**Disciplina de Sistemas Digitais (if675)**

**Centro de Informática - UFPE**

**ULA – Unidade Lógica Aritmética**

**Somador BCD**

**Equipe:**

**André de Souza Ferraz**

**Denyson José Silva Messias**

**Fernando Miguel Barros Lins**

**Rui Ricardo Dantas da Fonte Filho**

**Índice**

|  |  |
| --- | --- |
| **1. Introdução** | 3 |
|  |  |
| **2. ULA - Unidade Lógica Aritmética** | 4 |
| 2.1. Somador bit a bit | 4 |
| 2.2. Somador/Subtrator | 6 |
| 2.3. Complemento a 2 de B | 7 |
| 2.4. Comparadores | 8 |
| 2.4.1. A < B | 8 |
| 2.4.2. A > B | 11 |
| 2.4.3. A = B | 12 |
| 2.5. AND bit a bit | 13 |
| 2.6. XOR bit a bit | 14 |
| 2.7. Display da ULA | 15 |
| 2.8. ULA – Unidade Lógica Aritmética | 18 |
|  |  |
| **3. Somador BCD** | 21 |
| 3.1. Somador BCD de um dígito | 21 |
| 3.2. Somador BCD de 2 dígitos | 23 |
|  |  |
| **4. Conclusão** | 24 |

**1. Introdução**

O projeto apresenta a implementação de uma ULA – Unidade Lógica Aritmética e de um Somador BCD.

A ULA é capaz de realizar operações aritméticas entre vetores de quatro bits, tais como: soma, subtração, complemento a 2, comparações AND bit a bit e XOR bit a bit.

O somador BCD realiza a soma de números representados por código BCD (*Binary Coded Decimal*).

As implementações da ULA e do Somador BCD serão apresentadas com base nas tabelas-verdade de cada dispositivo, acompanhadas dos seus respectivos mapas de Karnaugh, circuitos e waveforms.

**2. ULA – Unidade Lógica Aritmética**

Implementações

A seguir, são mostradas as implementações de todas as unidades que compõem a *ULA*. Para cada unidade, é mostrada a Tabela Verdade, os mapas de Karnaugh correspondentes (quando aplicáveis), os circuitos usados na implementação e um exemplo de simulação, quando necessário, mostrando o resultado da referida operação.

**2.1 Somador Bit a Bit**

O *Somador bit a bit* é o circuito básico do Somador / Subtrador.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

*Tabela verdade: Somador bit a bit*

# A



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **00** | **01** | **11** | **10** |
| **0** CIn | 0 | 1 | 0 | 1 |
| **1** | 1 | 0 | 1 | 0 |

# B

S = (A xor B) xor CIn

# A

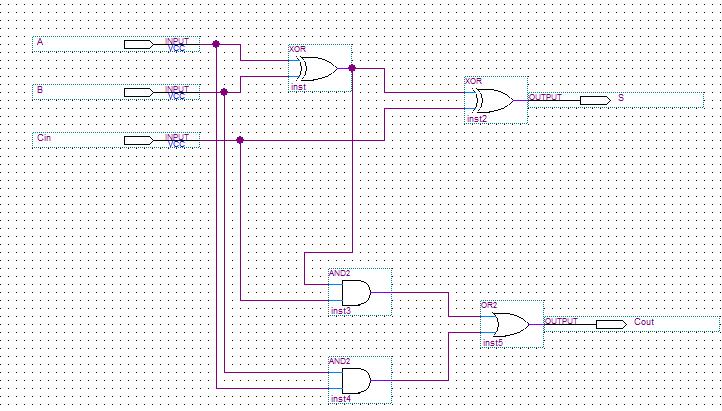


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **00** | **01** | **11** | **10** |
| **0** CIn | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 1 | 1 |

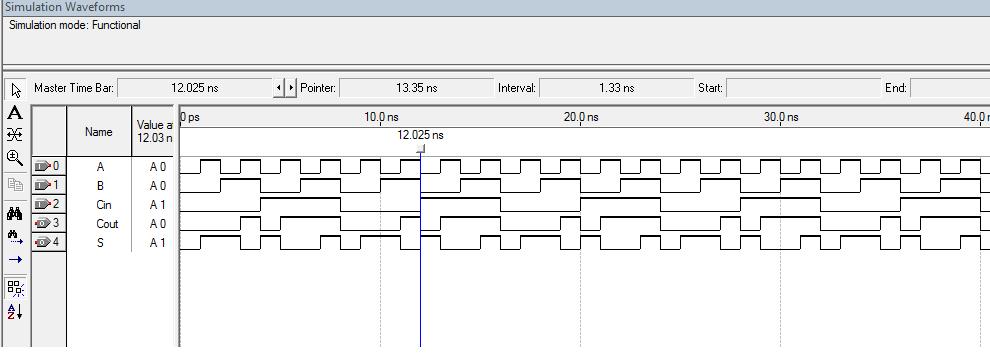
# B

Cout = AB + BCIn + ACIn

= AB + CIn(A+B)



*2.1.1 Circuito: Somador de um bit*

****

*2.1.2 Simulação: Somador de um bit*

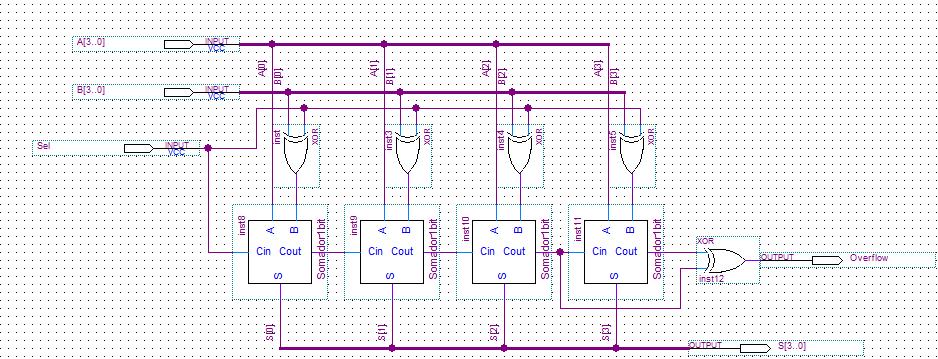
**2.2 Somador/Subtrator com Seletor.**

O *Somador / Subtrador* é uma unidade aritmética formada por unidades menores do *Somador bit a bit.* O bit de entrada *Seletor* indica qual das duas operações deve ser realizada, caso 0, é realizado a soma , caso contrário, uma subtração é efetuada.

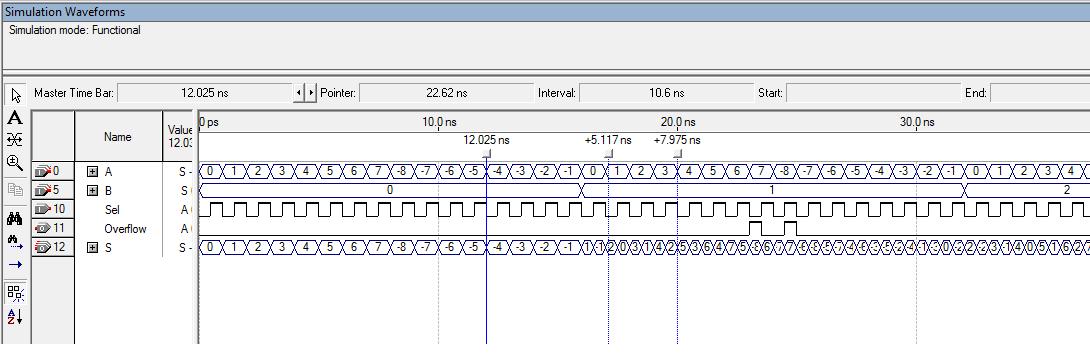
No circuito, além das 4 unidades do *somador bit a bit*, existem 4 portas lógicas do tipo XOR, que funcionam como seletor ligando o bit *somar/subtrair* de entrada e cada bit do vetor *B* às suas entradas, invertendo a entrada caso a operação selecionada seja uma subtração.

|  |  |  |
| --- | --- | --- |
| **B** | **Selet** | **Out** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Tabela verdade: Seletor do Somador / Subtrador (XOR)*

****

*Somador/Subtrator 4 bits*

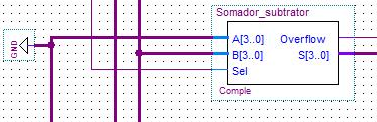


**2.3 Complemento a 2 de B**

A operação de *Complemento a 2 de B* (de 4 bits) funciona da seguinte maneira: os bits de B são invertidos e depois soma-se 0001.

Mas na implementação foi usado um somador/subtrator, subtraindo(seletor igual a 1) B de 0. Retornando assim o seu complemento a dois.

0 - ( B ) = (- B).

****

*Complemento a 2 de B*

**2.4 Comparadores**

**2.4.1 Comparador A < B**

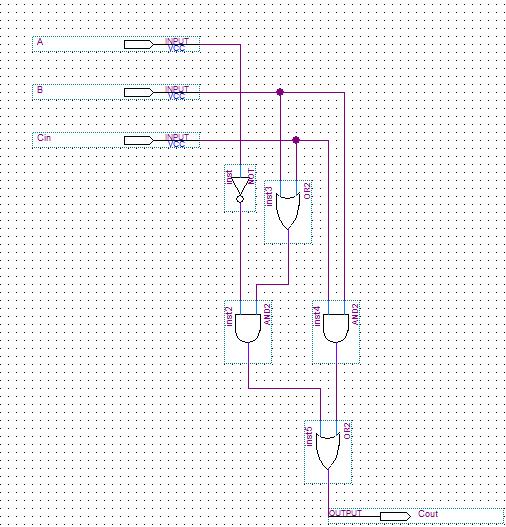
Comparador bit a bit:

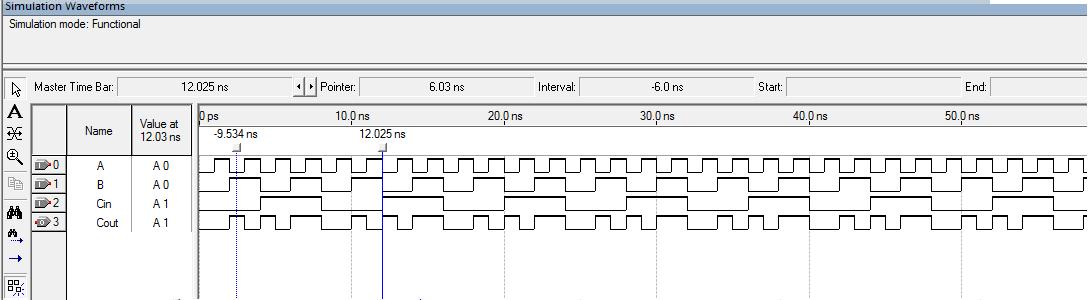
O comparador bit a bit é a unidade mais elementar do Comparador. Ele compara 2 bits e gera uma saída 1 caso o bit *A* seja menor que o bit *B*. Além dos bits *A* e *B* que serão comparados, há também uma outra entrada, aqui chamada de *Carry in*, que indica o resultado de uma comparação anterior.

|  |  |  |  |
| --- | --- | --- | --- |
| **B** | **A** | **CIn** | **R** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C\BA | **00** | **01** | **11** | **10** |
| **0** | 0 | 0 | 0 | 1 |
| **1** | 1 | 0 | 1 | 1 |

R = B.C + A’.(B + C)



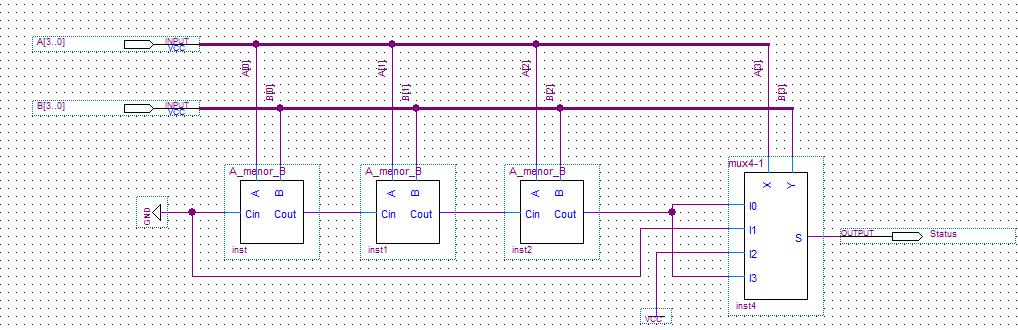
**

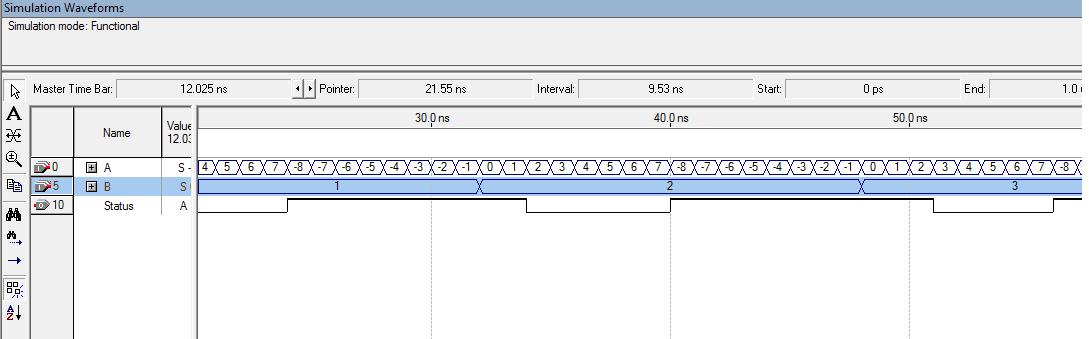
Comparador de 4 bits:

O comparador de 4 bits é formado por três unidades do Comparador bit a bit e uma do comparador de sinal.

**Comparador de bit de sinal:**

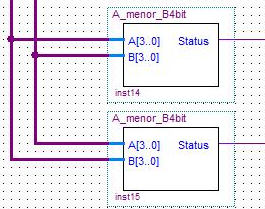
O *Comparador de Sinal* verifica se o vetor B é maior que o vetor A através do módulo e do sinal. Caso B seja maior que *A*, é gerada uma saída com nível lógico 1.

****



**2.4.2 Comparador A > B**

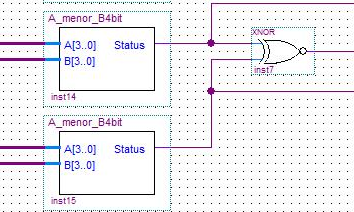
Simplesmente foram trocadas as entradas do comparador A < B

****

**2.6.2 Comparador A = B**

Caso os outros dois comparadores retornem 0, então A = B. Então é usada uma porta XNOR tendo como entradas as duas saídas dos outros comparadores:

|  |  |  |
| --- | --- | --- |
| **A<B** | **A>B** | **A = B** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

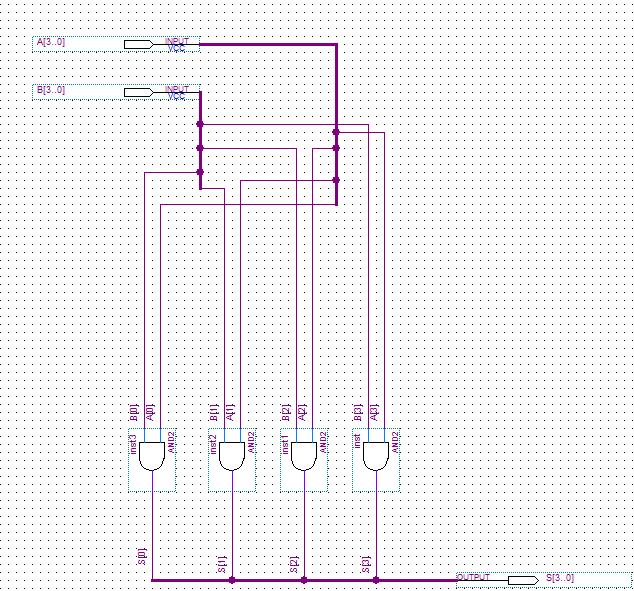


**2.7 AND**

Esta operação recebe como entrada dois vetores e realiza um *AND* entre os bits correspondentes de cada vetor.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Cin** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Tabela verdade: AND entre 2 bits*

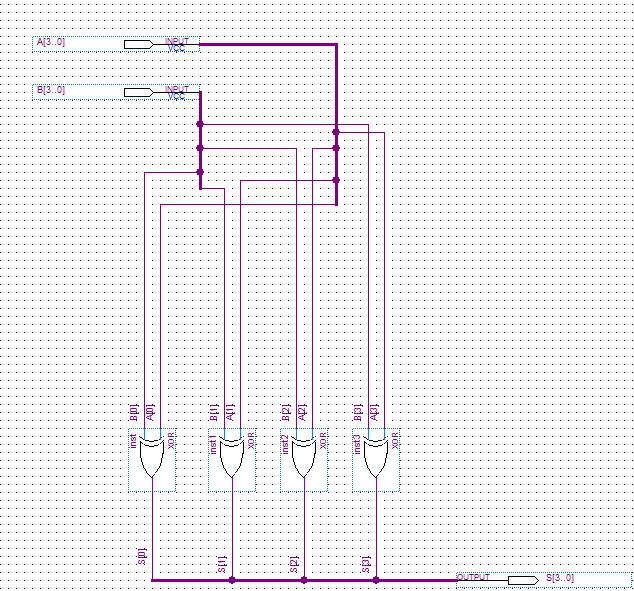
****

**2.8 XOR**

Esta operação recebe como entrada dois vetores e realiza um *XOR* entre os bits correspondentes de cada vetor.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Cin** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Tabela verdade: XOR entre 2 bits*

****

**2.9 Display da ULA**

O Decodificador do Display de 7 segmentos é usado para que os resultados obtidos nas operações efetuadas pela ULA possam ser exibidos no display da placa. Como entrada, ele recebe um vetor[3..0], 2 bits (status e overflow).

O Decodificador fará com que o display indique somente o resultado da operação escolhida. Para isso, o vetor de seleção é usado para forçar que saídas de uma operação que não foi escolhida sejam convertidas para nível lógico 1, apagando-as do display.

OBS: Supondo que o LED acende com o bit ‘1’.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | V[3] | V[2] | V[1] | V[0] |  | a | b | c | d | e | f | g |
| 0 | 0 | 0 | 0 | 0 |  | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |  | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 |  | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 |  | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |  | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 |  | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 |  | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 |  | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A V[1]V[0]  V[3] V[2] | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 1 | 1 |
| **01** | **0** | **1** | **1** | **1** |
| **11** | d | d | d | d |
| **10** | 1 | 1 | d | D |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| B V[1]V[0]  V[3] V[2] | 00 | 01 | 11 | 10 |
| 00 | 1 | 1 | 1 | 1 |
| 01 | 1 | 0 | 1 | 0 |
| 11 | d | d | d | d |
| 10 | 1 | 1 | d | d |

A = V[3] + V[1] + V[2]V[0] + ¬V[2]¬V[0] B = ¬ V[2] + ¬V[1]¬V[0] + V[1]V[0]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C V[1]V[0]  V[3] V[2] | 00 | 01 | 11 | 10 |
| 00 | 1 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | d | d | d | d |
| 10 | 1 | 1 | d | d |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D V[1]V[0]  V[3] V[2] | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | d | d | d | d |
| 10 | 1 | 1 | d | d |

C= ¬ V[1] + V[0] + V[2] D=V[3]+¬V[2]¬V[0]+V[1]¬V[2]+V[1]¬V[0] +V[2]¬V[1]+V[0]

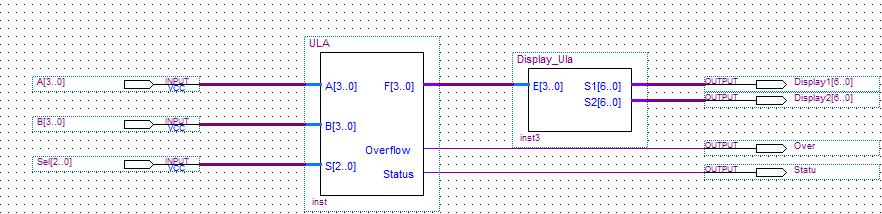
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| E V[1]V[0]  V[3] V[2] | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 0 | 1 |
| 01 | 0 | 0 | 0 | 1 |
| 11 | d | d | d | d |
| 10 | 1 | 0 | d | d |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *F V[1]V[0]*  *V[3] V[2]* | *00* | *01* | *11* | *10* |
| 00 | 1 | 0 | 0 | 0 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | d | d | d | d |
| 10 | 1 | 1 | d | d |

E = ¬v[2] ¬V[0] + V[1] ¬V[0] F=V[3] + ¬V[1] ¬V[0] + ¬V[1] V[2] + V[2] ¬V[0]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| G V[1]V[0]  V[3] V[2] | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | d | d | d | d |
| 10 | 1 | 1 | d | d |

G= V[3] + V[2]¬V[2] + ¬V[2]V[1] + V[2]¬V[0]

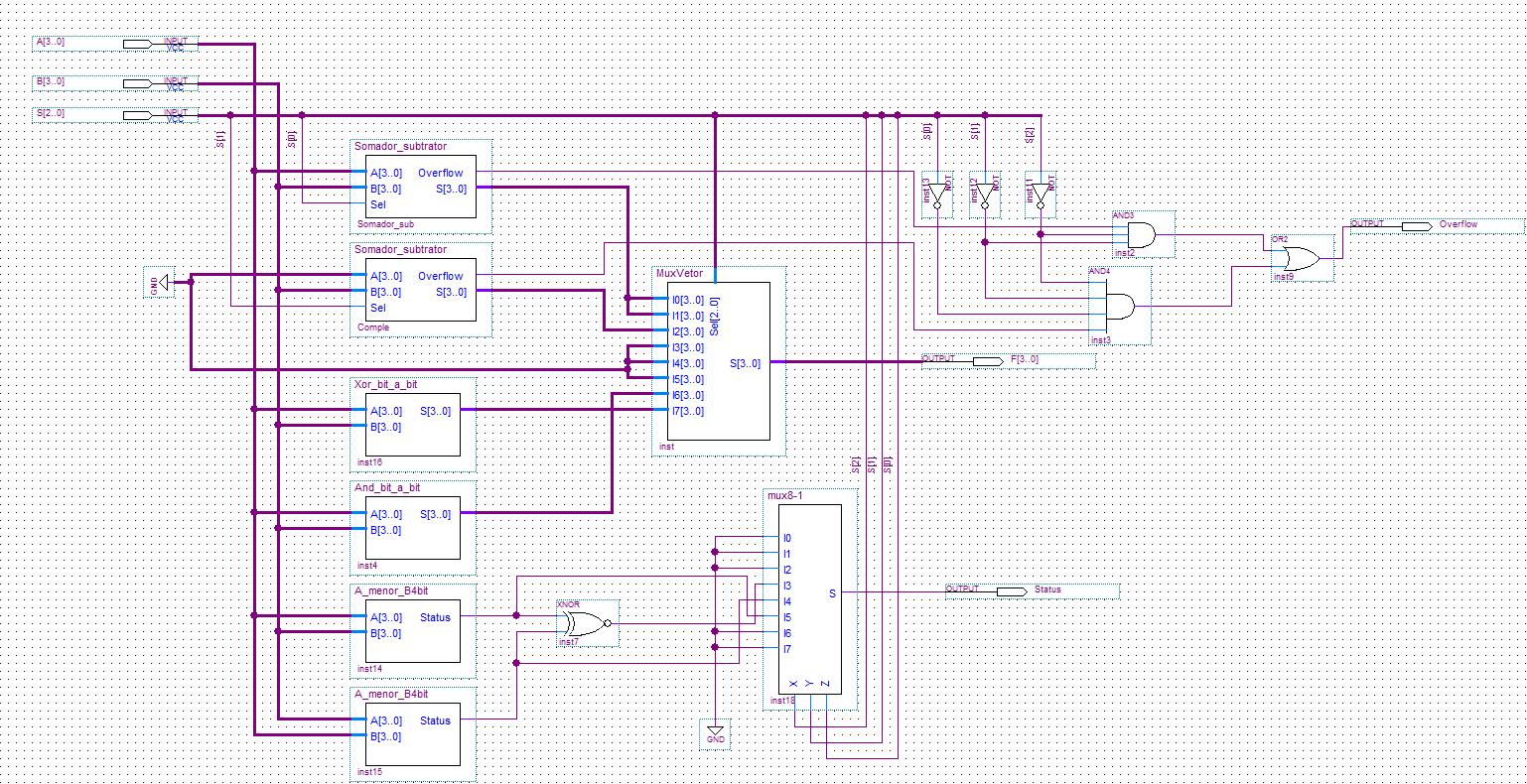


**Unidade Lógica Aritmética**

A ULA completa é formada pela união dos circuitos apresentados até aqui. Ela recebe como entrada 2 vetores de 4 bits e realiza a operação indicada pelo vetor de seleção. Possui um vetor de 4 bits de saída para operações que geram um vetor como resposta e 2 bits, o *overflow,* que indica se a operação *Soma, Subtração ou Complemento a 2* gerou um excesso e o *status,* usado nas respostas das operações de comparação.

No exemplo de simulação, pode se observar as diferentes operações sendo efetuadas quando se varia o vetor seleção.

|  |  |  |  |
| --- | --- | --- | --- |
| **Seleção** | | | **Função** |
| 0 | 0 | 0 | F = A + B |
| 0 | 0 | 1 | F = A - B |
| 0 | 1 | 0 | F = Complemento a 2 de B |
| 0 | 1 | 1 | F = A = B |
| 1 | 0 | 0 | F = A > B |
| 1 | 0 | 1 | F = A < B |
| 1 | 1 | 0 | F = A AND B |
| 1 | 1 | 1 | F = A XOR B |

*Tabela de operações efetuadas pela ULA.*****

**3. Somador BCD**

O *Somador BCD* aqui implementado é capaz de somar dois números de 00 + 00 à 99 + 99.

Em BCD, representamos cada dígito do número separadamente.

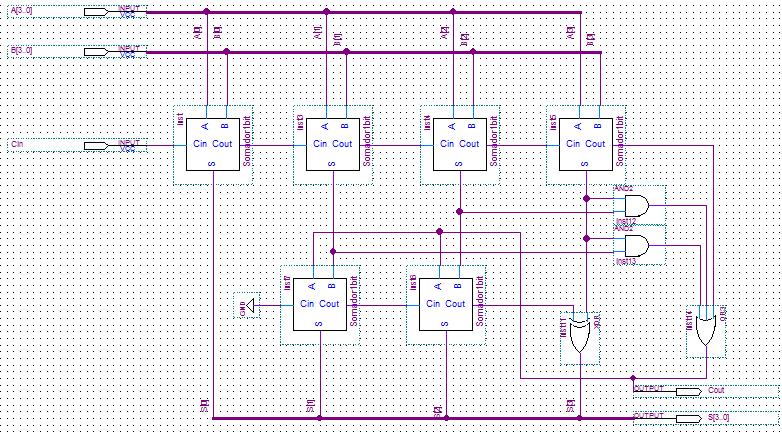
Implementações:

**3.1 Somador BCD de um dígito**

O *Somador BCD* de um digito é capaz de somar números de um único digito, de 0 + 0 a 9 + 9.

Como entrada, *o Somador BCD* recebe dois vetores de 4 bits, que representam os dois dígitos a serem somados. Na saída, são gerados um vetor resultado da soma, e um bit que indica se a soma gerou overflow.

Como base, o somador é formado por somadores simples, de um bit, mostrados na implementação da *ULA.* Depois da soma dos vetores, pode ser necessário separar os dígitos decimais, caso a soma ultrapasse 9. Esses casos são aqueles em que a soma gera um resultado maior ou igual a 10, ou seja, nos casos da forma 11xx, 1x1x e 1xxxx. Para esses resultados, é necessário a subtrair 10 de resultado da soma, para que assim tenhamos os bits separadamente. Esta subtração é conseguida efetuando-se a soma do resultado com o complemento a 2 de 10 em binário (0110).

****

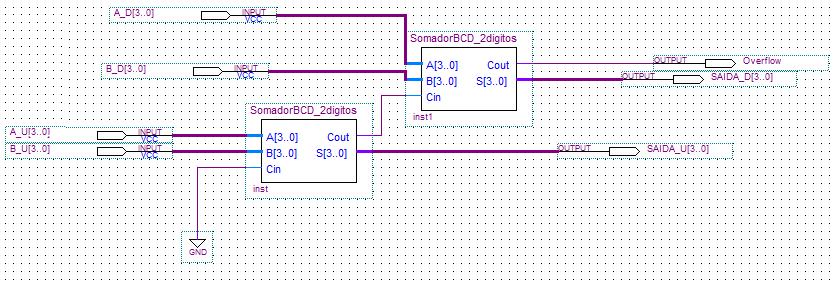
**3.2. Somador BCD de dois dígitos**

O *Somador BCD de dois dígitos* é formado por 2 unidades do *Somador BCD de 1 dígito*. Casas decimais correspondentes são somadas separadamente.

Como entrada, o circuito recebe 2 vetores de 4 bits representando os dígitos das unidades de cada um dos números e 2 vetores de 4 bits representando os dígitos das dezenas de cada um dos números.

Como saída, o circuito gera 2 vetores que representam os dígitos da unidade e da dezena, e uma terceira saída, o overflow, que indica se um resultado gerou um número maior que 99.

O primeiro Somador de um dígito recebe o *CarryIn* como 0, e o segundo recebe o overflow do primeiro.

****

**Conclusão**

Através desse projeto, concluímos que podemos implementar uma Unidade Lógica Aritmética, um Somador BCD e outros circuitos mais complexos pela construção de suas tabelas verdades, e seus respectivos mapas de Karnaugh. Percebemos também, que em muitos casos é necessário “Dividir para Conquistar”, separando o problema em pedaços menores facilitando sua solução. Através desses passos, fomos capazes de realizar o projeto e deixá-lo pronto para ser implementado na placa experimental em laboratório, para que possamos verificar os resultados obtidos.