UNIVERSIDADE FEDERAL DE PERNAMBUCO
CIN-CENTRO DE INFORMÁTICA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ALINE FRANCIELE CORREIA DA SILVA - AFCS
BRUNO SOARES DA SILVA – BSS3
MARIAMA CELI SERAFIM DE OLIVEIRA – MCSO
VINÍCIUS CARNEIRO PEREIRA SOUZA – VCPS
LUIZ ANTONIO DE VASCONCELOS FILHO - LAVF

Projeto de Sistemas Digitais

(ULA e Somador BCD)

RECIFE
2011

ALINE FRANCIELE CORREIA DA SILVA - AFCS
BRUNO SOARES DA SILVA – BSS3
MARIAMA CELI SERAFIM DE OLIVEIRA – MCSO
VINÍCIUS CARNEIRO PEREIRA SOUZA – VCPS
LUIZ ANTONIO DE VASCONCELOS FILHO - LAVF

Projeto de Sistemas Digitais

(ULA e Somador BCD)

Este trabalho acadêmico está relacionado à primeira unidade da disciplina de Sistemas Digitais (if-675), ministrada pelo professor Manoel Eusébio.

RECIFE
2011

Apresentação

Este projeto está vinculado à disciplina de Sistemas Digitais (cin.ufpe.br/~if675), e visa à aplicação prática de conceitos teóricos vistos em sala de aula. Dentre as especificações do projeto, estão: desenvolvimento de uma Unidade Lógica e Aritmética (ULA) e o Somador BCD.

O trabalho contou com a participação de cinco (05) componentes, que, durante quinze (15) dias, trabalharam em conjunto para que os maiores objetivos da disciplina sejam alcançados. Ou seja, os alunos, ao fim da disciplina, serão capazes de manipular e analisar, de forma eficiente, sistemas simples de hardware.

Sumário

1 Introdução

Neste trabalho acadêmico, relativo à disciplina de sistemas digitais, estão contidas as etapas de elaboração de uma Unidade Lógica e Aritmética (ULA) e um somador BCD. Para o desenvolvimento dos sistemas de hardware citados acima, seguiu-se uma filosofia de modularização e dividir para conquistar. Ou seja, iniciou-se o projeto a partir do estudo de cada unidade básica que compõe uma ULA ou um somador BCD, posto que a união de cada uma dessas partes funcionando corretamente formará o sistema final.

Devido a este processo de desenvolvimento, ao longo deste trabalho foram utilizados os seguintes métodos e técnicas na seguinte sequência: reconhecimento do problema a partir do uso de tabelas verdade, formulação de equações com o auxilio de mapas de Karnaugh, elaboração de circuitos lógicos a partir do uso de portas lógicas e, por último, a fase de testes. Este processo inicia-se em uma unidade básica e é repetido até o momento em que se chega ao sistema maior (ULA e somador BCD completo).

Além dessas técnicas, foram utilizadas, também, ferramentas computacionais, como o Quartus, que possibilitaram sua aplicação prática. Livros e anotações de aula serviram para o embasamento teórico do projeto.

Este trabalho teve o seu objetivo concluído quando foi possível chegar a um protótipo em hardware que não apresentasse falhas. No entanto, isto só foi possível devido ao uso das metodologias acima apresentadas.

2 Visão Geral da Unidade Logica e Aritmética (ULA)

Desde o surgimento dos primeiros computadores, a função de realizar cálculos está entre uma das principais atribuições desta invenção. A importância é tanta que muitas das operações aritméticas realizadas pelos computadores atuais são impossíveis de serem calculadas em tempo hábil por um ser humano. É devido a estas importância e necessidade que os computadores possuem uma unidade responsável pelas operações lógicas e aritméticas que são requisitadas à máquina. Este componente do computador é chamado de *Unidade de Lógica e Aritmética*, mais facilmente conhecido por sua abreviatura **ULA.**

O processo de resolução de cálculos em uma ULA é feita a partir do recebimento de informações em sua forma binária. Essa informação é manipulada em circuitos lógicos e aritméticos que são formados a partir da combinação de portas lógicas e flip-flops.

Neste projeto, ao implementar-se a ULA, fez-se a construção das seguintes partes que podem compor esta unidade:

-Somador/Subtrator;

-Complementador a 2;

-Função lógica AND;

-Função lógica XOR;

-Operações de Comparação: igualdade, maior que, menor que.

A seguir serão apresentadas essas partes, bem como a suas funções e como elas foram devidamente desenvolvidas nesse projeto. Além disso, serão mostrados outros componentes como multiplexadores e Decodificadores, visto que sem eles algumas funcionalidades da ULA proposta no projeto não poderiam ser implementadas, como o acendimento de LED em um display, que representa o resultado de uma operação realizada pela unidade.

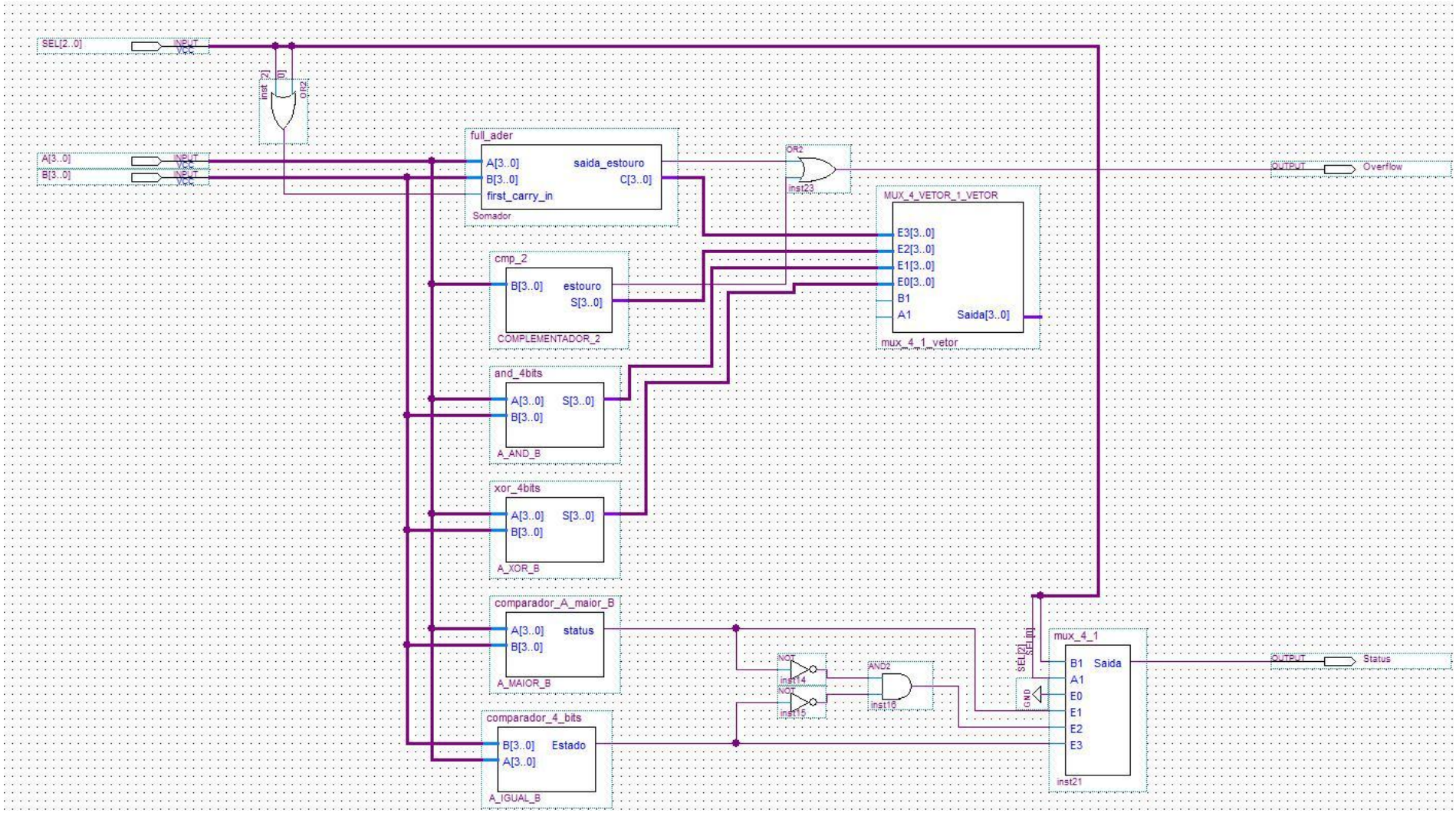

Figura 1 Implementação da ULA

2.1 Detalhamento dos Módulos da ULA

2.1.1 Somador / Subtrator

Este módulo tem a função de realizar as operações de soma e subtração, dada uma cadeia de números binários. Ele será composto de uma unidade básica responsável pela soma de dois bits (1 bit da entrada A e 1 bit da entrada B) e de uma unidade maior que reúne várias unidades básicas. Esta última tem a capacidade de realizar a soma de uma cadeia de vários bits.

Módulo básico do somador

Devido à complexidade de se realizar as operações de soma de forma conjunta, viu-se a necessidade de modularizar tal ação. Assim, partimos para a modularização da soma, deixando este trabalho abstraído em uma caixa. Sendo esta o somador de um bit, ela se encarrega apenas de somar dois bits da entrada e mostrar a saída. O processo de soma leva em conta a presença ou não de outro bit chamado *Carry In*. Este bit e o resultado de somas anteriores que levaram ao estouro da base (*overflow*) e consequentemente a necessidade de que houvesse a inclusão da sua soma no próximo passo.

Especificações do modulo:

Entradas:

1- Tipo: bit
   Nome: A
   Descrição: Primeiro bit a ser somado.
2- Tipo: bit
   Nome: B
   Descrição: Segundo bit a ser somado.
3- Tipo: bit
   Nome: Carry_in
   Descrição: Bit resultante do estoura da base de outras somas.

Saídas:

1- Tipo: bit
   Nome: Resultado
   Descrição: Bit resultante da operação de soma.
2- Tipo: bit
   Nome: Carry Out
   Descrição: Bit resultante do estouro da soma na base 2.

2.1.1.1 Tabela Verdade e Mapas de Karnaugh

O uso da tabela verdade e mapas de Karnaugh ao longo deste projeto será algo recorrente, uma vez que é a partir dela que será possível partir para uma resolução algébrica booleana do problema descrito. Por fim, chegaremos a uma função que resume o problema e que, além disso, possibilita a construção de um circuito lógico. Neste primeiro caso será calculada uma função para criar um circuito que represente o problema acima descrito do módulo do somado e subtrator, chamada de *Full Adder*.

A	B	CIn	Saída	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 1 Tabela- verdade do módulo:"Full-Adder"

Mapa de Karnaugh do módulo: "Full-Adder(1 bit)",para a função que gera a saída do módulo.

AB / CIn	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Mapa de Karnaugh do módulo: "Full-Adder(1 bit)",para a função que gera o Carry Out do módulo.

AB / CIn	00	01	11	10
0	0	0	1	0
1	0	1	1	1

A partir dos mapas foram encontradas as seguintes funções:

- FullAdder (soma1bit), carry out:
  AB + BCin + ACin
- FullAdder (soma1bit), soma:
  ¬A (B (+) CIn) + A (B (.) CIn)

2.1.1.2 Circuito Projetado e Simulação

O circuito encontrado após a aplicação da função com o uso de portas lógicas:

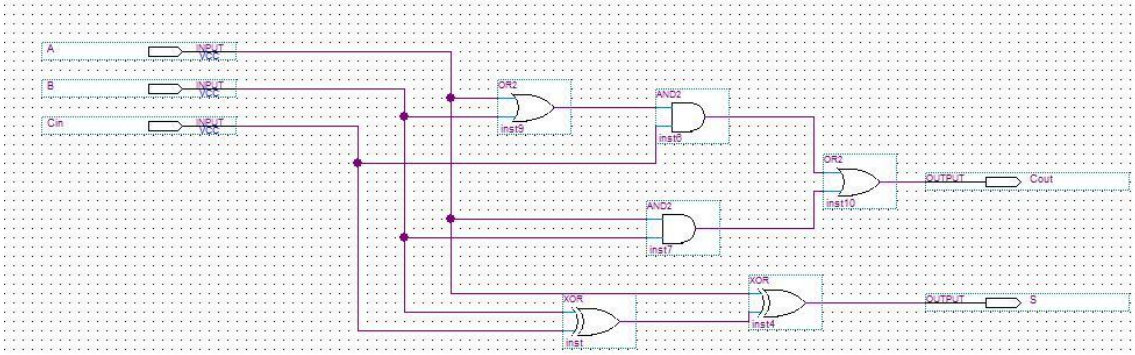

Figura 2 Implementação de um somador de um bit

Circuito do somador para uma cadeia com quatro bits:

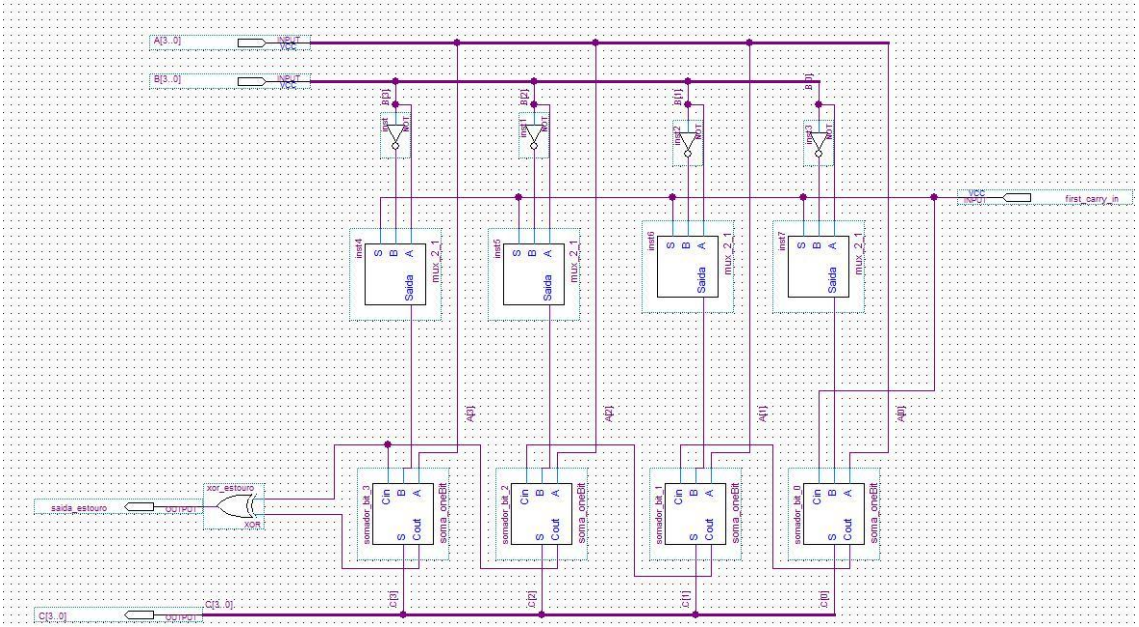

Figura 3 Implementação de um somador de quatro bits

Observações quanto ao somador de quatro bits:

- Devido a problemas de estouro (*overflow*) encontrados quando se faz a soma de números, utilizou-se a porta lógica XOR, que recebe o Cin e Cout, pois ela tem a capacidade de demonstrar as saídas nas quais ocorrem estouro quando Cin e Cout são respectivamente (0,1) e (1,0):

A[3]	B[3]	Cin	S	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Tabela 2 Casos de overflow no Full Adder

- O somador de quatro bits possui também uma porta lógica NOT e um multiplexador 2:1 (figura 2). Tais componentes são utilizados visando à inclusão da função de subtração a implementação. Tudo isto é feito adicionando um complementador a 2 ao sistema. Este complementador possibilita que um número seja transformado em negativo na notação complemento de 2 e consequentemente ele poderá ser manipulado da forma correta. Para tal, é feita sua inversão e logo após soma-se um bit à cadeia.

Ex:
2 = 0010
Invertendo os bits: 1101
Adicionando 1: 1101 + 0001 = 1110 = -2

Vantagens na aplicação: Só foi preciso implementar um módulo que realiza duas funções (subtrai e soma), visto que um número negativo na notação de complemento de dois pode ser somado com um positivo ou um negativo que a operação se realizará da forma correta.

Ex:
-2=1110
2=0010
1110 + 0010 = 0000 = 0

Simulação

Terminada a fase de implementação do sistema, foram feitos os devidos testes de verificação. Aqui estão apresentadas as simulações (waveform) realizadas com o auxílio do Quartus.

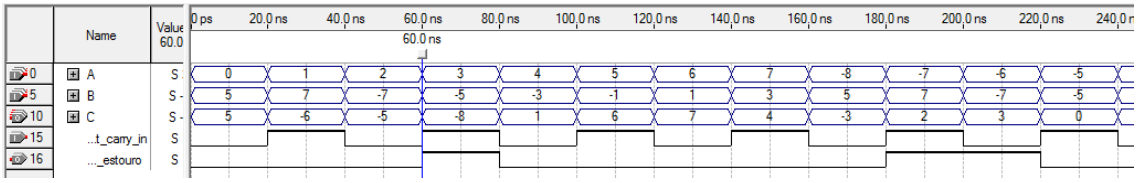

Figura 4 Waveform da simulação do somador de 4 bits

2.1.2 Complementador

A função do complementador na ULA é transformar cadeias binárias que representam um número em seu valor com sinal oposto, ou seja, caso um número seja 1, ele transformará em -1. Como, no caso da aplicação, são cadeias binárias de 4 bits, ele receberá um número que esteja entre este campo de codificação (7 a -8)* e o codificará em um número que tem sinal oposto.

Ex:
2 = 0010
Invertendo os bits: 1101

Adicionando 1: 1101 + 0001 = 1110 = -2

Isto é feito invertendo os bits da cadeia e adicionando 1 logo após. Como tal implementação já tinha sido feita no somador, foi reutilizado o seu módulo. No entanto, só há a entrada de uma cadeia de bits no somador, a outra que deveria ser um vetor de bits A está aterrada (figura 4).

*Somente os três últimos bits da cadeia codificam o número, o quarto bit restante, o mais significativo, codifica o sinal (1 é negativo e 0 é positivo).

2.1.2.1 Circuito Projetado e Simulação

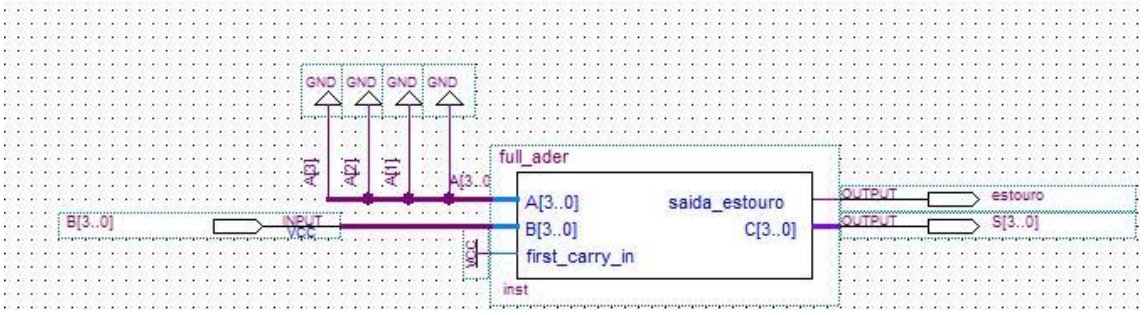

Figura 5 Implementação de um complemetador de quatro bits

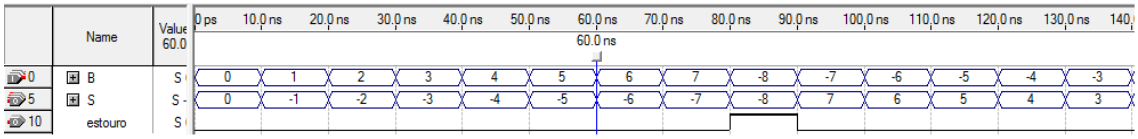

Figura 6 Waveform da simulação do complementador de 4 bits

2.1.3 And e Xor

2.1.3.1 And

Este módulo do sistema realiza a operação and a um vetor de bits de tamanho 4. Tal operação é realizada bit a bit a partir das portas lógicas AND, sendo aplicada a cada par de bits da mesma posição nos vetores de entrada. Assim, quando temos vetores iguais na entrada, por propriedades de álgebra booleana, temos que X AND X é X.

Especificações do módulo

Entradas:

1- Tipo: vetor
Nome: A
Descrição: Primeiro vetor de entrada com o bit 3 como mais significante.

1.1- Tipo: bit
Nome: A[0]
Descrição: bit 0 do vetor A.

1.2- Tipo: bit
Nome: A[1]
Descrição: bit 1 do vetor A.

1.3- Tipo: bit
Nome: A[2]
Descrição: bit 2 do vetor A.

1.4- Tipo: bit
Nome: A[3]
Descrição: bit 3 do vetor A.

1.5- Tipo: vetor
Nome: B
Descrição: Segundo vetor de entrada com o bit 3 como mais significante.

1.6- Tipo: bit
Nome: B[0]
Descrição: bit 0 do vetor B.

1.7- Tipo: bit
Nome: B[1]
Descrição: bit 1 do vetor B.

1.8- Tipo: bit
Nome: B[2]
Descrição: bit 2 do vetor B.

1.9- Tipo: bit
Nome: B[3]
Descrição: bit 3 do vetor B.

Saídas:

2- Tipo: vetor
Nome: Saida
Descrição: vetor com o resultado da operação com o bit 3 como mais significante.

2.1-Tipo: bit
Nome: Saída[0]
Descrição: bit 0 do vetor com o resultado da operação.

2.2-Tipo: bit
Nome: Saída[1]
Descrição: bit 1 do vetor com o resultado da operação.

2.3-Tipo: bit
Nome: Saída[2]
Descrição: bit 2 do vetor com o resultado da operação.

2.4-Tipo: bit
Nome: Saída[3]
Descrição: bit 3 do vetor com o resultado da operação.

2.1.3.2 Xor

Este módulo do sistema realiza a operação xor a um vetor de bit de tamanho 4. Tal operação é feita a partir de portas lógicas XOR, que são aplicadas a cada par de bits da mesma posição dos vetores de entrada. Assim, quando temos vetores iguais na entrada, teremos um vetor de valor 0 na saída devido a propriedade da operação lógica XOR.

Especificações do módulo

Entradas:

1- Tipo: vetor
Nome: A
Descrição: Primeiro vetor de entrada com o bit 3 como mais significante.
1.1- Tipo: bit
Nome: A[0]
Descrição: bit 0 do vetor A.
1.2- Tipo: bit
Nome: A[1]
Descrição: bit 1 do vetor A.
1.3- Tipo: bit
Nome: A[2]
Descrição: bit 2 do vetor A.
1.4- Tipo: bit
Nome: A[3]
Descrição: bit 3 do vetor A.

2- Tipo: vetor
Nome: B
Descrição: Segundo vetor de entrada com o bit 3 como mais significante.
2.1- Tipo: bit
Nome: B[0]
Descrição: bit 0 do vetor B.
2.2- Tipo: bit
Nome: B[1]
Descrição: bit 1 do vetor B.
2.3- Tipo: bit
Nome: B[2]
Descrição: bit 2 do vetor B.
2.4- Tipo: bit
Nome: B[3]
Descrição: bit 3 do vetor B.

Saídas:

3- Tipo: vetor
Nome: Saída
Descrição: Vetor com o resultado da operação com o bit 3 como mais significante.

3.1- Tipo: bit
Nome: Saída[0]
Descrição: bit 0 do vetor com o resultado da operação.

3.2- Tipo: bit
Nome: Saída[1]
Descrição: bit 1 do vetor com o resultado da operação.

3.3- Tipo: bit
Nome: Saída[2]
Descrição: bit 2 do vetor com o resultado da operação.

3.4- Tipo: bit
Nome: Saída[3]

Descrição: bit 3 do vetor com o resultado da operação.

2.1.3.2.1 Tabela Verdade e Mapas de Karnaugh

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 3 Tabela-verdade do módulo: “And para 1 bit”

A	B	Saída
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 4 Tabela-verdade do módulo: “Xor para 1 bit”

2.1.3.2.2 Circuito Projetado e Simulação

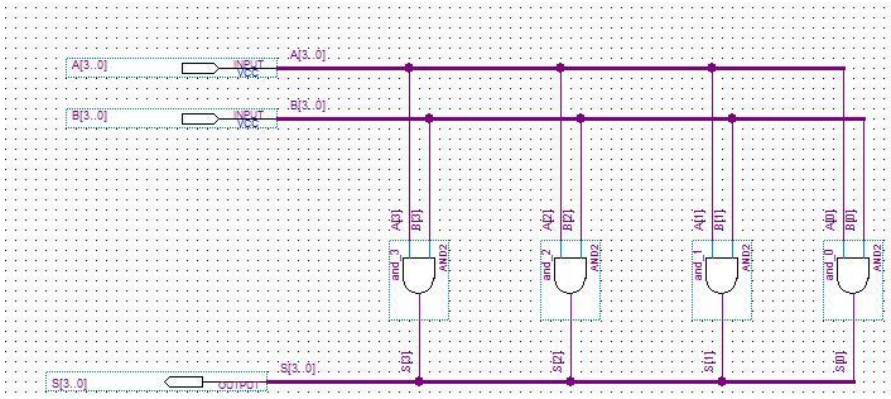

Figura 7 Implementação do AND de quatro bits

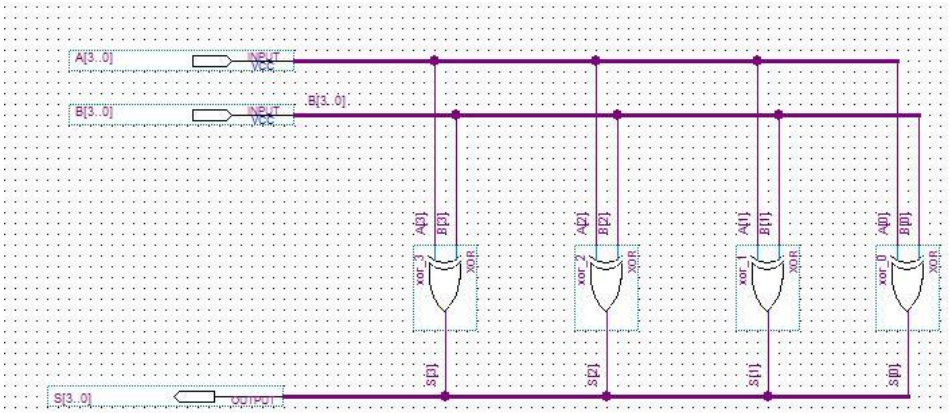

Figura 8 Implementação do XOR de quatro bits

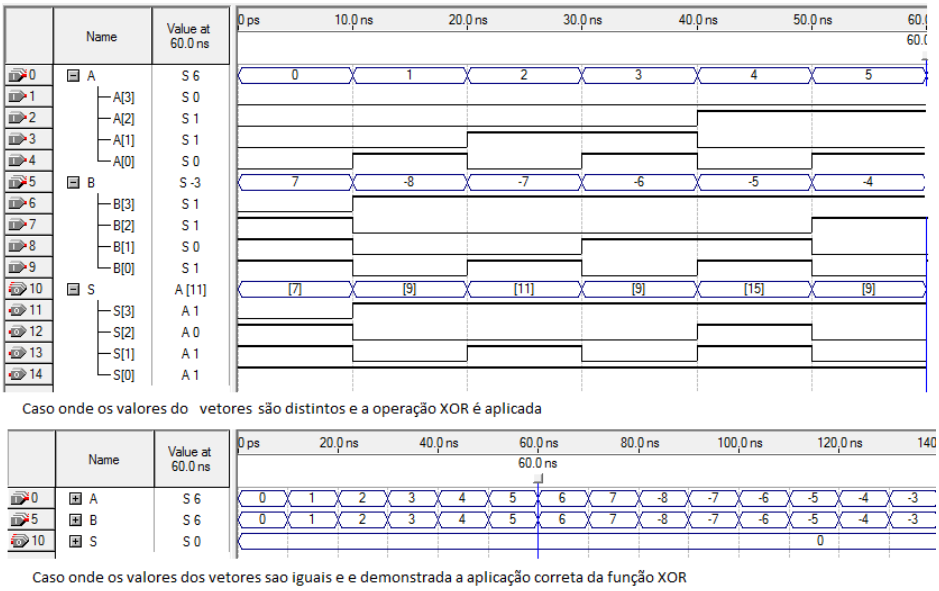

Figura 9 Waveform da simulação do XOR de 4 bits

2.1.4 Comparadores Lógicos

Estes módulos são responsáveis pela comparação de dois números, com as seguintes comparações: *maior que*, *menor que* e *igual a*.

2.1.4.1 Maior que

Devido à complexidade de se realizar a comparação entre duas cadeias de bits, realizamos a modularização do problema, quebrando-o em blocos. O bloco de comparação de bits funciona recebendo dois bits: um na entrada A, outro na entrada B, um status que diz se a comparação anterior já havia sido maior e uma saída. O módulo realiza a análise fornecendo 1 na saída, caso A seja maior que B, e 0, caso não seja maior. Em casos onde os bits são iguais (A=B), o módulo fornece na saída o resultado da ultima comparação.

Entradas:

1- Tipo: bit
   Nome: A
   Descrição: Primeiro bit, base da comparação.
2- Tipo: bit
   Nome: B
   Descrição: Segundo bit que e comparado com o bit A.
3- Tipo: bit
   Nome: Status
   Descrição: Bit que carrega a informação da ultima comparação.

Saídas:

1- Tipo: bit
   Nome: Saída
   Descrição: Bit que diz se A e maior que B(1) ou que não e maior (0).

Na tarefa de comparar uma cadeia de bits com sinal, existe um ponto critico; o bit que representa o sinal. Esta comparação é diferente da aplicada aos bits que representam o número, pois é inversa. Assim, modularizando o processo de comparação de cadeias de bits, implementamos o módulo de comparação do bit mais significativo. O bit da entrada B é comparado com o da entrada A e, em casos onde o bit A for 1 e o bit B for 1, é dado como saída o valor 0, pois A é negativo e B é positivo. Já em caso contrario, é dado como valor da saída 0. Quando temos o caso dos dois bits iguais (A = B), o módulo repassa a informação que tinha anteriormente sobre a última comparação de bits que foi realizada.

Entradas:

1- Tipo: bit
   Nome: A
   Descrição: Primeiro bit, base da comparação.
2- Tipo: bit
   Nome: B
   Descrição: Segundo bit que e comparado com o bit A.
3- Tipo: bit
   Nome: Status
   Descrição: Bit que carrega o status da comparação anterior.

Saídas:

- Tipo: bit Nome: Saída
  Descrição: Bit que carrega o status final da operação, informando se A e maior que B (1) ou que não e maior (0).

2.1.4.2 Menor que

Não foi necessária a criação do módulo menor que, pois o módulo *maior-que* foi reutilizado para essa aplicação. Para que o reuso fosse possível, colocou-se uma saída NOT a partir de uma extensão da saída do módulo *maior que*, visto que a negação de “A maior-que B” satisfaz a pergunta “A menor-que B” (figura 10).

2.1.4.3 Igual a

Devido à complexidade de se tratar a comparação de igualdade entre vetores de 4 bits, foi necessário recorrer à modularização da mesma, fazendo-a agora em blocos de 1 bit. Cada bloco desses faz a comparação dos bits correspondentes de cada vetor(sendo A e B os vetores de entrada, teremos, por exemplo, comparações entre A[3] e B[3],entre A[2] e B[2],e assim por diante).Cada módulo desses recebe os dois bits a comparar, um status anterior e gera um status atual. O status anterior é fornecido pelo módulo anterior e indica se até o presente momento a igualdade se verifica. É então gerado o status atual, que indica se, após a comparação efetuada pelo módulo atual, a igualdade dos vetores ainda se verifica. Se o status anterior for "0", então o status atual também será "0", visto que o módulo anterior está avisando ao atual que a igualdade não foi comprovada.

Entradas:

1- Tipo: bit
   Nome: A
   Descrição: Primeiro bit, base da comparação.

2- Tipo: bit
   Nome: B
   Descrição: Segundo bit que e comparado com o bit A.
3- Tipo: bit
   Nome: Status
   Descrição: Bit que carrega a informação da ultima comparação.

Saídas:

1- Tipo: bit
   Nome: Saída
   Descrição: Bit que diz se A e igual a B ou não.

Módulo para comparação de igualdade entre cadeias de 4 bits

Este módulo do sistema realiza a verificação de igualdade entre dois vetores de 4 bits, o que é realizado bit a bit por meio de módulos menores, os quais operam bit a bit (nos bits correspondentes de cada vetor de entrada, como em A[3] com B [3] e assim por diante(se os vetores A e B são as entradas do módulo maior que está sendo descrito)), e que são combinados para gerar a saída pra verificação da igualdade entre os vetores.

Especificações do módulo

Entradas:

1- Tipo: vetor
   Nome: A
   Descrição: Primeiro vetor de entrada com o bit 3 como mais significante.
1.1- Tipo: bit
   Nome: A[0]
   Descrição: bit 0 do vetor A.
1.2- Tipo: bit
   Nome: A[1]
   Descrição: bit 1 do vetor A.
1.3- Tipo: bit
   Nome: A[2]
   Descrição: bit 2 do vetor A.
1.4- Tipo: bit
   Nome: A[3]
   Descrição: bit 3 do vetor A.

2- Tipo: vetor
   Nome: B
   Descrição: Segundo vetor de entrada com o bit 3 como mais significante.
2.1-Tipo: bit
   Nome: B[0]
   Descrição: bit 0 do vetor B.

2.2-Tipo: bit
Nome: B[1]
Descrição: bit 1 do vetor B.
2.3-Tipo: bit
Nome: B[2]
Descrição: bit 2 do vetor B.
2.4-Tipo: bit
Nome: B[3]
Descrição: bit 3 do vetor B.

Saídas:

3- Tipo: bit
Nome: Saída
Descrição: Bit que indica se o vetor A é igual ao vetor B

2.1.4.3.1 Tabela Verdade e Mapas de Karnaugh

A	B	Status	Saída
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

Tabela 5 Tabela-verdade do módulo: "comparador (A > B) 1 bit, em que o bit comparado não é o mais significativo"

A	B	Status	Saída
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tabela 6 Tabela-verdade do módulo: "comparador (A > B) 1 bit, em que o bit comparado é o mais significativo"

A	B	Status	Saída
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tabela 7 Tabela-verdade do módulo: "comparador (A = B)1 bit"

Mapa de Karnaugh do módulo: “comparador (A > B) 1 bit, em que o bit comparado não é o mais significativo”

AB / Status	00	01	11	10
0	0	0	0	1
1	1	0	1	1

Mapa de Karnaugh do módulo: “comparador (A > B) 1 bit, em que o bit comparado é o mais significativo”

AB / Status	00	01	11	10
0	0	1	0	0
1	1	1	1	0

Mapa de Karnaugh do módulo: “comparador (A = B) 1 bit”

AB / Status	00	01	11	10
0	0	0	0	0
1	1	0	1	0

Funções encontradas a partir dos mapas de Karnaugh

- Comparador igualdade (S1 bit):
  E (A (·) B)
- Comparador maior, bit normal (1 bit):
  ¬B (A + E) + E (A + ¬B)
- Comparador maior, bit mais significativo (1 bit):
  B (E + ¬A) + ¬A (E + B)

2.1.4.3.2 Circuito Projetado e Simulação

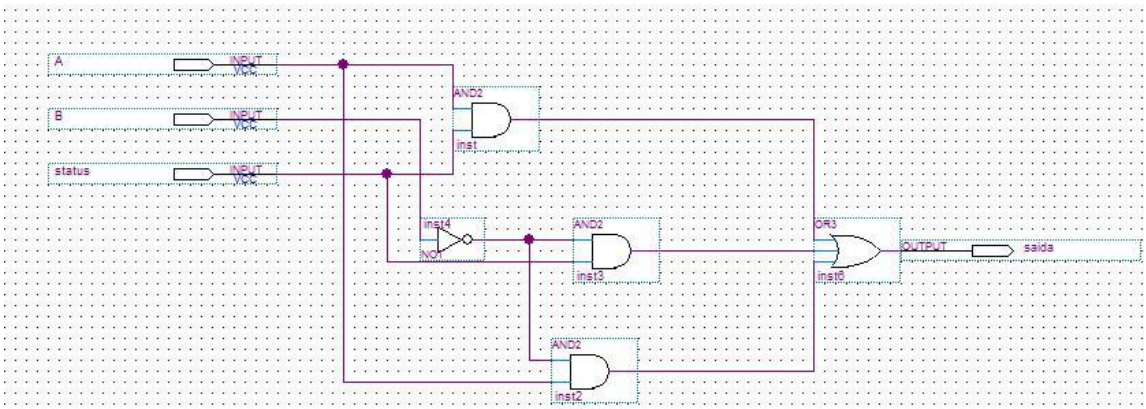

Figura 10 Implementação do maior-que para os bits antes do mais significativo

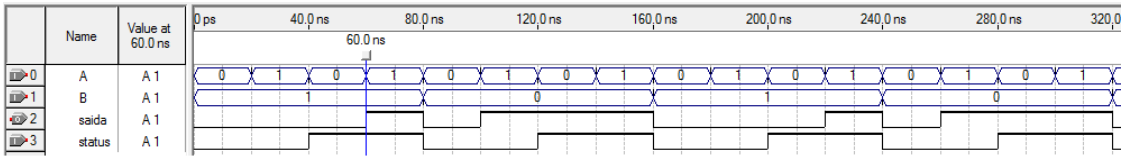

Figura 11 Waveform da simulação do maior-que para os bits antes do mais significativo

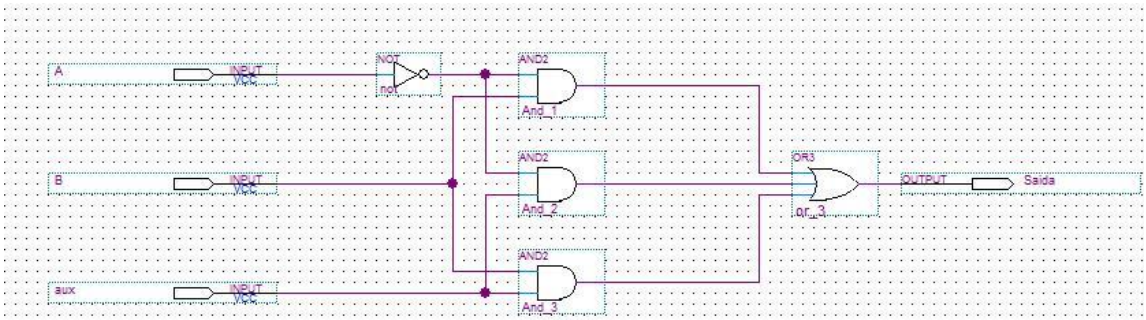

Figura 12 Implementação do maior-que para o bit mais significativo

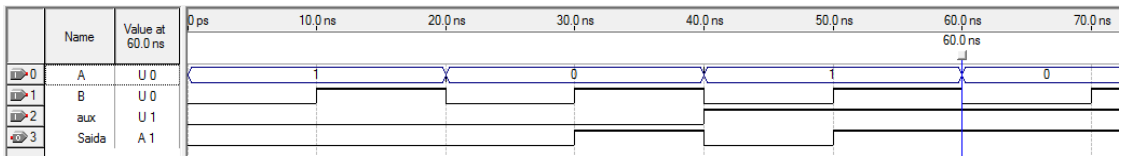

Figura 13 Waveform da simulação do maior-que para o bit mais significativo

Juntando os circuitos apresentados nas figuras 10 e 12, tem-se o comparador maior que:

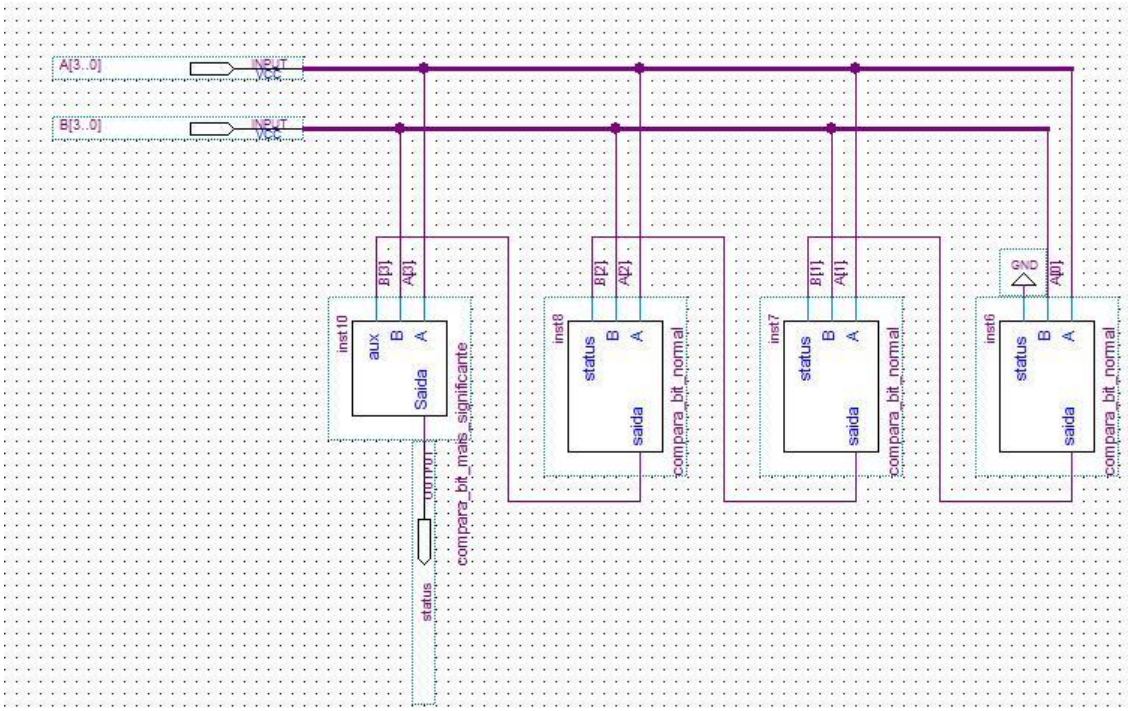

Figura 14 Implementação do comparador maior-que de 4 bits

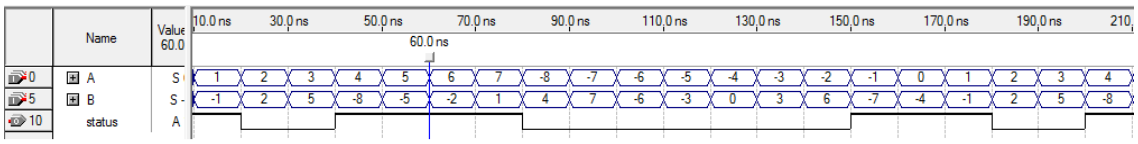

Figura 15 Waveform da simulação do comparador maior-que

Os esquemas do comparador de igualdade estão representados a seguir:

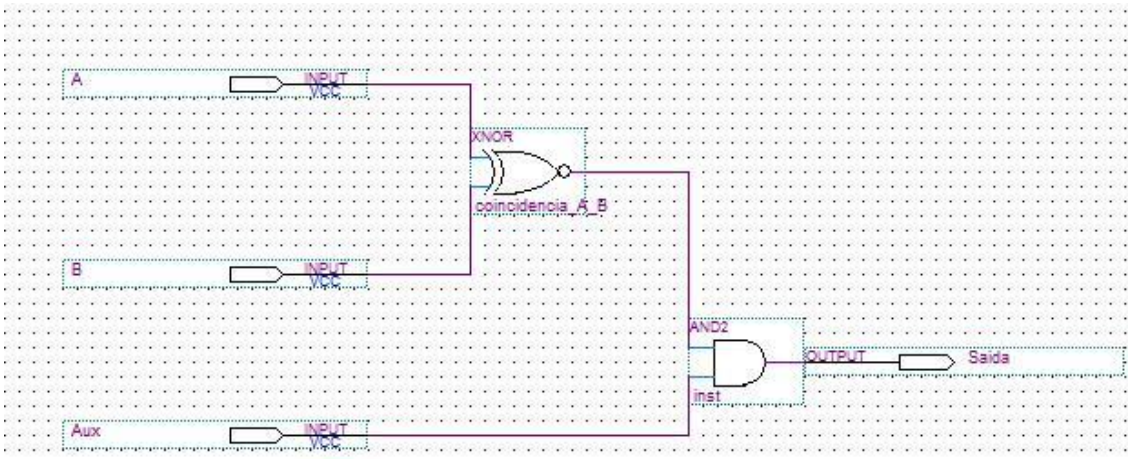

Figura 16 Implementação do comparador igual de 1 bit

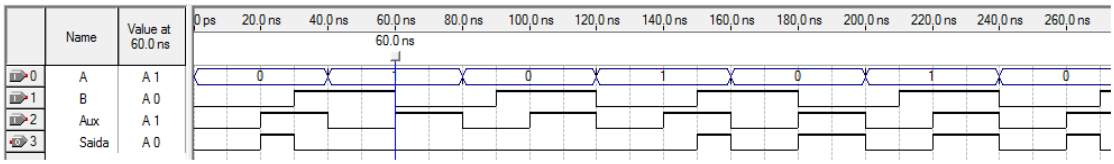

Figura 17 Waveform da simulação do comparador igual de 1 bit

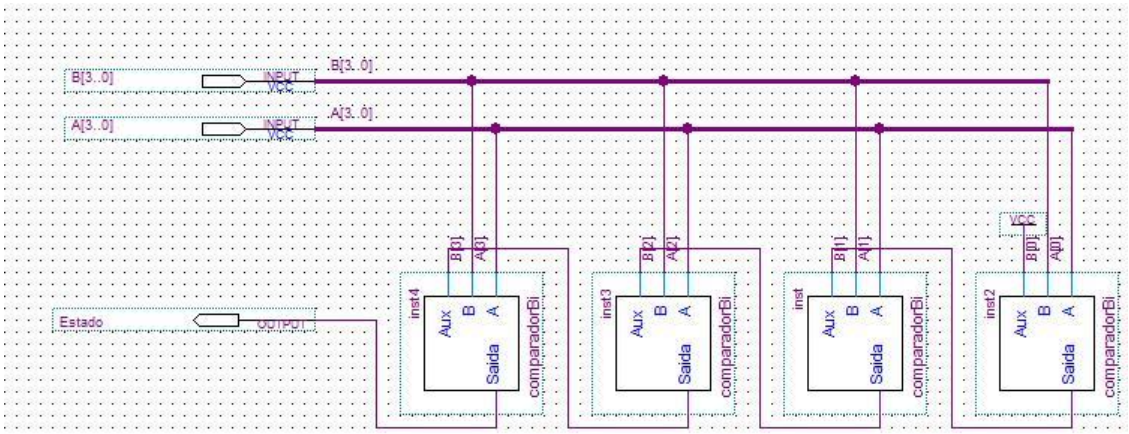

Figura 18 Implementação do comparador igual de 4 bits

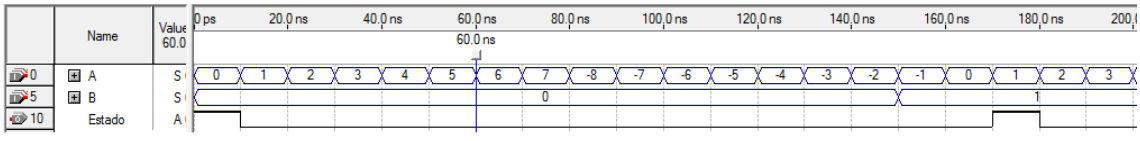

Figura 19 Waveform da simulação do comparador igual de 4 bits

2.1.5 Multiplexadores

Os multiplexadores exercem um importante papel em um circuito. Isso se deve ao fato deles servirem como chaves seletoras que selecionam uma entre varias entradas dependendo da chave que lhes forem passadas. Na aplicação do projeto, foram utilizados vários tipos de multiplexadores. São eles: 2:1 para bits, 2:1 para vetores de 4 bits e 4:1 para vetores de 4 bits.

Multiplexador 2:1 de 1 bit

Devido à necessidade de selecionar entre duas entradas, fez-se necessária a implementação de um Multiplexador, este servindo como seletor entre entradas, se utilizando de chaves seletoras que lhe são passadas. Assim, temos abaixo a descrição dos módulos do multiplexador que seleciona entre duas entradas aquela que deve passar, utilizando-se de um bit para tal ação. Esta seleção se baseia no conceito de que o bit que selecionará entre as entradas conterá o endereço da entrada que passará pela multiplexação. Dessa forma, temos que, se o bit de seleção for 0, o valor da entrada A passará. Caso contrário, será o valor da entrada B.

Especificações do módulo:

Entradas:

3- Tipo: bit
   Nome: A
   Descrição: Primeiro bit a ser multiplexado.
2- Tipo: bit
   Nome: B
   Descrição: Segundo bit a ser multiplexado.
3- Tipo: bit
   Nome: Seletor
   Descrição: Bit que selecionara uma das entradas.

Saídas:

1- Tipo: bit
   Nome: Saída
   Descrição: Bit com o resultado da multiplexação.

2.1.5.1 Tabela Verdade e Mapas de Karnaugh

As tabelas verdade abaixo resumem o funcionamento de dada uma chave (*Seletor A* ou *Seletor B)*.

Seletor A	Seletor B	Saída
0	0	E_0
0	1	E_1
1	0	E_2
1	1	E_3

Tabela 8 Tabela-verdade do módulo: "Mux (4:1)"

Seletor A	Saída
0	E_0
1	E_1

Tabela 9 Tabela-verdade do módulo:"Mux (2:1)"

Função do Multiplexador:

- Mux (2:1), (1 bit):

  Bse + AB + A¬Se

2.1.5.2 Circuito Projetado e Simulação

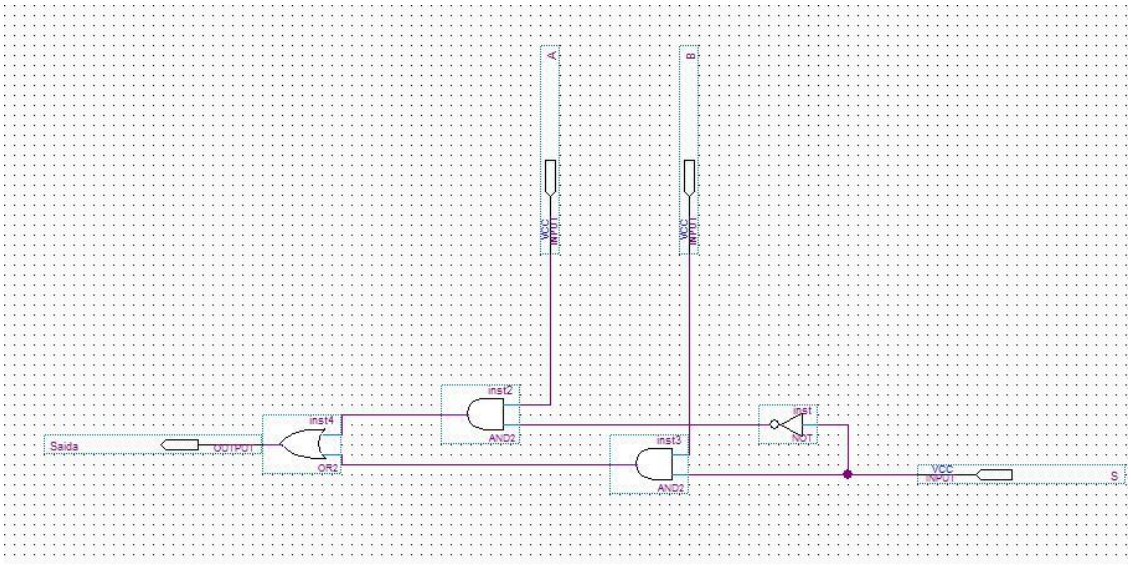

Figura 20 Implementação do multiplexador 2:1 de 1 bit

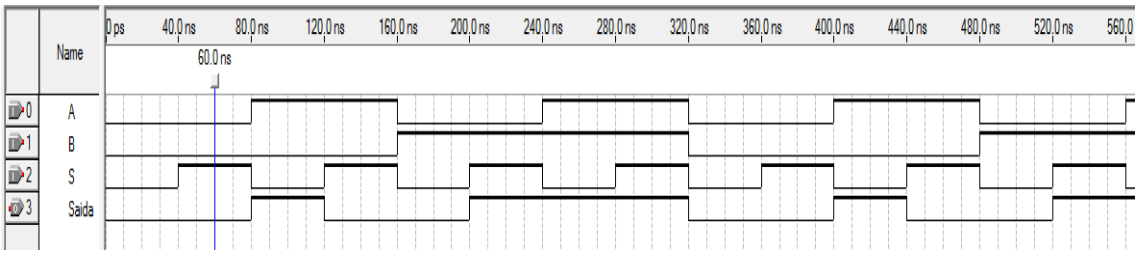

Figura 21 Waveform da simulação do multiplexador 2:1 de 1 bit

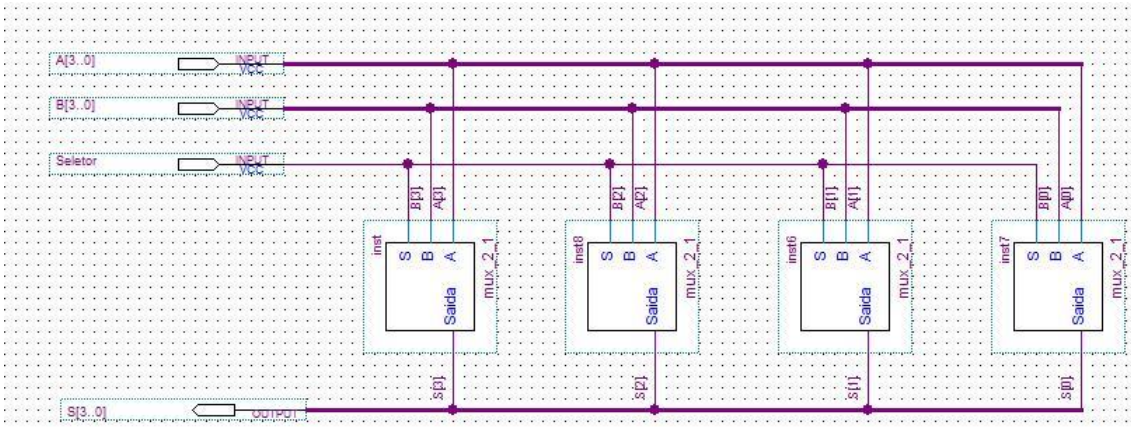

Figura 22 Implementação do multiplexador2:1 de 4 bits

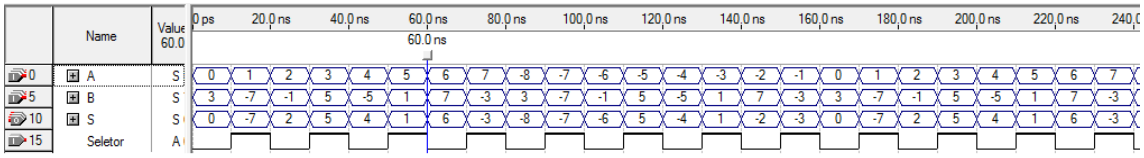

Figura 23 Waveform da simulação do multiplexador 2:1 de 4 bits

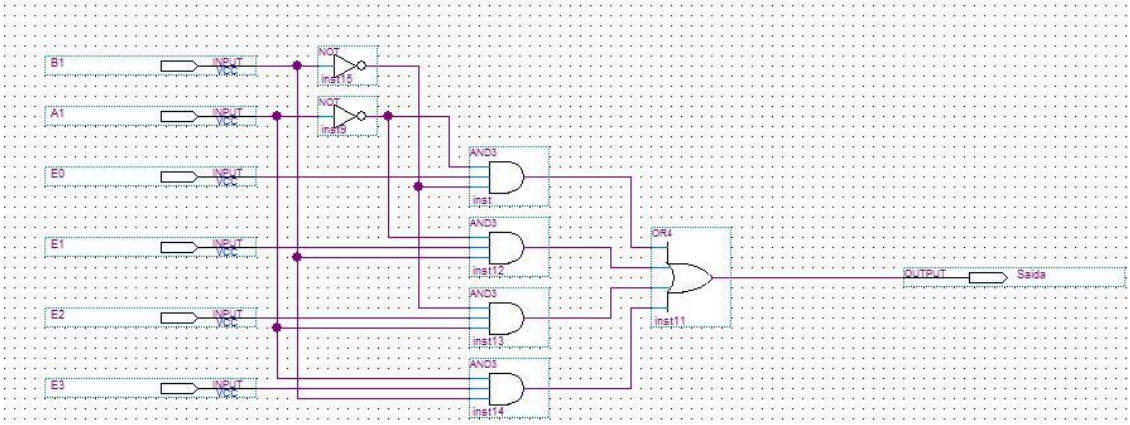

Figura 24 Implementação do multiplexador 4:1 de 1 bit

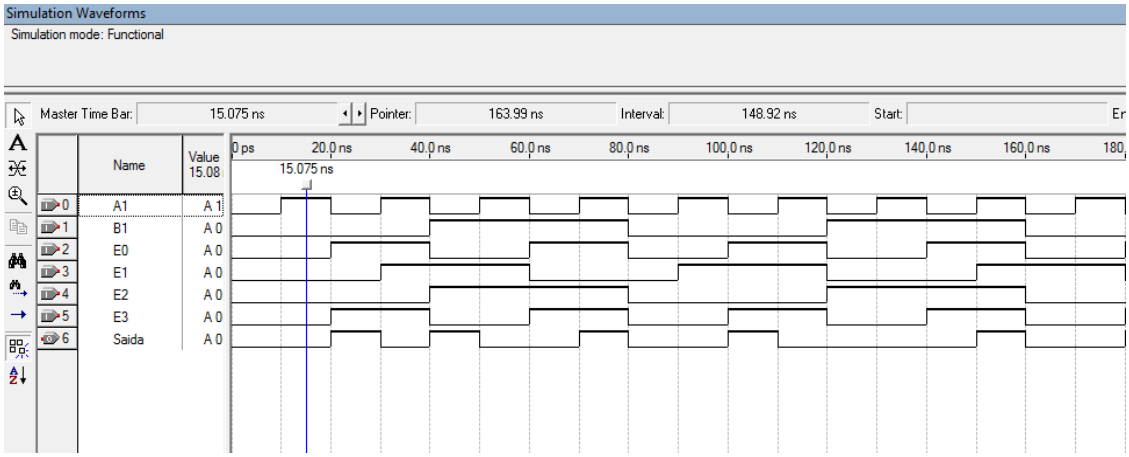

Figura 25 Waveform da simulação do multiplexador 4:1 de 1 bit

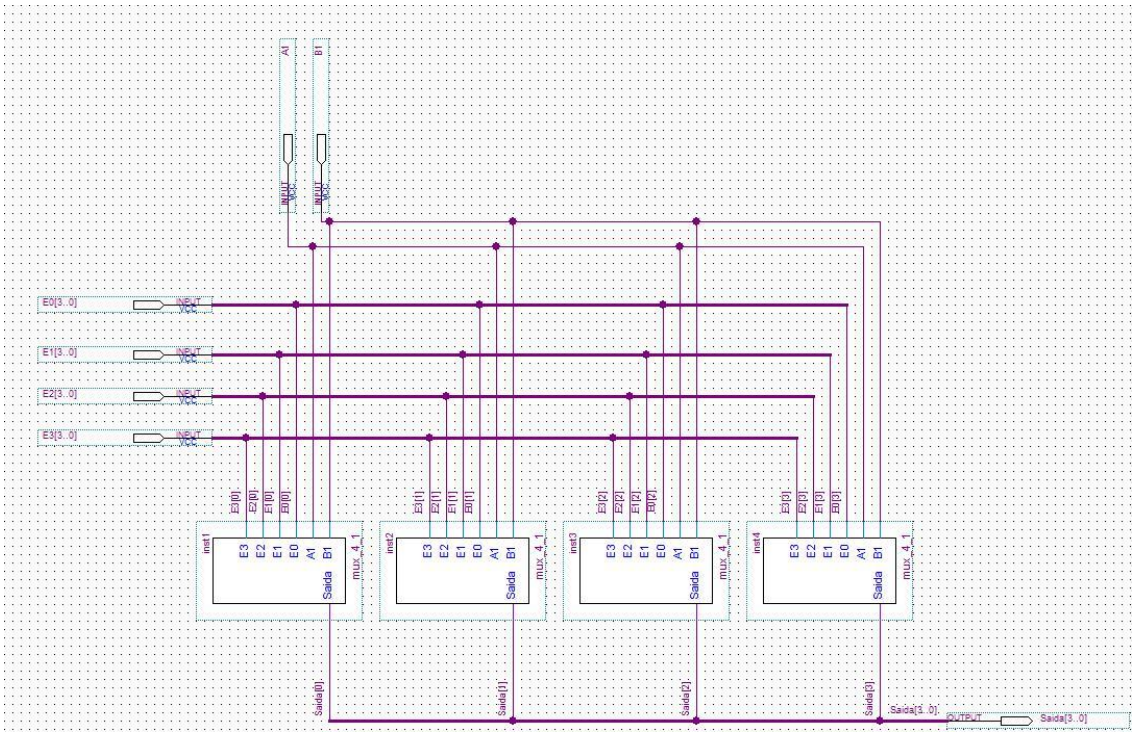

Figura 26 Implementação do multiplexador 4:1 de 4 bits

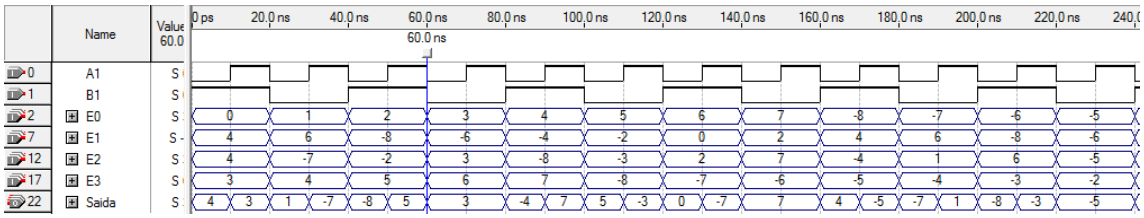

Figura 27 Waveform da simulação do multiplexador 4:1 de 4 bits

2.1.6 Decodificador

Na ULA se fará uso do decodificador para ativar os leds do display, no qual será mostrado o resultado do processo que foi pedido para ULA realizar. Este display possui 7 leds, que, para facilitar o seu manejo nessa aplicação, foram nomeados com as letras de **a** até **g**, como mostra a figura abaixo:

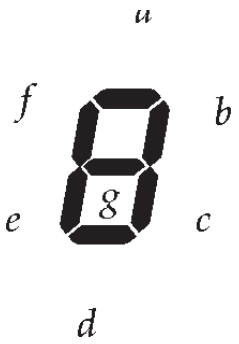

Figura 28 Representação do display de leds

Valor decimal da cadeia $A_3A_2A_1A_0$	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

Tabela 10 Tabela que correlaciona os valores decimais com as cadeias binária de 4 bits.

Cada led possui sua própria tabela verdade. Cada uma foi elaborada de forma a decodificar uma sequência de um vetor de 4 bits. Essa cadeia representa um número de 0 à 8 em sua forma binária. Se decodificada corretamente, a luz do display se acenderá de acordo com o valor correspondente da cadeia binária em valor decimal. Como exemplo para ilustrar essa aplicação, tem-se o vetor “0000”, que representa 0 em valor decimal. Vê-se que os leds que se acenderão serão os representados pelas letras: a, b, c, d, e, f. Para que isso aconteça, estabelece-se todos os casos na tabela verdade, e posteriormente elabora-se o circuito lógico. Verifique as tabelas verdade apresentadas abaixo, para melhor compreensão.

Obs.: A luz do circuito é ativada caso o valor de saída seja igual a zero.

2.1.6.1 Tabela Verdade e Mapas de Karnaugh

Seletor	A_3	A_2	A_1	A_0	A	b	c	d	E	f	g
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	0
1	0	0	1	0	0	0	1	0	0	1	0
1	0	0	1	1	0	0	0	0	1	1	0
1	0	1	0	0	1	0	0	1	1	0	0
1	0	1	0	1	0	0	0	0	1	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	0	1	1	1	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	-	-	-	-	-	-	-
0	x	x	x	x	1	1	1	1	1	1	1

Tabela 11 Tabela-verdade do módulo: Decodificador Completo

Mapa de Karnaugh do módulo: “Demultiplexador a”

SeletorA_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	0	1	-	0
01	1	1	1	1	1	0	-	-
11	1	1	1	1	0	0	-	-
10	1	1	1	1	0	0	-	-

Mapa de Karnaugh do módulo: “Demultiplexador b”

SeletorA_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	0	0	-	0
01	1	1	1	1	0	1	-	-
11	1	1	1	1	0	0	-	-
10	1	1	1	1	0	1	-	-

Mapa de Karnaugh do módulo: “Demultiplexador c”

SeletorA_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	0	0	-	0
01	1	1	1	1	0	0	-	-
11	1	1	1	1	0	0	-	-
10	1	1	1	1	1	0	-	-

Mapa de Karnaugh do módulo: “Demultiplexador d”

Seletor A_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	0	1	-	0
01	1	1	1	1	1	0	-	-
11	1	1	1	1	0	1	-	-
10	1	1	1	1	0	0	-	-

Mapa de Karnaugh do módulo: “Demultiplexador e”

Seletor A_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	0	1	-	0
01	1	1	1	1	1	1	-	-
11	1	1	1	1	1	1	-	-
10	1	1	1	1	0	0	-	-

Mapa de Karnaugh do módulo: “Demultiplexador f”

Seletor A_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	0	0	-	0
01	1	1	1	1	1	0	-	-
11	1	1	1	1	1	1	-	-
10	1	1	1	1	1	0	-	-

Mapa de Karnaugh do módulo: “Demultiplexador g”

Seletor A_3A_2 / A_1A_0	000	001	011	010	100	101	111	110
00	1	1	1	1	1	0	-	0
01	1	1	1	1	1	0	-	-
11	1	1	1	1	0	1	-	-
10	1	1	1	1	0	0	-	-

Equações encontradas a partir dos Mapas:

a = (¬S) + (¬A2¬A1A0) + (A2¬A1¬A0)

b = (¬S) + (A2¬A1A0) + (A2A1¬A0)

c = (¬S) + (¬A2A1¬A0)

d = (¬S) + (¬A2¬A1A0) + (A2¬A1¬A0) + (A2A1A0)

e = (¬S) + (A0) + (A2¬A1)

f = (¬S) + (¬A2A0) + (¬A2A1) + (A1A0)

g = (¬S) + (¬A3¬A2¬A1) + (A2A1A0)

2.1.6.2 Circuito Projetado e Simulação

No waveform dos circuitos do decodificador, foram testados os casos em que a chave seletora tem valor igual a 1 e quando ela tem valor igual 0. Quando ela tem valor de entrada igual a 0, todas as saídas terão valor igual a 1, visto que o seu valor sempre será 1 quando ela estiver desativada. Quando a chave seletora apresenta valor igual 1, as saídas do decodificador apresentarão saídas de acordo com o valor que lhe foi passado nas entradas.

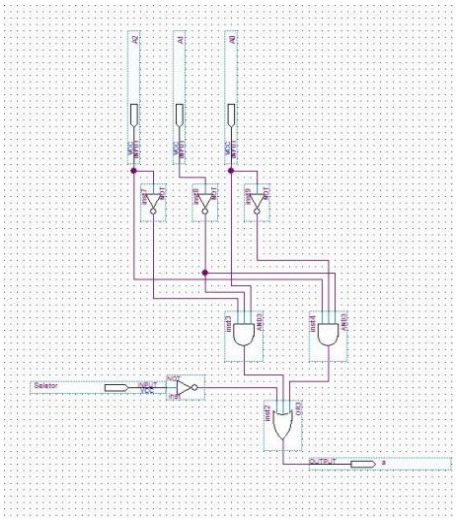

Figura 29 Modulo do decodificador parte “a”

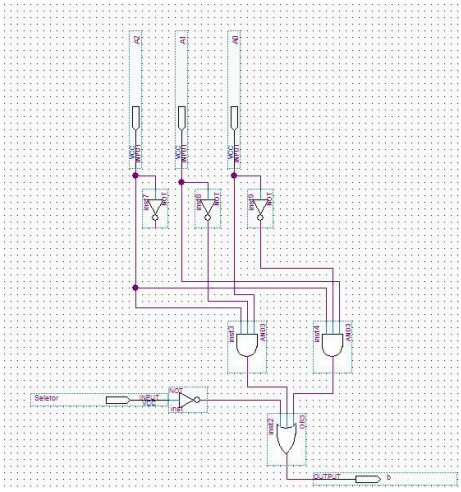

Figura 30 Modulo do decodificador parte “b”

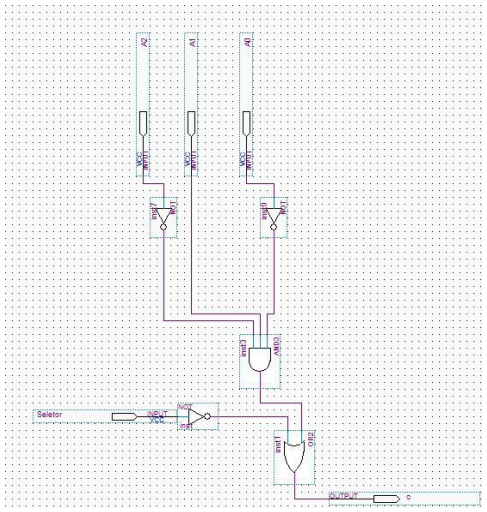

Figura 31 Modulo do decodificador parte “c”

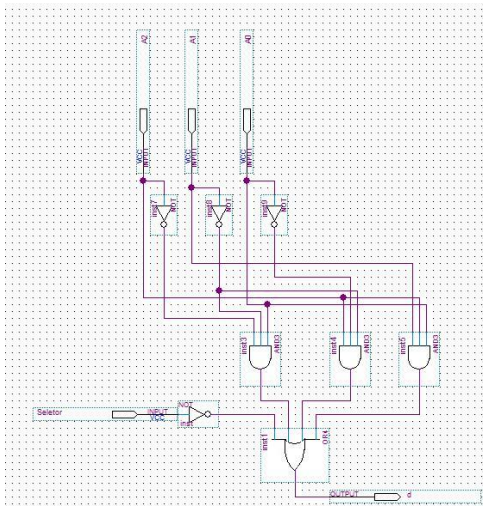

Figura 32 Modulo do decodificador parte “d”

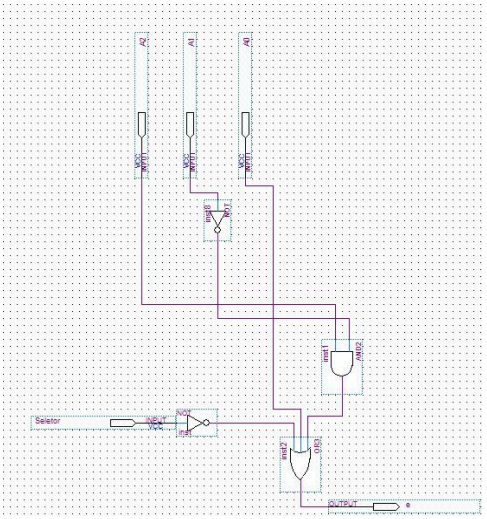

Figura 33 Modulo do decodificador parte “e”

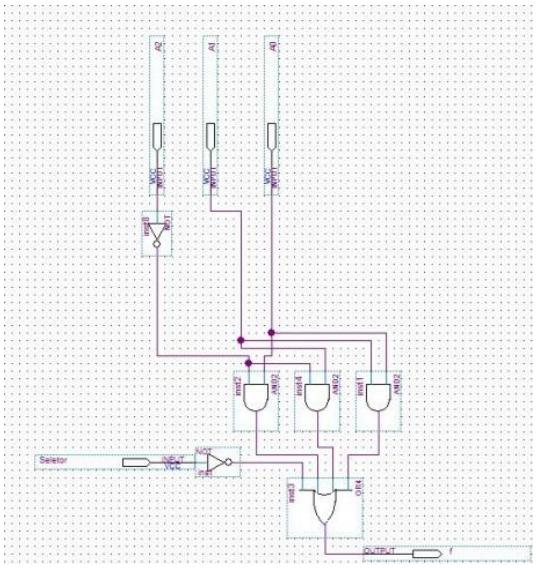

Figura 34 Modulo do decodificador parte “f”

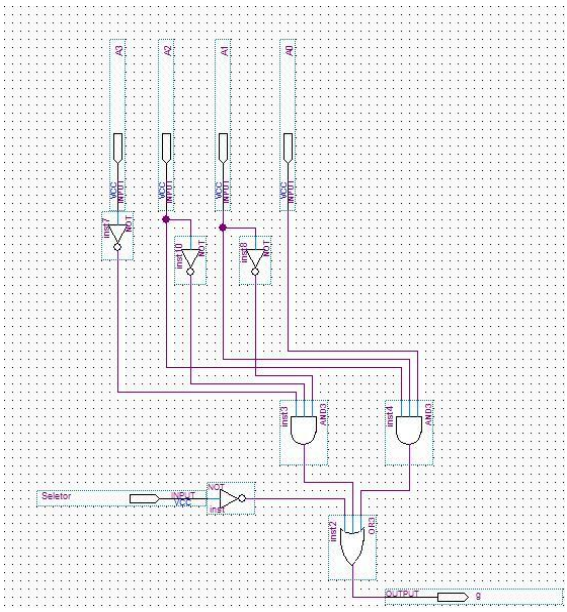

Figura 35 Modulo do decodificador parte “g”

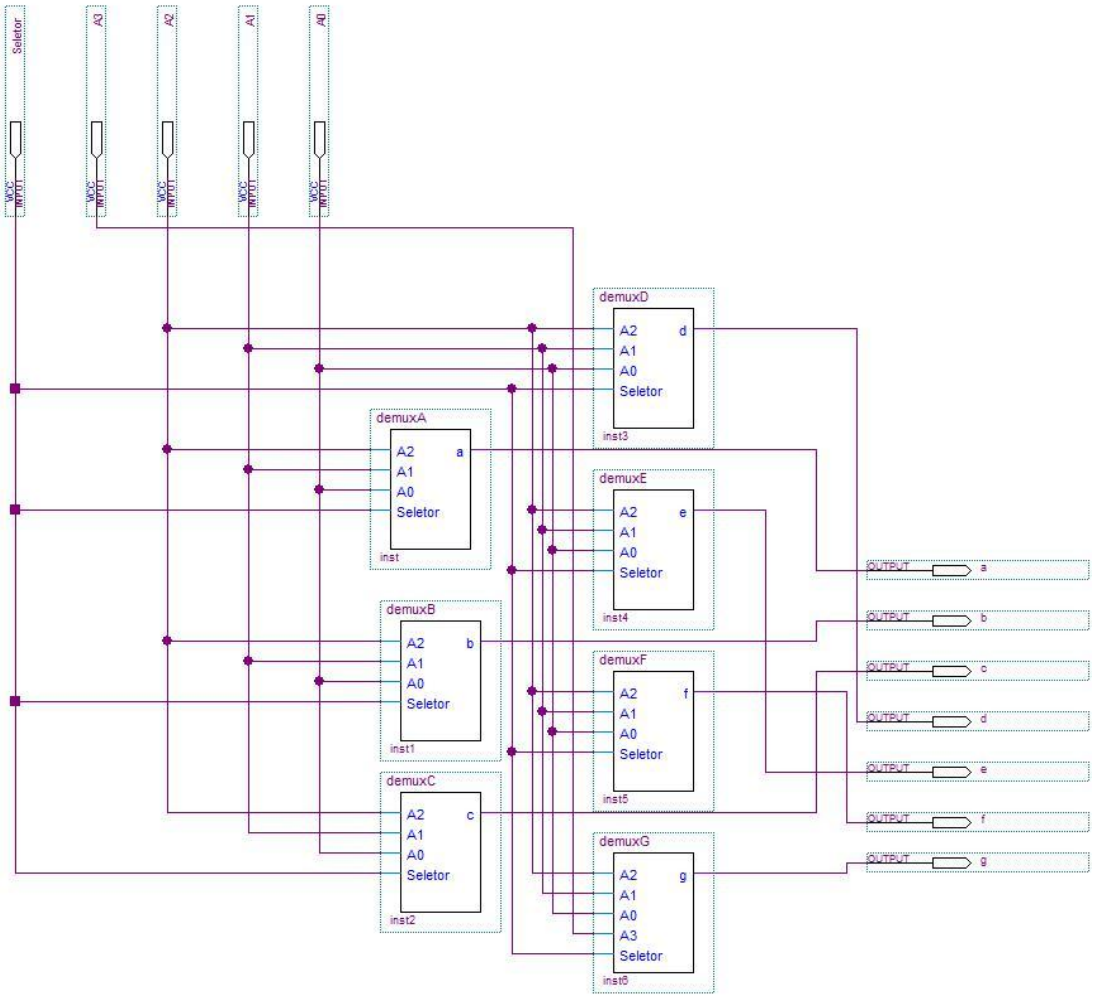

Figura 36 Implementação do decodificador

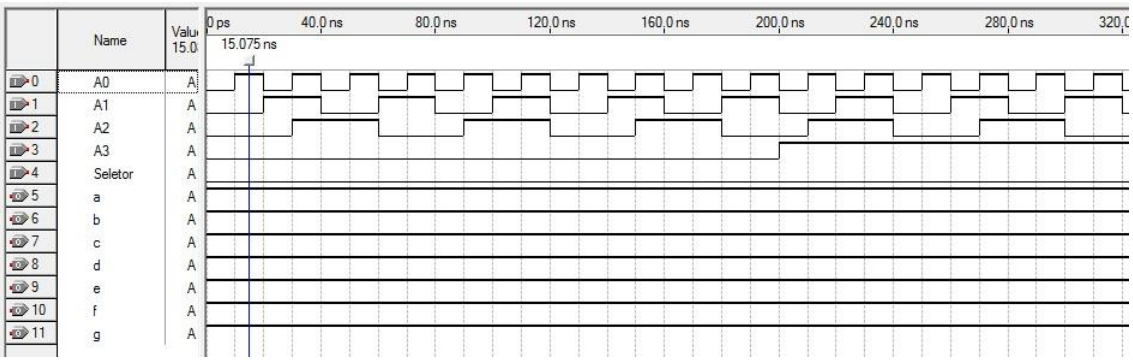

Figura 37 Waveform da simulação do decodificador quando a chave seletora está baixa

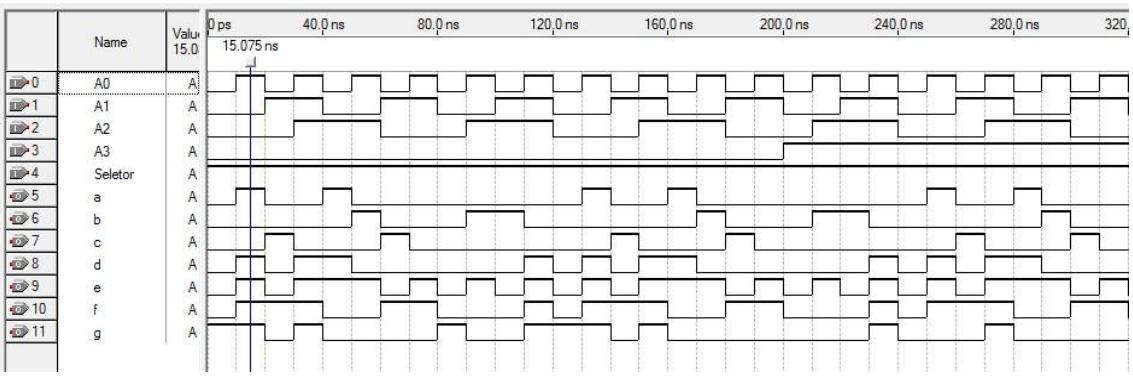

Figura 38 Waveform da simulação do decodificador quando a chave seletora está alta

2.1.7 Seletor Especial

O seletor especial tem por objetivo fornecer uma saída informando se as operações que estão sendo realizadas gerarão ou não um algum tipo de overflow ou se as operações irão requerer alguma espécie de status. O modulo foi desenvolvido para fins de apresentação do projeto com o intuito de fornecer apenas a saída do que se pede e nada a mais.

Especificações do modulo

Entradas:

1- Tipo: vetor Nome: S Descrição: Vetor com o endereçamento da operação da ULA
1.1- Tipo: bit Nome: A[0] Descrição: bit 0 do vetor S.
1.2- Tipo: bit Nome: A[1] Descrição: bit 1 do vetor S.
1.3- Tipo: bit Nome: A[2] Descrição: bit 2 do vetor S.

Saídas:

Tipo: bit Nome: isOperationOverflow: Entrega 1 para operações que possuem overflow e 0 caso contrário;
Tipo: bit Nome: isOperationVector: Entrega 1 caso se esteja trabalhando com uma operação que usa vetor e 0 em caso contrario;

2.1.7.1 Tabela Verdade e Mapas de Karnaugh

A	B	C	IOO	IOV
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

Tabela 12 Tabela - Verdade do módulo: "Seletor Especial"

Mapa de Karnaugh do Módulo: “Seletor Especial” para IOO

AB / C	00	01	11	10
0	1	1	0	0
1	1	0	0	0

Mapa de Karnaugh do Módulo: “Seletor Especial” para IOV

AB / C	00	01	11	10
0	1	1	1	0
1	1	0	1	0

Equações encontradas a partir dos mapas

IOO = ¬A(¬C +¬ B)

IOV = ¬A(¬C +¬ B) + AB

2.1.7.2 Circuito Projetado e Simulação

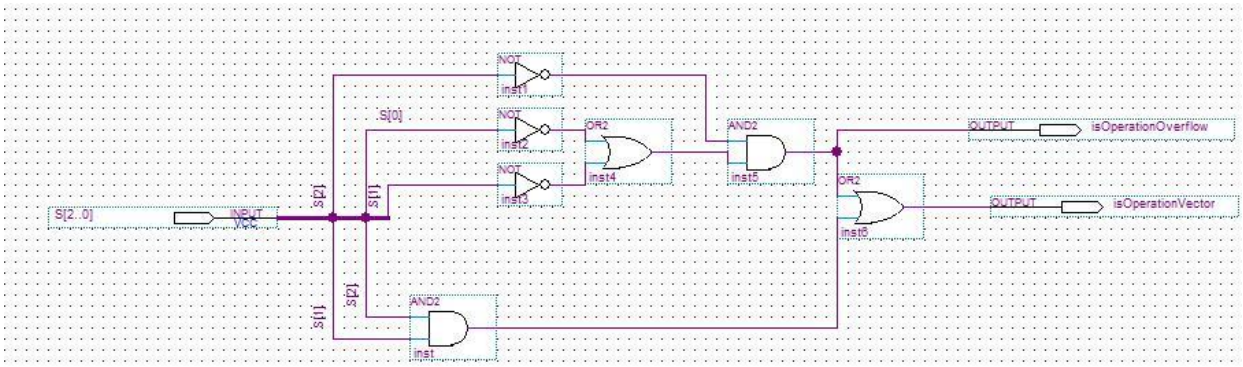

Figura 39 Implementação do circuito do Seletor Especial

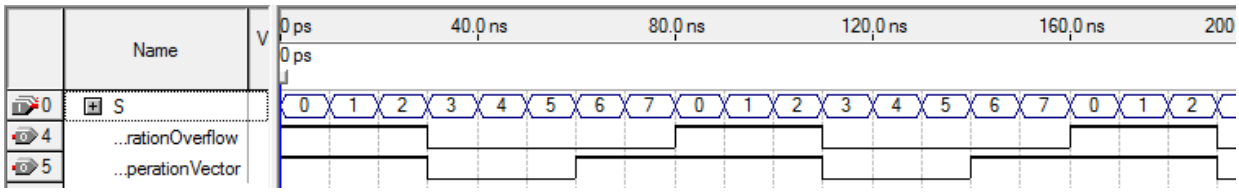

Figura 40 Waveform do módulo Seletor Especial

2.1.8 Seletor de Operações de Estouro

O seletor de operações de Estouro tem por objetivo fornecer uma saída quanto a operação que se esta realizando, dentre as operações que resultam em estouro(Operações do Full-Adder e o Complementador a 2) viu-se a necessidade de fornecer uma saída correta no momento certo uma vez que para alguns valores que são processados pelo circuitos há um estouro por parte de um modulo e por outro não. Assim com o objetivo de resolver tal problema se criou esta caixa extra.

Entradas:

1- Tipo: vetor Nome: S Descrição: Vetor com o endereçamento da operação da ULA
1.1- Tipo: bit Nome: A[0] Descrição: bit 0 do vetor S.
1.2- Tipo: bit Nome: A[1] Descrição: bit 1 do vetor S.
1.3- Tipo: bit Nome: A[2] Descrição: bit 2 do vetor S.

Saídas:

Tipo: bit Nome: S: Entrega 1 caso a operação realizada esteja sendo realizada na ULA seja no complementador a 2 e 0 caso esta operação que pode gerar overflow esteja sendo realizada no Full-Adder.

2.1.8.1 Tabela Verdade e Mapas de Karnaugh

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

Tabela 13 Tabela - Verdade do módulo : "Seletor Operação - Estouro"

Mapa de Karnaugh do módulo: “Seletor Operação - Estouro”

C \ AB	00	01	11	10
0	0	1	0	0
1	0	0	0	0

Equações encontradas a partir dos mapas

S = ¬AB¬C

2.1.8.2 Circuito Projetado e Simulação

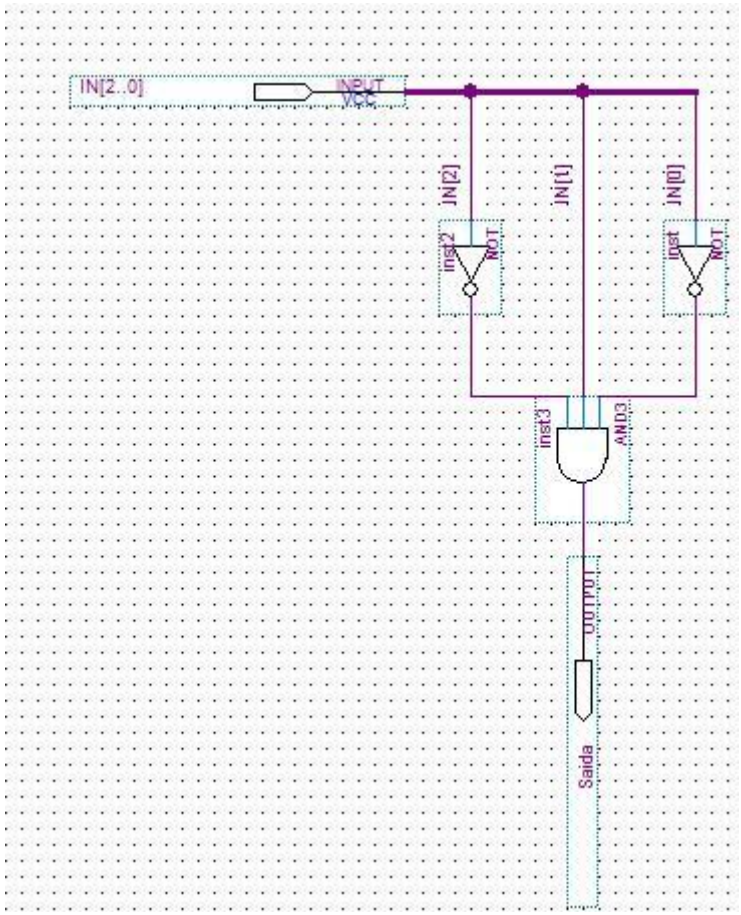

Figura 41 Implementação do módulo Seletor de Operações de Estouro

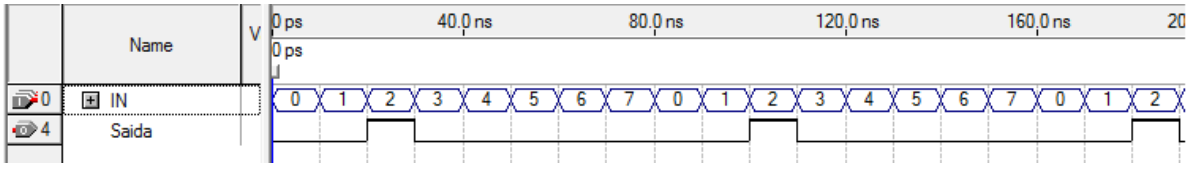

Figura 42 Waveform do módulo Seletor de Operações de Estouro

2.1.9 Seletor de Operações Vetoriais

Como as operações que envolvem vetor não possuem um endereço sequencial viu-se a necessidade de criar uma caixa para fornecer uma saída de forma sequencial para as operações realizadas de forma a simplificar o entendimento de qual operação esta sendo multiplexada no vetor que repassa aquele que corresponde à informação processada corretamente. Assim foi mapeado da seguinte forma:

Entradas:
1- Tipo: vetor Nome: S Descrição: Vetor com o endereçamento da operação da ULA
1.1- Tipo: bit Nome: A [0] Descrição: bit 0 do vetor S.
1.2- Tipo: bit Nome: A [1] Descrição: bit 1 do vetor S.
1.3- Tipo: bit Nome: A [2] Descrição: bit 2 do vetor S.

Saídas:
- 0 0 para quando se seleciona a operação de soma e subtração que vem do modulo que realiza a operação XOR binaria
- 0 1 para quando se seleciona a operação de soma e subtração que vem do modulo que realiza a operação AND binaria;
- 1 0 para quando se seleciona a operação de soma e subtração que vem do modulo que realiza a operação de COMPLEMENTO A 2;
- 1 1 para quando se seleciona a operação de soma e subtração que vem do modulo que realiza a operação de SOMA ou SUBTRAÇÂO;

2.1.9.1 Tabela Verdade e Mapas de Karnaugh

A	B	C	S_1	S_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	*	*
1	0	0	*	*
1	0	1	*	*
1	1	0	1	0
1	1	1	1	1

Tabela 14 Tabela -Verdade do módulo : "Seletor Operação - Vetor"

Mapa de Karnaugh para o módulo: “Seletor Operação - Vetor” para S_1

C \ AB	00	01	11	10
0	0	0	1	*
1	0	*	1	*

Mapa de Karnaugh para o módulo : “Seletor Operação – Vetor para S_2”

C \ AB	00	01	11	10
0	0	1	0	*
1	0	*	1	*

2.1.9.2 Equações encontradas a partir dos mapas

$S_1 = AB$

$S_2 = B(\neg A \neg C + AC)$

2.1.9.3 Circuito Projetado e Simulação

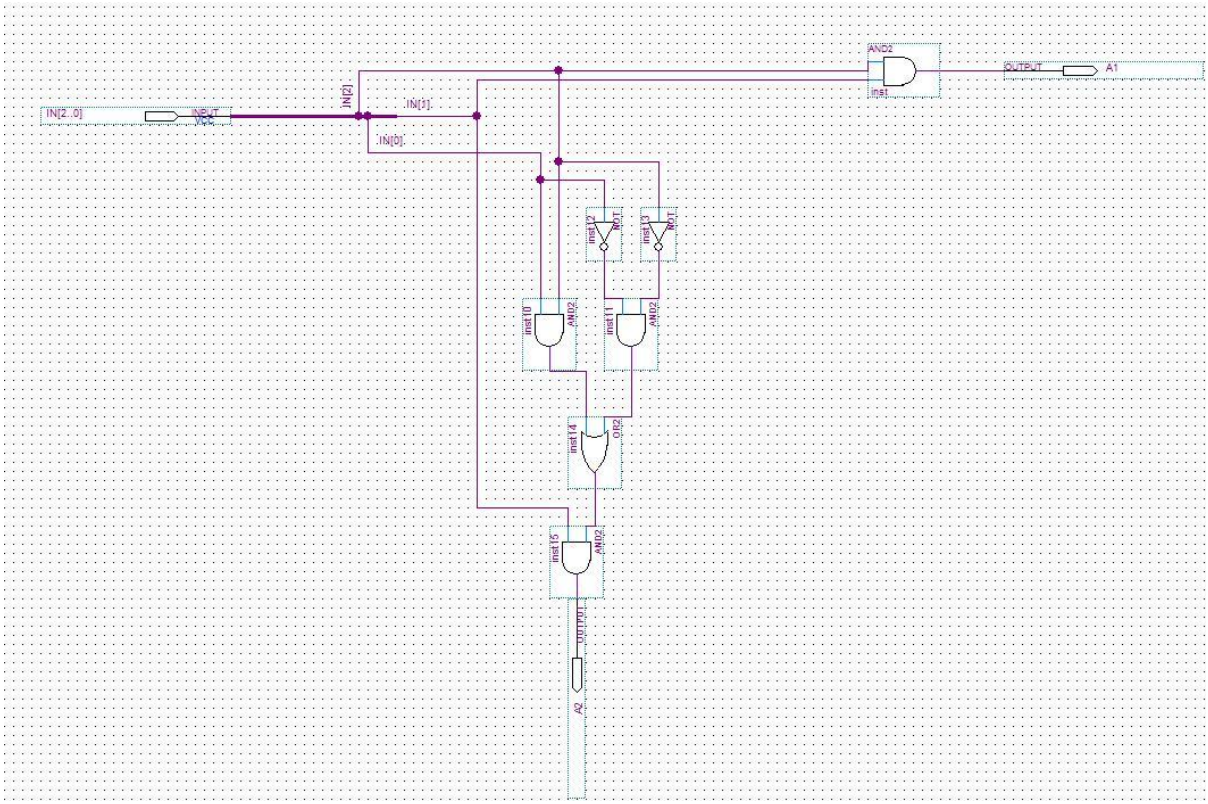

Figura 43 Implementação do módulo Seletor de Operações Vetor

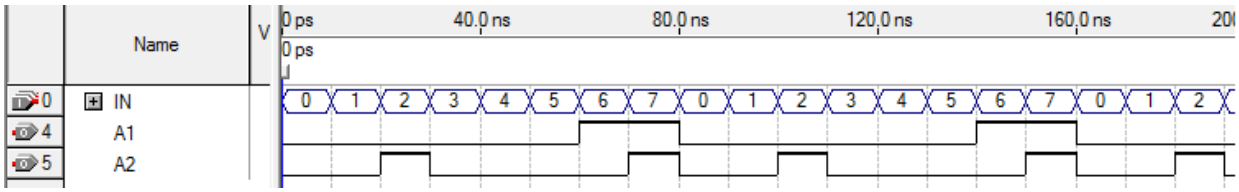

Figura 44 Waveform do módulo Seletor Operações Vetor

3 Visão Geral do Somador BCD

O Somador BCD é uma dos métodos utilizados para somar números binários na forma BCD (Binary Coded Decimal), em que as entradas variam de 00 a 99. Como estas são muito grandes para serem resolvidas de uma única vez, é preciso “quebrar” esse problema em outros menores (modularização). Em tal módulo, as entradas variam de 0 a 9. Portanto, para explicar e caracterizar o Somador BCD com entradas variando de 00 a 99 é preciso explicar primeiro o funcionamento dos seus módulos menores.

3.1 Detalhamento dos Módulos do Somador BCD

3.1.1 Módulo de soma de dois dígitos:

Descrição do somador para a soma de dois números de 1 dígito decimal cada (ou seja, 0 à 9):

O módulo de soma de dois dígitos, o qual será usado como componente do Somador BCD completo (que é o alvo dessa descrição), tem como entradas e saídas:

Entradas:

- A: Um vetor de 4 bits que representa o primeiro dígito a ser somado
- B: Um vetor de 4 bits que representa o segundo dígito a ser somado
- Carry In: Um bit que indica se a soma anterior causou overflow

Saídas:

- C: Um vetor de 4 bits que representa o resultado da soma
- Carry Out: Um bit que indica se a soma causou um overflow

O módulo tem como componentes:

- 1 Full Adder, módulo explicado e especificado na parte do relatório referente à ULA (Unidade Lógica e Aritmética)
- Módulo Is Over Flow (explicado com mais detalhes no tópico adiante)
- 1 porta lógica XOR

- 2 somadores de 1 bit que complementam o número com 6, caso haja overflow.

Lógica do módulo:

Primeiramente, é feita a soma, por meio do Full Adder (bit a bit) dos vetores A e B. No entanto, temos o problema de quando a soma desses dígitos é maior do que 9. Nesse caso, é preciso gerar um carry out (para indicar o estouro da base adotada) e subtrair 10 da soma atual. Mas, por conta da representação adotada, subtrair 10 é o mesmo que somar 6 (0110) a essa forma, uma vez que estamos em um modulo.

Agora é preciso identificar o formato da soma que indica se houve carry out. Sabe-se que esses casos são (considerando o vetor de 4 bits de saída), esta tarefa de identificação se resume ao módulo IsOverFlow:

- Formato 1: 1X1X
- Formato 2: 11XX
- Formato 3: 1XXXX (houve overflow)

Caso seja identificado o caso de overflow no módulo IsOverFlow é somado 6 ao número atual, uma vez que estamos em módulo de 16 para a representação binária. O processo é feito da seguinte forma ao se detectar o caso de overflow no módulo IsOverFlow, manda-se como sinal um bit de valor 1 para os somadores de 1 bit, que estão ligados a saída do Full Adder de forma a permitir a soma do valor 6 na cadeia resultante.

Deste modo, obtém-se o valor correto do resultado da soma.

Explicado o funcionamento do módulo para a soma de dois dígitos de 0 a 9, é hora de resolver o problema proposto incialmente (O problema de se fazer a soma de números binários na representação BCD (Binary Coded Decimal) em que as entradas variam de 00 a 99) interligando módulos menores que fazem a soma de dois dígitos. Temos então que, para esse circuito, teríamos como entradas e saídas:

Entradas:

- A: Um vetor de 8 bits que representa o primeiro número que será somado;
- B: Um vetor de 8 bits que representa o segundo número que será somado;
- Carry In : Um bit que indica se a soma anterior causou overflow.

Saídas:

- C : Um vetor de 8 bits que representa o resultado da soma;
- Carry Out - Um bit que indica se a soma causou overflow;

Seus componentes seriam:

- 2 módulos de soma de 2 dígitos, módulo o qual já foi descrito aqui.

Lógica do módulo:

Os dois vetores de 8 bits serão manipulados da seguinte maneira:

Serão formados dois vetores de 4 bits a partir dos 4 bits da direita de A e B(um vetor de 4 bits pra A e um pra B) e esses vetores serão dados como entrada do primeiro Full Adder.
Serão formados dois vetores de 4 bits a partir dos 4 bits da esquerda de A e B(um vetor de 4 bits pra A e um pra B) e esses vetores serão dados como entrada do segundo Full Adder.

O módulo que faz a soma do primeiro dígito (das unidades) tem como carry in um fio-terra (ground), o qual força um 0 nessa entrada. Esse módulo gera o vetor E, cujos bits serão fornecidos como os quatro bits da direita do vetor de saída C, e seu carry out aproveitado como carry in do próximo módulo de soma de dois dígitos.

O módulo que faz a soma do segundo dígito (das dezenas) tem como carry in o carry out do módulo anterior e gera o vetor F, cujos bits serão fornecidos como os 4 bits da esquerda do vetor de saída C, e seu carry out é fornecido como o Carry Out do módulo como um todo.

Essa foi a descrição do circuito responsável por resolver o problema citado no começo da mesma, o Somador BCD.

3.1.1.1 Circuito Projetado e Simulação

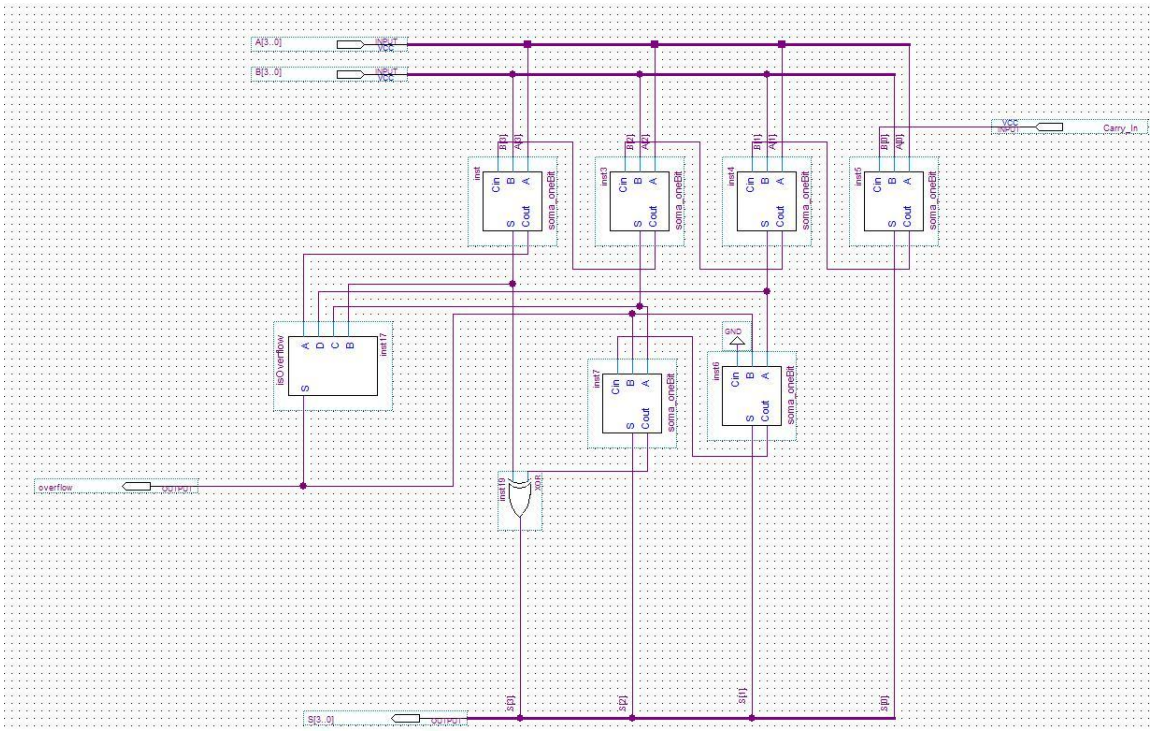

Figura 45 Circuito do Somador BCD para 4 bits

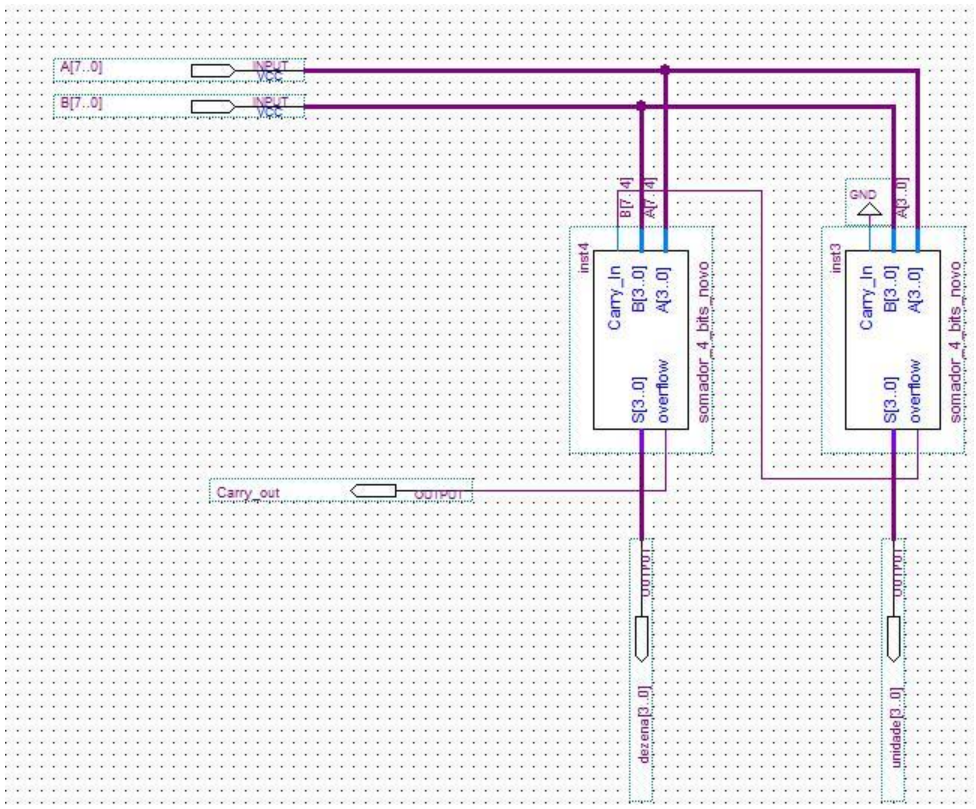

Figura 46 Circuito do Somador BCD para 8 bits

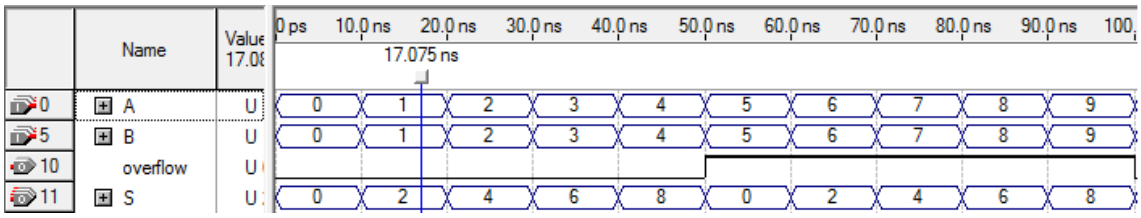

Figura 47 Waveform do somador BCD de 4 bits

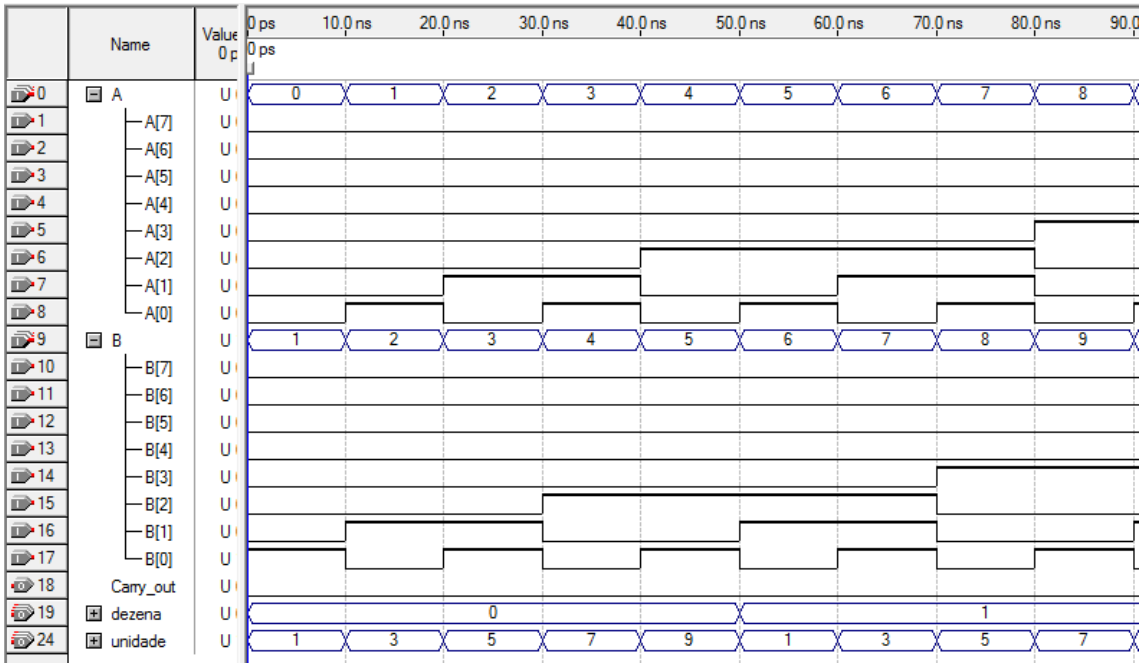

Figura 48 Waveform do somador BCD de 8 bits

3.1.2 Is Over Flow

O modulo intitulado isOverflow tem por objetivo abstrair a tarefa de determinar se uma dada soma entre dois números resulta em estouro da base decimal. Para tal tarefa o modulo necessita dos 3 bits mais significantes do numero BCD de 4 bits e do Carry-Out da soma binaria ao final da soma da cadeia de quatro bits. Isso pelo fato de o LSB (Last Significant Bit) não influenciar na tarefa de determinar se a entrada e de algo que estourou a base 10. Assim o processo é feito da seguinte maneira. Se uma cadeia binaria que é dada como entrada contiver o primeiro elemento igual a 1 então ele vem de um carry-out que estourou a base 10 obrigatoriamente pois já estourou uma base 16, e então são verificados se os bits remanescentes são partes de um dos números que podem ser representados dentro da faixa(com 4 bits) e que tenham estourado a base 10.

Assim o modulo retorna como resposta 1, em casos onde a entrada e parte de um numero que atende as condições de estouro da base 10 ou 0 caso contrario.

Especificações do Modulo

Entradas

1- Tipo: bit Nome: A Descrição: Bit resultante da operação de soma binaria de todos os 4 bits que formam o numero BCD, para este caso (último carry out).
2- Tipo: bit Nome: B Descrição: O bit mais significante da operação de soma sobre as entradas passadas para o somador de dígitos(vide especificações do BCD).
3- Tipo: bit Nome: c Descrição: Segundo bit mais significativo para o caso em trabalho.
4- Tipo: bit Nome: D Descrição: Penúltimo bit significativo da cadeia binaria dada como entrada para o somador de dígitos.

Saída

1- Tipo: bit Nome: S Descrição: Valor 1 caso a entrada venha de um numero que estoura a base 10, 0 caso contrario.

3.1.2.1 Tabela Verdade e Mapas de Karnaugh

Tabela 15 Tabela verdade do modulo "Is Over Flow BCD"

A	B	C	D	S
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Mapa de Karnaugh do módulo: “Is Over Flow”

AB / CD	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	1	1	1	1
10	1	1	1	1

Fórmula do IsOverFlow

A+BC+BD

3.1.2.2 Circuito Projetado e Simulação

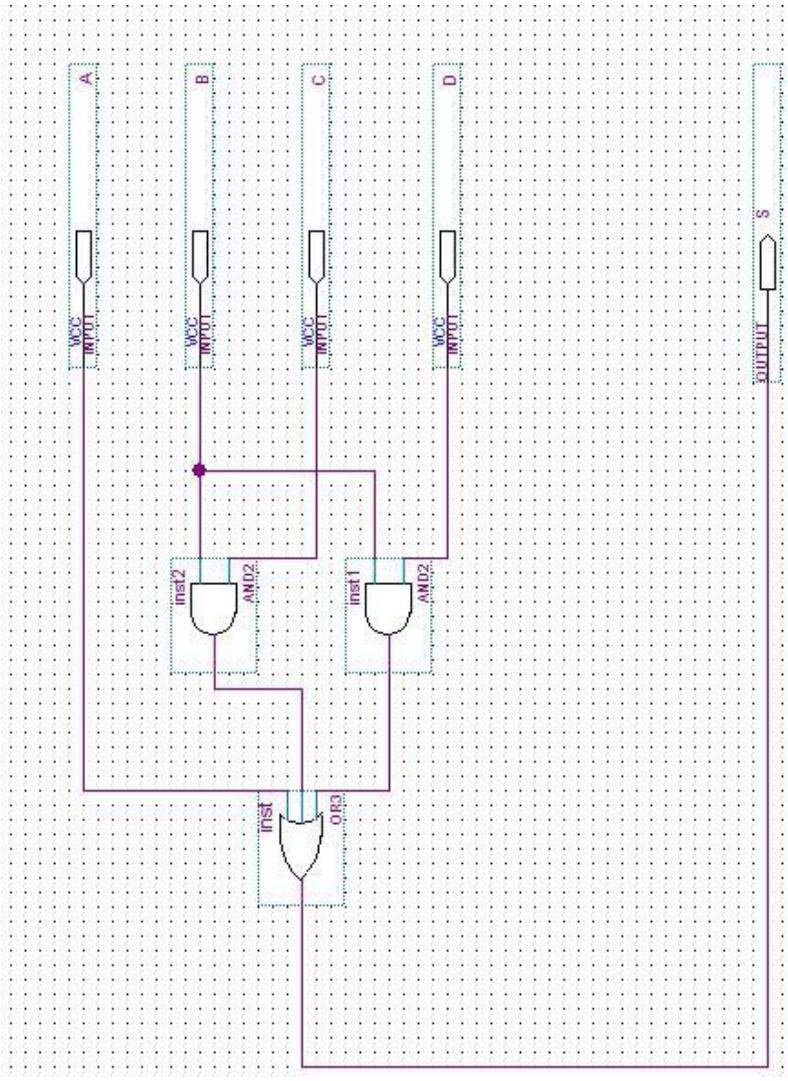

Figura 50 Implementação do módulo Is Over Flow

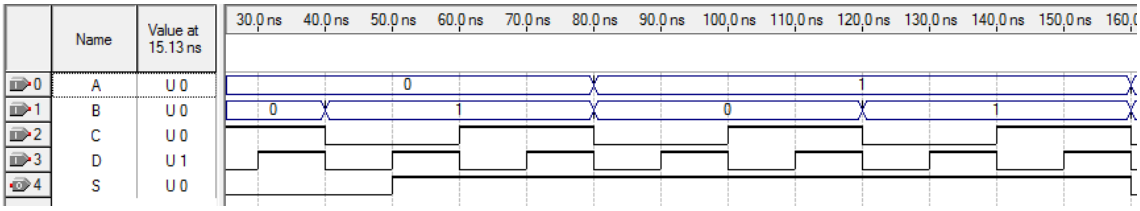

Figura 51 Waveform do módulo IsOverFlow

3.1.3 Decoder tela do BCD

Devido a dificuldade enfrentada com a implementação do demultiplexador da ULA, baseando-se em fórmulas resolve-se aplicar uma nova estratégia. Para tal ação foi feito um circuito que recebe o numero como entrada e o endereça a uma saída esta que por sua vez e ligada a um OU para cada led da tela. Assim temos que para fins de explicação temos que foi dada a para o demultiplexador da tela o numero oito (8) em binário. Assim o circuito trabalha da seguinte forma. Primeiramente o valor binário de e associado a seu Maxitermo e partir de então a sua saída e ligada a um ou de cada tela com o intuito de acendê-la apenas no momento certo e com o auxilio de portas logicas OU foi implementado o circuito.

3.1.3.1 Tabela Verdade

A	IN[3]	IN[2]	IN[1]	IN[0]	0	1	2	3	4	5	6	7	8	9
0	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	1	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0	1
1	1	0	1	0	x	x	x	x	x	x	x	x	x	x
1	1	0	1	1	x	x	x	x	x	x	x	x	x	x
1	1	1	0	0	x	x	x	x	x	x	x	x	x	x
1	1	1	0	1	x	x	x	x	x	x	x	x	x	x
1	1	1	1	0	x	x	x	x	x	x	x	x	x	x
1	1	1	1	1	x	x	x	x	x	x	x	x	x	x

Tabela 16 Tabela verdade do Demux para a tela do BCD

3.1.3.2 Circuito Projetado e Simulação

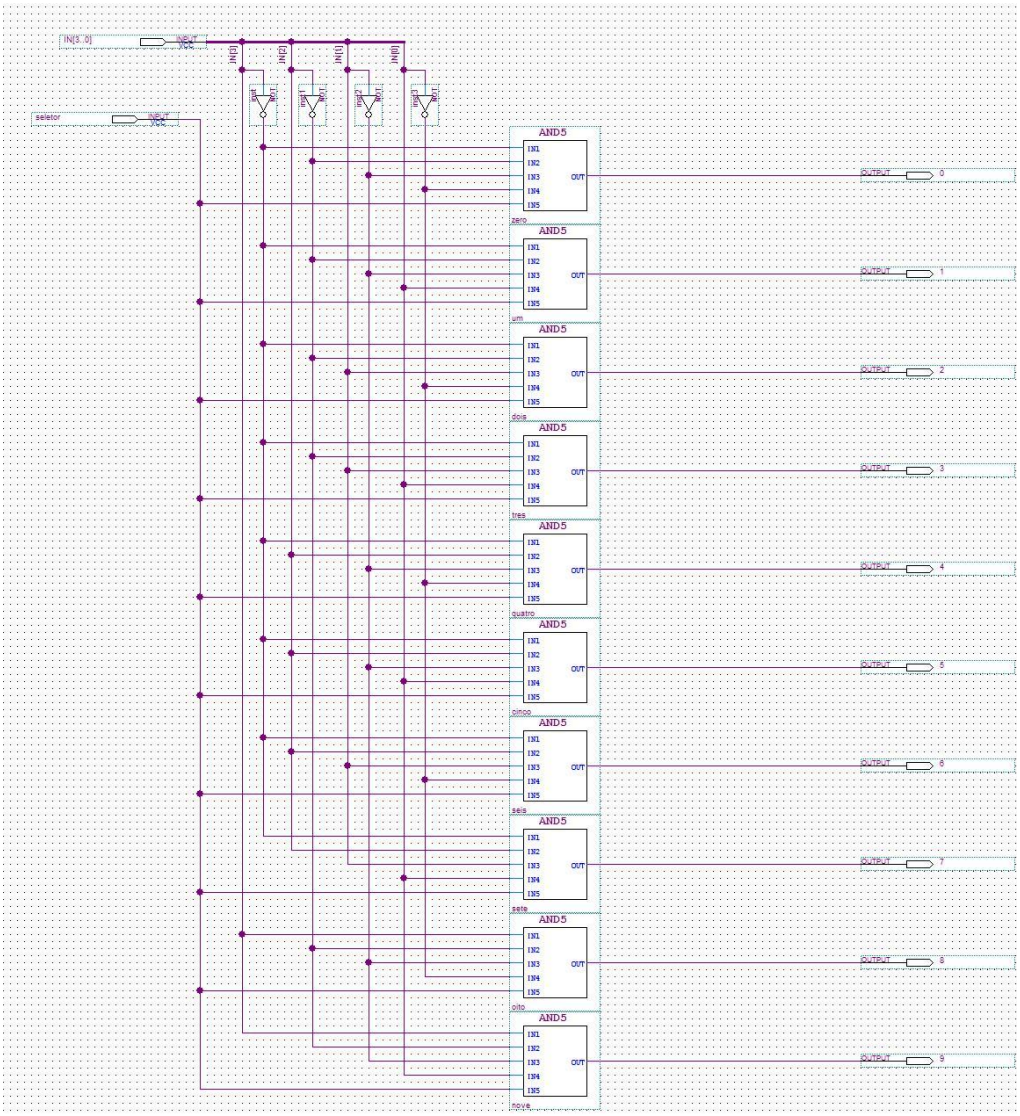

Figura 52 Implementação do Demux da tela do BCD

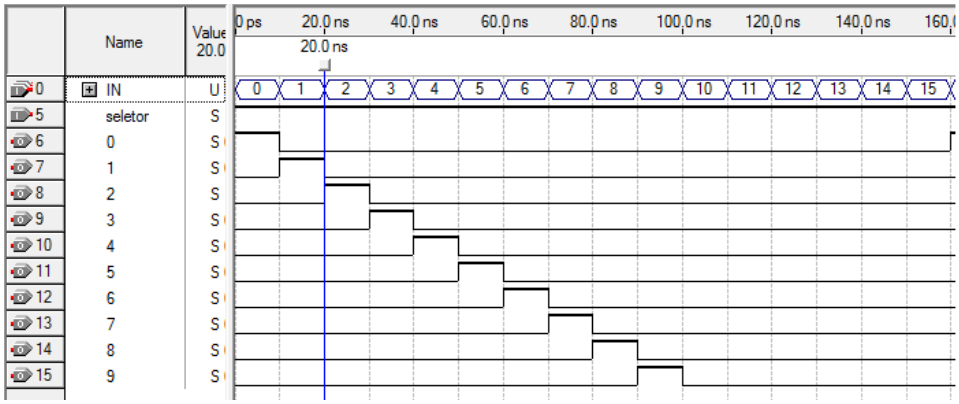

Figura 53 Waveform do demux da tela do BCD

4 Conclusão

Na implementação deste projeto, utilizou-se a ferramenta de design de dispositivos lógicos programáveis Quartus II. Tal software teve extrema importância na modelagem dos circuitos utilizados para construção dos sistemas pedidos.

A construção de uma ULA, Unidade Lógica e Aritmética, é um objetivo demasiado grande para ser solucionado sem que haja uma divisão do sistema em partes menores. Tal divisão é chamada de modularização e se trata da resolução do problema em módulos menores, seguida da junção dessas soluções parciais, de modo a solucionar o problema como um todo.

A construção do Somador BCD (Binary Coded Decimal), além de ter se mostrado extremamente válida, instigante e recompensadora do ponto de vista do aprendizado do assunto da disciplina de Sistemas Digitais, se mostrou também uma ótima oportunidade de conhecer mais sobre mais essa forma de representação e codificação de números.

Esse projeto é de grande importância no entendimento de sistemas mais elaborados, tais como a CPU, Central Processing Unit, visto que introduz unidades mais simples, como comparadores, somadores/ subtratores e multiplexadores. Tais módulos servem de base para a confecção de circuitos mais complexos.

Todo conhecimento adquirido no projeto será certamente utilizado, não só na disciplina de Sistemas Digitais, como em todo o decorrer do curso de Ciência da Computação.

5 Referências Bibliográficas

GAJSKI, Daniel D.. *Principles of digital design.* Universidade de Michigan: Prentice Hall, 1997.

TOCCI, Ronald J.; WIDMER, Neal S.. *Sistemas digitais: princípios e aplicações.* Prentice Hall, 2003.

http://www.cin.ufpe.br/~if675/ Acesso em: 09 de outubro de 2011.