

# Universidade Federal do Rio de Janeiro

Departamento de Engenharia Eletrônica e de Computação

EEL480- Laboratório de Sistemas Digitais

Unidade-Lógico-Aritmética (ULA) de 4 Bits

Data: 10/05/2019

Turma: EL4

Nomes:

Iverton Darlan

DRE:

Ass.:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Paulo Henrique Bucco dos Santos Caetano

DRE: 116169635

Ass.:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Introdução**

Este projeto tem por objetivo o desenvolvimento de uma Unidade-Lógico-Aritmética (ULA) de 4 Bits que realiza 8 operações. Ela deve conter um sistema de interface para o teste em uma placa de desenvolvimento. Determinamos essas 8 operações divididas em dois grupos, o Lógico, que atribui as funções AND, OR, NOT e XOR, e o Aritmético, que atribui as funções de soma (+), subtração (-), multiplicação (\*) e incremento de 1 (+1).

Uma Unidade-Lógico-Aritmética (ULA) é parte fundamental de sistemas mais complexos, pois se trata de um circuito digital que realiza operações aritméticas e booleanas entre operandos de N bits. Possui uma forma de seleção que determina a operação a ser realizada, chamada de “chave seletora”. A saída indica o resultado da operação e, também, pode ter N bits, dependendo do que foi realizado na ULA.

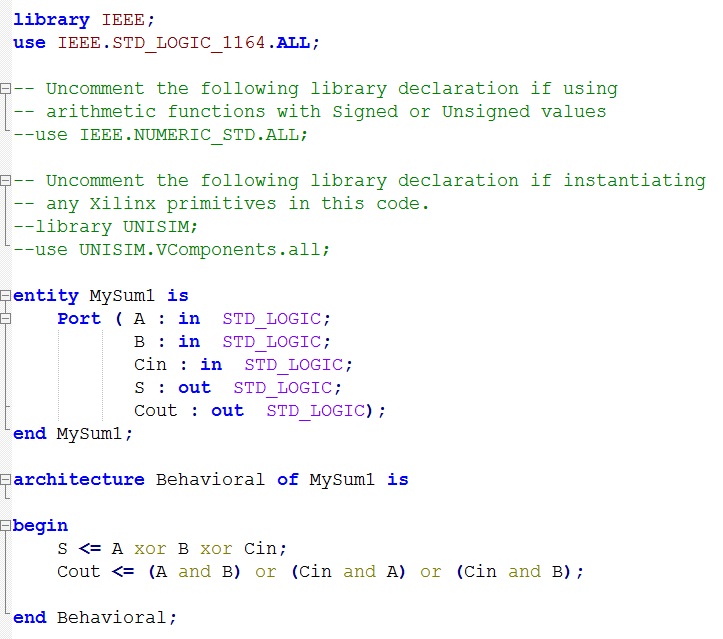
1. **Descrição da Implementação**

A ULA foi desenvolvida a partir de blocos menores que se relacionam dentro de um sistema gerenciador. O particionamento em módulos menores, em certo ponto de vista, torna mais fácil e simples a estruturação lógica da tarefa a ser realizada. Dito isso, podemos listar abaixo cada módulo e descrição do que foi realizado no projeto.

2.1 - Operações Aritméticas:

2.1.1 - Soma (+)

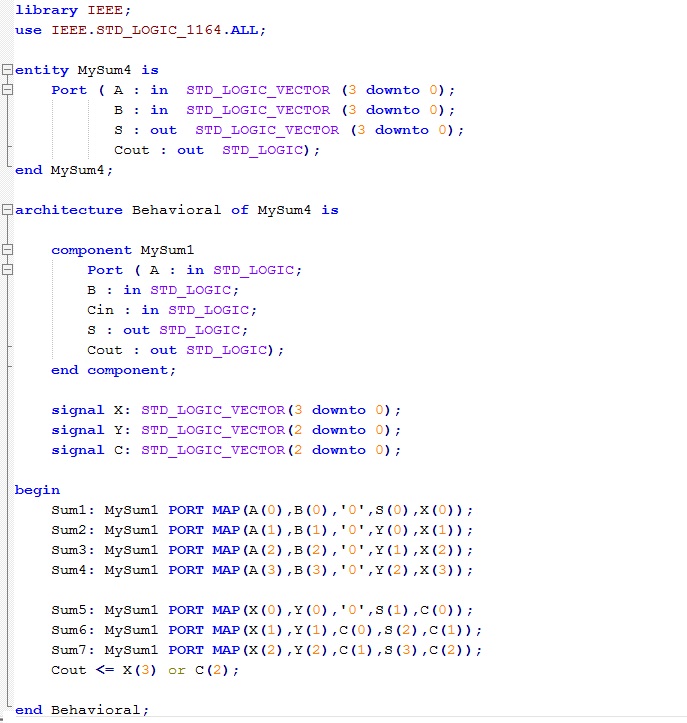
Definimos que a partir de um bloco somador de 1 Bit, ficaria mais fácil a realização da soma de 4 Bits, pois esse último bloco usaria repetidas vezes o bloco mais simplista.



Módulo MySum1 (Somador de 1 bit)

“S” é a saída do bloco que contém o resultado e ela recebe as entradas “A”, “B” e “Cin” (carry de entrada). A saída é atribuída após a realização da operação lógica XOR, padronizada pela biblioteca do sistema IEEE, entre as entradas. “Cout” (carry de saída) também é uma saída do bloco, porém não carrega o resultado da operação, apenas uma informação a respeito do que foi realizado. Comumente é chamada de bit de transporte e leva esse nome porque indica que houve uma extrapolação na operação de soma. No “Cout” as entradas se relacionam por meio de AND’s e OR’s, que também são padronizados pelo sistema IEEE.

A soma de 4 Bits dentro da ULA usará o bloco acima da seguinte forma:

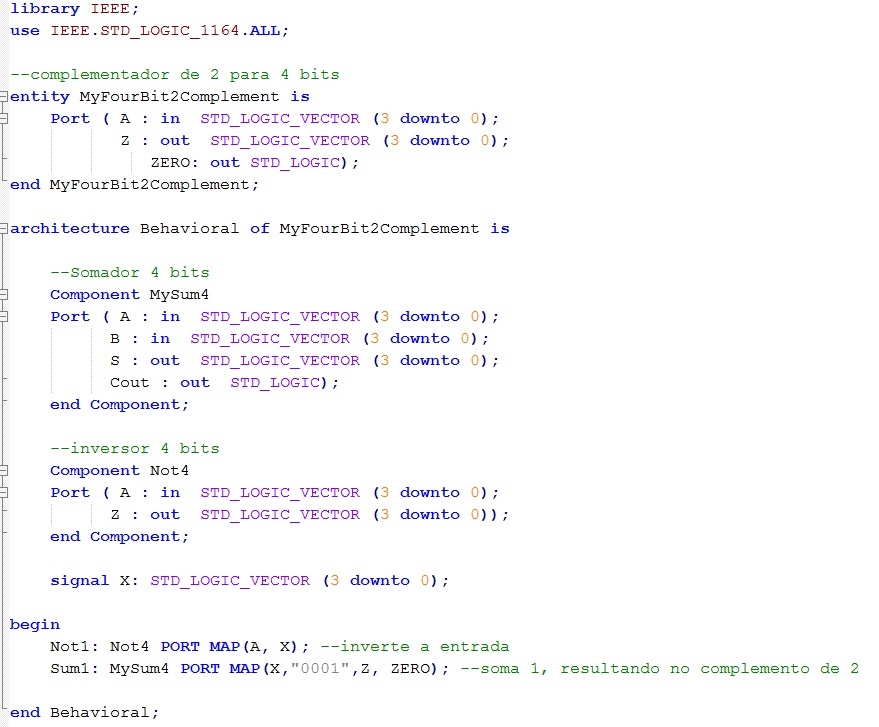


Módulo MySum4 (Somador de 4 bits)

Para cada bit de saída “S” (S0, S1, S2 e S3) que relaciona os bits de entrada de “A” (A0, A1, A2 e A3) e “B” (B0, B1, B2 e B3) e “C” (carry de entrada), a função “MySum1” (soma de 1 bit) é chamada. No módulo acima, é realizada a soma bit a bit das entradas A e B da ULA e é retornado o valor da soma em 4 bits de saída (S0, S1, S2 e S3). Neles está contido, em binário, o valor resultado da soma binária de dois números de 4 bits cada. Desta forma, o processo de soma aritmética das entradas da ULA se tornou mais simples e eficaz após a modularização.

2.1.2 - Subtração (-)

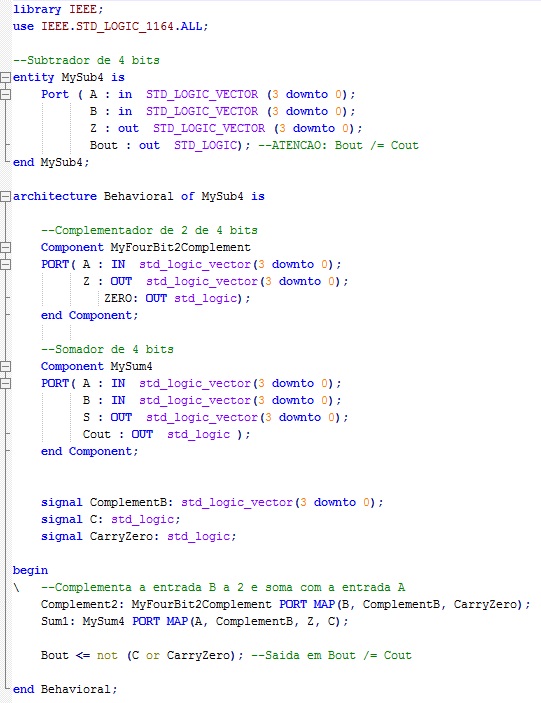
Na realização da operação aritmética de subtração, utilizamos do sistema Complemento a 2 para tornar a subtração uma soma de um número binário positivo com um número binário negativo representado em complemento de 2, afim de torná-la mais simples. Na mudança do segundo número para deixá-lo no sistema de complemento a 2, utilizamos o seguinte módulo conversor:



Módulo MyFourBit2Complement (Complementador a 2 de 4 bits)

No módulo acima da nossa ULA, utilizamos um módulo aritmético de soma de números com 4 bits (“MySum4”, em 2.1.1) e um módulo lógico inversor de números de 4 bits (“Not4”), que será mostrado na continuação deste relatório, para modificar o número desejado e deixá-lo em sistema de complemento a 2. O número em questão será sempre o composto dos bits de entradas B (B0, B1, B2 e B3) da nossa ULA, pois será ele que queremos deixar como um número negativo no sistema complemento de 2. Nesse sistema, invertemos as N bits entradas do número e depois somamos a ele o valor 1 (em N bits também). A saída desse módulo é o valor da entrada selecionada em complemento de 2 e, portanto, pronto para ser realizada a operação de subtração com a outra entrada não modificada.

Para realizarmos a subtração, agora que o segundo número de entrada está em sistema complemento a 2, deveremos apenas somar a entrada não modificada ao complemento a 2 da segunda entrada, obtendo assim a soma dos dois números de 4 bits. Utilizamos o seguinte módulo para isso:

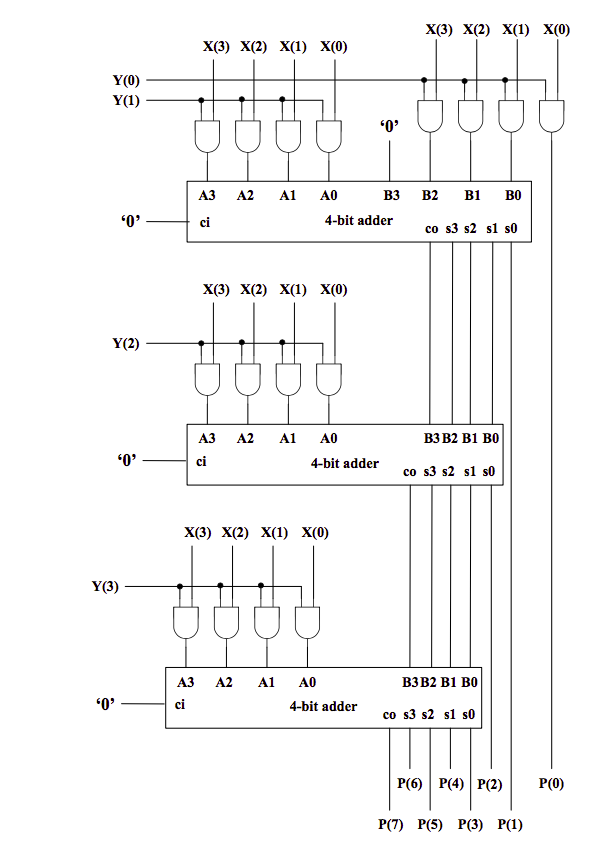


Módulo MySub4 (Subtrator de 4 bits)

No módulo acima da nossa ULA, utilizamos os módulos de soma de 4 bits (“MySum4”, em 2.1.1) e complementador a 2 de 4 bits (“MyFourBit2Complement”, em 2.1.2) para realizar a subtração de 2 números de 4 bits cada. Nele primeiro se complementa a 2 a entrada B e depois soma com a entrada A, dando o resultado “Sum1” e outra saída apenas operacional, que é o “Bout” (Borrow de saída). O “Bout” é atribuído a partir de operações NOT e OR, padronizadas pelo sistema IEEE, do “C” e “CarryZero”, que são os carry’s da soma e do complemento a 2.

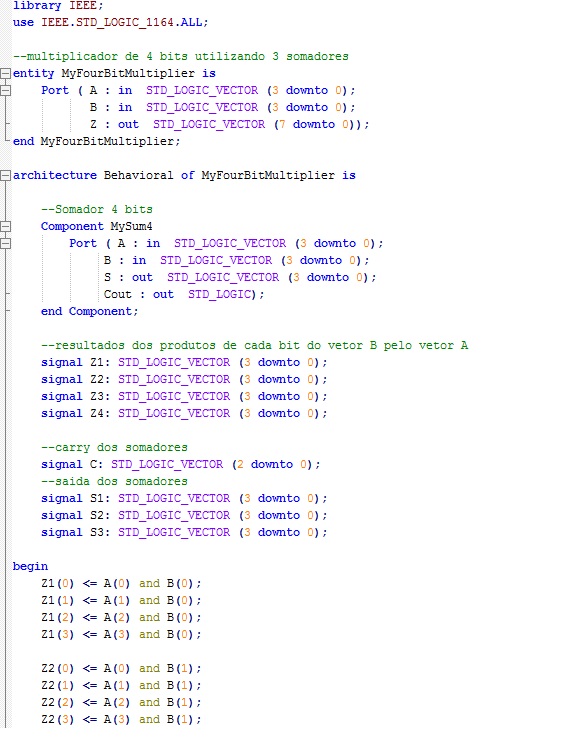
2.1.3 - Multiplicação (\*)

Na realização da operação aritmética de multiplicação da nossa ULA, utilizamos repetidas vezes o módulo somador de 4 bits (“MySum4”, em 2.1.1) afim de tornar mais simples e entendível a operação por completo. Baseamos o nosso módulo e em outro esquemático, mostrado a seguir:

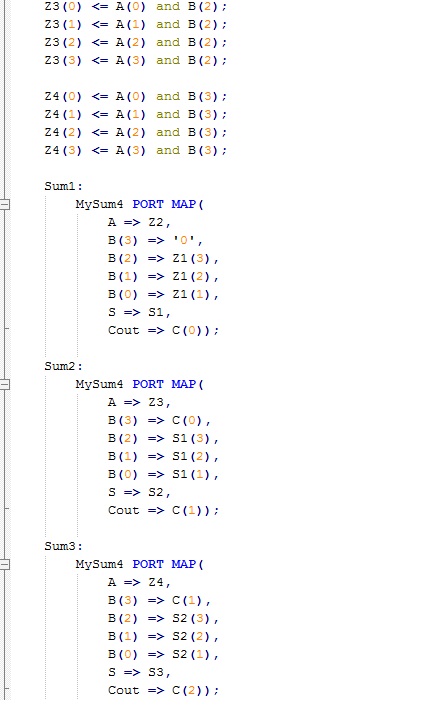


Esquemático do multiplicador de 4 bits

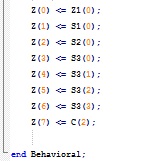
A partir da ideia desse esquemático fizemos o módulo abaixo:



Módulo MyFourBitMultiplier (Multiplicador de 4 bits) - parte 1



Módulo MyFourBitMultiplier (Multiplicador de 4 bits) - parte 2

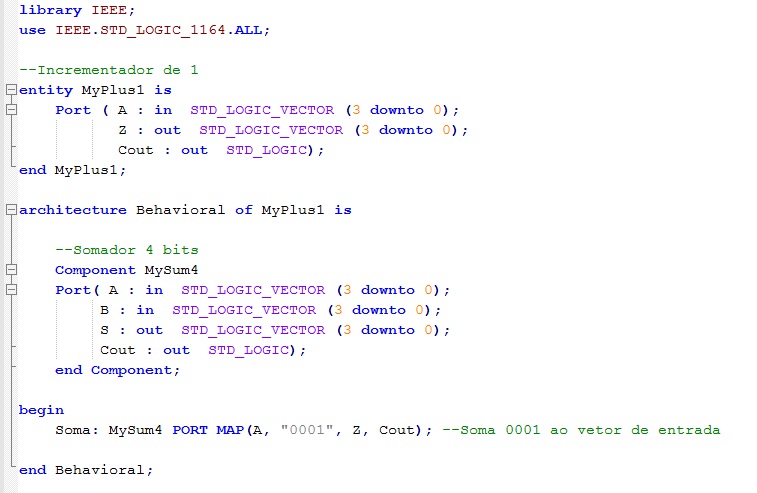


Módulo MyFourBitMultiplier (Multiplicador de 4 bits) - parte 3

No módulo multiplicador da nossa ULA, usamos 3 módulos somadores de 4 bits (“MySum4”, em 2.1.1) e a operação lógica AND, padronizada pelo sistema IEEE. Como sabemos, a saída com o resultado da operação deve conter a soma do número de bits das duas entradas, portanto Z irá de 0 a 7. Cada saída Z (Z0, Z1, Z2, Z3, Z4, Z5, Z6 e Z7) recebe os valores atribuídos no corpo do código acima.

2.1.4 - Incremento de 1 (+1)

Na realização da operação aritmética de incrementação de 1 da nossa ULA, usamos o módulo de soma de 4 bits (“MySum4”, em 2.1.1) para incrementar “0001” aos 4 bits de entrada. Ele se resume a nada mais do que uma soma fixa da entrada de N bits com o vetor de N bits com valor unitário. O módulo está descrito abaixo:

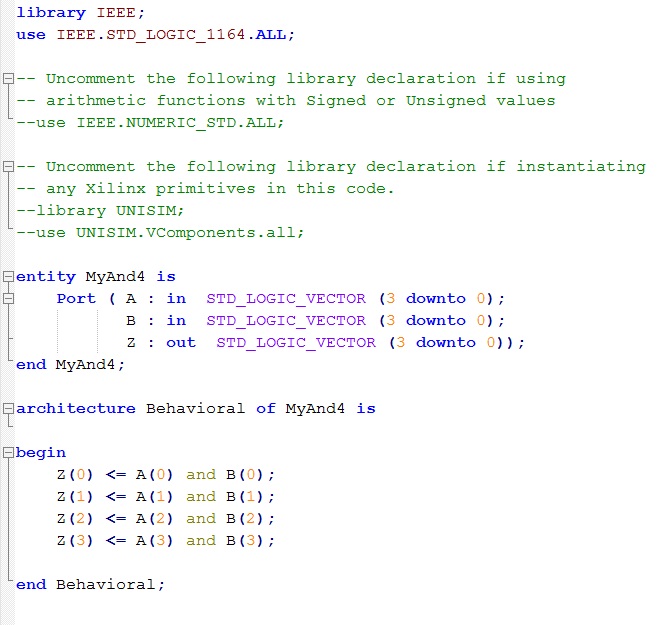


Módulo MyPlus1 (Incrementador de 1 em 4 bits)

2.2 - Operações Lógicas:

2.2.1 - AND

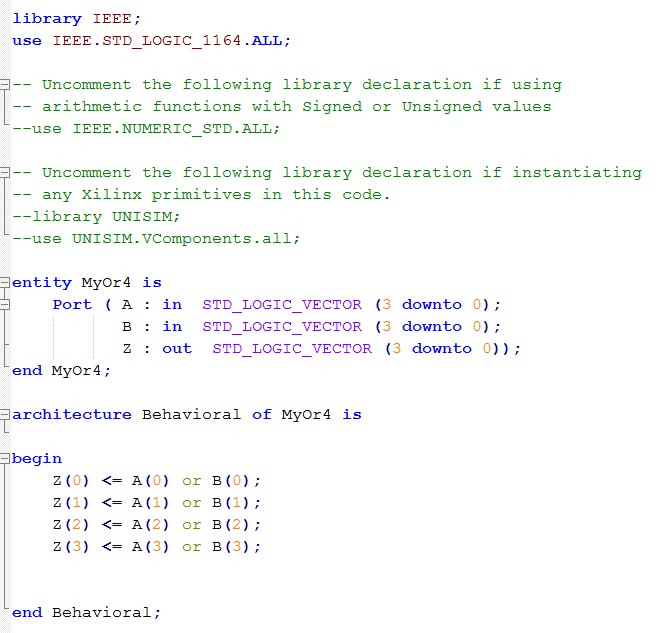
Na realização operação lógica AND da nossa ULA, usamos a função lógica AND padronizada pelo sistema IEEE 4 vezes relacionando as entradas A (A0, A1, A2 e A3) e B (B0, B1, B2 e B3), respectivamente. As saídas Z (Z0, Z1, Z2 e Z3) são atribuídas com essas relações e formam o resultado da nossa operação. O módulo está descrito abaixo:



Módulo MyAnd4 (Operação lógica AND de 4 bits)

2.2.2 - OR

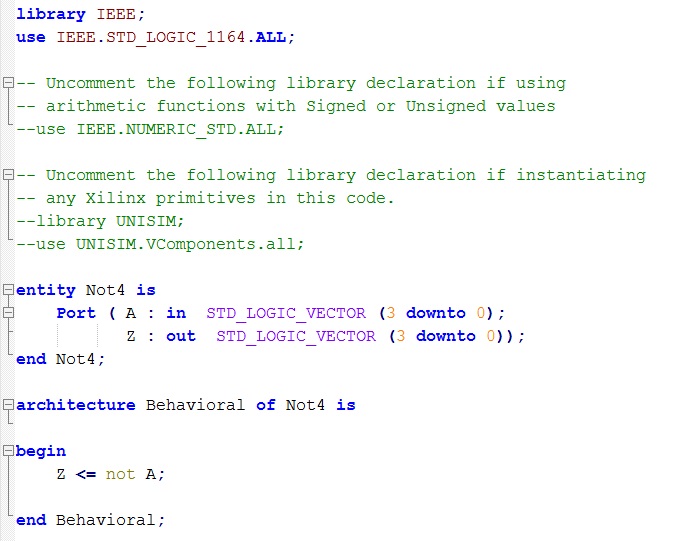
Na realização operação lógica OR da nossa ULA, usamos a função lógica OR padronizada pelo sistema IEEE 4 vezes relacionando as entradas A (A0, A1, A2 e A3) e B (B0, B1, B2 e B3), respectivamente. As saídas Z (Z0, Z1, Z2 e Z3) são atribuídas com essas relações e formam o resultado da nossa operação. O módulo está descrito abaixo:



Módulo MyOr4 (Operação lógica OR de 4 bits)

2.2.3 - NOT

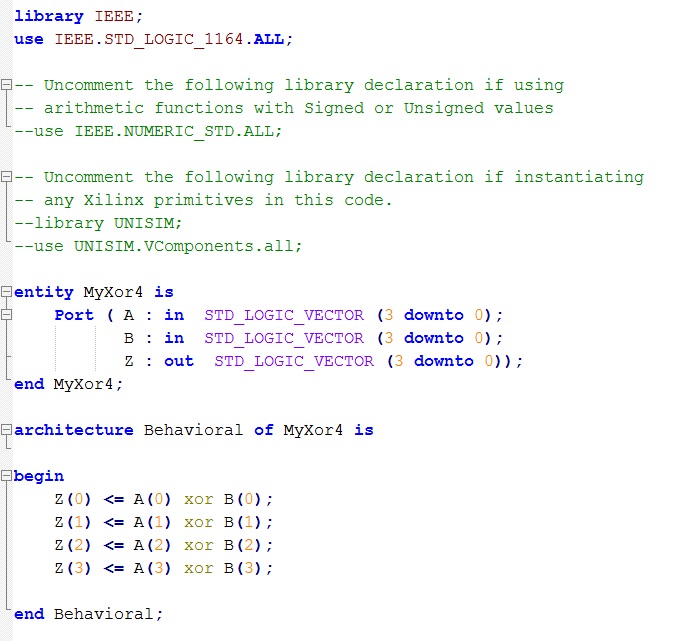
Na realização operação lógica NOT da nossa ULA, usamos a função lógica NOT padronizada pelo sistema IEEE uma vez, relacionando a entrada A (vetor de 4 bits) a saída Z (vetor de 4 bits). Esse último vetor forma o resultado da nossa operação inversora. O módulo está descrito abaixo:



Módulo Not4 (Operação lógica NOT de 4 bits)

2.2.4 - XOR

Na realização operação lógica XOR da nossa ULA, usamos a função lógica XOR padronizada pelo sistema IEEE 4 vezes relacionando as entradas A (A0, A1, A2 e A3) e B (B0, B1, B2 e B3), respectivamente. As saídas Z (Z0, Z1, Z2 e Z3) são atribuídas com essas relações e formam o resultado da nossa operação. O módulo está descrito abaixo:



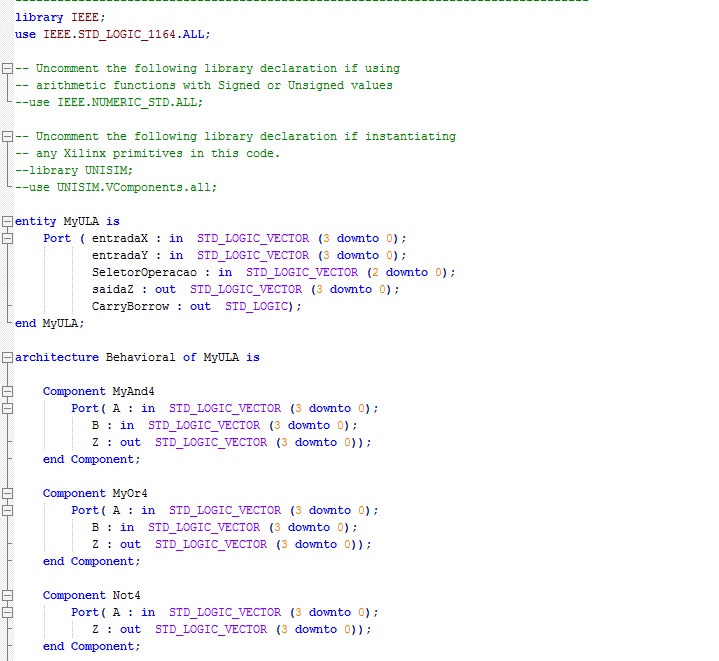
Módulo MyXor4 (Operação lógica XOR de 4 bits)

2.3 - Unidade-Lógico-Aritmética:

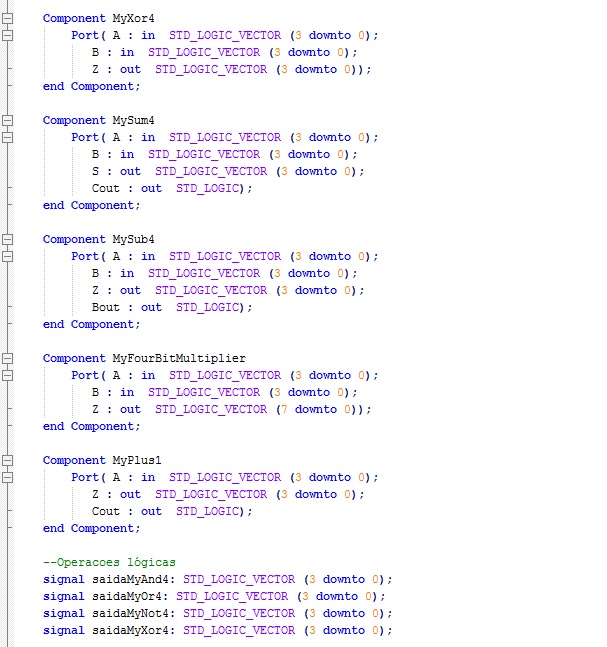
A Unidade-Lógico-Aritmética funciona basicamente chamando os módulos lógicos ou aritméticos descritos acima, nas seções 2.1 e 2.2 do capítulo 2. Escolhemos um seletor para determinar qual operação o sistema da ULA deve realizar. Chamamos esse vetor seletor de “SeletorOperacao” e ele é um vetor de 3 bits, no qual podem ser descritas as 8 operações do sistema. As operações tem valores fixos nessa seleção, que são determinados conforme lista abaixo:

* + “000” - AND de 4 bits (“MyAnd4”, em 2.2.1);
  + “001” - OR de 4 bits (“MyOr4”, em 2.2.2);
  + “010” - NOT de 4 bits (“Not4”, em 2.2.3);
  + “011” - XOR de 4 bits (“MyXor4”, em 2.2.4);
  + “100” - Soma de 4 bits (“MySum4”, em 2.1.1);
  + “101” - Incrementador de 1 de 4 bits (“MyPlus1”, em 2.1.4);
  + “110” - Subtração de 4 bits (“MySub4”, em 2.1.2);
  + “111” - Multiplicador de 4 bits (“MyFourBitMultiplier”, em 2.1.3).

Após o módulo ter sido selecionado, a ULA atribui o resultado a saída “saidaZ”. Há, também, uma saída dependente da operação selecionada no sistema da ULA, chamada de “CarryBorrow”. Essa saída carrega o Carry ou Borrow da operação de soma (“MySum4”, em 2.1.1) ou subtração (“MySub4”, em 2.1.2), respectivamente. O módulo da Unidade-Lógico-Aritmética (ULA) está descrito abaixo:



Módulo MyULA - parte 1



Módulo MyULA - parte 2

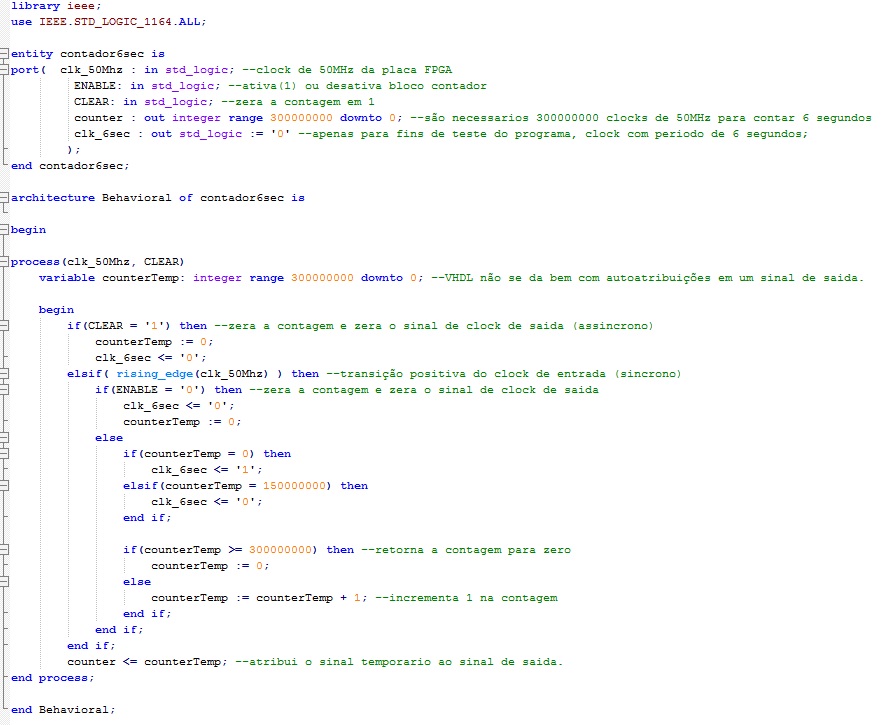


Módulo MyULA - parte 3

2.4 - Interface:

2.4.1 - Contador:

A interface da nossa Unidade-Lógico-Aritmética, precisou, primeiramente, de um módulo contador para que pudéssemos exibir os resultados na sequência determinada em projeto (entrada A, entrada B, resultado da operação). Este módulo contador, descrito abaixo, utiliza o clock de 50 MHz da placa para converter em um pulso de 6 segundos, contando 300000000 transições de subida de 50MHz, essa contagem será usada para mostrar os números e resultados na ordem e tempo determinados em projeto. Este módulo possui ainda uma entrada para habilitar ou desabilitar o contador (síncrona) e uma entrada para zerar o contador (assíncrona) .



Módulo contador6sec (contador 6 segundos)

O programa da interface possui como entradas, o clock de 50MHz da placa, um botão para reset, um botão para mudar de estado e 4 switches para escolher os vetores de entrada. E possui como saídas, 4 leds responsáveis por mostrar o vetor de 4 bits e mais 4 bits para mostrar qual dentre os 4 vetores de bits está sendo mostrado (entrada A, entrada B, seleção de Operação, Resultado).

Começamos declarando as componentes do contador e da ULA, após isso, declaramos os estados referentes à máquina de estados, state\_A, state\_B, state\_Operação, state\_Result. Declaramos então variáveis para guardar o estado atual e indice do estado. Também declaramos variáveis para guardar os valores da entrada A, entrada B, seleção de operação, Carry/Borrow da ULA, Saida da ULA, contagem do contador, guardados nas respectivas variáveis Xtemp, Ytemp, SeletorOperacao, CarryBorrow, SaidaZ, numeroContador.

Começamos o programa com um “port map” da Unidade Logico Aritmética e do contador de 6 segundos. Com o detalhe que usaremos o botaoEntrada, ou seja, o botao que muda de estado, como clear do contador, ou seja, após clicar nesse botão o contador é zerado, e a contagem recomeça.

Após entrar no processo da máquina de estado, iniciamos declarando o efeito do botão de reset, o circuito retorna para o estado A. Temos então, um bloco responsável por mudar de estado e gravar os valores de entrada A, entrada B, seleção de operação.

O bloco de código a seguir é responsável por mostrar os vetores entrada A, entrada B, resultado, sequencialmente com intervalos de 2 segundos. O bloco final é responsável por atualizar os leds da esquerda com o valor atual dos switches durante os estados, state\_A, state\_B, state\_Operacao.

Ao clicar no botãoEntrada no state\_A, a entrada A é gravada com os valores do switches.

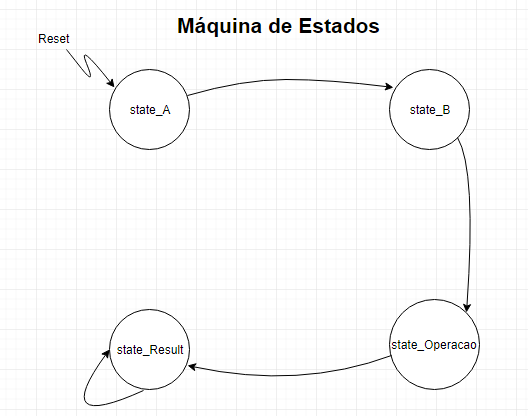
Ao clicar no botãoEntrada no state\_B, a entrada B é gravada com os valores do switches.

Ao clicar no botãoEntrada no state\_Operacao, o vetor de operação é gravado com os valores do switches.

Ao clicar no botãoEntrada no state\_Result, é selecionada uma nova operacao.

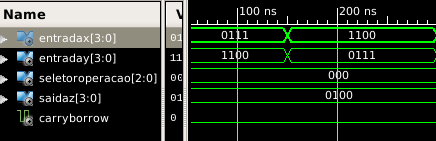
IMAGEM DA INTERFACE

A seguir temos um diagrama de estados representando a máquina de estados implementada na interface.



1. **Resultados**

**3.1 – AND**



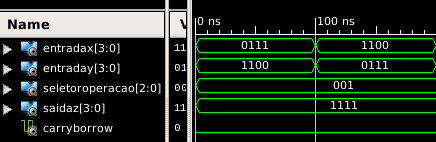
Na imagem começamos com A = 0111 e B=1100, após 100ns alternamos as entradas, A=1100 e B=0111. Vemos que carryborrow = 0, pois não é utilizado nessa operação. Observamos que alternar as entradas não altera a saída.

saídaz = 0111 and 1100 = 0100

saidaz = 1100 and 0111 = 0100

O resultado está correto.

**3.2 – OR**



Na imagem começamos com A = 0111 e B=1100, após 100ns alternamos as entradas, A=1100 e B=0111. Vemos que carryborrow = 0, pois não é utilizado nessa operação. Observamos que alternar as entradas não altera a saída.

saídaz = 0111 or 1100 = 1111

saidaz = 1100 or 0111 = 1111

O resultado está correto.

**3.3 – INVERTER**



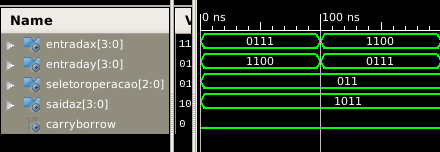
Na imagem começamos com A = 0111 e B=1100, após 100ns invertemos as entradas, A=1100 e B=0111. Vemos que carryborrow = 0, pois não é utilizado nessa operação. Apenas a entrada A é utilizada para este caso.

saídaz = not 0111 = 1000

saidaz = not 1100 = 0011

O resultado está correto.

**3.4 – XOR**



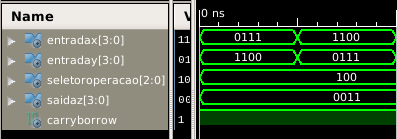
Na imagem começamos com A = 0111 e B=1100, após 100ns alternamos as entradas, A=1100 e B=0111. Vemos que carryborrow = 0, pois não é utilizado nessa operação. Observamos que alternar as entradas não altera a saída.

saídaz = 0111 and 1100 = 1011

saidaz = 1100 and 0111 = 1011

O resultado está correto.

**3.5 – SUM**



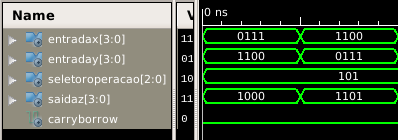
Na imagem começamos com A = 0111 e B=1100, após 100ns alternamos as entradas, A=1100 e B=0111. Observamos que alternar as entradas não altera a saída e que o carry é 1 nos dois casos.

saídaz = 0111 + 1100 = 0011 com carry = 1

saidaz = 1100 + 0111 = 0011 com carry = 1

O resultado está correto.

**3.6 – INCREMENTO DE 1**



Na imagem começamos com A = 0111 e B=1100, após 100ns alternamos as entradas, A=1100 e B=0111. Apenas a entrada A é utilizada neste caso, observamos que mudar a entrada A altera a saída e que carryborrow se mantém em 0.

saídaz = incrementa1( 0111 ) = 1000 com carry = 0

saidaz = incrementa1(1100) = 1101 com carry = 0

O resultado está correto.

**3.7 – SUBTRATOR**



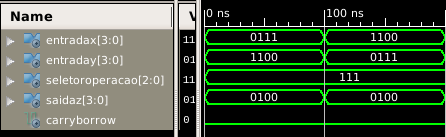
Na imagem começamos com A = 0111 e B=1100, após 100ns alternamos as entradas, A=1100 e B=0111. Observamos que alternar as entradas altera a saída. Observamos também que carry se mantém em 1 até a primeira metade e 0 na segunda meta.

saídaz = 0111 - 1100 = 1011 com borrow = 1

saidaz = 1100 - 0111 = 0101 com borrow = 0

O resultado está correto.

**3.8 – MULTIPLIER**



Na imagem começamos com A = 0111 e B=1100, após 100ns invertemos as entradas, A=1100 e B=0111. Vemos que carryborrow = 0, pois não é utilizado nessa operação. Observamos que alternar as entradas não altera a saída. Apenas os últimos 4 bits são mostrados nessa operação.

saídaz = 0111 \* 1100 = 01010100

saidaz = 1100 \* 0111 = 01010100

O resultado está correto.

1. **Conclusões**

O projeto funciona perfeitamente bem conforme o esperado, entretanto foi observado um número excessivo de saídas não utilizadas, pelo fato de serem implementadas em vista de seu modelo real.

Foi observado em certo momento as diferenças entre simulação e funcionamento real. Onde foi possível observar o efeito de bouncing (trepidação) na placa e no entanto, não foi possível observar na simulação, sendo necessário adicionar uma pequena linha de código a mais para corrigir este efeito.