

Relatório do 1º Projeto de Laboratório

Abraão Carvalho Gomes, Gabriel Rodrigues da Silva

¹Universidade Federal do Rio de Janeiro (UFRJ)

²EEL480 - Laboratório de Sistemas Digitais

abraao.cg@poli.ufrj.br, gabriel.rodrigues129@poli.ufrj.br

Resumo. Este relatório tem como objetivo mostrar o processo de construção de uma Unidade Lógica e Aritmética (ULA) de 4 bits e 8 operações, bem como o funcionamento dos módulos que a compõem. No fim do relatório, os dados de entrada e o resultado das operações foram simulados na placa FPGA DE2-115.

1. Estrutura da ULA

A ULA é dividida em 8 operações que podem ser selecionadas por meio de um seletor de 3 bits, com a seguinte correspondência:

- 000 - Soma;
- 001 - Subtração;
- 010 - Incremento de 1 (em ‘a’);
- 011 - Troca de Sinal (em ‘a’);
- 100 – Decremento de 1 (em ‘a’);
- 101 - Normalização (em ‘a’);
- 110 - Operação Lógica “AND”;
- 111 - Operação Lógica “XOR”;

As 4 operações principais são: soma, subtração, incremento de 1 e troca de sinal, sendo que todas elas são realizadas utilizando um mesmo módulo chamado somador_4bits que, para cada uma das operações, recebe diferentes valores como entradas. Além disso, a quinta operação ”Decremento de 1” também irá utilizar o somador.

Inicialmente, é necessário analisar a estrutura do somador e ver como é feita a alternação de suas entradas e resultados entre diferentes operações.

1.1. Somador de 4 Bits

O somador utiliza o componente `full_adder` que realiza a soma de dois bits e um carry in (“cin”), resultando em um bit de saída e um carry out (“cout”).

A partir disso, o somador de 4 bits foi construído de forma a somar dois números de N bits, sendo estabelecido inicialmente N = 4 devido às exigências do projeto. Para tanto, utiliza-se uma estrutura de repetição que faz proveito do módulo `full_adder` para somar bit a bit, cada qual é atribuído ao vetor de saída ”sum”.

O somador tem 3 entradas: ‘a’, ‘b’ e “cin” (carry in) e 3 saídas: “sum”, “cout” (flag carry out da ULA) e “overflow” (flag overflow da ULA). Dessa forma, as três entradas são manipuladas na ULA em função do seletor, a fim de obter o resultado desejado, como será visto à frente.

1.2. Seleção de Entradas do Somador

Diversos sinais intermediários foram criados no módulo principal da ULA para que fosse possível fornecer variadas entradas ao somador de 4 bits. São eles:

- B_{comp1} : representa o complemento de um de ‘b’;
- A_{comp1} : representa o complemento de um de ‘a’;
- A_2 : representa o valor de ‘a’ que será utilizado no somador;
- B_2 : representa o valor de ‘b’ que será utilizado no somador;
- $Mux1_zero$: um sinal de 4 bits (“0000”) para ser utilizado no somador no lugar de ‘b’ caso a operação apenas envolva ‘a’ (ou seja, nas operações incremento de 1 e troca de sinal);
- $Mux1_1comp1$: um sinal de 4 bits que representa o complemento de um de ‘0001’ = “1110”, para ser utilizado no somador no lugar de ‘b’ caso seja selecionada a operação de decremento de um. Com isso, também é possível utilizar o módulo somador para realizar essa operação;
- Cin : um sinal de um bit representando o carry in que será usado no somador e terá nível lógico alto quando for necessário fazer complemento de dois de ‘a’, de ‘b’, de ‘1’(para a operação decremento de 1) ou então na operação de incremento de ‘1’;
- $Verifik$: um sinal que tem nível lógico alto quando a operação é de troca de sinal.

Nesse sentido, dois módulos que operam como dois multiplexadores para selecionar ‘a2’ e ‘b2’ foram implementados no projeto.

O módulo VHDL que seleciona “a2” é chamado de `troca_sinal`, pois essa é a única operação que requer utilizar o complemento de um de ‘a’. Isso funciona de forma simples: o sinal intermediário `Verifik` na ULA tem nível lógico alto quando a operação de troca de sinal é selecionada. Esse sinal intermediário entra no módulo `troca_sinal`, de forma que quando `Verifik` é 1, o módulo seleciona o complemento de um de ‘a’ como saída, caso contrário seleciona ‘a’:

Já o módulo que seleciona ‘b2’ é chamado de `mux_4bits` e funciona da seguinte forma: para operação de soma, ‘b’ é selecionado. Para a operação de subtração, “`bcomp1`” é selecionado, a fim de obter o complemento de dois de ‘b’ durante a soma. Para a operação de decremento de 1, `mux1_1comp1` é selecionado, a fim de obter o complemento de 2 de ‘0001’. Para quaisquer outros casos, `mux1_zero` (“0000”) é selecionado.

A entrada “`cin`” (carry in) do somador, se tratando de uma entrada de um bit, é regulada por meio da seguinte operação lógica na estrutura da ULA envolvendo o seletor:

$$cin = seletor(0) \text{ OR } seletor(1) \text{ OR } seletor(2).$$

Dessa maneira, quando a operação de soma for selecionada, o “`cin`” será 0, mas quando qualquer outra operação for selecionada , o “`cin`” será 1.

Esses filtros, portanto, permitem que o somador opere na seguinte configuração:

- **Soma:** ‘a’ e ‘b’ originais da entrada da ULA e `carry in` = ‘0’. A operação realizada equivale a “ $a + b$ ”;

- **Subtração:** ‘a’ original, complemento de dois de ‘b’ e carry in = ‘1’. A operação realizada equivale a “a - b”;
- **Incremento de 1:** ‘a’ original, b igual a “0000” e Carry In = ‘1’. A operação realizada equivale a “a + 1”;
- **Troca de sinal:** complemento de dois de ‘a’, b igual a “0000” e carry in = ‘1’. A operação realizada equivale a “-a”;
- **Decremento de 1:** ‘a’ original, b igual a ”1110”e carry in = ‘1’. A operação realizada equivale a “a - 1”.

1.3. Conclusão do Somador de 4 Bits

Com isso, o resultado do módulo somador, encaminhado a um sinal intermediário chamado Soma, representa o produto de uma das 5 primeiras operações, dependendo da seleção.

Nesse sentido, o módulo SELMux_4Bits, que é um multiplexador responsável por selecionar a saída da ULA, irá selecionar o mesmo sinal Soma para qualquer uma das cinco primeiras seleções : ”000, 001, 010, 011, 100”.

1.4. Operação de Normalização

A operação de normalização de vetores desloca o vetor para a esquerda até que seu bit mais significativo seja 1.

Com um módulo VHDL chamado Normalizador_4Bits, monta-se uma estrutura dividida em dois componentes:

- Um codificador de prioridade de 2 bits que identifica quantas casas à esquerda o vetor de 4 Bits de entrada ’a’ deve ser deslocado para que o bit mais significativo seja 1;
- Um deslocador lógico à esquerda de 4 bits, o qual vai deslocar o vetor de acordo com o resultado do codificador de prioridade.

O codificador de prioridade implementado funciona com a lógica invertida, identificando o número de casas com ’0’ à direita do último ’1’. Portanto, no módulo VHDL Normalizador_4Bits, o vetor de entrada ’a’ é inserido com seus bits em ordem invertida no codificador, de forma a ser possível encontrar o número de casas com ’0’ à esquerda do primeiro ’1’.

Esse número, portanto, é utilizado na estrutura do deslocador de bits implementado, de modo a normalizar o vetor de entrada ’a’. O resultado é armazenado em um sinal intermediário Numintermed que será atribuído à saída.

Exemplificando, essa operação transforma um vetor:

- ”0101”em ”1010”.
- ”0011”em ”1100”.
- ”0010”em ”1000”.
- ”1000”em ”1000”(sem alterações).

1.5. Operação AND

Na operação AND, é retornado um valor verdadeiro somente se todos os valores de entrada forem verdadeiros. A tabela-verdade da operação é:

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

Figura 1. Tabela-verdade da porta AND

1.6. Operação XOR

Na operação XOR, é retornado um valor verdadeiro somente se todos os valores de entrada forem diferentes. A tabela-verdade da operação é:

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

Figura 2. Tabela-verdade da porta XOR

1.7. Conclusão da Estrutura da ULA

A ULA contém 8 operações e o resultado será alterado por uma chave seletora de 3 bits "Sel".

As 5 primeiras operações utilizam do somador e de variações nas entradas desse somador, que são feitas em função de dois multiplexadores: um para 'a' e outro para 'b', em " $a + b$ ", e também em função da entrada carry in.

As últimas 3 operações são feitas utilizando módulos independentes com suas respectivas entradas e saídas.

Dessa forma, as saídas do somador e as saídas dos módulos independentes entram no multiplexador SELMux_4Bits, assim como o Seletor, que é uma entrada da ULA.

A seguinte correspondência operação - resultado é adotada:

- Soma - 000
- Subtração - 001
- Incremento de 1 - 010
- Troca de sinal - 011
- Decremento de 1 - 100
- Normalização de bits - 101
- Operação Lógica AND - 110
- Operação Lógica XOR - 111

O diagrama a seguir representa o módulo da Unidade Lógica e Aritmética¹

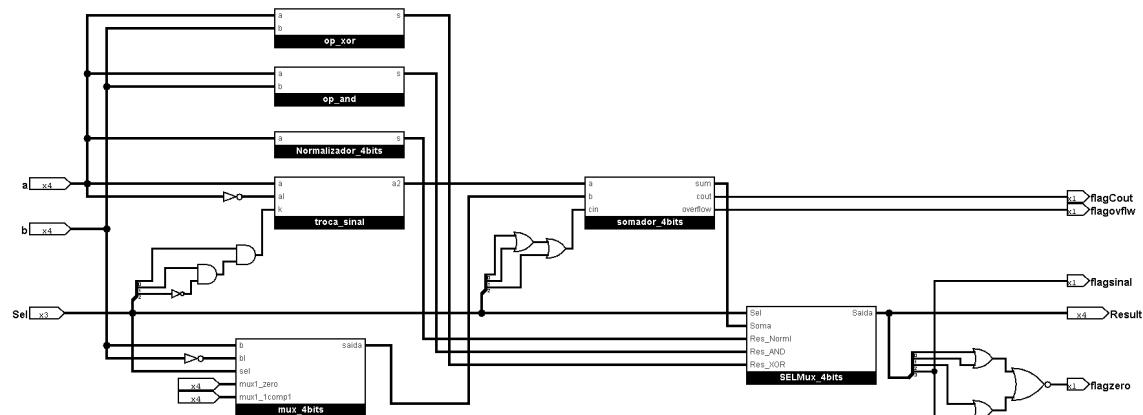


Figura 3. Diagrama da ULA

¹Os módulos estão disponíveis no GitHub *por meio deste link*.

2. Módulo Contador de 4 Bits

O módulo `contador_4bits` é o arquivo top level do projeto, e será utilizado no LabsLand. Por conta disso, é necessário que o nome de suas entradas e saídas estejam de acordo com a documentação. As entradas e saídas virtuais utilizadas foram:

- `G_CLOCK_50` - clock de 50 MHz;
- `V_BT` - botão virtual da placa FPGA;
- `V_SW` - interruptores virtuais da placa FPGA;
- `G_LEDG` - LEDs verdes da placa FPGA;
- `G_LEDR` - LEDs vermelhos da placa FPGA;
- `G_HEX7...G_HEX0` - Displays de 7 segmentos da placa FPGA.

A entrada da ULA que representa o Seletor de operação com 3 bits virá da placa FPGA por meio de 3 interruptores virtuais. Esses 3 interruptores são associados a um vetor de 3 bits denominado `V_SW`.

As entradas da ULA 'a' e 'b' geradas pelo contador, bem como o resultado da operação selecionada, serão exibidos em três conjuntos de displays de 7 segmentos:

- `G_HEX6` e `G_HEX7` representam, respectivamente, 'a' e seu sinal.
- `G_HEX4` e `G_HEX5` representam, respectivamente, 'b' e seu sinal.
- `G_HEX0` e `G_HEX1` representam, respectivamente, o resultado e seu sinal .

Já as flags da ULA serão representadas por 4 LEDs verdes da placa FPGA:

- `G_LEDG (3)` representa a flag carry out;
- `G_LEDG (2)` representa a flag overflow;
- `G_LEDG (1)` representa a flag zero;
- `G_LEDG (0)` representa a flag sinal.

As operações de normalização, AND e XOR irão utilizar uma representação binária dos números. Dessa maneira, tanto as entradas 'a' e 'b' quanto o resultado serão mostrados nos leds vermelhos, da seguinte ordenação:

- `G_LEDR (17)` ao `G_LEDR (14)` representam 'a';
- `G_LEDR (12)` ao `G_LEDR (9)` representam 'b';
- `G_LEDR (7)` ao `G_LEDR (4)` representam o resultado da operação.

Depois de definidas as entradas virtuais, pode-se prosseguir ao funcionamento do contador. Para gerar os dados de entrada da ULA, utilizou-se dois contadores de 4 bits, que percorrem toda a faixa de números representável em complemento de 2 (-8 a 7).

Sabe-se que o período do clock de 50 MHz é igual a 20 ns. Se a cada 20 ns conta-se 1, então é necessário 1 segundo para que contemos até 50.000.000. Para que cada contador identifique que passou 1 segundo, utilizou-se um divisor de frequência que conta até 50.000.000 e faz com que o primeiro contador adicione 1 à contagem.

Sendo assim, a cada segundo, o primeiro contador incrementa 1 ao operando B, até que chegue a "1111". Em seguida, volta a zero e o segundo contador incrementa 1 no operando A.

Os números gerados pelos contadores e o resultado da operação selecionada são mostrados nos displays de 7 segmentos da placa FPGA através de um decodificador BCD para 7 segmentos (veja a Figura 4). Para se acender um dos segmentos, é necessário aplicar o nível lógico baixo (0).

Decimal	X3	X2	X1	X0	g	f	e	d	c	b	a
0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	0	0	1
2	0	0	1	0	0	1	0	0	1	0	0
3	0	0	1	1	0	1	1	0	0	0	0
4	0	1	0	0	0	0	1	1	0	0	1
5	0	1	0	1	0	0	1	0	0	1	0
6	0	1	1	0	0	0	0	0	0	1	0
7	0	1	1	1	1	1	1	1	0	0	0
-8	1	0	0	0	0	0	0	0	0	0	0
-7	1	0	0	1	1	1	1	1	0	0	0
-6	1	0	1	0	0	0	0	0	0	1	0
-5	1	0	1	1	0	0	1	0	0	1	0
-4	1	1	0	0	0	0	1	1	0	0	1
-3	1	1	0	1	0	1	1	0	0	0	0
-2	1	1	1	0	0	1	0	0	1	0	0
-1	1	1	1	1	1	1	1	1	0	0	1

Figura 4. Entradas BCD e os segmentos de saída correspondentes no display

Para mostrar números negativos visualmente, é necessário incluir o sinal de menos utilizando o display ao lado do número. Sendo assim, no código, sempre que o bit mais significativo do número for '1' (indicando que o número é negativo), apenas o segmento "g" do display ao lado do número irá acender (veja a Figura 5), mostrando visualmente que o número é negativo.

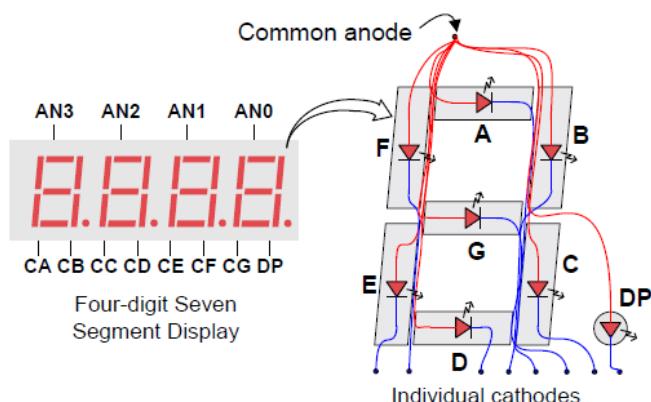


Figura 5. Display de sete segmentos (anodo comum)

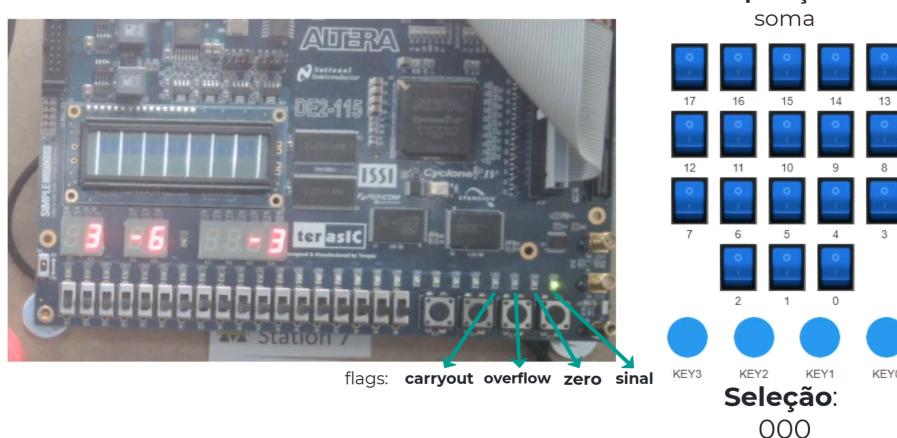
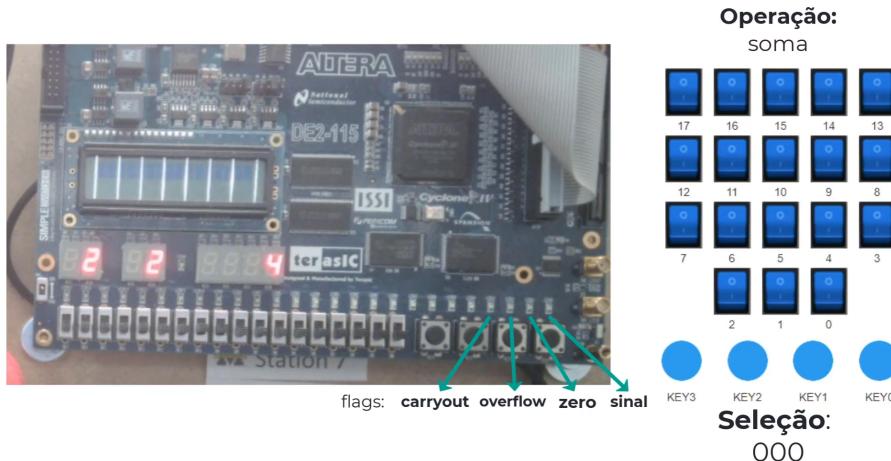
3. Funcionamento no Labsland

Por fim, observa-se um total de 10 módulos VHDL:

- contador_4bits (arquivo top level);
- ula_4bits;
- full_adder;
- somador_4bits;
- mux_4bits;
- troca_sinal;
- Normalizador_4bits;
- op_and;
- op_xor;
- SELMux_4bits.

Todos os módulos VHDL são inseridos no LabsLand². Após isso, esse conjunto é sintetizado, tendo o contador como módulo principal e, dessa forma, é possível observar o funcionamento prático da ULA³ na placa FPGA DE2-115.

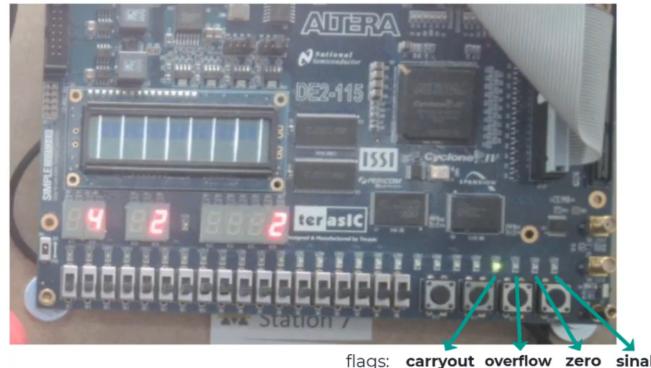
Soma:



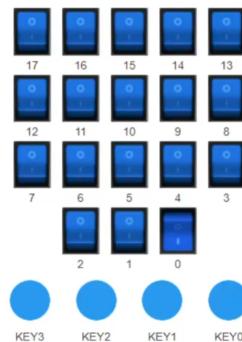
²Link de acesso ao LabsLand: https://labsland.com/pt_BR

³Os vídeos demonstrando o funcionamento da ULA podem ser acessados *neste link de apresentação*.

Subtração:



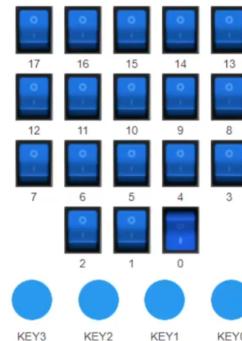
Operação:
subtração



Seleção:
001

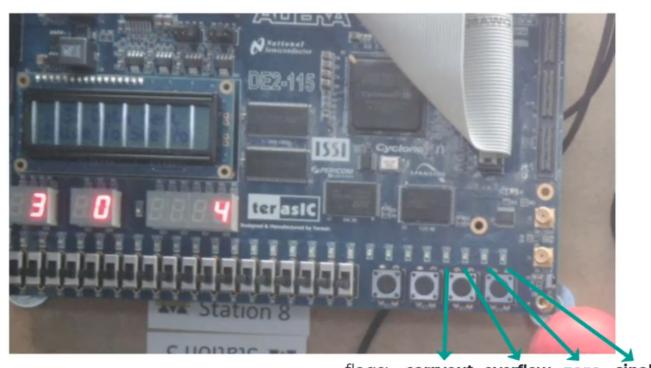


Operação:
subtração

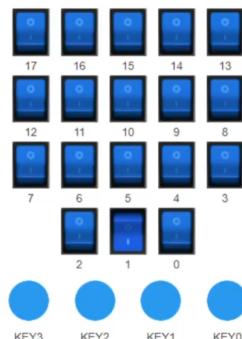


Seleção:
001

Incremento de 1:

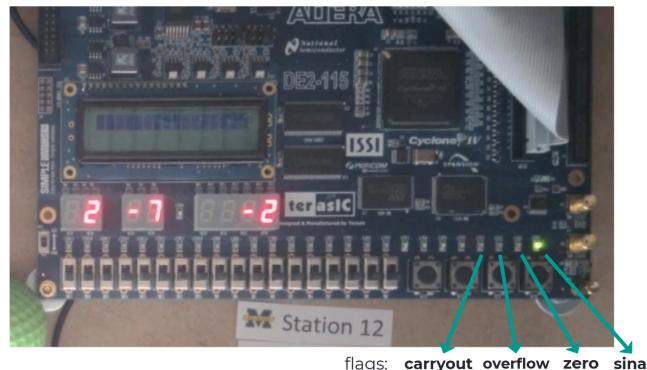


Operação:
incrementa 1 em A

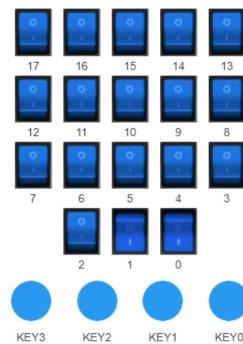


Seleção:
010

Troca Sinal:

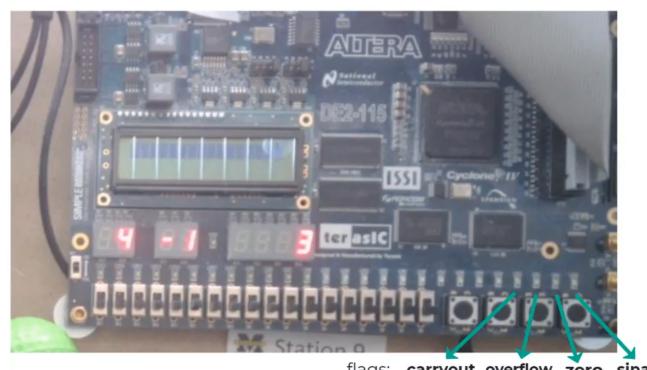


Operação:
troca de sinal de A

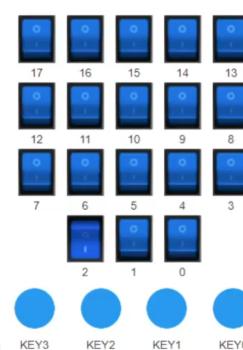


Seleção:
011

Decremento de 1:

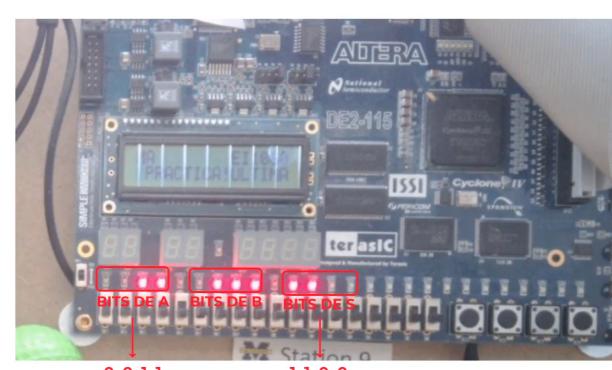


Operação:
decrementa 1 em A

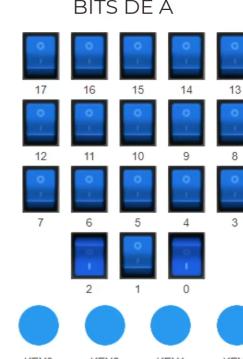


Seleção:
100

Normalização:

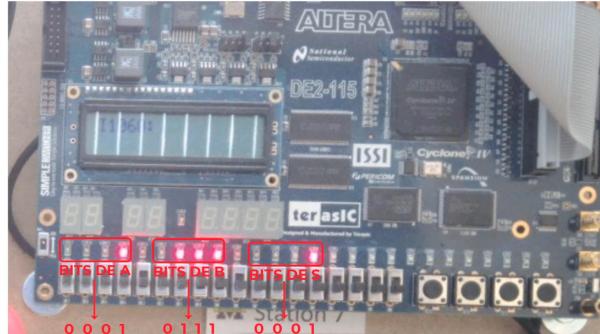


Operação:
NORMALIZAÇÃO DOS
BITS DE A

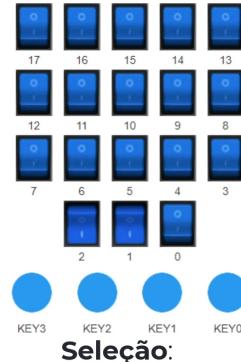


Seleção:
101

Operação Lógica AND:

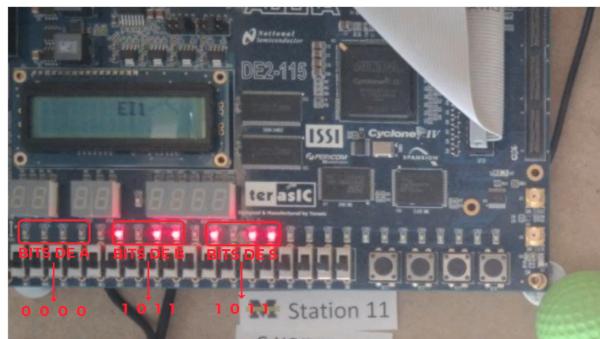


Operação:
AND

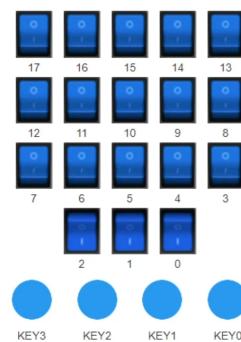


Seleção:
110

Operação Lógica XOR:



Operação:
XOR



Seleção:
111

4. Considerações Finais

Portanto, concluímos a construção de uma Unidade Lógica e Aritmética de 4 bits, implementamos 8 operações e simulamos os dados de entrada e as saídas em uma placa FPGA DE2-115 através do simulador LabsLand⁴.

Essa prática permitiu que conceitos importantes da disciplina de Sistemas Digitais fossem consolidados, mostrando-se uma forma efetiva para a compreensão dos circuitos combinacionais e a forma como eles são planejados.

⁴Os módulos estão disponíveis no GitHub *por meio deste link*.