

UNIVERSIDADE FEDERAL DE PERNAMBUCO  
CIN-CENTRO DE INFORMÁTICA

Camila Xavier de Medeiros (c xm)

Lorenzo Fontenelle Chaves (lfc4)

Maria Luísa Mendes de Siqueira Passos (mlmsp)

Mário da Mota Limeira Neto (mmln)

Projeto de Sistemas Digitais

(ULA)

RECIFE  
2021

## Sumário

1	Introdução	3
2	Visão Geral da Unidade Lógica e Aritmética (ULA)	4
2.1	Módulos da ULA	13
2.1.1	Somador	13
2.1.2	Subtrator	18
2.1.3	Complementador	20
2.1.4	And e Xor	22
2.1.5	Comparadores Lógicos	25
2.1.6	Demultiplexadores	30
2.1.7	Decodificador	33
3	Conclusão	47
4	Referências Bibliográficas	48

## 1 Introdução

Neste trabalho, relativo à disciplina de sistemas digitais, estão contidas as etapas de elaboração de uma Unidade Lógica e Aritmética (ULA) acoplada a um display de 7 segmentos que reproduz os números. A fim de otimizar o seu desenvolvimento, os estudantes seguiram uma filosofia de modularização. Ou seja, o projeto foi iniciado a partir do estudo de cada unidade básica que compõe uma ULA e, em seguida, ocorreu a união de cada uma dessas partes funcionando corretamente, formando o sistema final desejado.

Dessa forma, ao longo deste trabalho, realizado através da ferramenta Quartus, foram utilizados os seguintes métodos: uso de tabelas verdade, formulação de equações, mapas de Karnaugh, elaboração de circuitos lógicos a partir do uso de portas lógicas e a realização de simulações. Esse processo inicia-se em uma unidade básica e é repetido até o momento em que se chega ao sistema maior (ULA) seguido da implementação do display.

## 2 Visão Geral da Unidade Lógica e Aritmética (ULA)

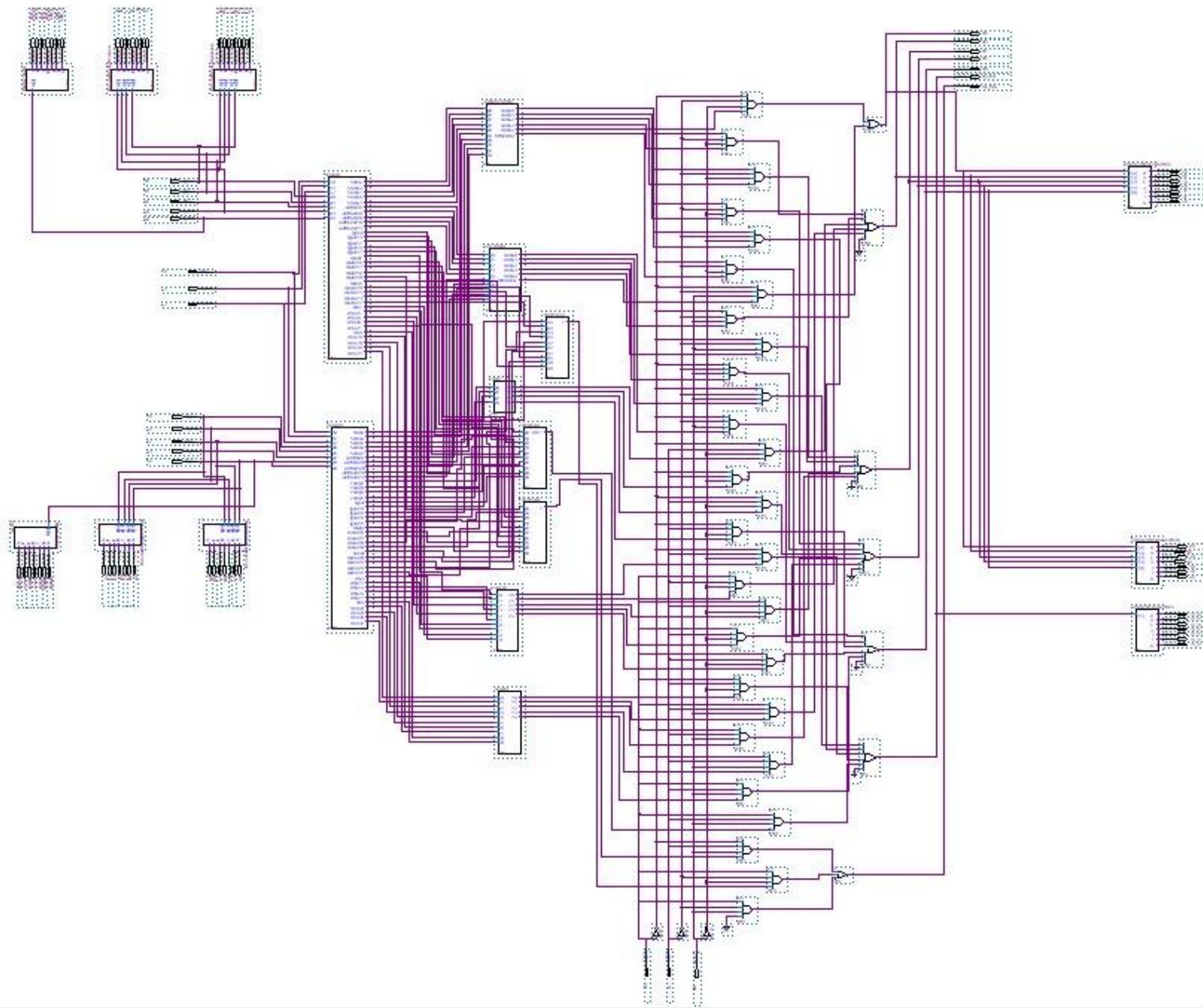
Desde o surgimento dos primeiros computadores, a função de realizar cálculos está entre uma das principais atribuições desta invenção. A importância é tanta que muitas das operações aritméticas realizadas pelos computadores atuais são impossíveis de serem calculadas em tempo hábil por um ser humano. É devido a estas importâncias e necessidades 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 feito 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.

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

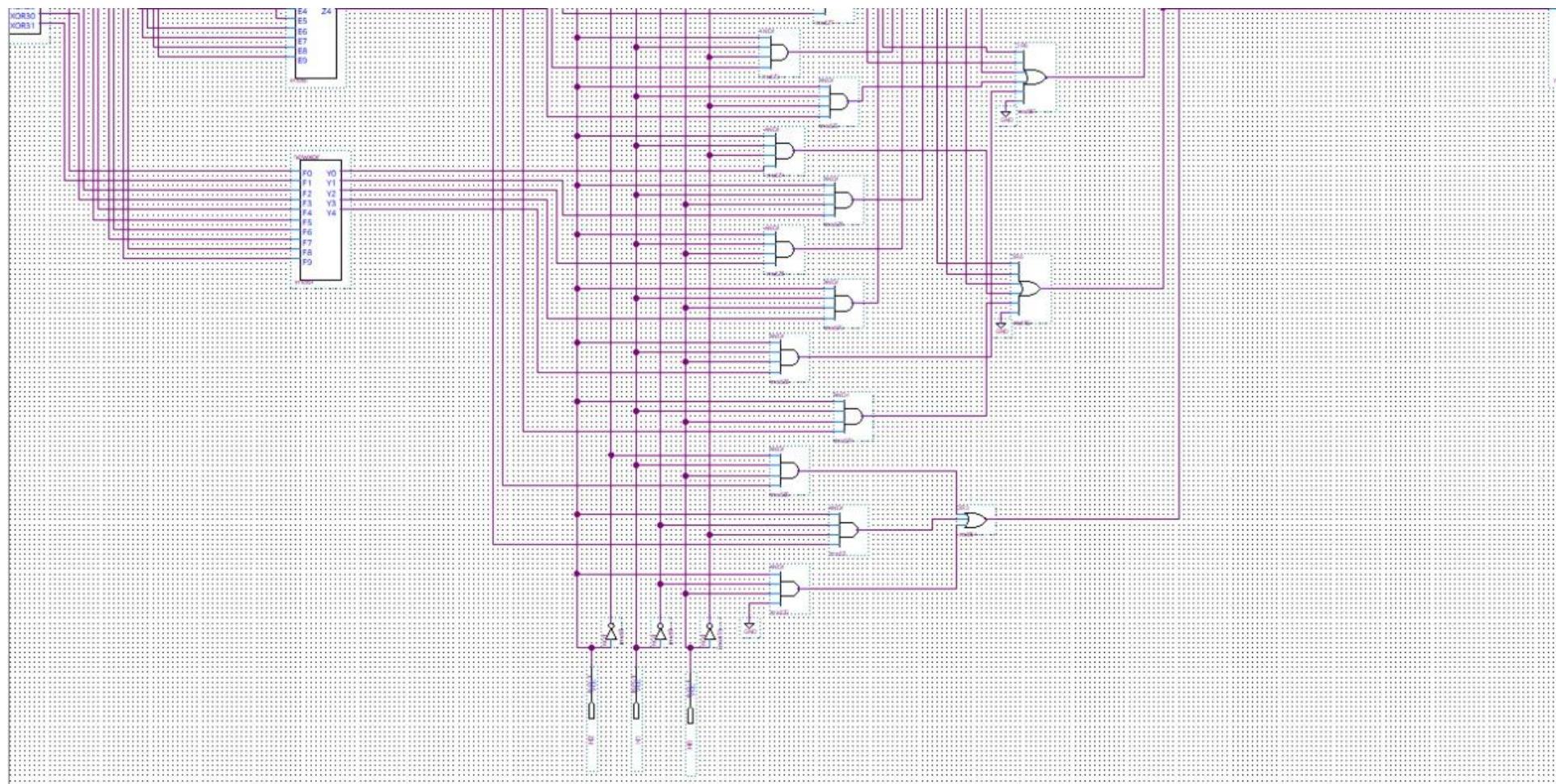
- Somador;
- Subtrator;
- Complementador a 2;
- Função lógica AND;
- Função lógica XOR;
- Operações de Comparação: igualdade, maior que, menor que.

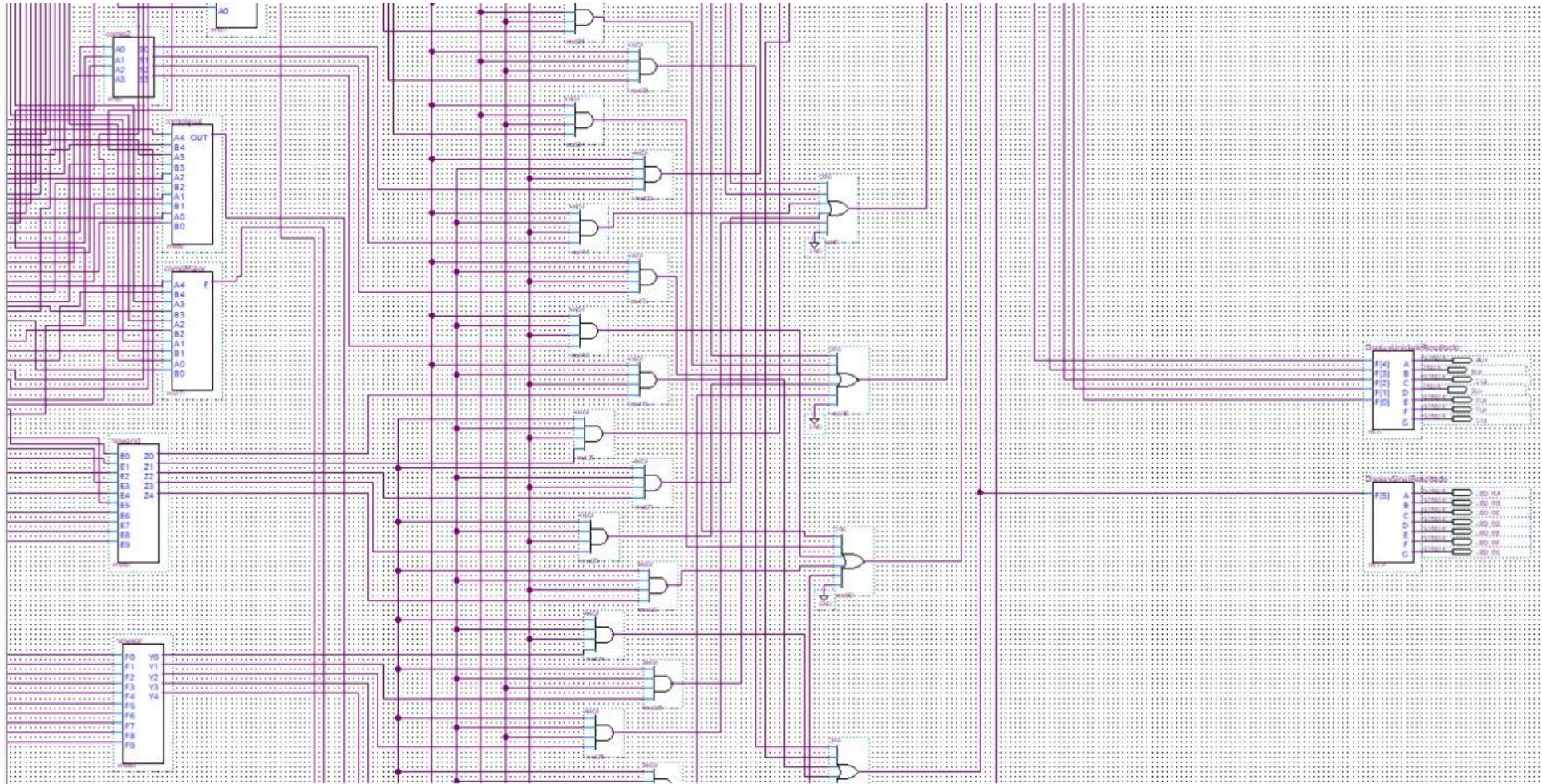
A seguir serão apresentadas essas partes, bem como as suas funções e como elas foram devidamente desenvolvidas nesse projeto.

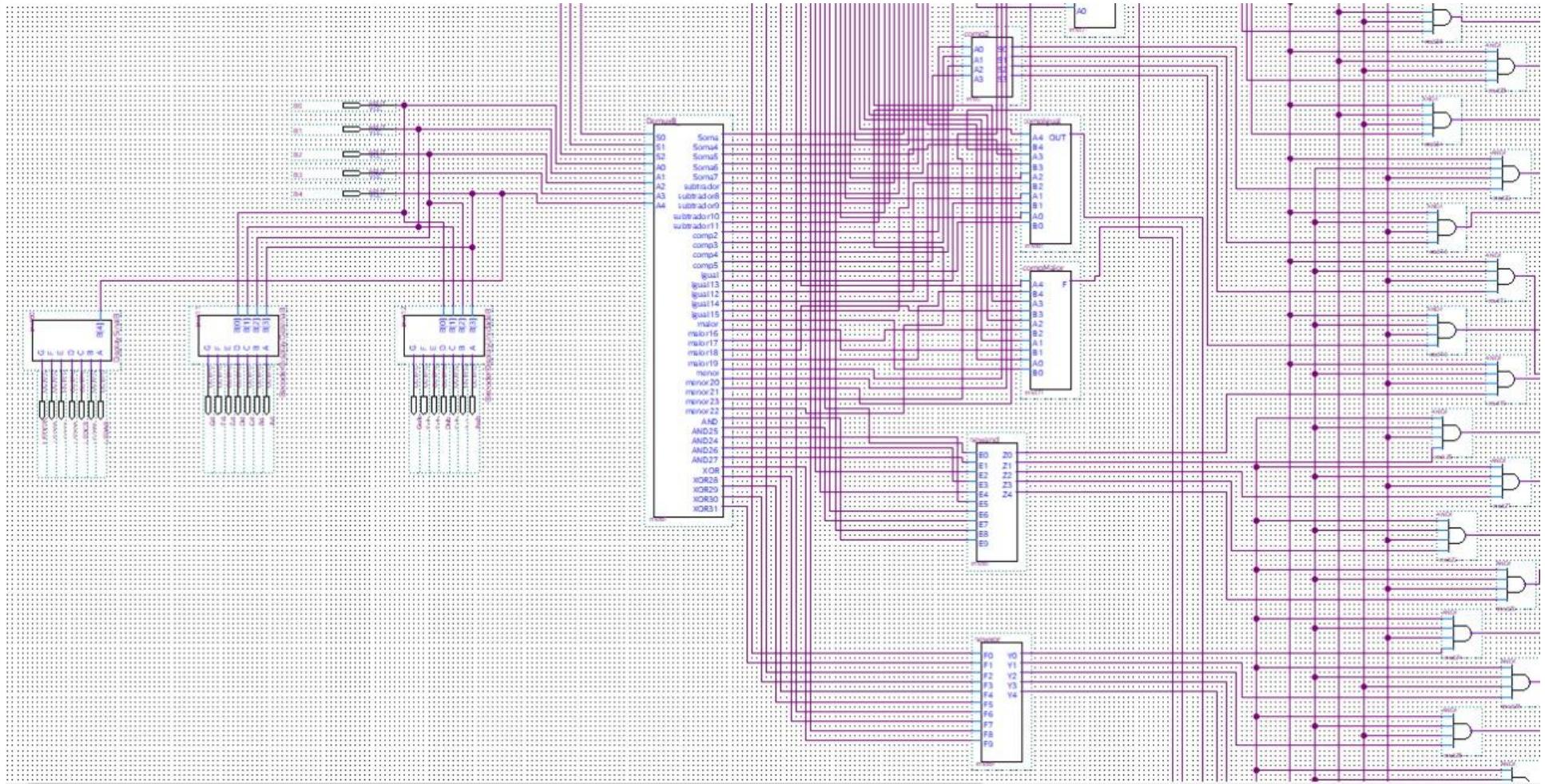


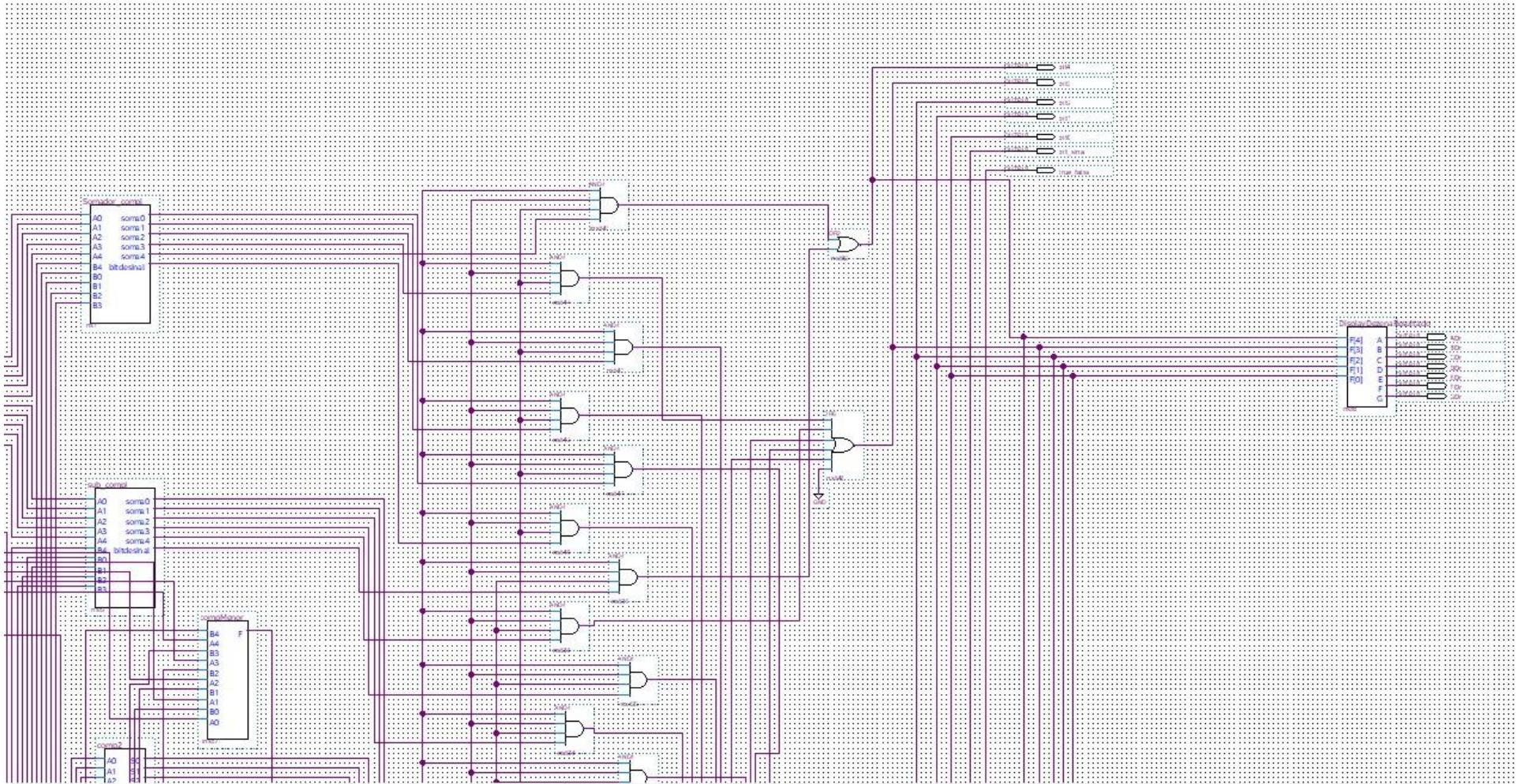
**Figura 1 Implementação da ULA e do display de 7 segmentos (no início e no final).**

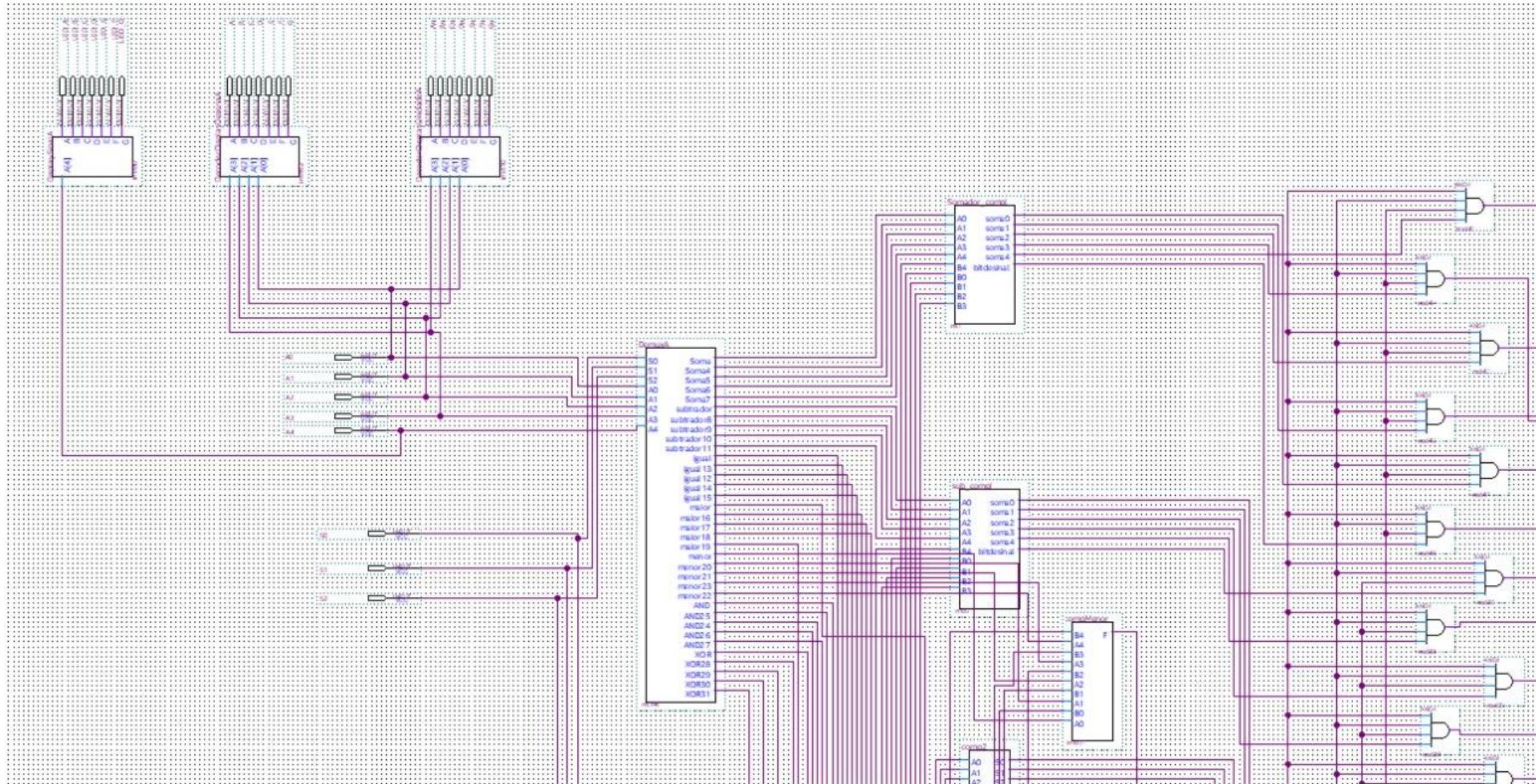
**Detalhamento da implementação da ULA:**





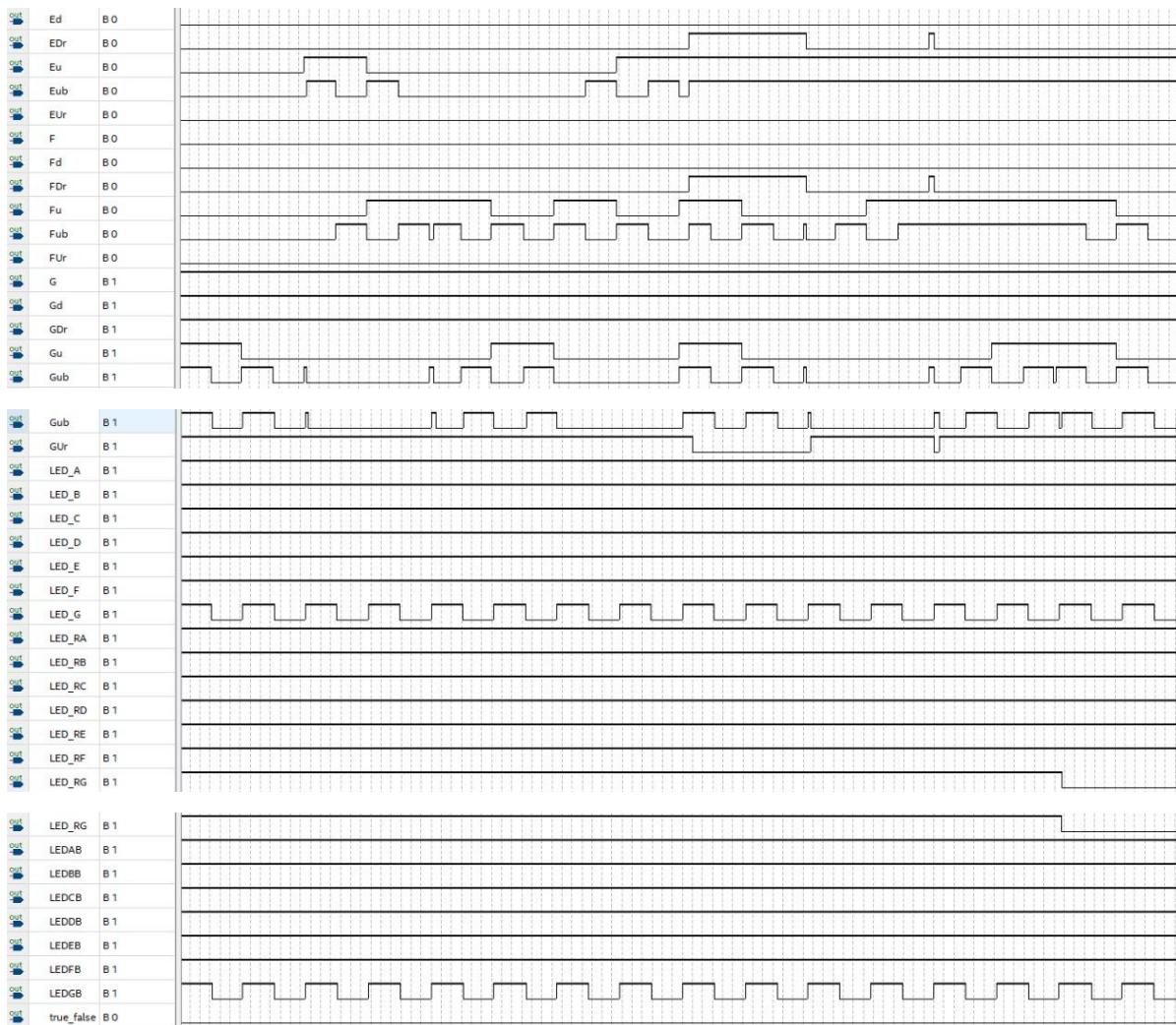






## Waveform ULA:







## 2.1 Módulos da ULA

### 2.1.1 Somador

Esse módulo tem a função de realizar as operações de soma, dada uma cadeia de números binários. Em todos os sistemas digitais o circuito que realiza a adição pode efetuar uma operação apenas com dois números de cada vez, com isso, o resultado é somado com o terceiro e assim sucessivamente. Em nosso projeto, realizamos um somador bit a bit.

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 abstruído 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 é 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.

Além do full-adder, também adicionamos o complemento de 2 (que iremos explicar melhor ao longo do projeto) no somador, caso o usuário queira somar um número negativo. Esse complemento foi colocado em blocos ligados diretamente aos números binários da entrada, que, em seguida, são ligados ao somador em si e, depois, a um último bloco para verificar se a soma precisa ser complementada ou não.



### 2.1.1.1 Tabela Verdade e Mapas de Karnaugh

A tabela verdade é uma ferramenta matemática muito utilizada no campo do raciocínio lógico. Seu objetivo é verificar a validade lógica de uma proposição composta. Já o Mapa de Karnaugh é um método gráfico usado para simplificar uma equação lógica ou converter uma tabela verdade no seu circuito lógico correspondente. Essas duas ferramentas foram bastante utilizadas ao longo do projeto para analisar as entradas e saídas booleanas da ULA.

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	<u>1</u>	<u>1</u>

**Tabela 1 Tabela- verdade do módulo:“Full-Adder”(usado no somador)**

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:  
 $\neg A (B (+) CIn) + A (B (.) CIn)$



### 2.1.1.2 Circuito Projetado e Simulação

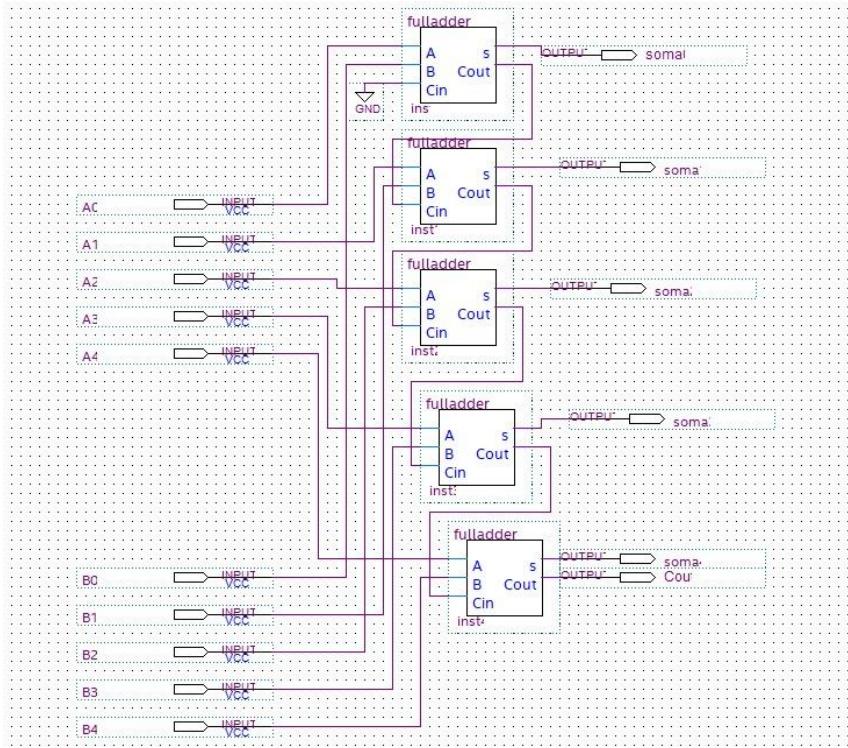


Figura 2 Implementação de um somador bit a bit

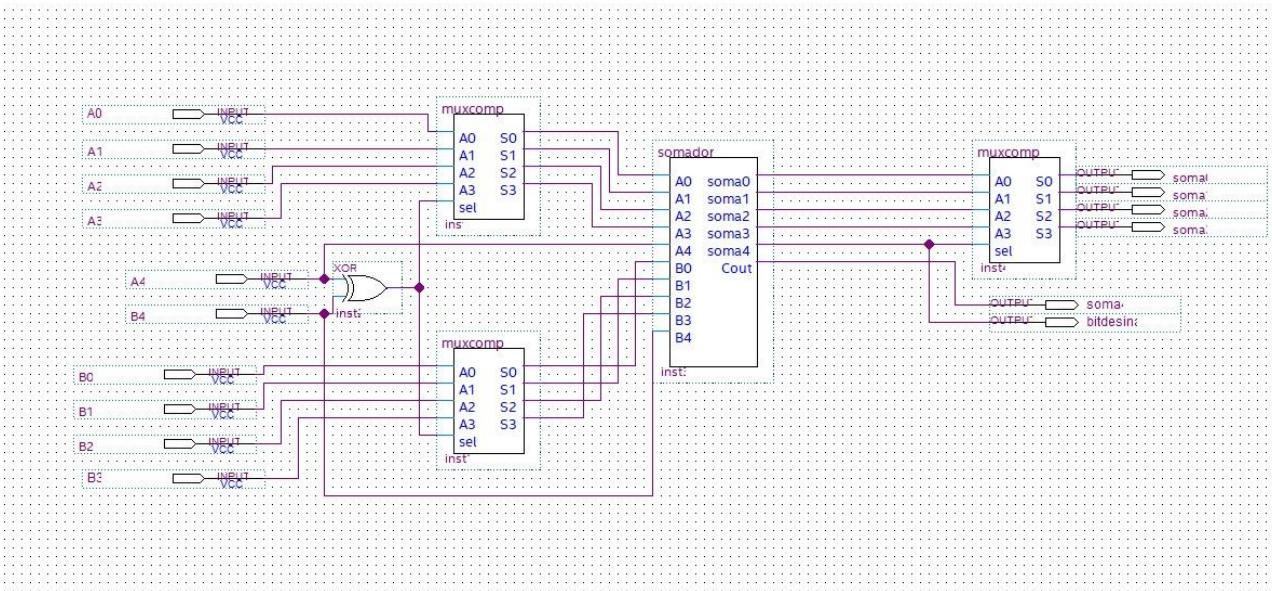


Figura 3 Implementação de um somador de quatro bits (junção do somador bit a bit)

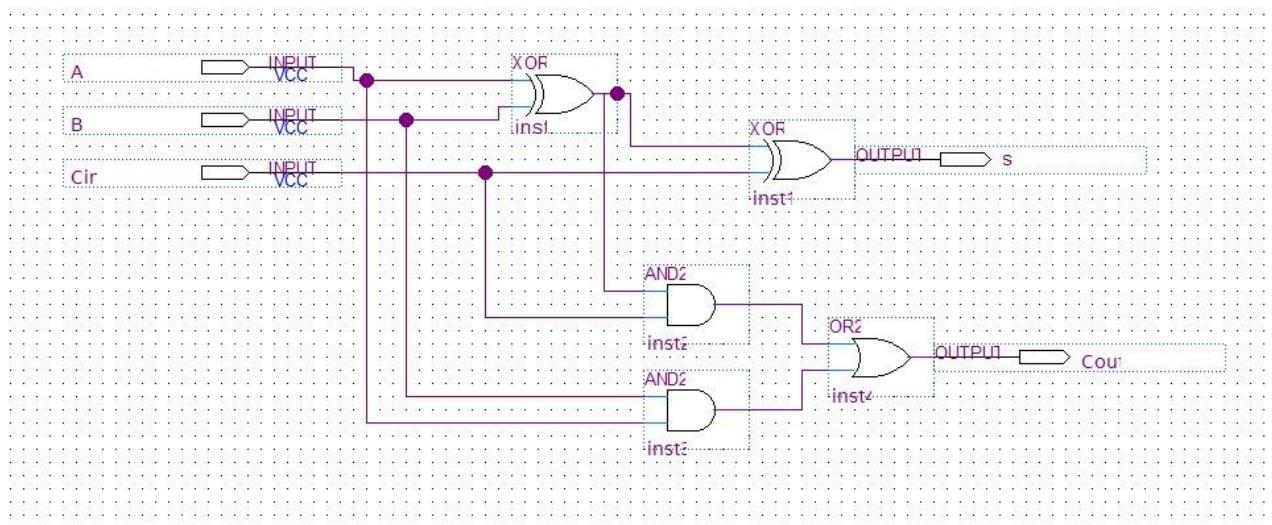


Figura 4 FullAdder do somador

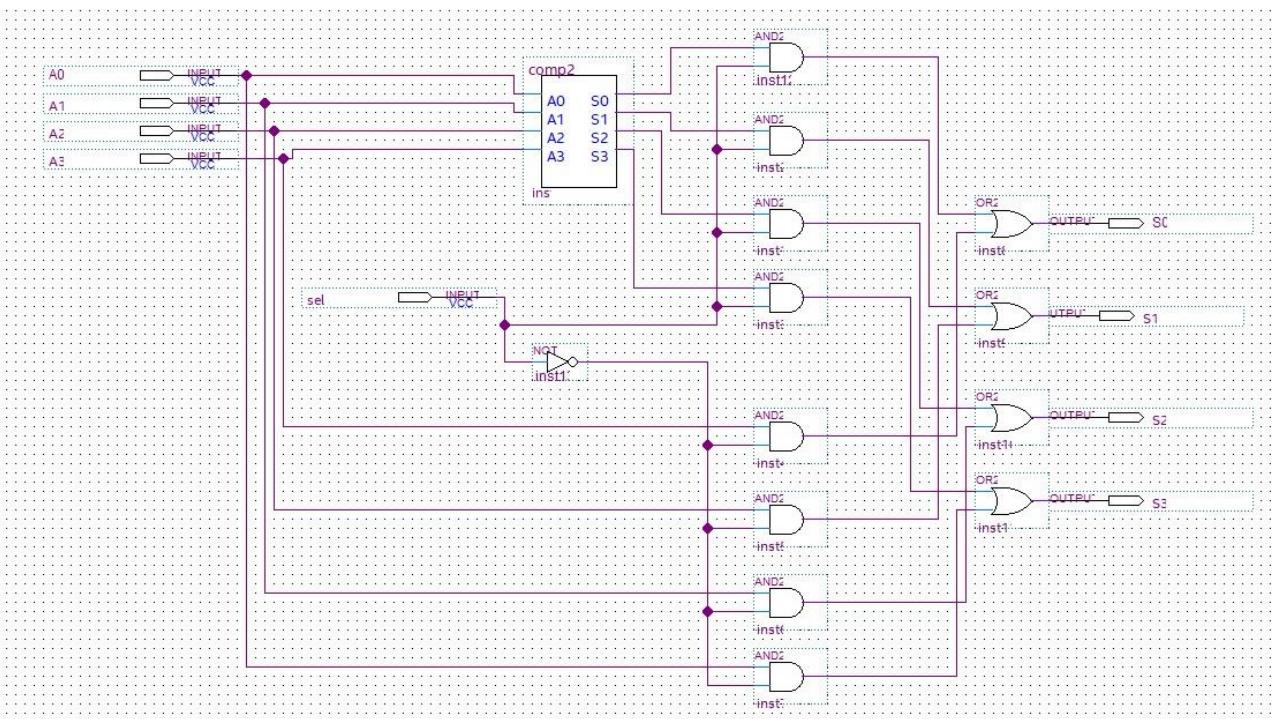


Figura 5 Implementação multiplexador somador e subtrator

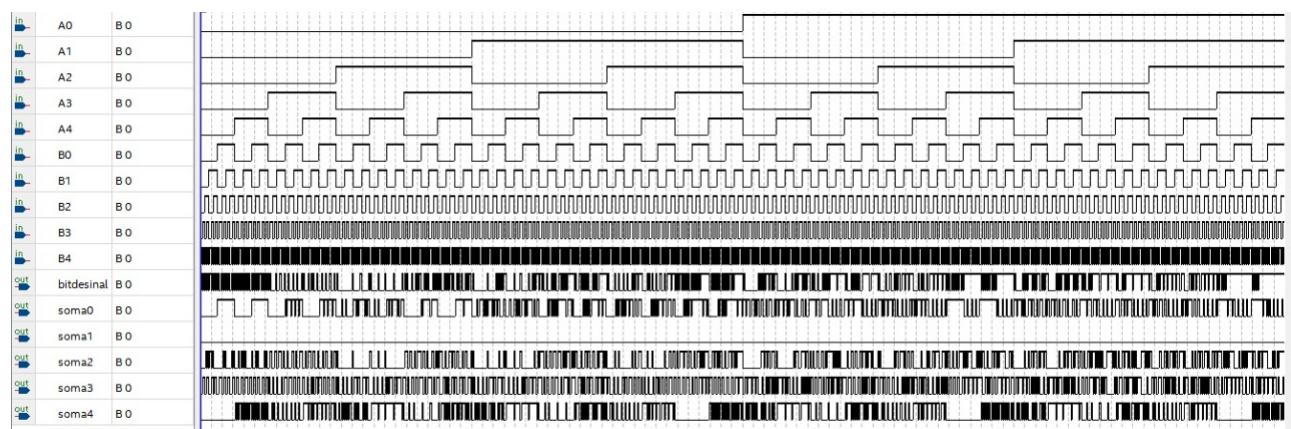


Figura 6 Waveform da simulação do somador de 4 bits



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**



### 2.1.2 Subtrator

A operação de subtração que usa o sistema de complemento de 2, na verdade, envolve a operação de soma e não é diferente dos casos de adição tratados anteriormente.

Esse módulo tem a função de realizar as operações de subtração, dada uma cadeia de números binários. Possui a mesma implementação do somador, mas com uma porta NOT, o que faz com que a função e as saídas sejam diferentes. Em nosso projeto, realizamos um somador bit a bit que compõe o subtrator.

Para efetuar-se a subtração de um número binário de um outro número binário usa-se os seguintes procedimentos:

- Operação de negação do subtraendo. Isso mudará o subtraendo para o seu valor equivalente com sinal oposto.
- Adicione esse número obtido ao minuendo. O resultado dessa adição representa a diferença entre o subtraendo e o minuendo.



### 2.1.2.1 Circuito Projetado e Simulação

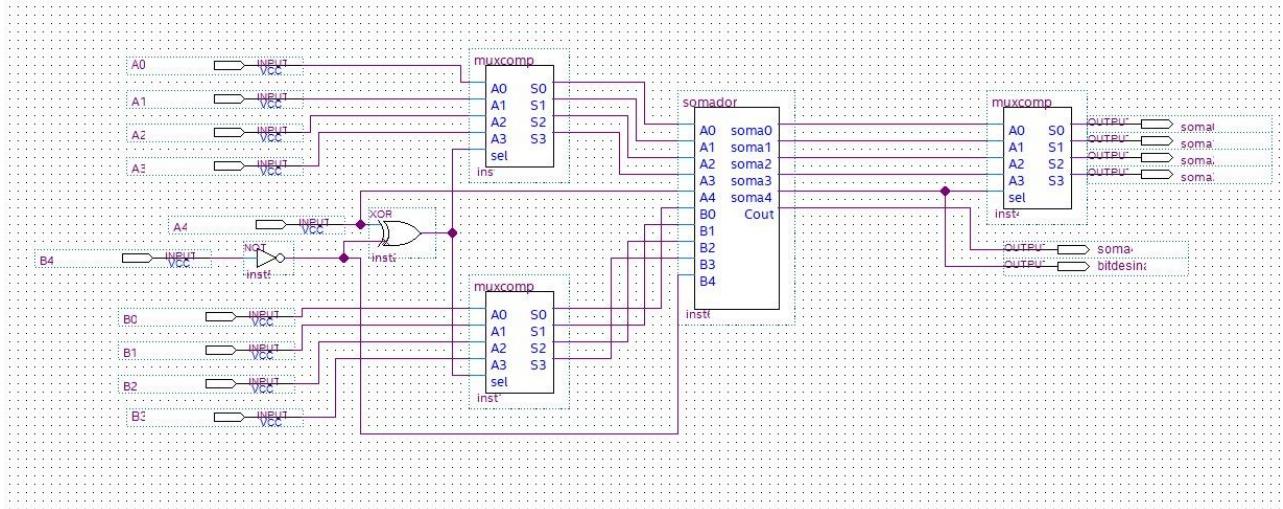


Figura 7 Implementação de um subtrator de quatro bits

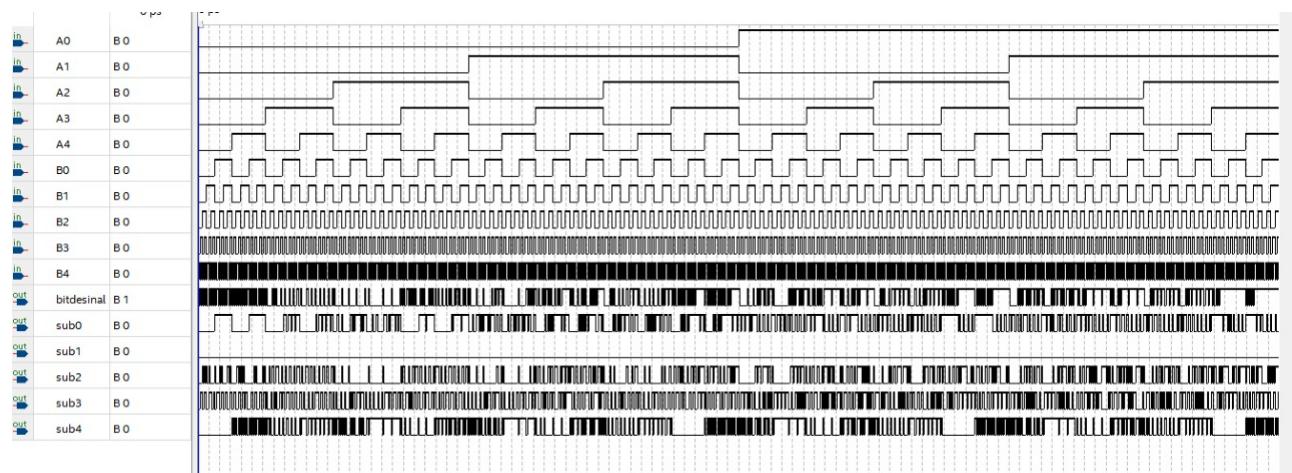


Figura 8 waveform subtrator de quatro bits



### 2.1.3 Complementador (Complemento de dois)

A função do complemento de dois na ULA é transformar cadeias binárias que representam um número em seu valor com sinal oposto, ou seja, caso um número seja 1, ele se transformará em -1. Isso é feito invertendo os bits da cadeia e adicionando 1 logo após.

Ex:

$$2 = 0010$$

Invertendo os bits: 1101

$$\text{Adicionando } 1: 1101 + 0001 = 1110 = -2$$

\*Somente quatro bits da cadeia codificam o número, o quinto bit, o mais significativo, codifica o sinal (1 é negativo e 0 é positivo).



### 2.1.3.1 Circuito Projetado e Simulação

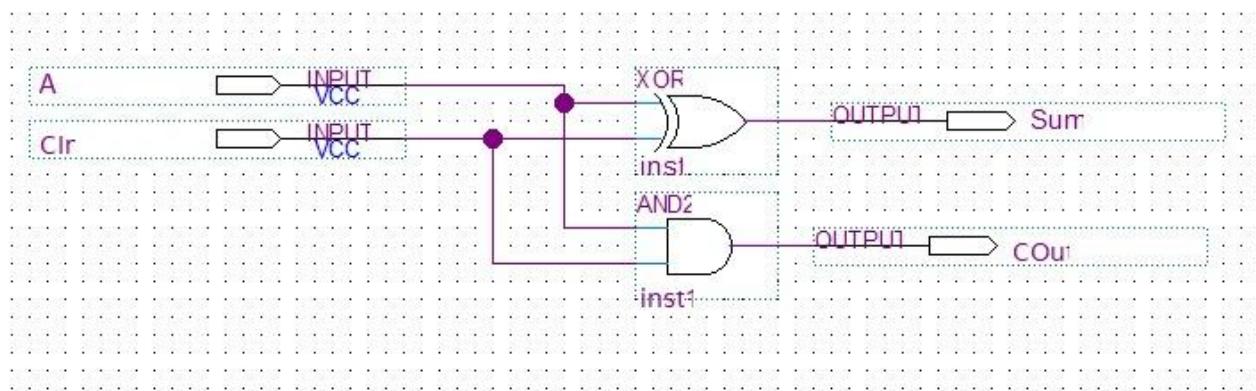


Figura 9 Adder usado no complementador de quatro bits

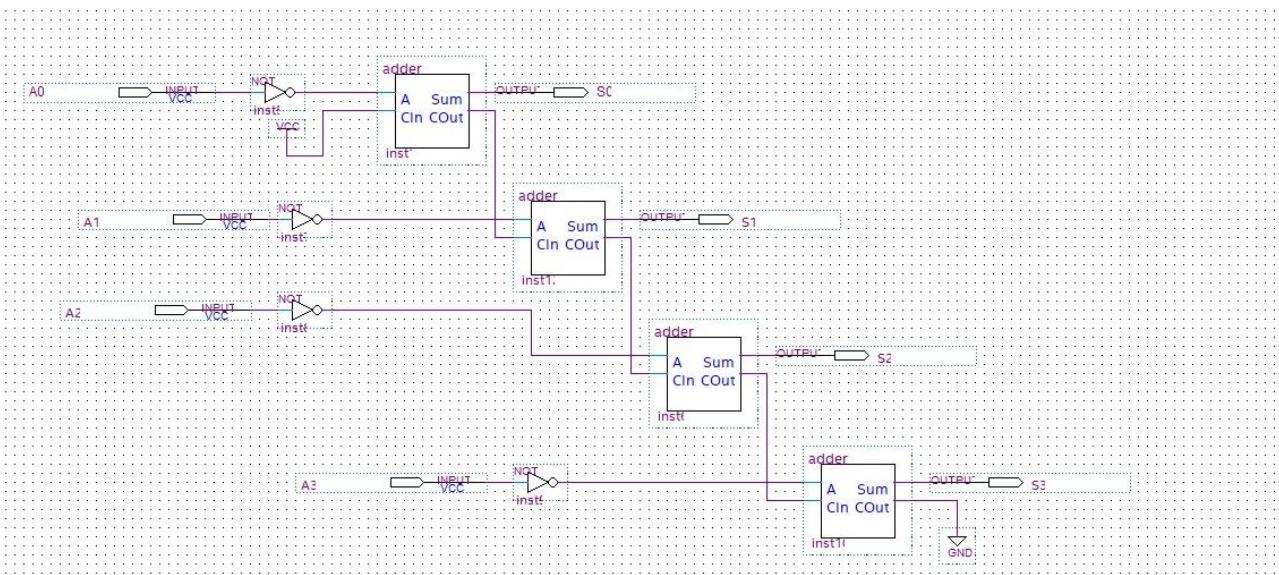


Figura 10 Implementação de um complementador de quatro bits

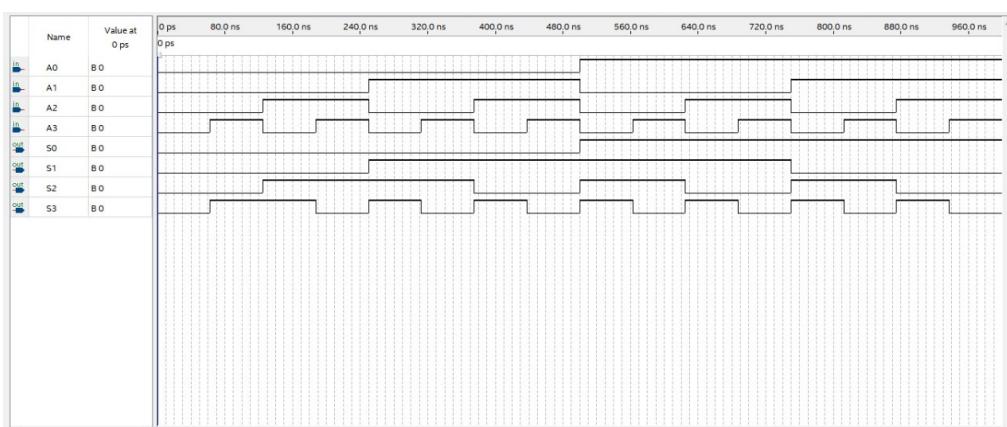


Figura 11 Waveform da simulação do complementador de 4 bits



## 2.1.4 And e Xor

### 2.1.4.1 And

Este módulo do sistema realiza a operação AND a um vetor de 5 bits (quatro referente ao número em si e um bit de sinal). Essa 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 A AND A é A.

### 2.1.4.2 Xor

Esse módulo do sistema realiza a operação XOR a um vetor de 5 bits (quatro referente ao número em si e um bit de sinal). 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, por propriedades de álgebra booleana, teremos um vetor de valor 0 na saída.

### 2.1.4.3 Tabela Verdade

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”

#### 2.1.4.4 Circuito Projetado e Simulação

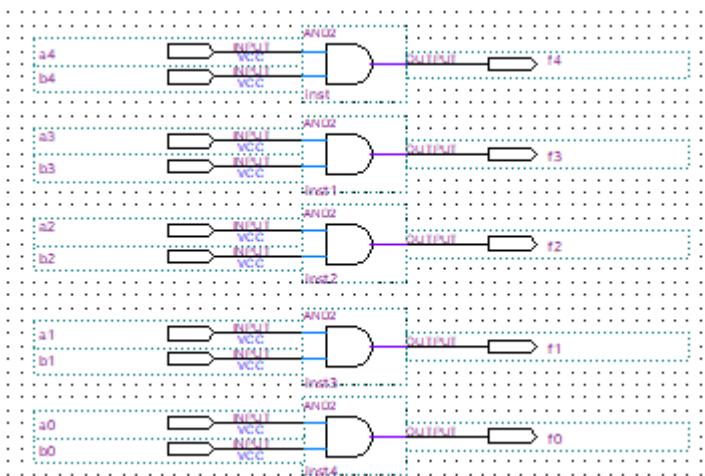


Figura 12 Implementação do AND de quatro bits

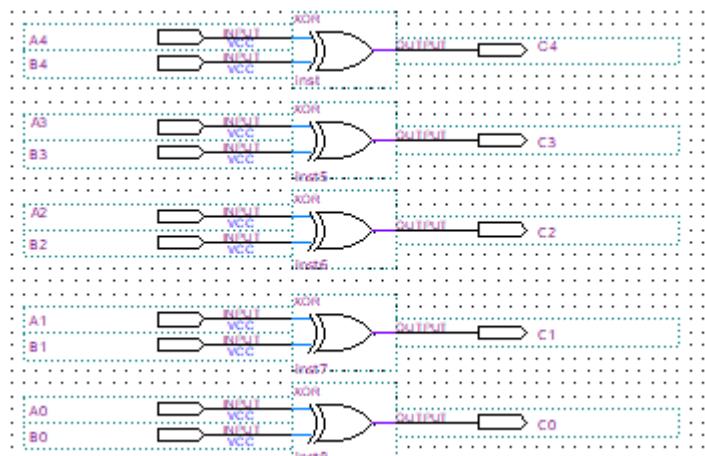


Figura 13 Implementação do XOR de quatro bits

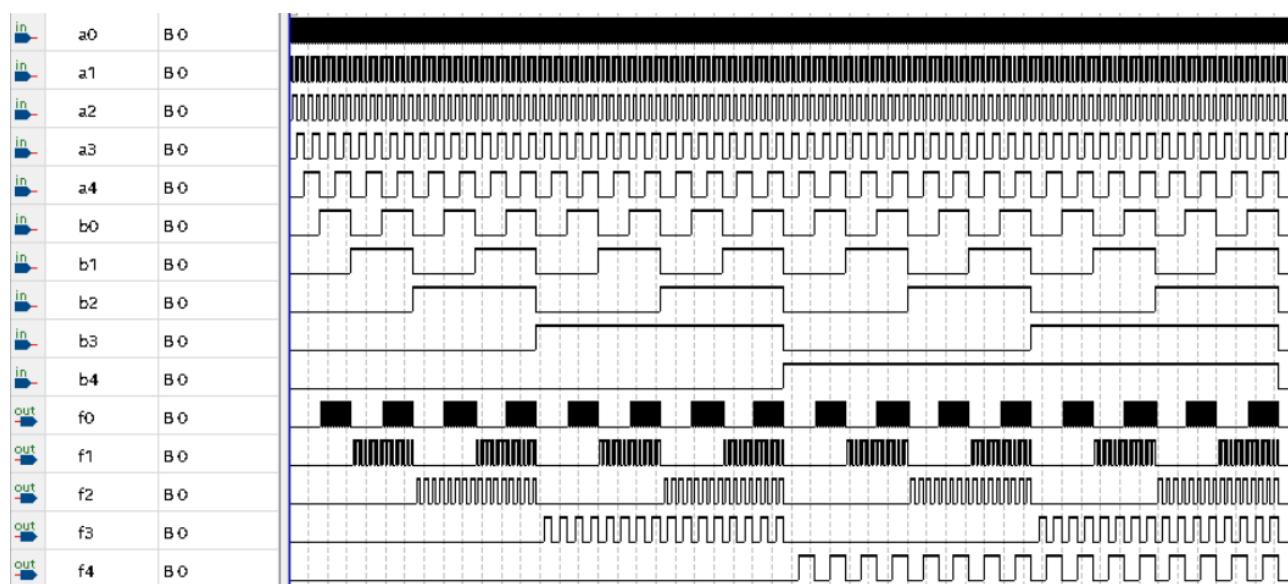


Figura 14 Waveform da simulação do AND de 4 bits

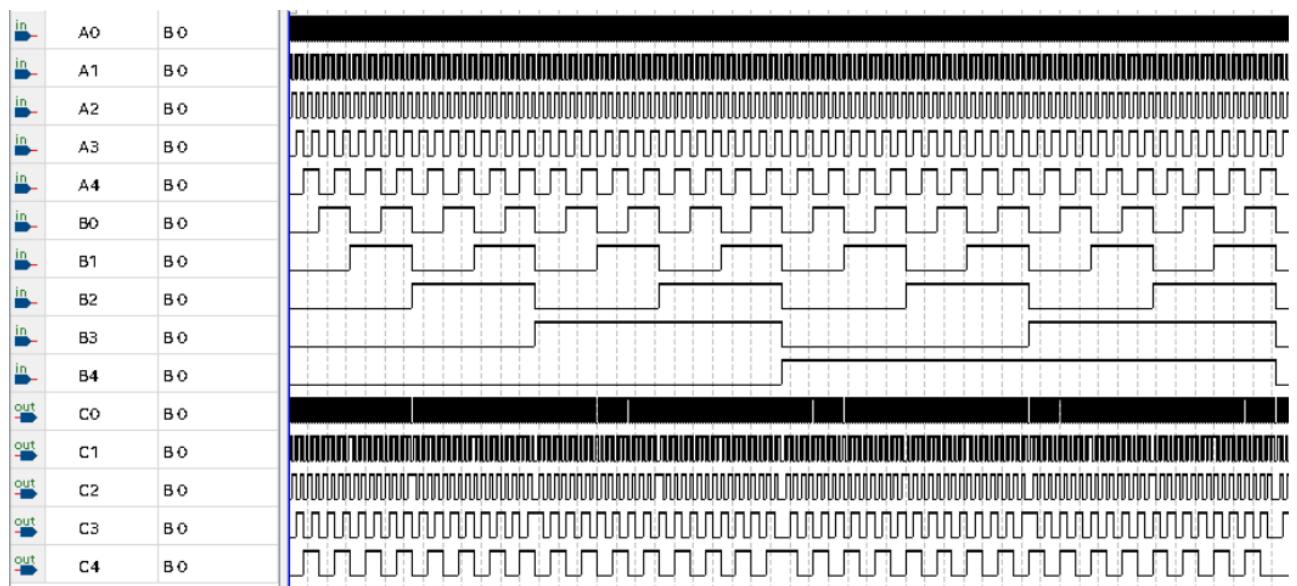


Figura 15 Waveform da simulação do XOR de 4 bits



## 2.1.5 Comparadores Lógicos

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

### 2.1.5.1 Maior que, menor que, igual a

**Igual:** no comparador igual primeiro fazemos XORs com os 4 bits do número e seu quinto bit de sinal, caso eles sejam iguais vai sair 0 e diferentes vai sair 1, além disso, usamos um not para inverter esses devidos valores, no final, colocamos em um and para saber se são iguais, pois, caso contrário, a saída será 0.

**Maior:** no comparador maior primeiro usamos XORs com os 4 bits de cada número e um quinto bit de sinal, comparando-os bit a bit através de ANDs. No final, utilizamos portas lógicas OR para recebermos o resultado. Ps: utilizamos o método de cascata bit a bit.

**Menor:** no comparador menor utilizamos a mesma lógica do maior, primeiro usamos XORs com os 4 bits de cada número e o quinto bit de sinal comparando-os bit a bit através de ANDs. No final utilizamos portas lógicas OR para recebermos o resultado final. Ps: também utilizamos o método de cascata, na qual o circuito vai comparando bit a bit de cima para baixo e através das portas lógicas oferece o resultado esperado.



### 2.1.5.2 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	<u>1</u>

**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):  
 $E(A \cdot B)$
- Comparador maior, bit normal (1 bit):  
 $\neg B(A + E) + E(A + \neg B)$
- Comparador maior, bit mais significativo (1 bit):  
 $B(E + \neg A) + \neg A(E + B)$

### 2.1.5.3 Circuito Projetado

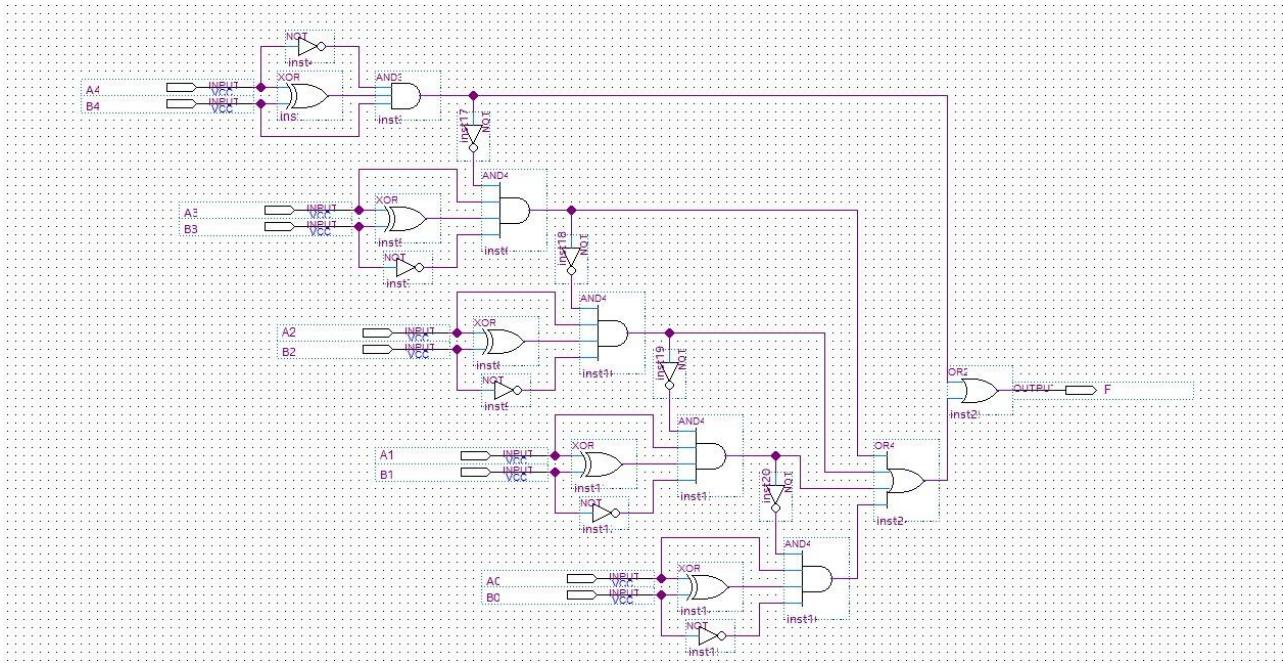


Figura 16 Implementação do comparador maior

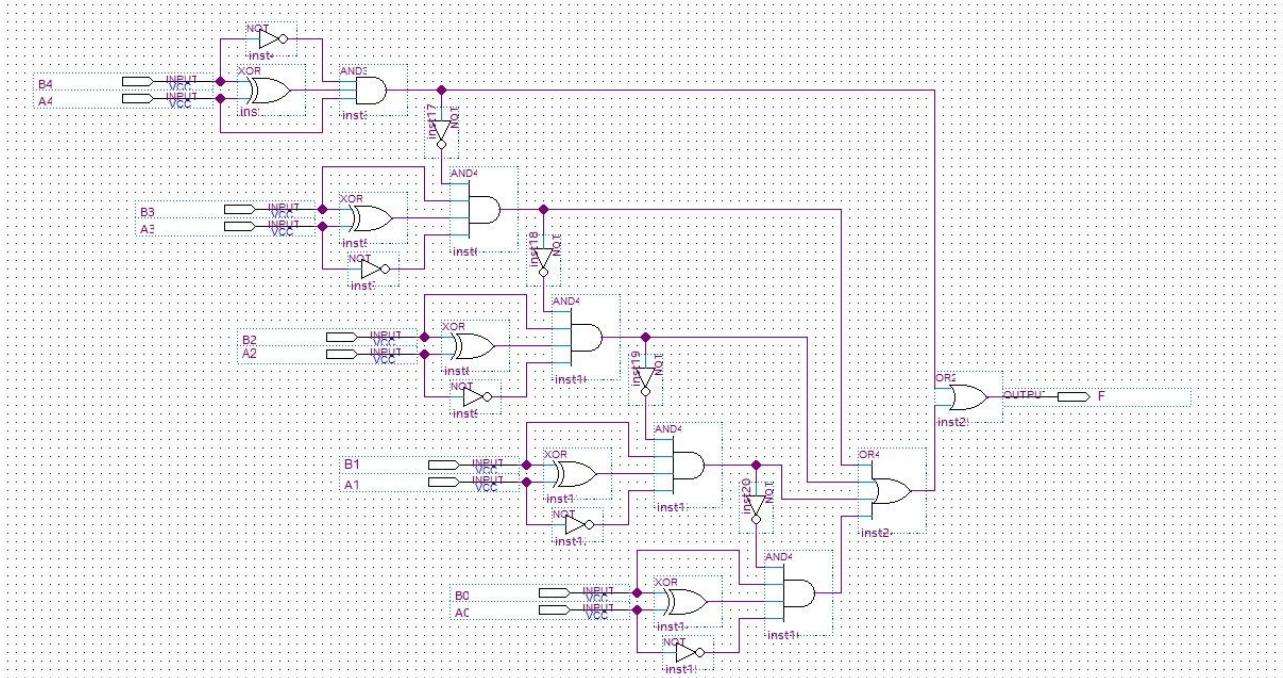


Figura 17 Implementação do comparador menor

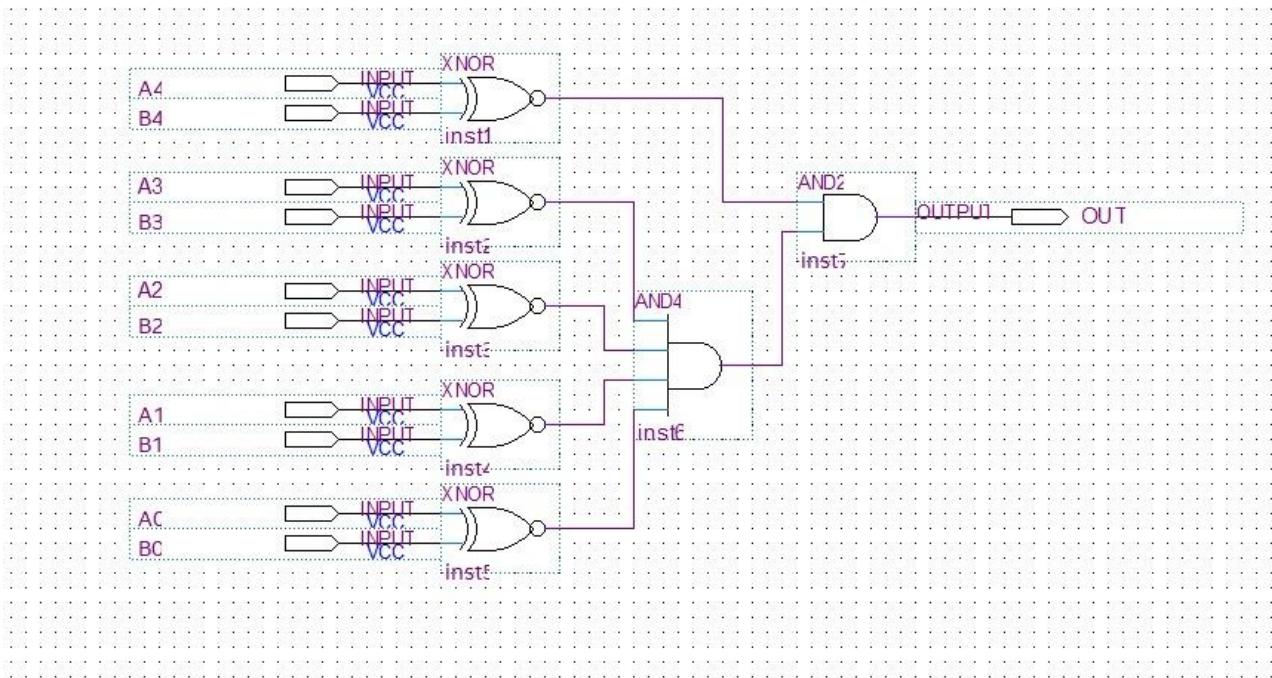


Figura 18 Implementação do comparador igual

## 2.1.6 Demultiplexadores

Esses componentes servem como chaves seletoras que selecionam uma entre várias entradas dependendo da chave que lhes forem passadas. Nesse projeto, os DEMUX recebem os números binários (bit a bit) e redistribuem para as próximas funções da ULA, de acordo com a operação (seleção) desejada, podendo ser de soma, subtração, comparação (maior, menor ou igual), AND, XOR e complemento de 2. Foram utilizados 2 Demultiplexadores na aplicação do projeto.

### **2.1.6.1 Tabela Verdade**

A tabela verdade abaixo resume o funcionamento de uma chave com seletores.



### 2.1.6.2 Circuito Projetado e Simulação

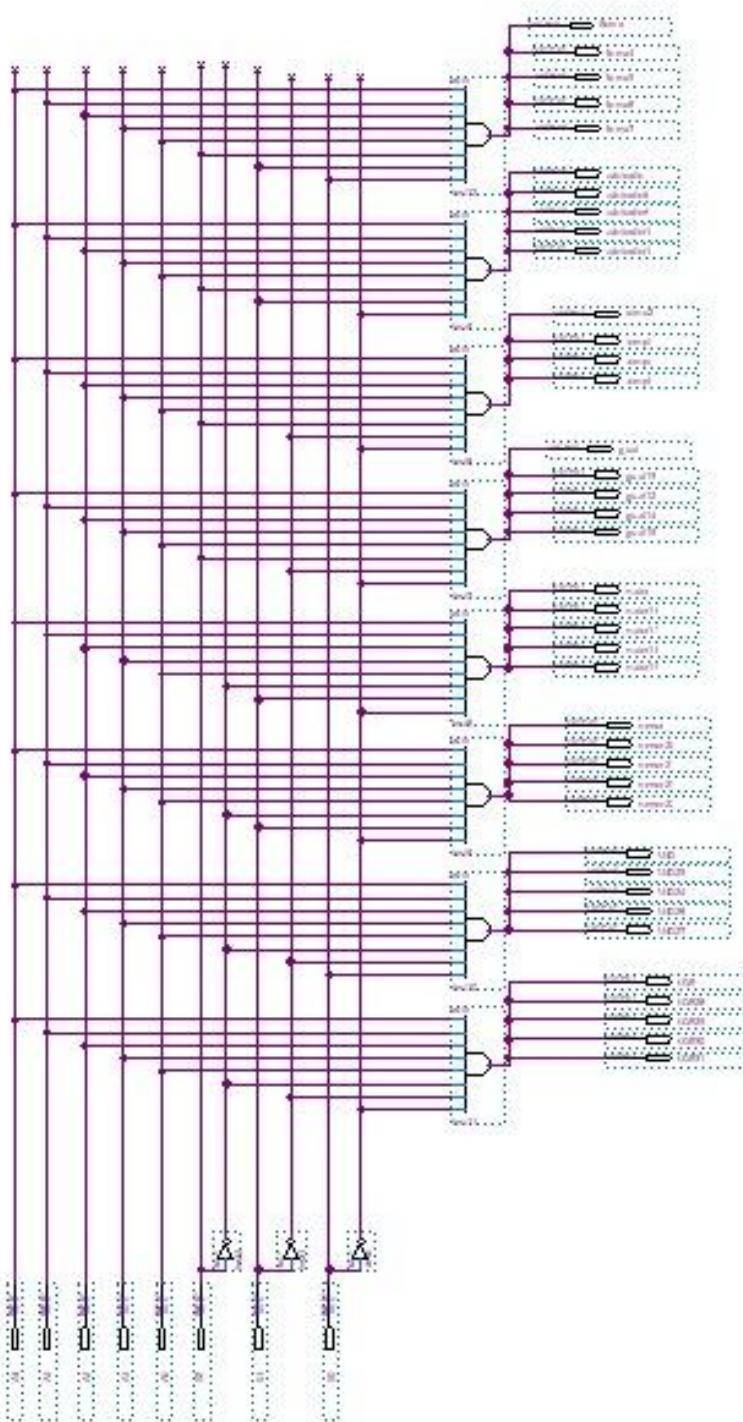


Figura 19 Implementação do demultiplexador B

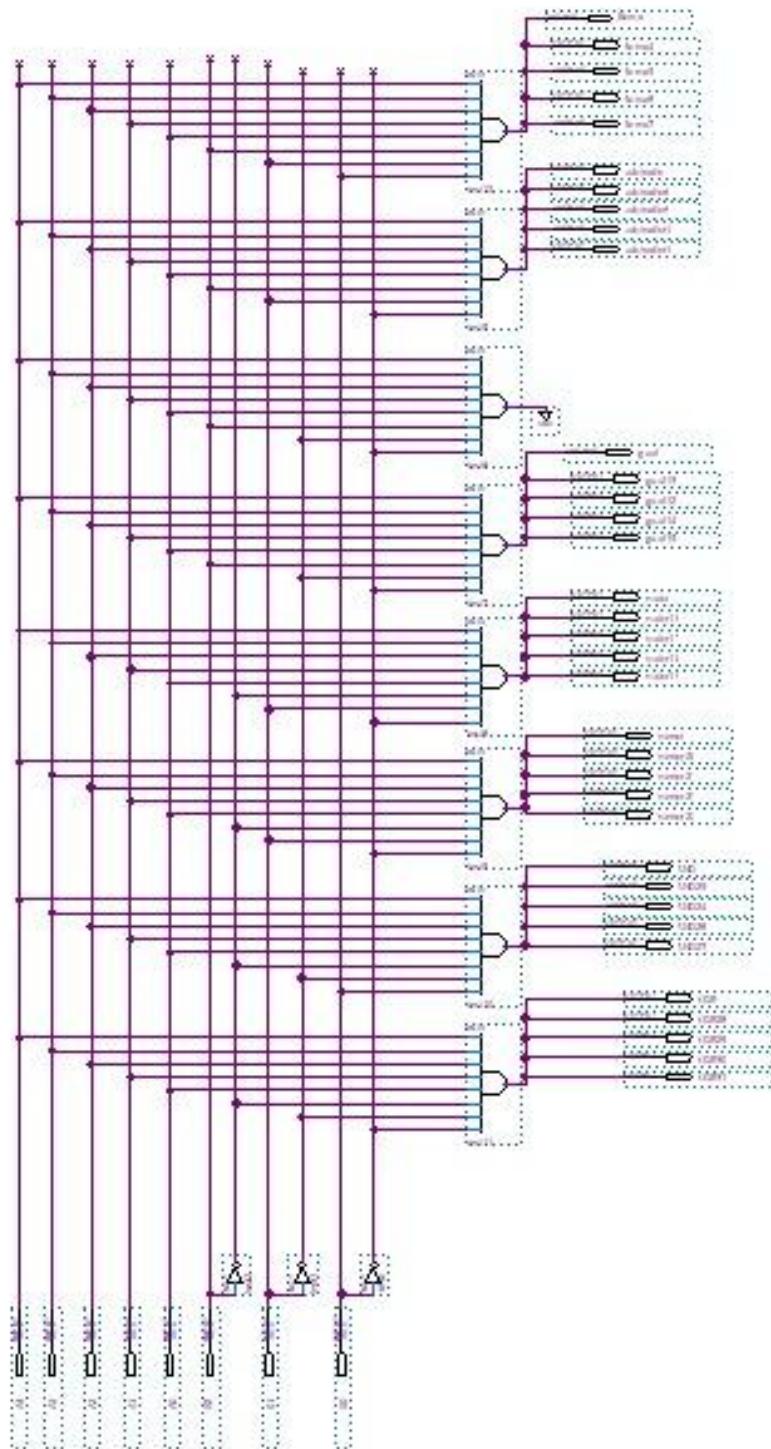
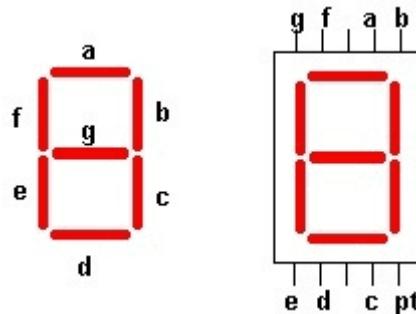


Figura 20 Implementação do demultiplexador A



### 2.1.7 Decodificador

Na ULA, o uso do decodificador será para ativar os leds do display, no qual será mostrado o resultado do que foi pedido para ULA realizar. Este display será um de 7 segmentos, ou seja, possui 7 leds, que foram nomeados com as letras de **a** até **g**, como mostra a figura abaixo:



**Figura 16 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
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

**Tabela 8 Tabela que correlaciona os valores decimais com as cadeias binárias de 5 bits**

(apenas para os valores positivos, no nosso projeto, vai de -16 até 15).

Nessa tabela, estamos considerando o bit  $A_4$  como o bit de sinal, ele está sendo sempre 0.



**Obs.:** Cada led possui sua própria tabela verdade. Cada uma foi elaborada de forma a decodificar uma sequência de um vetor de 5 bits. Essa cadeia representa um número de 0 a 15 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. Em nosso projeto, foi criado um decodificador para a representação das unidades e outro para a representação das dezenas, a fim de facilitar o entendimento. Implementamos o mesmo no início da ULA para representar os números de entrada, como também colocamos um no final para representar o número de saída, fizemos também um display para o sinal dos respectivos números tanto no início como no final. Além disso, adicionamos 7 leds finais, 1 (LED) para indicar que o resultado é negativo (aceso quando negativo e apagado quando positivo), 1 (LED) representando o status (para as operações que retorna um booleano) e 5 LEDs para replicar a saída F que representam o resultado da operação aritmética devidamente escolhida na ULA. Como exemplo para ilustrar essa aplicação, tem-se o vetor “00000”, 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 acima e abaixo, para melhor compreensão. A luz do circuito é ativada caso o valor de saída seja igual a zero.



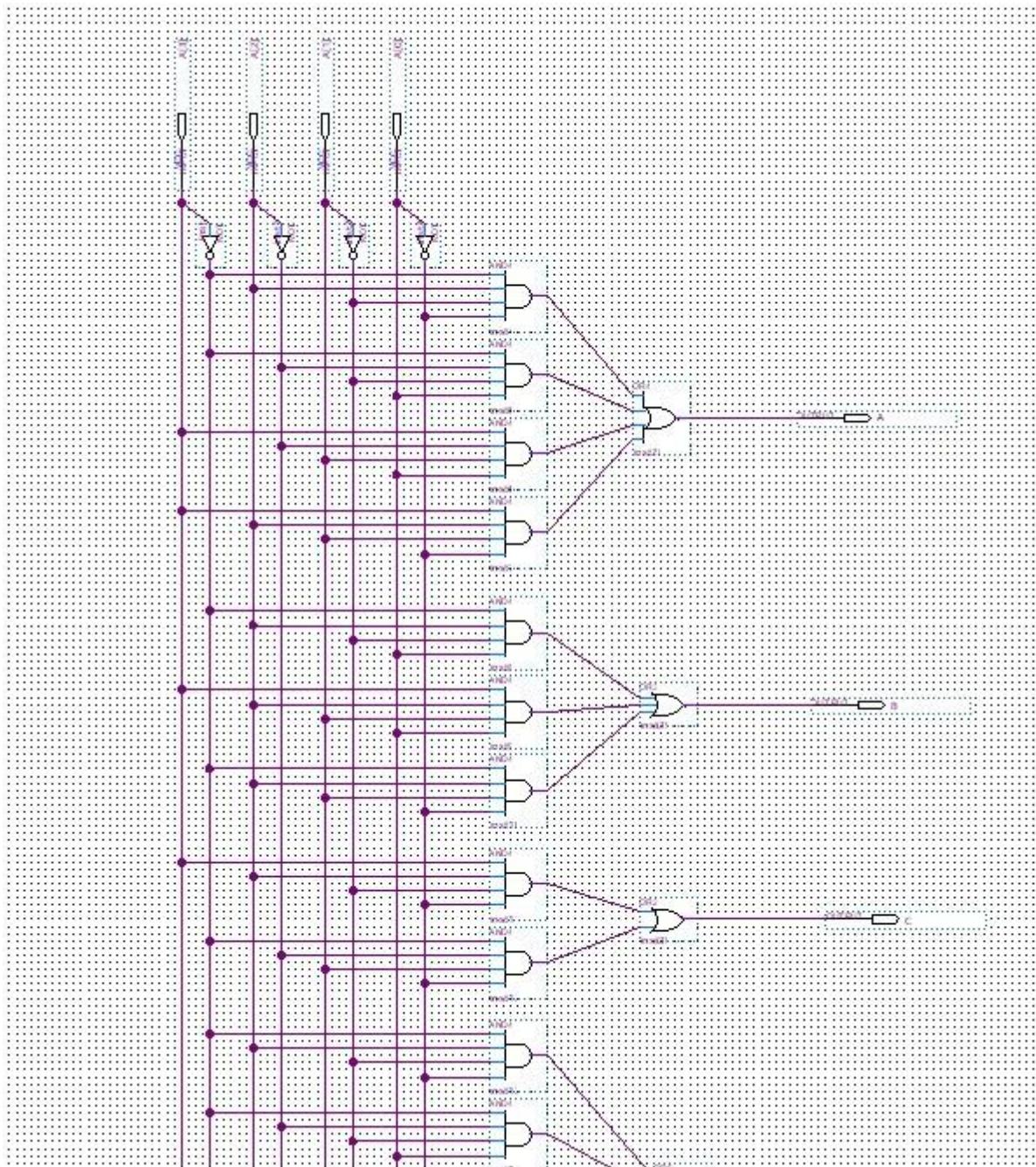
### 2.1.7.1 Tabela Verdade e Mapas de Karnaugh

A	B	C	D	E	ad	bd	cd	dd	ed	fd	gd	au	bu	cu	du	0	fu	gu
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1	1	1
0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0
0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	1
0	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1
0	1	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	1	0
0	1	1	1	0	1	0	0	1	1	1	1	1	0	0	0	1	1	0
0	1	1	1	1	1	0	0	1	1	1	1	0	0	1	0	0	1	0
1	0	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1
1	0	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1
1	0	1	0	1	0	0	1	0	0	1	0	1	0	0	1	1	1	1
1	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0
1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
1	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0
1	1	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1
1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0
1	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0	1
1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1

Tabela 9 Tabela-verdade do módulo: Decodificador Completo



### 2.1.7.2 Circuito Projetado e Simulação



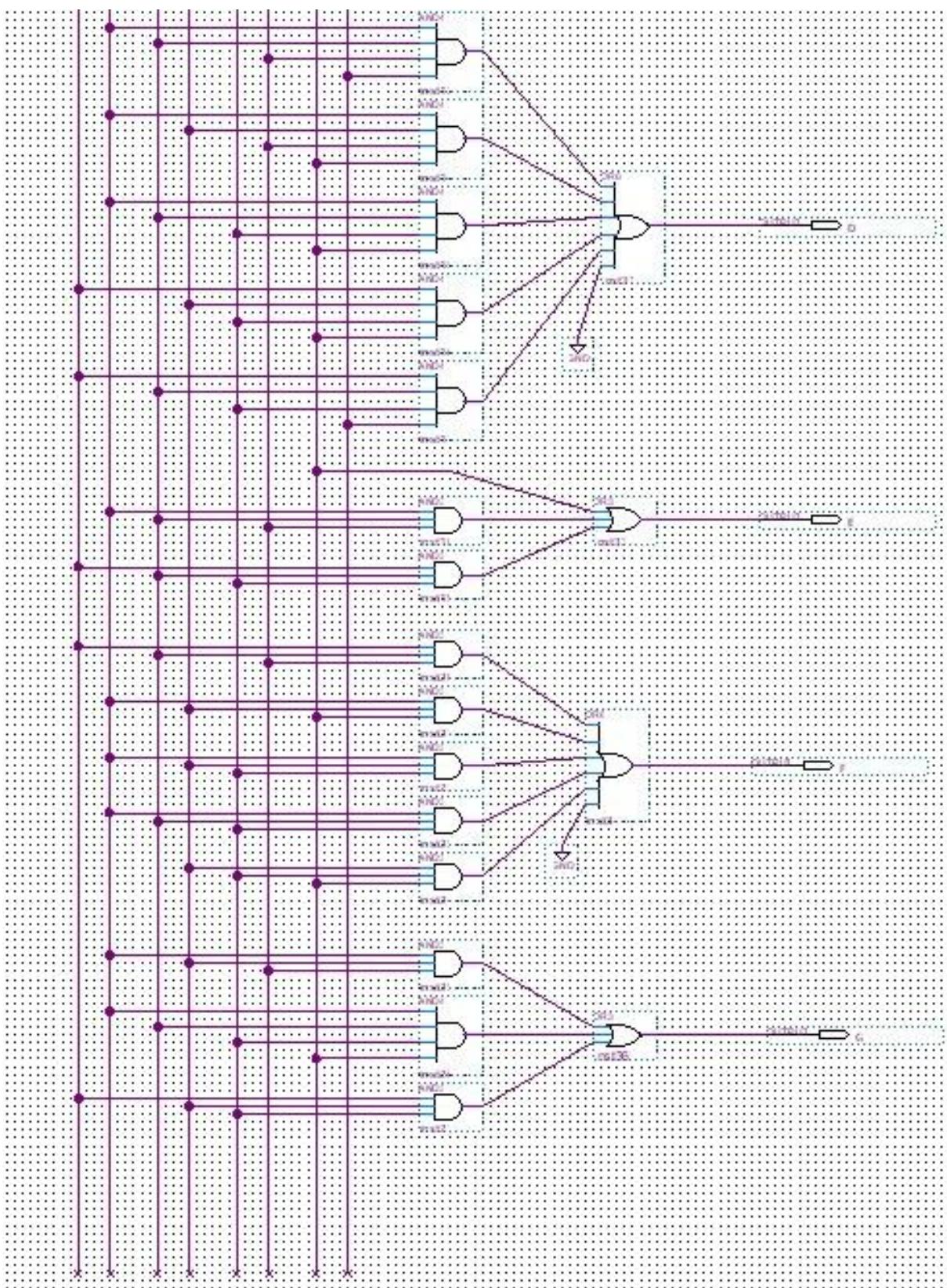


Figura 21 Implementação do DecodificadorA unidade

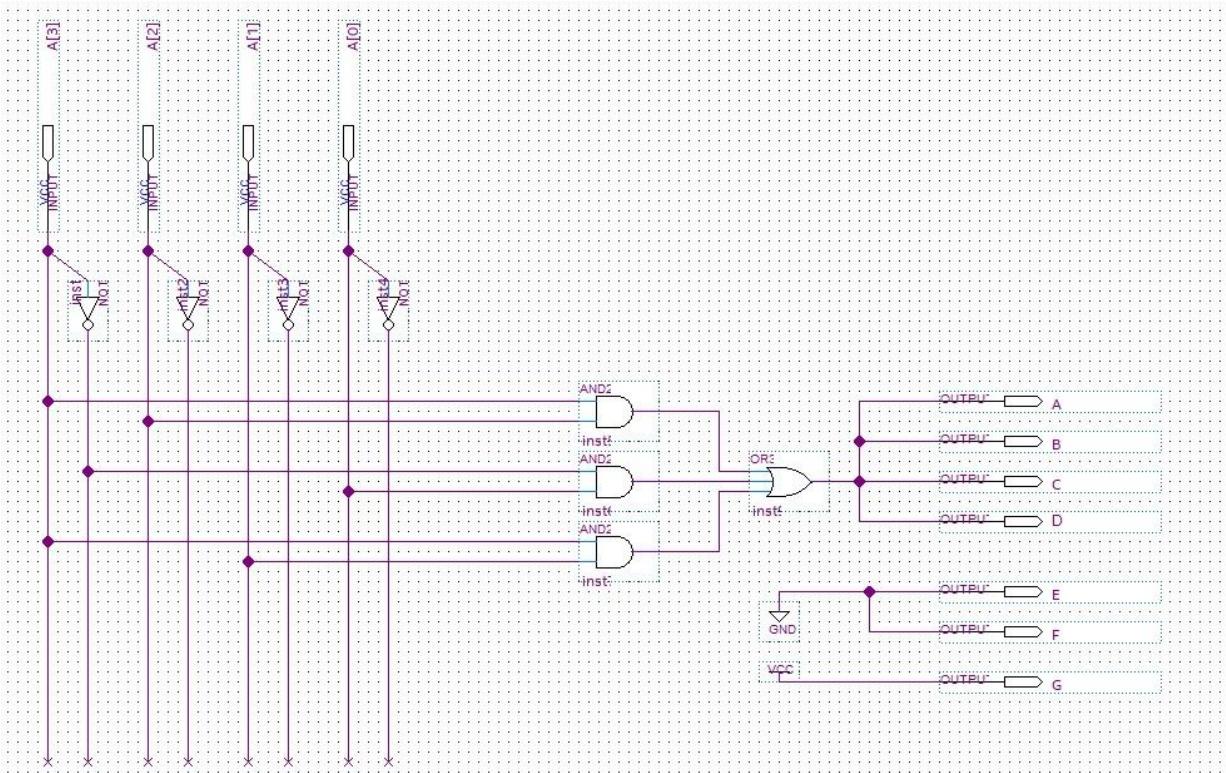


Figura 22 Implementação Decodificador A dezena

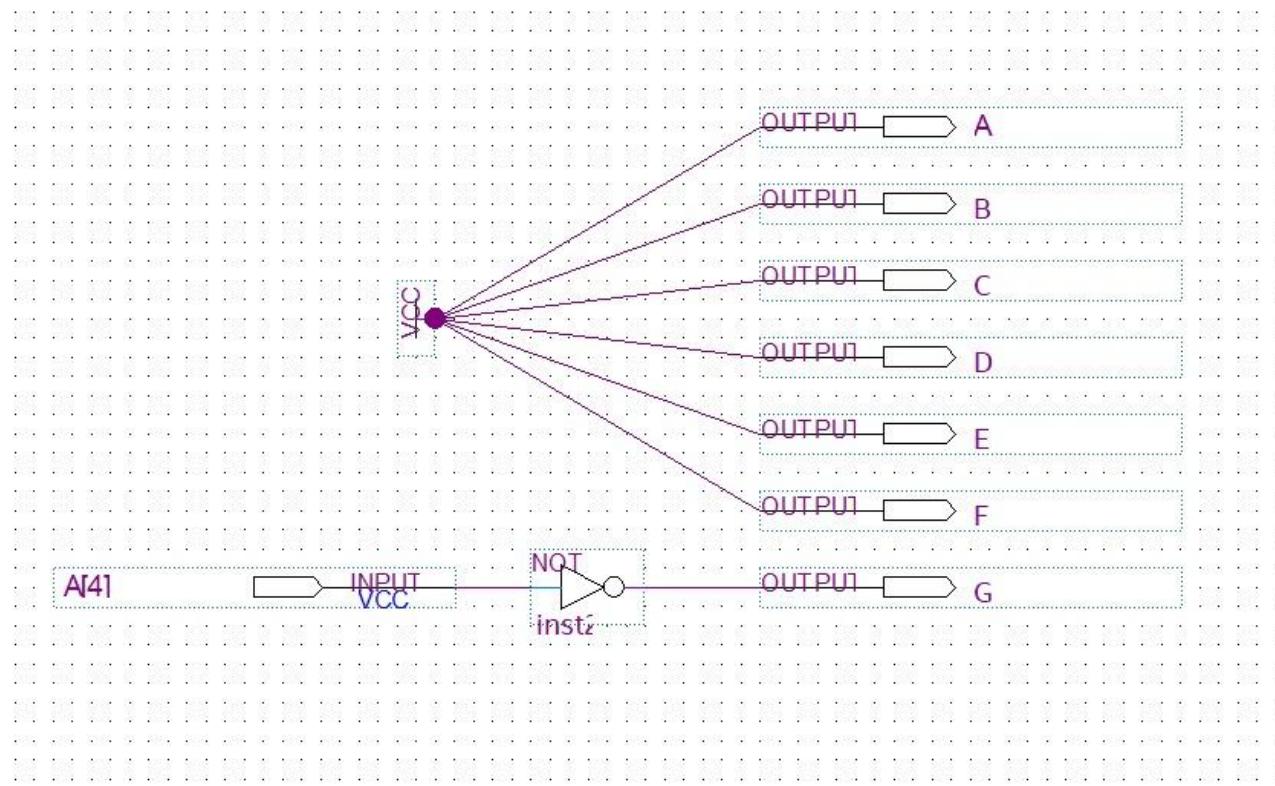
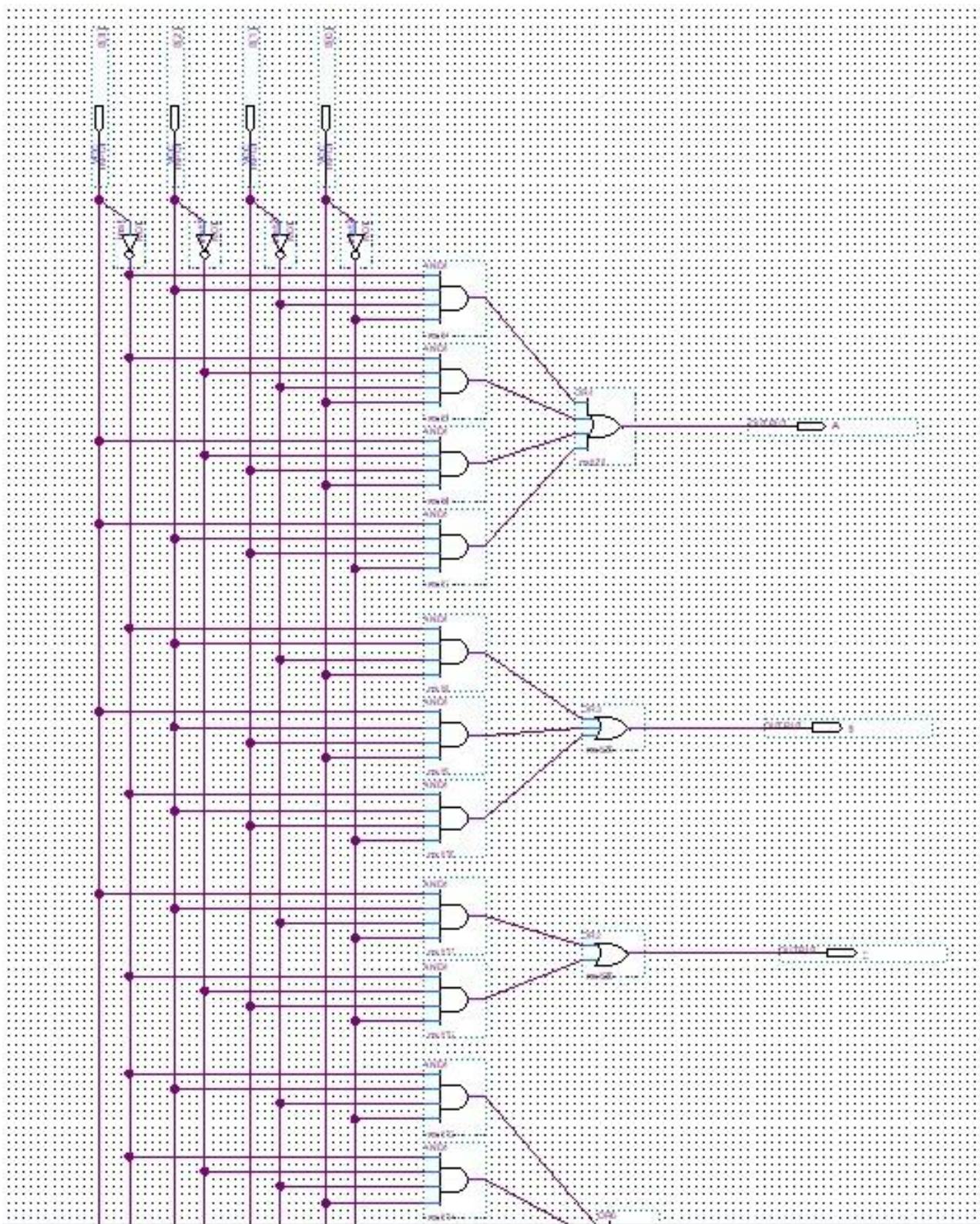


Figura 23 Implementação sinal A



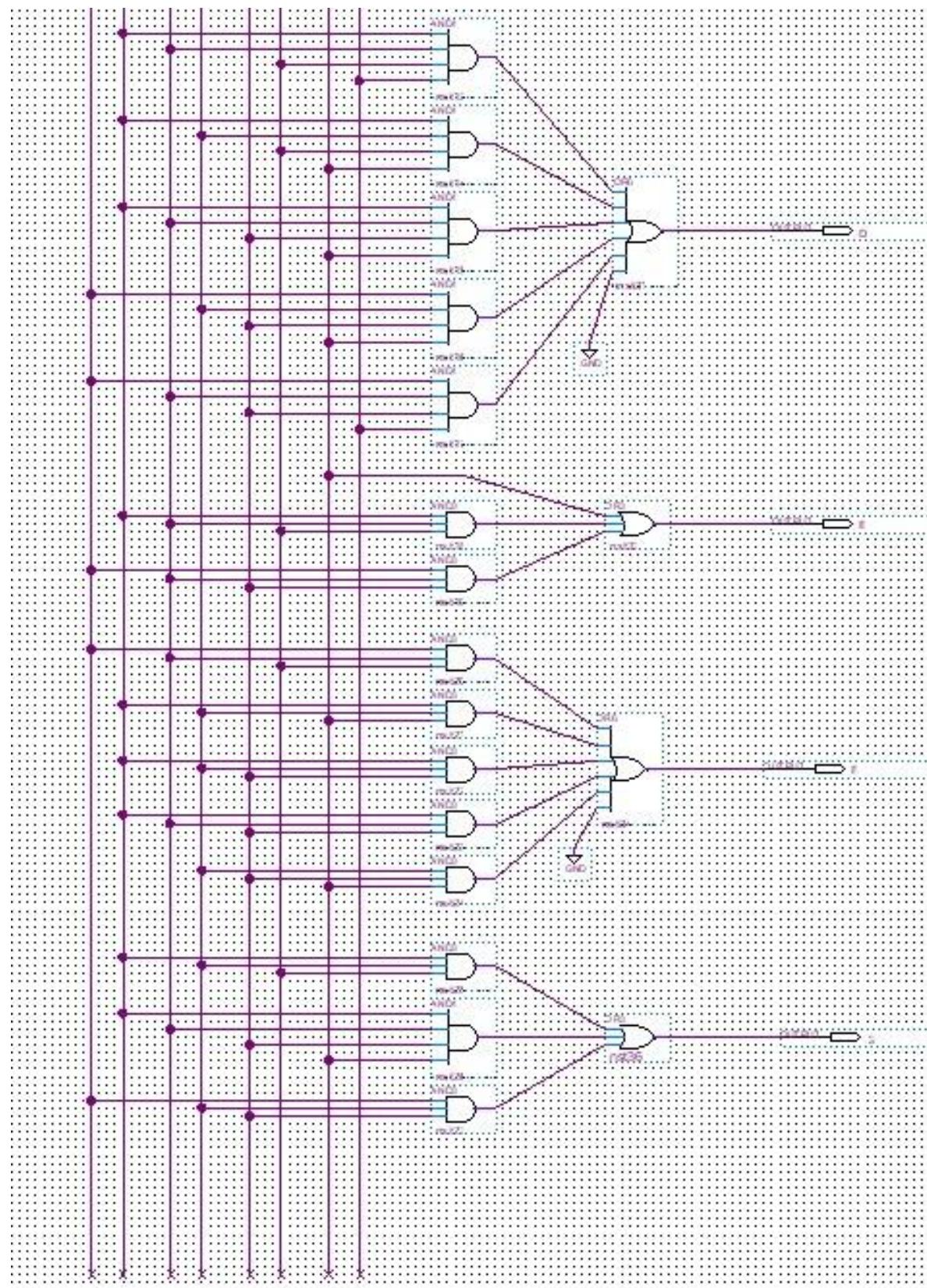
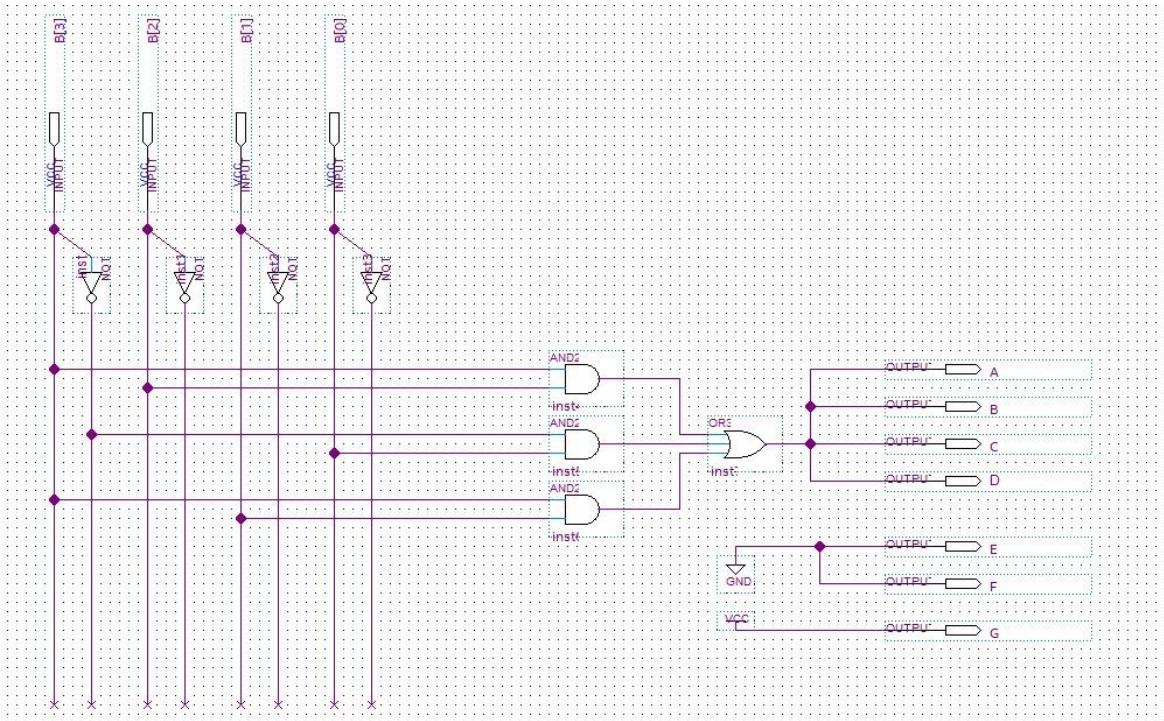
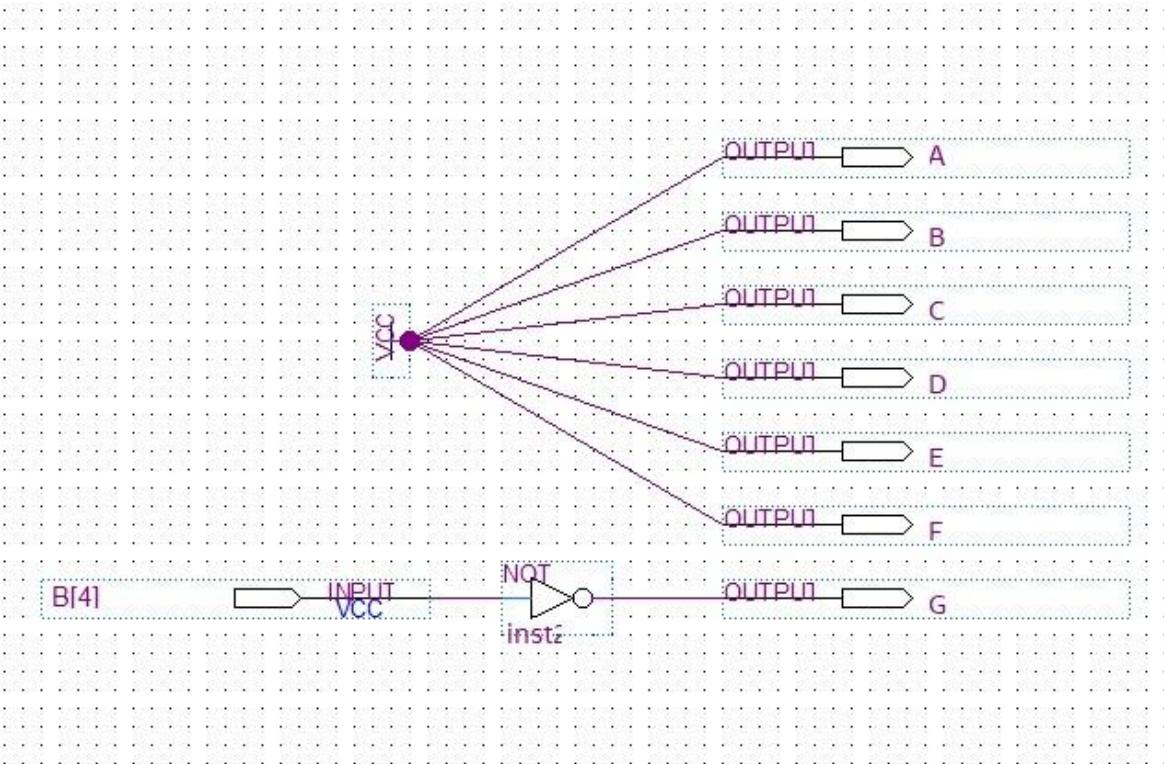


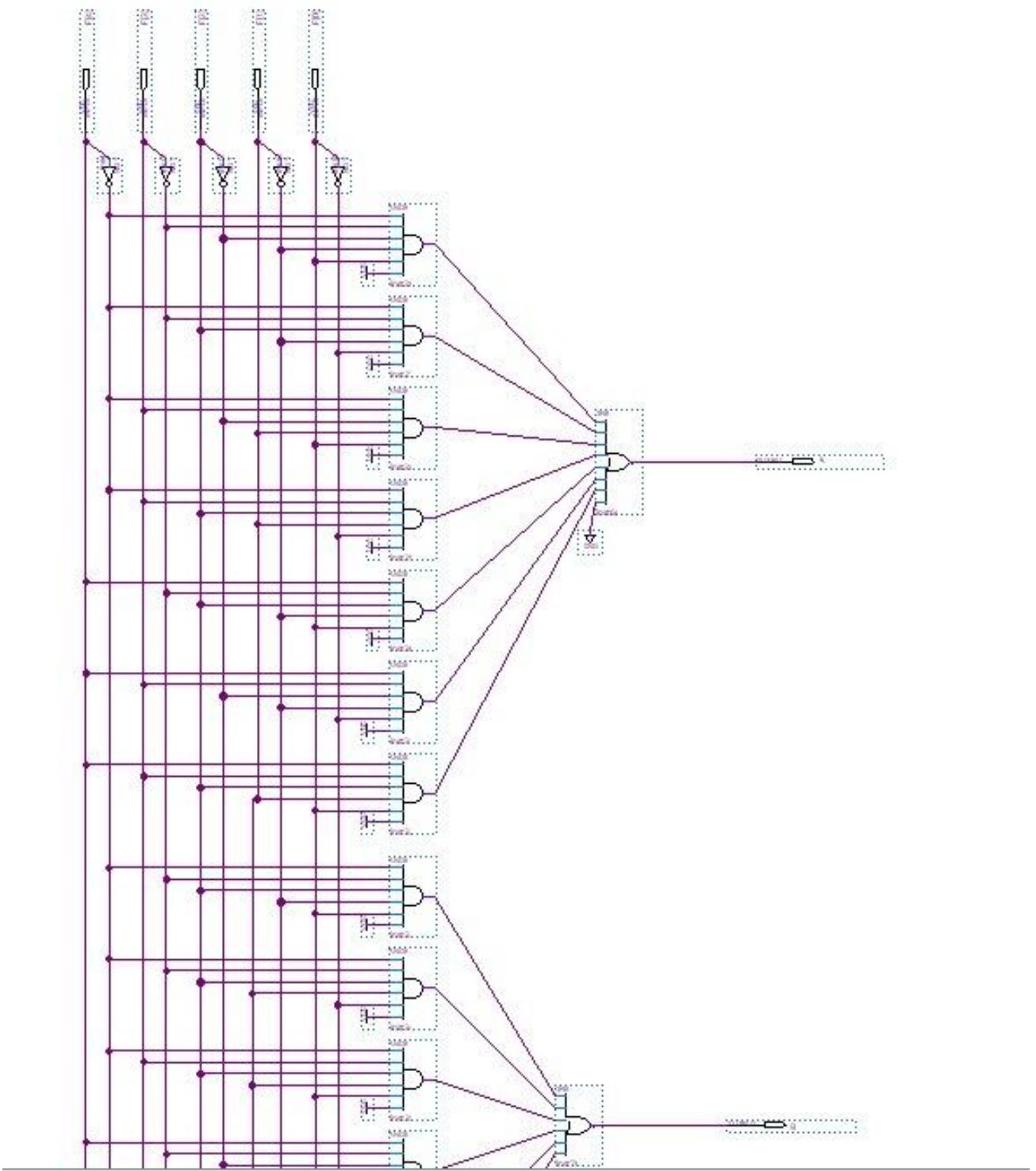
Figura 24 Implementação do DecodificadorB unidade

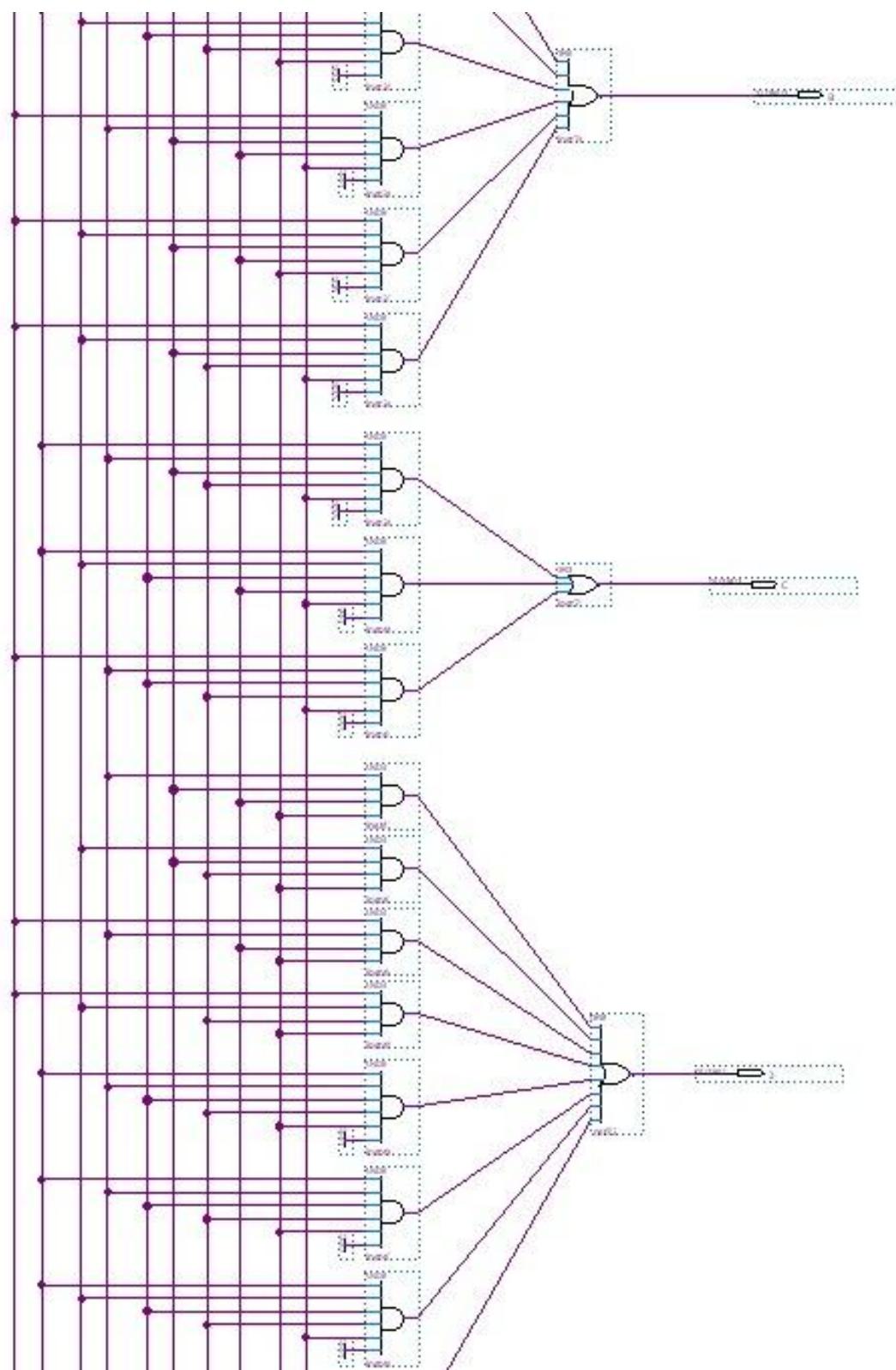


**Figura 25 Implementação DecodificaodordB dezena**



**Figura 26 Implementação sinalB**





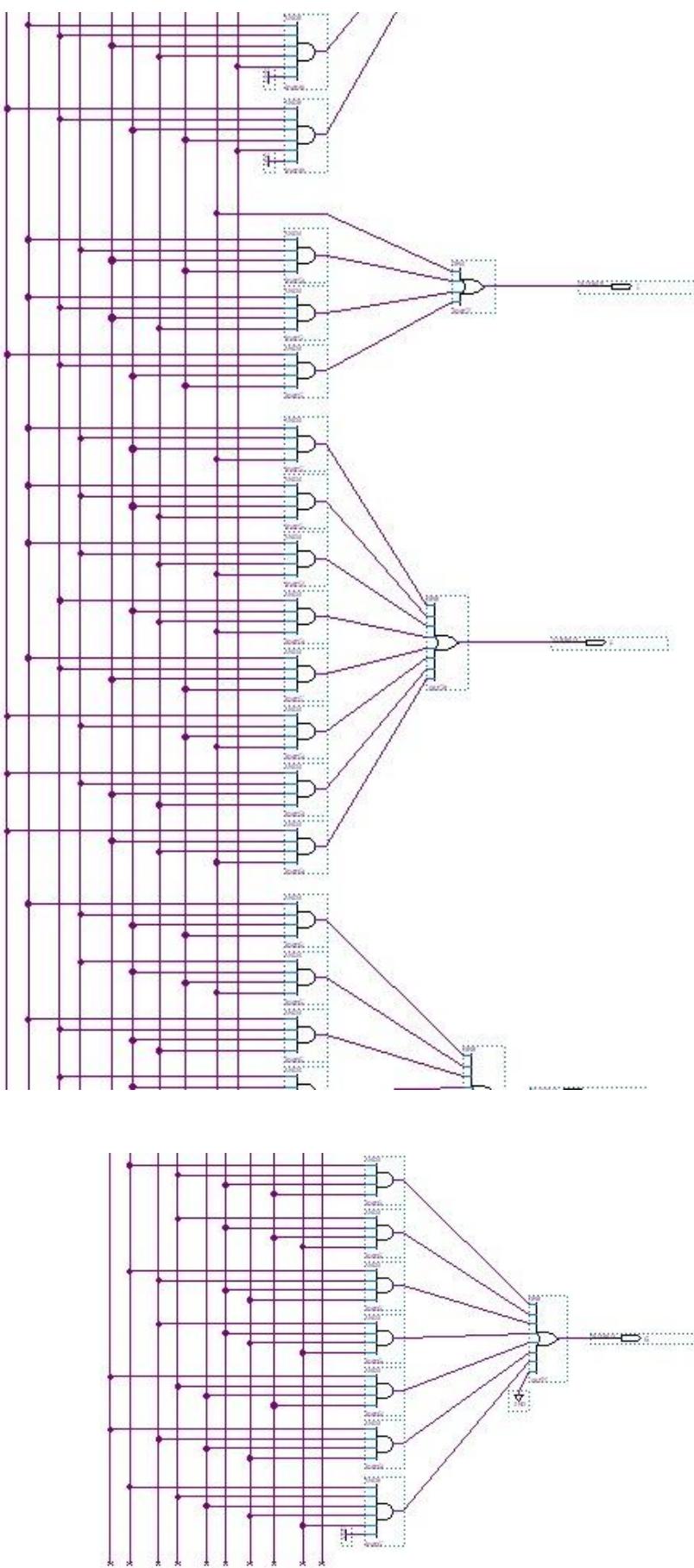
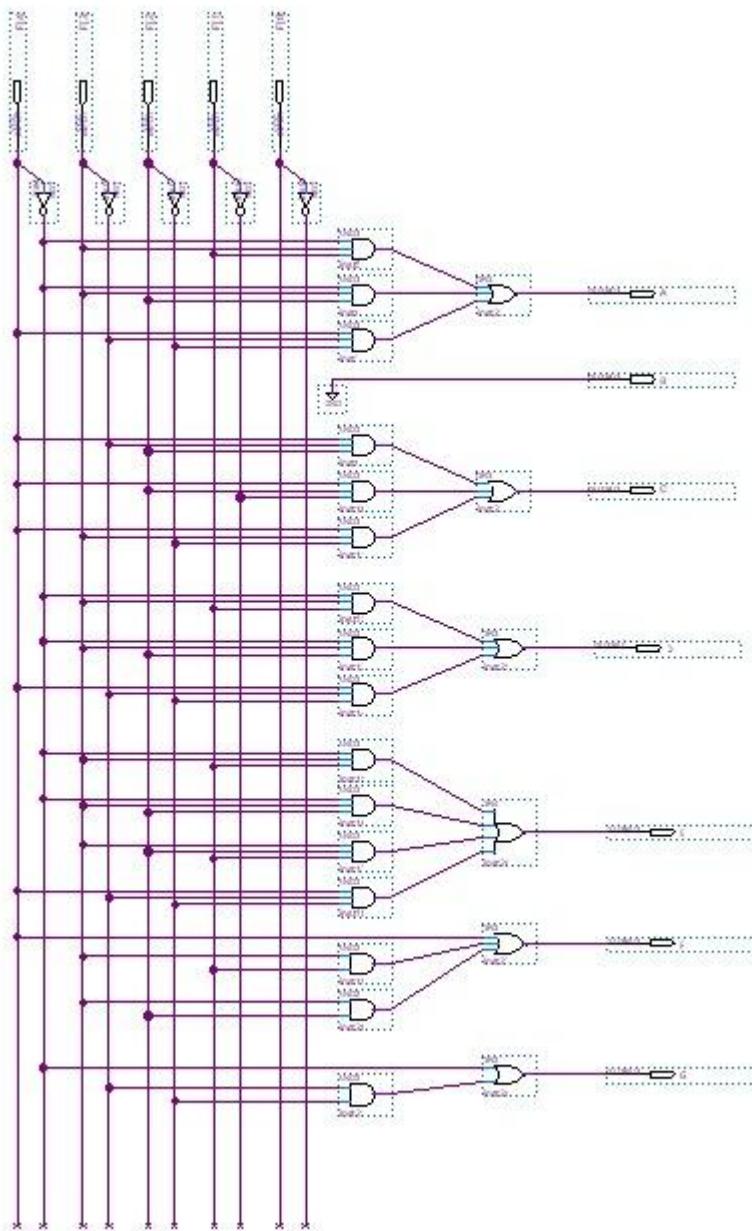
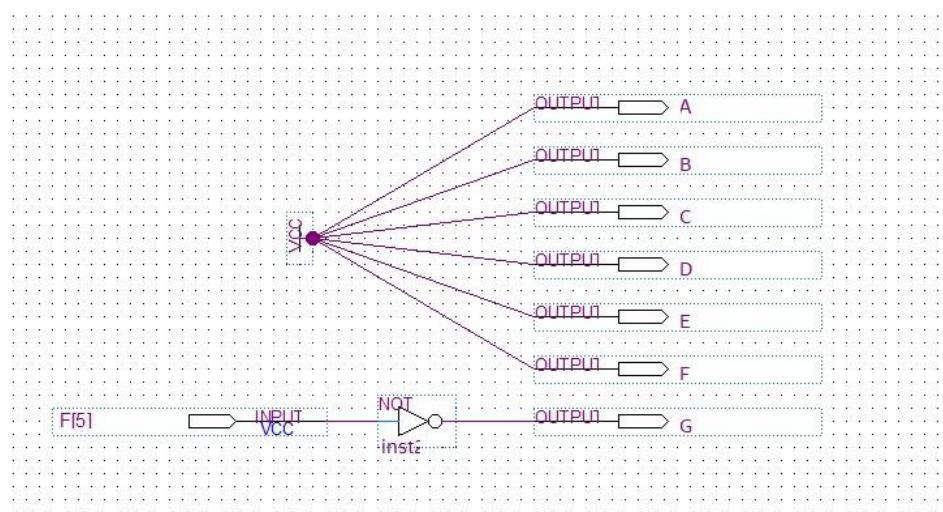


Figura 27 Implementação Decodificador resultado unidade



**Figura 28 Implementação Decodificador resultado dezena**



**Figura 29 Implementação sinal resultado**

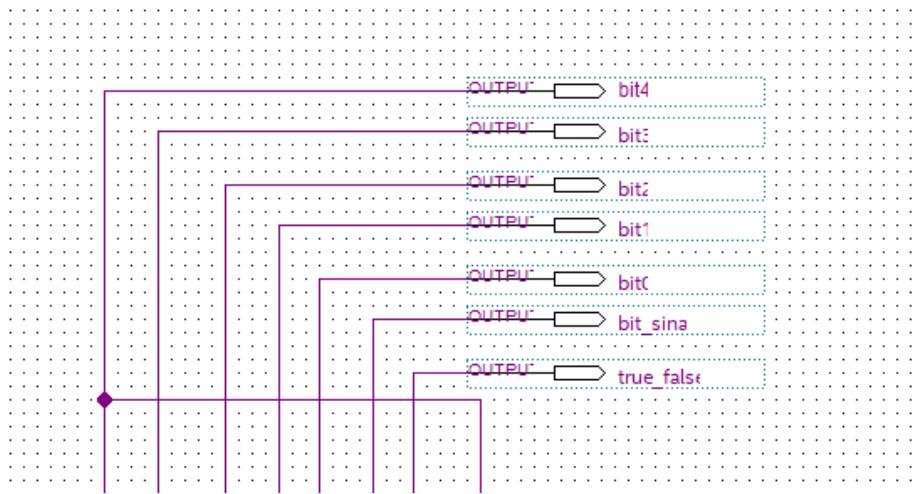


Figura 30 7 leds que representam as saídas (5) o sinal (1) e o booleano (1)



### 3 Conclusão

Neste trabalho, concluímos que há uma grande importância em conhecer a ULA e suas aplicações, pois ela apresenta as principais operações lógicas e aritméticas de um computador. Ela soma, subtrai, divide, determina se um número é positivo, negativo ou se é zero. Além de executar funções aritméticas, uma ULA deve ser capaz de determinar se uma quantidade é menor ou maior que outra e quando quantidades são iguais. Com isso, colocamos em prática nossos conhecimentos das aulas e desenvolvemos um somador, substrator, comparador, entre outros.

É importante, também, destacar seu peso histórico, já que a mesma é, na verdade, uma "grande calculadora eletrônica" desenvolvida durante a II Guerra Mundial e sua tecnologia já estava disponível quando os primeiros computadores modernos foram construídos. Logo, ter o conhecimento do seu funcionamento é de grande importância para o desenvolvimento de um profissional da área de tecnologia. Ainda, pudemos observar, na prática, como são suas waveforms (desenvolvidas no Quartus).

Tudo isso gerou uma experiência incrível, juntando nossos conhecimentos em Sistemas Digitais, principalmente os de portas lógicas, conseguimos aplicar de forma usual um NOT, um XOR, um AND e, até mesmo, montar funções com a união das mesmas. Dessa forma, esse trabalho nos forneceu não só conhecimento teórico, mas também, prático e usual.



#### 4 Referências Bibliográficas

<http://fpgaparatodos.com.br/2012/02/02/decodificador-para-display-de-set-e-segmentos/>

<https://mundoprojetado.com.br/multiplexador-e-demultiplexador-aula-8-ed/>

[http://www.telecom.uff.br/tecdig1/arquivos/apostilas/apostila\\_td1\\_parte2.pdf](http://www.telecom.uff.br/tecdig1/arquivos/apostilas/apostila_td1_parte2.pdf)

[http://usuarios.upf.br/~appel/arquil/ULA\\_projeto\\_logico.pdf](http://usuarios.upf.br/~appel/arquil/ULA_projeto_logico.pdf)

<https://www.cin.ufpe.br/~if675ec/>