais significativo de B;

### Expressão Lógica 3

Para o detector de estouro, teremos a seguinte tabela verdade, de modo que podemos obter o seu resultado através da Soma de Produtos (SOP).

S (bit sig.)	A (bit sig.)	B (bit sig.)	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	0	1	0

A expressão lógica para o circuito que verifica a ocorrência de Overflow no operador será a seguinte:

$$X = (\sim S)AB + S(\sim A)(\sim B)$$

### Implementação em Verilog 3

Para o problema do circuito Detector de estouro, teremos a seguinte implementação em Verilog:

**Figura 19 - Código em Verilog para o verificador de estouro**

```
1 module estouro
2   (
3     input s, a, b,
4     output x
5   );
6
7   assign x = (~s & a & b) | (s & ~a & ~b);
8
9 endmodule
```

Fonte: Imagem Autoral

Agora foi implementado um módulo de teste para o Detector de estouro:

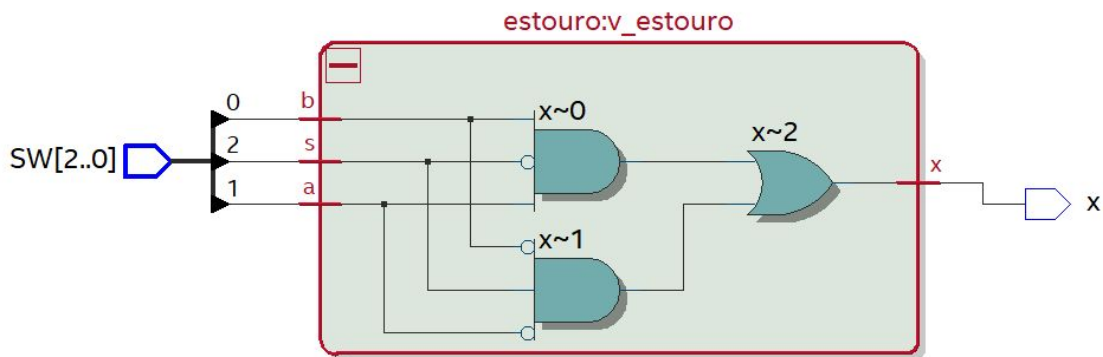
**Figura 20 - Módulo de Teste para o Detector de Estouro**

```
1 module Mod_Test3
2   (
3     input [2:0] sw,
4     output x
5   );
6
7   estouro v_estouro(sw[2], sw[1], sw[0], x);
8
9 endmodule |
```

Fonte: Imagem Autoral

- **SW[2]** = Bit mais significativo da soma de SW[1] e SW[0];
- **SW[1]** = Bit mais significativo de uma das parcelas da soma;
- **SW[0]** = Bit menos significativo de uma das parcelas da soma;

**Figura 21 - Diagrama Lógico do Detector de Estouro**



Fonte: Imagem Autoral

Testando algumas entradas e algumas saídas no circuito acima, é possível obter a seguinte tabela verdade:

SW[2]	SW[1]	SW[0]	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	0	1	0

#### Objetivo 4

Especificação e implementação de um Somador/Subtrator Binário, com o projeto realizado utilizando a linguagem Verilog. Para a realização deste somador/subtrator completo observe que a subtração deve ser realizada pela soma do complemento de dois do segundo operando.

#### Bloco Lógico 4

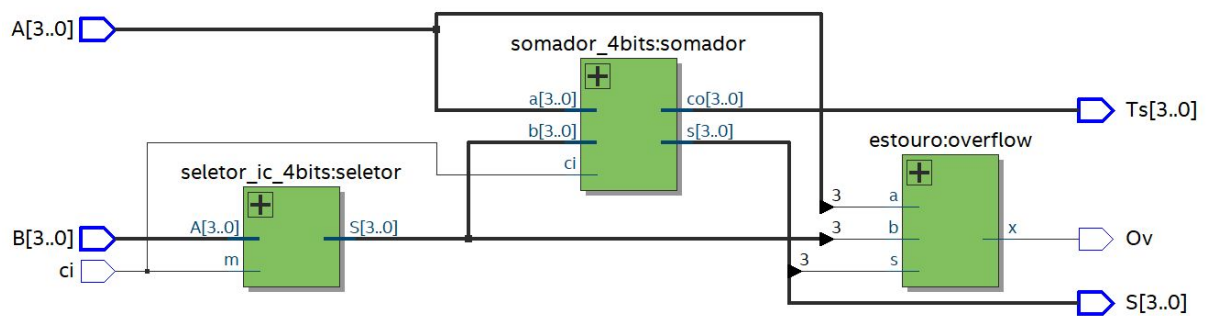
Para implementar esse circuito Somador/Subtrator, devemos utilizar todos os módulos anteriores. Através de uma chave de seleção (Ci), podemos realizar a soma de A e B quando Ci = 0 e a subtração de A e B quando Ci = 1.

A forma de realizar a implementação é aplicando o módulo Seletor de Igualdade/Complemento de 1 em B, em que Ci também estará como chave desse módulo. Ou seja, Quando  $Ci = 0$ , B não mudará e A e B será propagado para o somador com Carry inicial 0. Entretanto, quando  $Ci = 1$ , B será invertido (Complemento de 1) e A e B será propagado para o somador com Carry inicial igual a 1 (Soma mais 1 bit), de modo que tenha sido aplicado Complemento de 2 em B ao final de todo o processo.

Vale a pena ressaltar, que para esse processo funcionar, deve-se considerar na entrada números binários com sinal.

Por fim, o módulo que detecta o estouro estará ao final do circuito, recebendo o bit mais significativo de A, B e S (soma de A com B). Observe o Diagrama Lógico abaixo:

**Figura 22 - Diagrama Lógico do Somador/Subtrator**



Fonte: Imagem Autoral

## Implementação em Verilog 4

Abaixo vamos analisar a implementação dessa ideia do circuito Somador/Subtrator em Verilog:

**Figura 23 - Implantação do Somador/Subtrator**

```

1  module somador_subtrator
2  (
3      input [3:0] A, B,
4      input ci,
5      output [3:0] S, Ts,
6      output Ov
7  );
8
9      wire [3:0] B_aux;
10     seletor_ic_4bits seletor(B, ci, B_aux);
11     somador_4bits somador(A, B_aux, ci, S, Ts);
12     estouro overflow(S[3], A[3], B_aux[3], Ov);
13
14 endmodule

```

Fonte: Imagem Autoral

Para Testar esse módulo, devemos criar um módulo de teste e analisar os resultados obtidos.

**Figura 24 - Módulo de Teste do Somador/Subtrator**

```

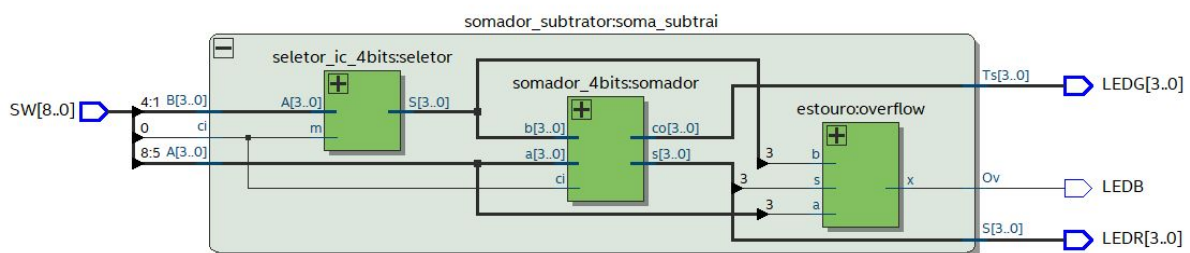
1  module Mod_Test4
2  (
3      input  [8:0]SW,
4      output [3:0]LEDR, LEDG,
5      output LEDB
6  );
7
8      somador_subtrator soma_subtrai(SW[8:5], SW[4:1], SW[0], LEDR, LEDG, LEDB);
9
10 endmodule
11

```

Fonte: Imagem Autoral

- **SW[8:5]** = Número binário com sinal de 4 bits (uma das parcelas da soma);
- **SW[4:1]** = Número binário com sinal de 4 bits (uma das parcelas da soma);
- **SW[0]** = Chave seletora de entrada (Somador/Subtrator);
- **LEDR** = Resultado da operação de A + B ou A - B (binário com sinal de 4 bits);
- **LEDG** = Número binário de 4 bits que armazena os "Carrys" da operação;
- **LEDB** = Número binário de 1 bit que indica se houve overflow na operação;

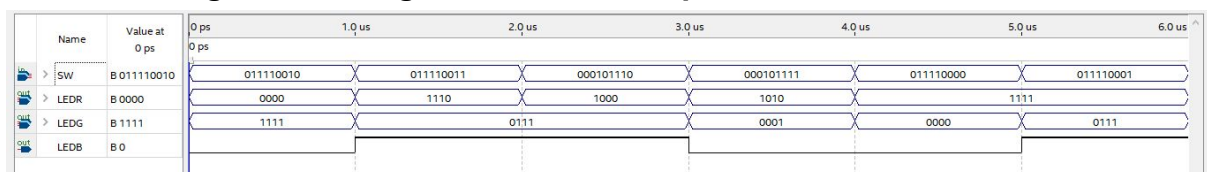
**Figura 25 - Diagrama Lógico do Módulo de Teste**



Fonte: Imagem Autoral

Implementando o diagrama de tempo para o circuito acima, teremos 6 exemplos arbitrários que demonstram a utilização do circuito:

**Figura 26 - Diagramas de Estado para o módulo de teste**



Fonte: Imagem Autoral



Os resultados do diagrama de tempo podem ser observados na seguinte tabela verdade:

SW [8]	SW [7]	SW [6]	SW [5]	SW [4]	SW [3]	SW [2]	SW [1]	SW[ 0]	LED R[3]	LED R[2]	LED R[1]	LED R[0]	LED B
0	1	1	1	1	0	0	1	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	1	1	1	0	1
0	0	0	1	0	1	1	1	0	1	0	0	0	1
0	0	0	1	0	1	1	1	1	1	0	1	0	0
0	1	1	1	1	0	0	0	0	1	1	1	1	0
0	1	1	1	1	0	0	0	1	1	1	1	1	1