**Relatório do laboratório 08 - Projeto da ULA do Neander**

Nome: Arthur Ferreira Ely

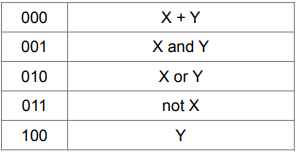
Cartão: 338434

Data: 21/02/2023

Número da sala: 103

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foi realizado desenvolvido a ULA (unidade lógica aritmética) do Neander, computador hipotético, no programa Quartus II. O circuito foi projetado como se fossem várias ULAs 1 bit, conectadas recursivamente para formar uma ULA de 8 bits. Nesse circuito, também, foi utilizado um decodificador para selecionar as funções da ULA. Elas são:



**Tabela 1: funções do Neander**

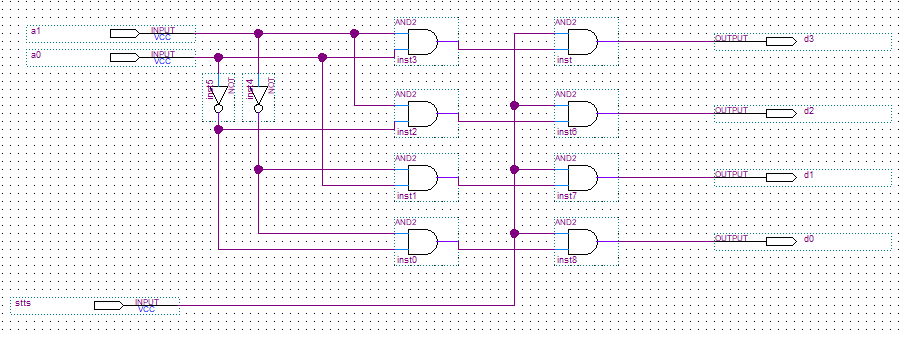
Outra possibilidade era usar a ferramenta Karma para otimização do circuito, mas foi optado o caminho mais simples, que funciona da mesma forma, apesar de ser maior.

Cada bit da ULA conta com 6 variáveis: X, Y, opcode (3) e carry in. Além disso, essa ULA em específico conta com 5 funções (saídas), que são escolhidas pelo decodificador 3x8.

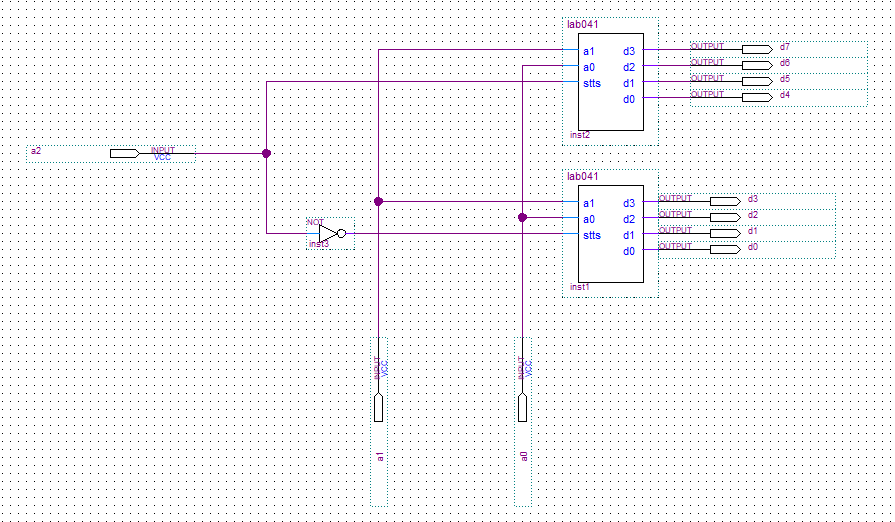
**Primeiro passo: ULA de 1 bit**

Para a ULA de 1 bit, foram postas todas as funções esperadas do computador, sendo elas ativadas conforme o código de operação de entrada, que será chamado de opcode daqui pra frente.

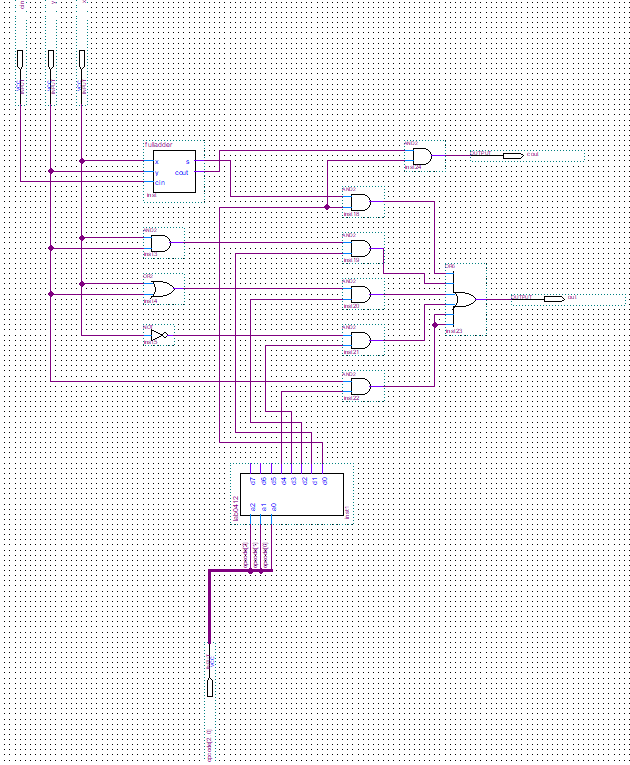
Cada opcode corresponde a uma função, como pode-se observar na tabela 1. O opcode 000 seleciona a função X + Y, o opcode 001 seleciona a função X and Y, e assim por diante.

A lógica utilizada consiste em fazer essas seleções com um decodificador 3x8, que vai pegar os 3 bits do opcode e escolher uma função para aplicar em X e Y. Abaixo, a figura do decodificador 2x4 utilizado no decodificador maior.

**Figura 1: decodificador 2x4**

Agora, a figura do decodificador 3x8.

**Figura 2: decodificador 3x8**

Com o decodificador pronto, é possível integrá-lo ao nosso circuito lógico para selecionar as funções desejadas. Abaixo, a figura da ULA de 1 bit completa.

X + Y

X and Y

X or Y

not X

Y

**Figura 3: ULA de 1 bit**

carry out

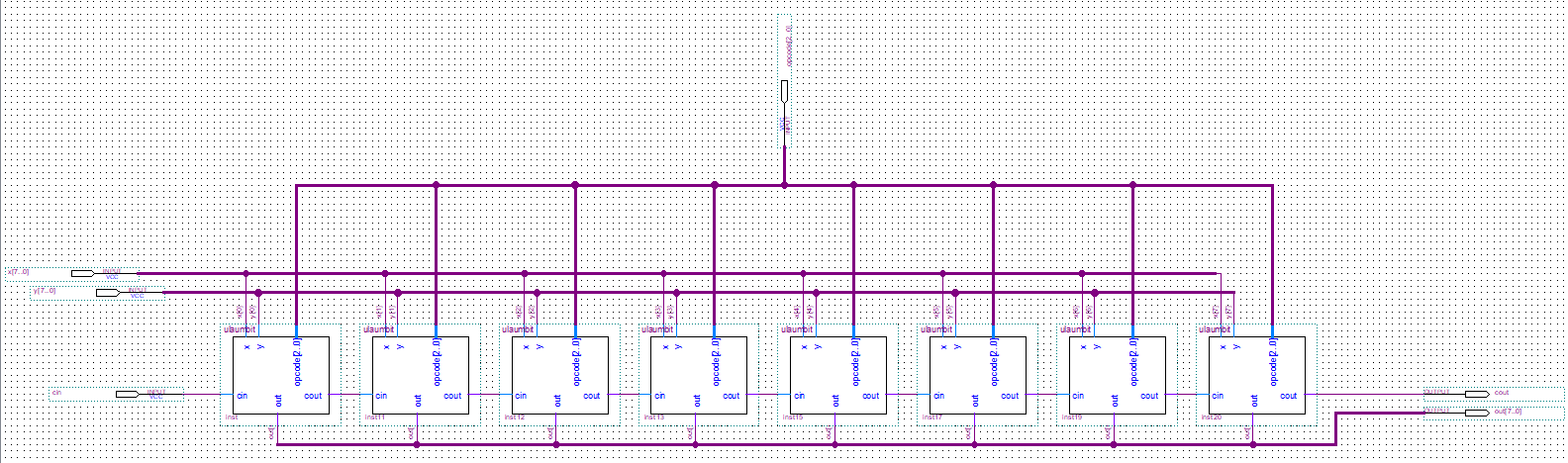
saída

Y e X, respectivamente

carry in

opcode

**Segundo passo: ULA de 8 bit, em *ripple-carry*.**

Com a ULA de 1 bit pronta, basta conectar recursivamente 8 ULAs de 1 bit para formar uma de 8 bits. Para X, Y e a saída, foram utilizados barramentos. Abaixo, a figura da ULA de 8 bits.

Carry out

Saída

X

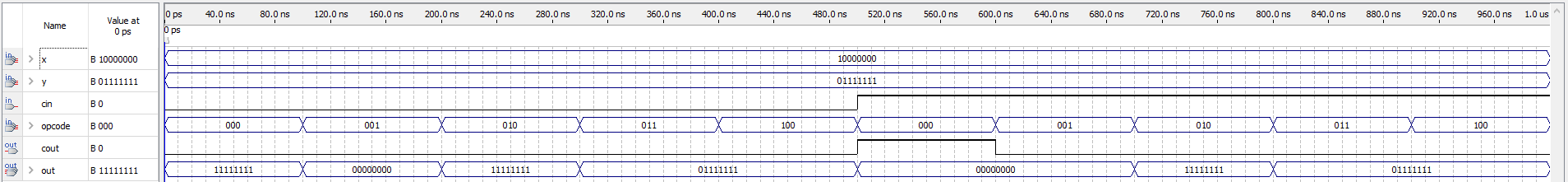
Y

Carry in

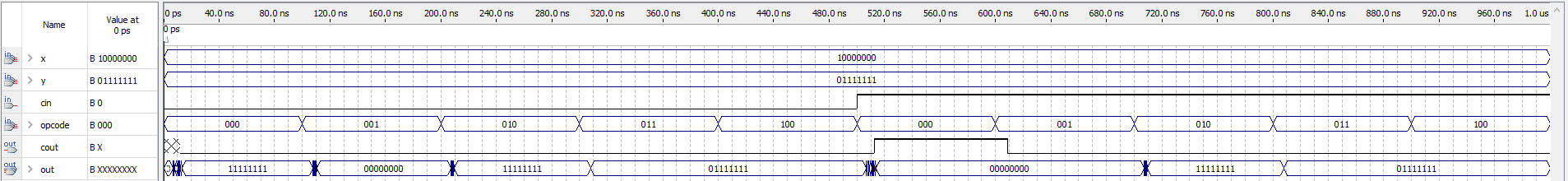
opcode

**Figura 4: ULA de 8 bits**

**Conclusão**

Ao realizar as simulações, foi observado o correto funcionamento da ULA de 8 bits. Abaixo, a simulação sem e com atraso, respectivamente.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X + Y | X and Y | X or Y | not X | Y | X + Y | X and Y | X or Y | not X | Y |

**Figura 5: simulação da ULA do Neander, sem atraso.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X + Y | X and Y | X or Y | not X | Y | X + Y | X and Y | X or Y | not X | Y |

**Figura 6: simulação da ULA do Neander, com atraso.**

Como foi visto, o circuito funciona perfeitamente, propagando o carry quando precisa e trocando de função desejada corretamente. Também, assumimos que esses circuitos não fornecem nenhuma saída se estiverem desligados.