ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais

PCS3635 – LABORATÓRIO DE PROJETO DE SISTEMAS DIGITAIS I

EXPERIÊNCIA 2 – Um Fluxo de Dados Simples

Planejamento da Bancada B6 – Turma 2 – Prof. Antônio

Data de Emissão: 18 de Janeiro de 2026.

Nome: Gabriel Agra de Castro Motta	**Número USP: 15452743**
Nome: Mateus Silva de Araújo	**Número USP: 15497076**
Nome: Vinicius Akira Durante Tahara	**Número USP: 12547002**

1 Introdução

Esta experiência tem como objetivo o desenvolvimento e a validação de um fluxo de dados simples, integrando um contador binário de 4 bits (contador_163) e um comparador de magnitude (comparador_85). O projeto visa a familiarização com a descrição estrutural em Verilog, a verificação lógica via simulação no software Digital e a implementação física em uma placa FPGA DE0-CV, utilizando o Intel Quartus Prime para síntese e pinagem.

2 Descrição do Projeto

A experiência 2 se trata de um aprofundamento da aula anterior, de modo a continuar o estudo dos módulos do comparador e do contador. Desta vez, entretanto, há um maior foco no processo de criação do projeto, como a interação com arquivos em verilog e a simulação no digital.

O sistema consiste em um fluxo de dados onde a saída de um contador síncrono é constantemente monitorada por um comparador de magnitude. O usuário interage com o circuito através de chaves de entrada que definem tanto o valor de carga do contador quanto o valor de referência para a comparação.

Além disso, dentro do programa Quartus, há uma primeira interação com etapas como a escolha da pinagem e a compilação dos arquivos “.v”. Por fim, resta a etapa da programação da FPGA, que será realizada durante a próxima aula.

3 Detalhamento do Projeto Lógico

3.1 Projeto do Componente contador_163

O módulo contador_163 é um contador binário de 4 bits com sincronismo em todas as suas funções principais, operando na borda de subida do clock.

- Há uma hierarquia de controle. O sinal de clear (clr) tem prioridade máxima. Se inativo (nível baixo), zera a saída Q no próximo ciclo. Em seguida, vem o load (ld), que carrega o valor de D para Q.
- A contagem só ocorre se ambos os sinais de enable (ent e enp) estiverem em nível alto. Caso contrário, o estado é mantido.
- Ripple Carry Out é um sinal combinacional que indica o fim da contagem (quando Q atinge 15 e o enable ent está ativo). É uma implementação puramente lógica, sem depender da borda do clock para sua atualização imediata.

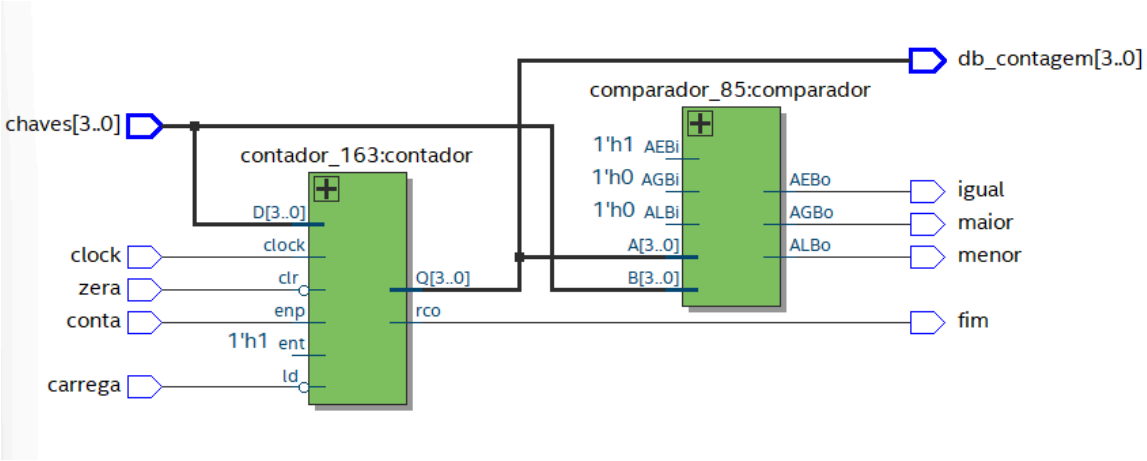
3.2 Projeto do Componente comparador_85

Diferente do contador, este é um módulo puramente combinacional que realiza a comparação de magnitude entre dois vetores de 4 bits (A e B).

- O componente aceita entradas de comparação prévia (ALBi, AGBi, AEBi), permitindo a expansão para mais bits.
- O código utiliza Verilog estrutural: em vez de apenas operadores lógicos simples, ele utiliza somas e inversões (~A + B + ALBi) para determinar a magnitude através do bit de carry.
    - O sinal ALBo (A < B) é derivado de uma soma com o complemento de A e o sinal de cascata ALBi. Da mesma forma, AGBo (A > B) utiliza o complemento de B e AGBi.
- A saída de igualdade AEBo (A = B) é uma porta lógica AND que verifica se A é igual a B e se a entrada de cascata AEBi também é verdadeira.

3.3 Projeto do Fluxo de Dados

Através do RTLViewer, no Quartus, foi obtida a seguinte imagem:



As chaves controlam simultaneamente a carga do contador e a entrada “B” do comparador, enquanto a entrada “A” do comparador é a saída do contador. Os sinais de ZERA, CARREGA e CONTA são ativados em nível lógico alto, e o módulo é sensível à borda de subida do clock. ZERA funciona como um reset, ou seja, retorna o contador para 0, enquanto CARREGA copia o valor da carga para o contador. Conta faz com que o contador avance em 1 na próxima subida do clock. A saída do contador será comparada com sua carga, e o comparador indicará se A é maior, menor ou igual a B. Quando o contador estiver em 1111 (15 ou F), o sinal de fim será ativado (desde que CONTA também esteja).

4 Plano de Testes do Sistema e Simulações

4.1 Cenário de Teste 1 – Validação Individual do Contador:

Conforme solicitado no item (c) do roteiro, foi realizada a simulação isolada do componente contador_163. A tabela abaixo apresenta a comparação entre os resultados esperados e os obtidos na simulação.

Teste	Descrição	Entradas Relevantes	Saídas Esperadas	Resultado Simulado
C.I.	Condições Iniciais	CLR=1, LD=1, ENP=0, ENT=0	Q=0000, RCO=0	OK
1	Clock inativado	ENP=0, ENT=0, Clock x2	Q=0000, RCO=0	OK
2	Clear ativado	CLR=0, LD=1	Q=0000, RCO=0	OK
3	Enable ativado (5x)	ENP=1, ENT=1, Clock x5	Q=0101, RCO=0	OK
4	Load ativado (1011)	LD=0, D=1011	Q=1011, RCO=0	OK
5	Contar até 15	ENP=1, ENT=1, Clock x4	Q=1111, RCO=1	OK
6	Inibir ENP	ENP=0, ENT=1	Q=1111, RCO=1	OK
7	Inibir ENT	ENP=1, ENT=0	Q=1111, RCO=0	OK
8	Overflow (0 a 1)	ENP=1, ENT=1, Clock x2	Q=0001, RCO=0	OK
9	Prioridade Clear	CLR=0, LD=0	Q=0000, RCO=0	OK
10	Load Direto	LD=0, D=1001	Q=1001, RCO=0	OK
11	Contagem Final	ENP=1, ENT=1, Clock x6	Q=1111, RCO=1	OK

4.2 Cenário de Teste 2 – Validação Individual do Comparador:

Conforme o item (f), a tabela abaixo preenche as lacunas do plano de testes para o comparador_85.

Teste	Entradas (A vs B)	Cascata (ALBi, AGBi, AEBi)	Saídas Esperadas (ALBo, AGBo, AEBo)	Obs.
4	A > B (0001, 0000)	Menor (1, 0, 0)	0, 1, 0	Magnitude A>B prevalece
5	A > B (0001, 0000)	Maior (0, 1, 0)	0, 1, 0	Magnitude A>B prevalece
6	A > B (0001, 0000)	Igual (0, 0, 1)	0, 1, 0	Magnitude A>B prevalece
7	A < B (0011, 1100)	Igual (0, 0, 1)	1, 0, 0	Magnitude A<B prevalece
8	A < B (0011, 1100)	Maior (0, 1, 0)	1, 0, 0	Magnitude A<B prevalece
9	A < B (0011, 1100)	Menor (1, 0, 0)	1, 0, 0	Magnitude A<B prevalece

Os resultados obtidos na simulação são totalmente compatíveis com a análise teórica do código Verilog realizada na seção 3.2. Confirmou-se que as entradas de cascata (ALBi, AGBi, AEBi) só influenciam a saída quando as entradas principais A e B são idênticas. Nos testes 4 a 9, onde há diferença de magnitude entre A e B, o comparador ignora corretamente os sinais de cascata, validando a lógica aritmética implementada.

4.3 Cenário de Teste 3 – Integração Completa do Contador

Validar se o contador incrementa corretamente e se o comparador atualiza as saídas de magnitude conforme a contagem evolui.

Entradas (Zera, Carrega, Conta, Chaves)	Saída Esperada (Contagem, Maior, Igual, Menor)	Resultado Simulado OK?	Análise de Não Conformidades
ZERA=1, CONTA=0, D=0010	0000, MAIOR=0, IGUAL=0, MENOR=1	Sim	Nenhuma.
ZERA=1, CONTA=1, CLOCK↑	0001, MAIOR=0, IGUAL=0, MENOR=1	Sim	Operação síncrona validada.
ZERA=1, CONTA=1, CLOCK↑	0010, MAIOR=0, IGUAL=1, MENOR=0	Sim	Ponto de igualdade atingido.
ZERA=1, CONTA=1, CLOCK↑	0011, MAIOR=1, IGUAL=0, MENOR=0	Sim	Transição para "Maior" OK.

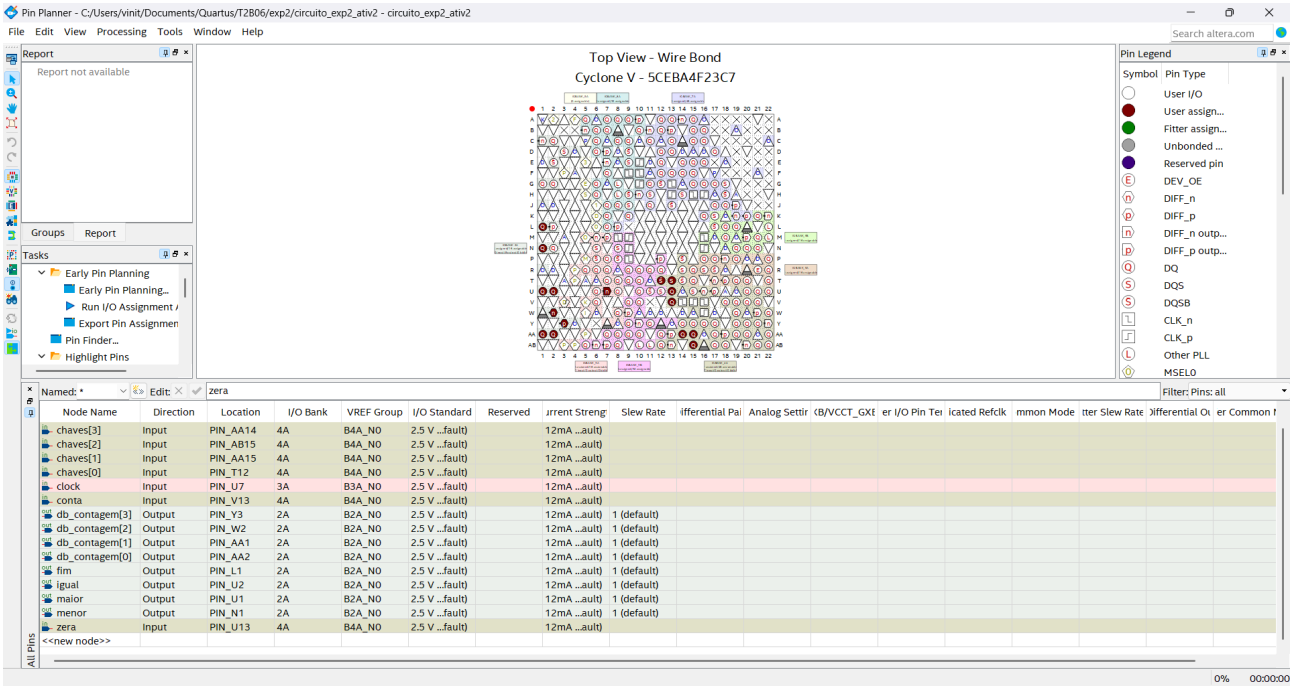
5 Implantação do Projeto

5.1 Pinagem da Placa FPGA

Ainda não foram vistos métodos para a escolha da pinagem, por isso ela foi feita seguindo a tabela disponibilizada na apostila da experiência, disposta a seguir:

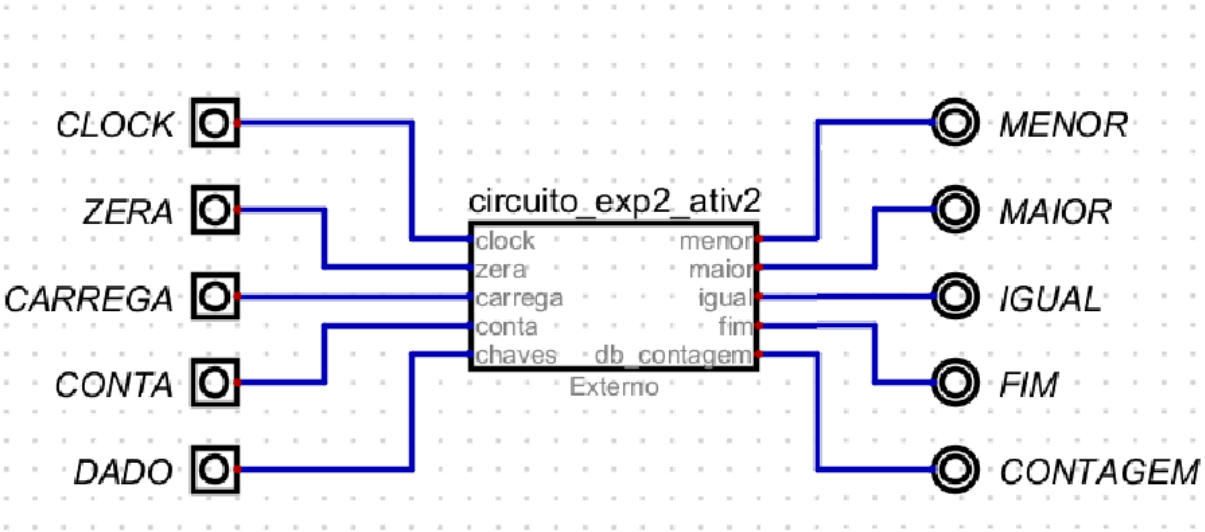
Sinal	Pino na Placa DE0-CV	Pino na FPGA
CLOCK	botão KEY0	PIN_U7
ZERA	chave SW0	PIN_U13
CONTA	chave SW1	PIN_V13
CARREGA	chave SW2	PIN_T13
CHAVES(0)	chave SW3	PIN_T12
CHAVES(1)	chave SW4	PIN_AA15
CHAVES(2)	chave SW5	PIN_AB15
CHAVES(3)	chave SW6	PIN_AA14
DB_CONTAGEM(0)	Led LEDR0	PIN_AA2
DB_CONTAGEM(1)	Led LEDR1	PIN_AA1
DB_CONTAGEM(2)	Led LEDR2	PIN_W2
DB_CONTAGEM(3)	Led LEDR3	PIN_Y3
MENOR	Led LEDR5	PIN_N1
IGUAL	Led LEDR6	PIN_U2
MAIOR	Led LEDR7	PIN_U1
FIM	Led LEDR9	PIN_L1

No Quartus, a pinagem final, seguindo a tabela, está representada abaixo:



5.2 Estratégia de Montagem

No Digital, foi feito um esquema simulando o circuito completo num componente Verilog. As entradas e saídas estão mostradas a seguir:



Para a programação da placa FPGA, serão necessários os códigos em verilog e o Quartus. Após a depuração (que foi, inicialmente, testada com êxito na etapa de planejamento), o resultado será transferido para a placa, ou seja, programando-a.

De modo a verificar o sucesso da programação, serão feitos os testes indicados na apostila (já detalhados em itens anteriores do planejamento).

5.3 Estratégia de Depuração

Nesta experiência, os códigos dos módulos do comparador e do contador foram dados, enquanto o fluxo de dados foi parcialmente fornecido, de forma que apenas pequenas alterações foram feitas (preenchimento de campos incompletos). O fato de os códigos em verilog estarem prontos ou próximos da finalização tornou o processo de depuração (dentro do Intel Quartus) rápido e simples, sem complicações.

6 Respostas das perguntas da apostila

Item h)

O código em Verilog a seguir descreve o circuito em estudo do experimento: A ligação de um contador com um comparador. O circuito possui como entradas de controle: clock, zera (que é reset), carrega (que é o load) e conta (que é o enable).

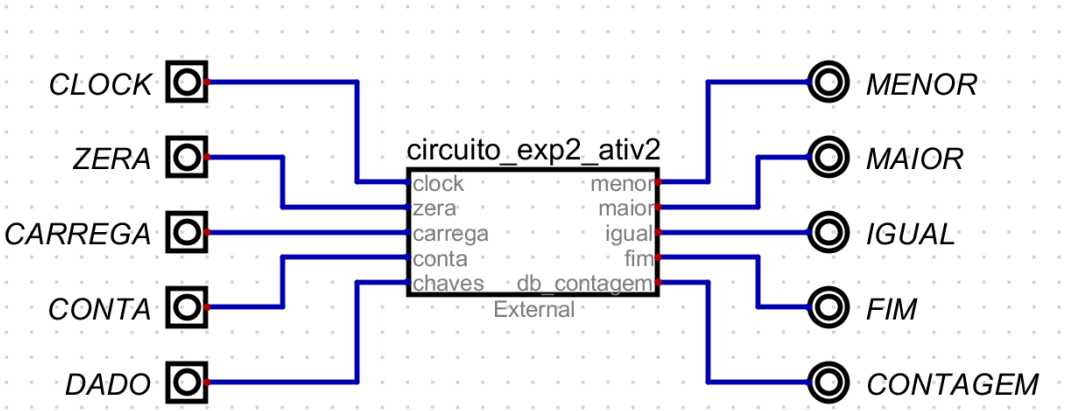
Descrevendo agora os dois componentes instanciados, temos primeiro o Contador (contador_163). Este bloco é uma instância de um contador binário síncrono de 4 bits. Quando o sinal carrega é ativado, a entrada ld recebe 0 (devido à inversão ~carrega), carregando o valor atual em chaves para dentro do contador. Se carrega estiver desativado e conta estiver ativo (1), o contador incrementa seu valor a cada pulso de clock. O valor atual é enviado para o fio interno s_contagem.

O outro componente no circuito é o Comparador (comparador_85). Este bloco é uma instância de um comparador de magnitude de 4 bits. Ele compara dois valores de 4 bits sendo A o valor atual do contador (s_contagem) e sendo B o valor das entradas externas (chaves). As saídas indicam se o valor do contador é menor, maior ou igual ao valor definido nas chaves.

```
/*
 * ---------------------------------------------------------------------
 *  Arquivo   : circuito_exp2_ativ2-PARCIAL.v
 *  Projeto   : Experiencia 2 - Um Fluxo de Dados Simples
 * ---------------------------------------------------------------------
 *  Descricao : Circuito PARCIAL do fluxo de dados da Atividade 2
 *
 *     1) COMPLETAR DESCRICAO
 *
 * ---------------------------------------------------------------------
 *  Revisoes  :
 *      Data        Versao  Autor             Descricao
 *      11/01/2024  1.0     Edson Midorikawa  versao inicial
 * ---------------------------------------------------------------------
 */

module circuito_exp2_ativ2 (clock, zera, carrega, conta, chaves,
                            menor, maior, igual, fim, db_contagem);
    input        clock;
    input        zera;
    input        carrega;
    input        conta;
```

```
    input  [3:0] chaves;
    output       menor;
    output       maior;
    output       igual;
    output       fim;
    output [3:0] db_contagem;

      wire    [3:0] s_contagem;   // sinal interno para interligacao dos
   componentes

    // contador_163
    contador_163 contador (
      .clock( clock ),
      .clr  ( ~zera ),
      .ld   ( ~carrega ),
      .ent  ( 1'b1 ),
      .enp  ( conta ),
      .D    ( chaves ),
      .Q    ( s_contagem ),
      .rco  ( fim )
    );

    // comparador_85
    comparador_85 comparador (
      .A   ( s_contagem ),
      .B   ( chaves ),
      .ALBi( 1'b0 ),
      .AGBi( 1'b0 ),
      .AEBi( 1'b1 ),
      .ALBo( menor ),
      .AGBo( maior ),
      .AEBo( igual )
    );

    // saida de depuracao
    assign db_contagem = s_contagem;

endmodule
```

i) A figura a seguir representa a software da atividade e logo depois segue também o código que sucede é o que foi inserido no software Digital para gerar a simulação.



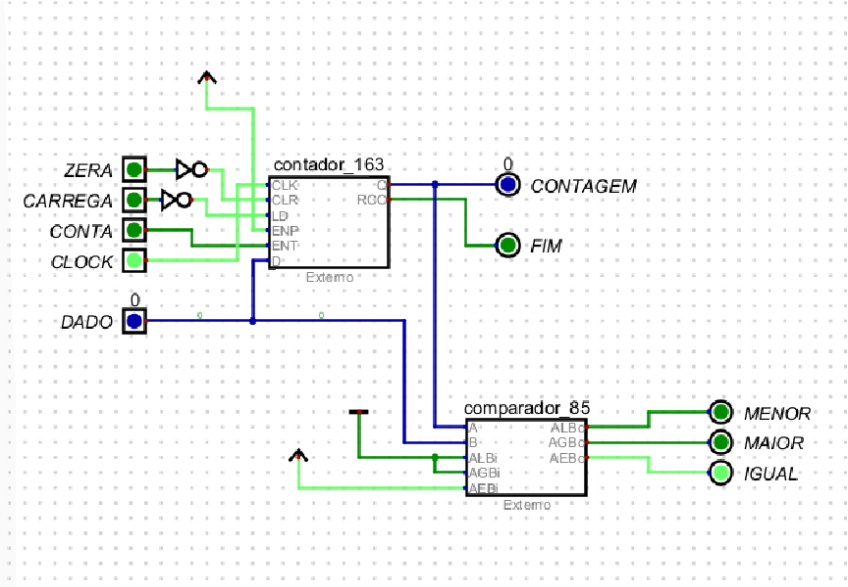

```
/*
 * ---------------------------------------------------------------------
 *  Arquivo    : circuito_exp2_ativ2-PARCIAL.v
 *  Projeto    : Experiencia 2 - Um Fluxo de Dados Simples
 * ---------------------------------------------------------------------
 *  Descricao : Circuito PARCIAL do fluxo de dados da Atividade 2
 *
 *      1) COMPLETAR DESCRICAO
 *
 * ---------------------------------------------------------------------
 *  Revisoes  :
 *      Data        Versao  Autor             Descricao
 *      11/01/2024  1.0     Edson Midorikawa  versao inicial
 * ---------------------------------------------------------------------
 */

module circuito_exp2_ativ2 (clock, zera, carrega, conta, chaves,
                            menor, maior, igual, fim, db_contagem);
    input        clock;
    input        zera;
    input        carrega;
    input        conta;
    input  [3:0] chaves;
    output       menor;
    output       maior;
    output       igual;
    output       fim;
    output [3:0] db_contagem;

      wire    [3:0] s_contagem;   // sinal interno para interligacao dos
   componentes

    // contador_163
```

```
    contador_163 contador (
      .clock( clock ),
      .clr  ( ~zera ),
      .ld   ( ~carrega ),
      .ent  ( 1'b1 ),
      .enp  ( conta ),
      .D    ( chaves ),
      .Q    ( s_contagem ),
      .rco  ( fim )
    );

    // comparador_85
    comparador_85 comparador (
      .A   ( s_contagem ),
      .B   ( chaves ),
      .ALBi( 1'b0 ),
      .AGBi( 1'b0 ),
      .AEBi( 1'b1 ),
      .ALBo( menor ),
      .AGBo( maior ),
      .AEBo( igual )
    );

    // saida de depuracao
    assign db_contagem = s_contagem;

 endmodule


//---------------------------------------------------------------------
// Arquivo   : contador_163.v
// Projeto   : Experiencia 2 - Um Fluxo de Dados Simples
//---------------------------------------------------------------------
// Descricao : Contador binario de 4 bits, modulo 16
//             similar ao componente 74163
//
// baseado no componente Vrcntr4u.v do livro Digital Design Principles
// and Practices, Fifth Edition, by John F. Wakerly
//---------------------------------------------------------------------
// Revisoes  :
//     Data        Versao  Autor             Descricao
//     14/12/2023  1.0     Edson Midorikawa  versao inicial
//---------------------------------------------------------------------
//
```

```
module contador_163 ( clock, clr, ld, ent, enp, D, Q, rco );
    input clock, clr, ld, ent, enp;
    input [3:0] D;
    output reg [3:0] Q;
    output reg rco;

    always @ (posedge clock)
        if (~clr)               Q <= 4'd0;
        else if (~ld)           Q <= D;
        else if (ent && enp)    Q <= Q + 1'b1;
        else                    Q <= Q;

    always @ (Q or ent)
        if (ent && (Q == 4'd15))   rco = 1;
        else                       rco = 0;
endmodule



/* -----------------------------------------------------------------
 *  Arquivo    : comparador_85.v
 *  Projeto    : Experiencia 2 - Um Fluxo de Dados Simples
 * -----------------------------------------------------------------
 * Descricao : comparador de magnitude de 4 bits
 *             similar ao CI 7485
 *             baseado em descricao comportamental disponivel em
 * https://web.eecs.umich.edu/~jhayes/iscas.restore/74L85b.v
 * -----------------------------------------------------------------
 * Revisoes  :
 *     Data        Versao  Autor             Descricao
 *     21/12/2023  1.0     Edson Midorikawa  criacao
 * -----------------------------------------------------------------
 */

module comparador_85 (ALBi, AGBi, AEBi, A, B, ALBo, AGBo, AEBo);

    input[3:0] A, B;
    input      ALBi, AGBi, AEBi;
    output     ALBo, AGBo, AEBo;
    wire[4:0]  CSL, CSG;

    assign CSL  = ~A + B + ALBi;
    assign ALBo = ~CSL[4];
    assign CSG  = A + ~B + AGBi;
```

```
    assign AGBo = ~CSG[4];
    assign AEBo = ((A == B) && AEBi);

endmodule /* comparador_85 */
```

j) A imagem a seguir é outra composição dos componentes, contador e comparador, utilizando o software Digital.



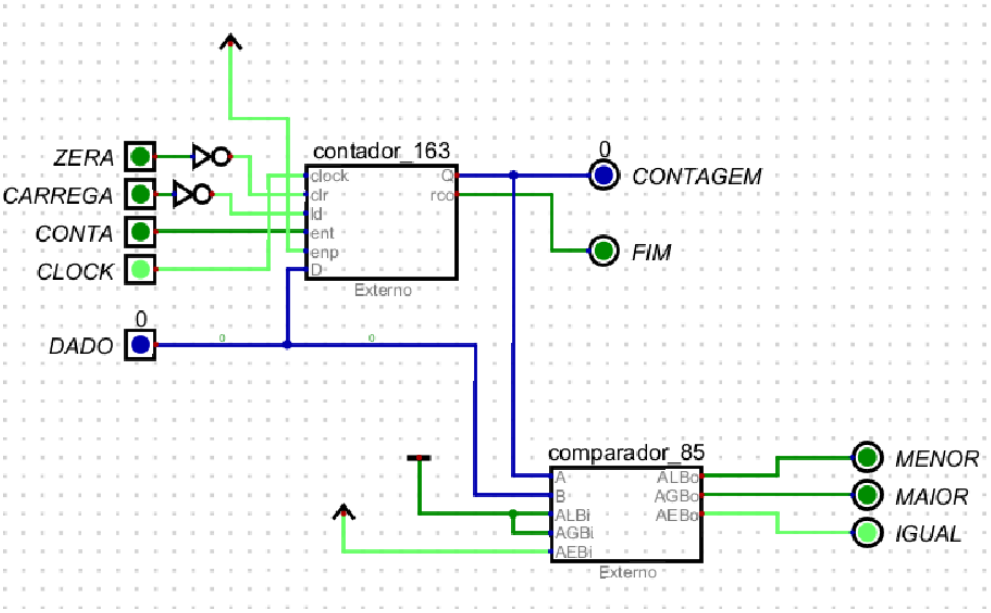
k) Nessa etapa, será feita uma série de testes para validar o projeto simulado no Digital para conseguir levá-lo para a FPGA sem falhas. A seguir estão os testes que serão realizados:

#	Caso de teste	Entradas de Dados e Sinais de controle	Resultado esperado	Resultado observado
c.i.	*Condições iniciais*	clock=1 zera=0 carrega=0 conta=0 chaves=0000	contagem=0, fim=0, maior=0, menor=0, igual=1	
1	*Zerar contador e observar a saída da contagem*	zera=1 clock ↑	contagem=0, fim=0, maior=0, menor=0, igual=1	
2	*Ajustar chaves para 0001*	chaves=0001	contagem=0, fim=0, maior=0, menor=1, igual=0	
3	*Incrementar contador e chaves=0001*	conta=1 clock ↑	contagem=1, fim=0, maior=0, menor=0, igual=1	
4	*Incrementar contador para 3 e ajustar chaves para 0010*	conta=1 clock ↑ (2x) chaves=0010	contagem=3, fim=0, maior=1, menor=0, igual=0	
5	*Ajustar chaves para 0110*	chaves=0110	contagem=3, fim=0, maior=0, menor=1, igual=0	
6	*Incrementar contador até 1110*	conta=1 clock ↑ (11x)	contagem=14, fim=0, maior=1, menor=0, igual=0	
7	*Incrementar contador*	conta=1 clock ↑	contagem=15, fim=1, maior=1, menor=0, igual=0	

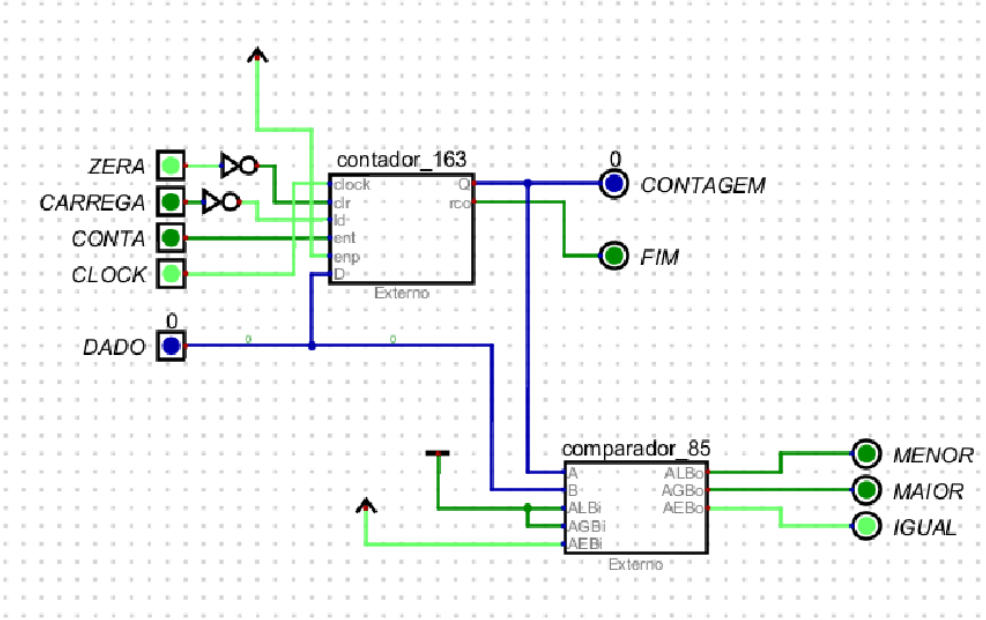
Observação: Ao fazer os testes, notou-se a inversão do sinal “ZERA”, que ativava em nível baixo. Após negar a entrada no código em verilog, o circuito passou a funcionar conforme pedido.

I) Agora serão feitos os testes elencados anteriormente um a um:

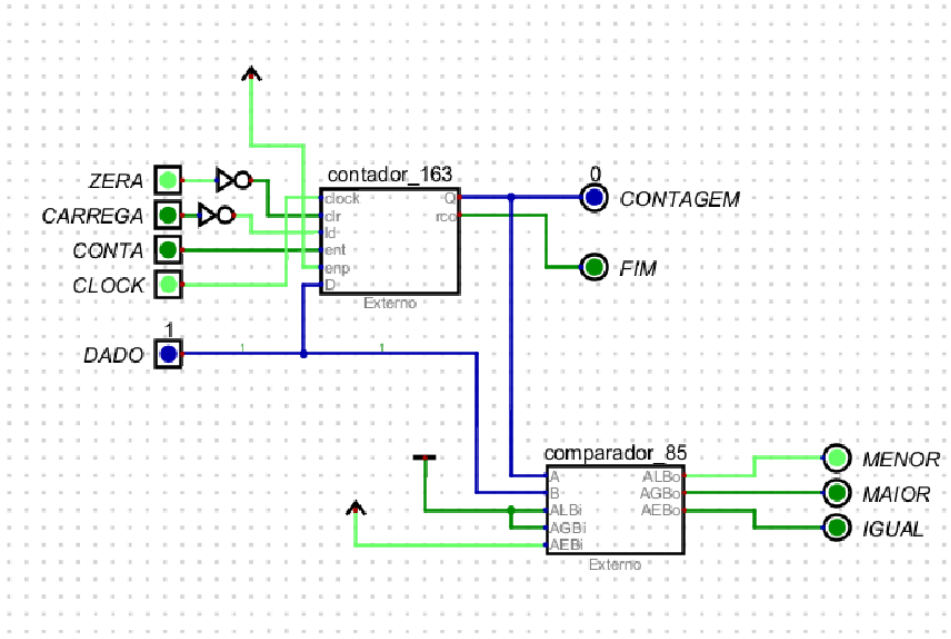
- C.i. - Resultado Observado: Esperado



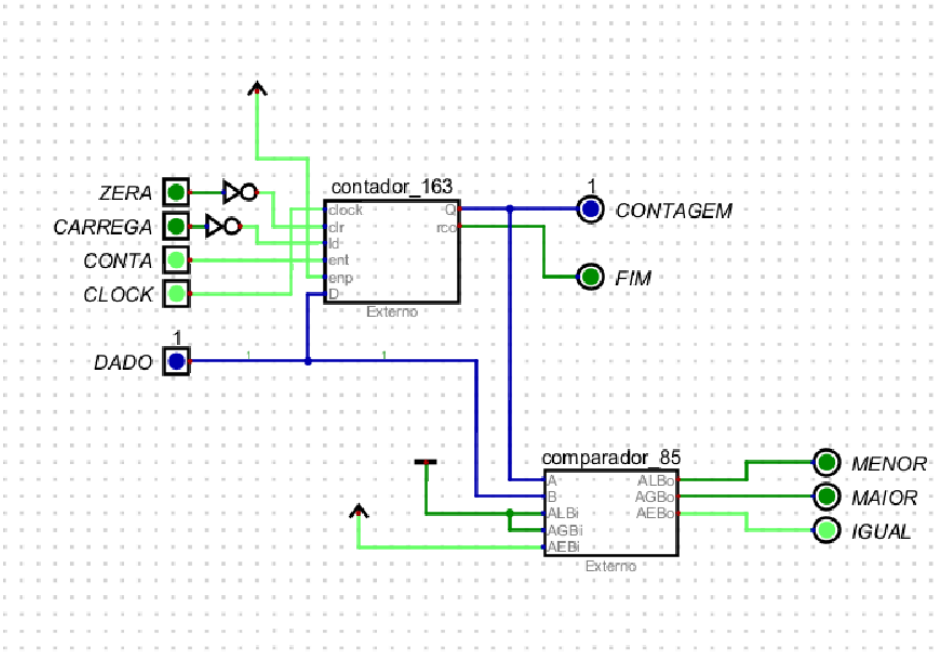
- 1. - Resultado Observado: Esperado



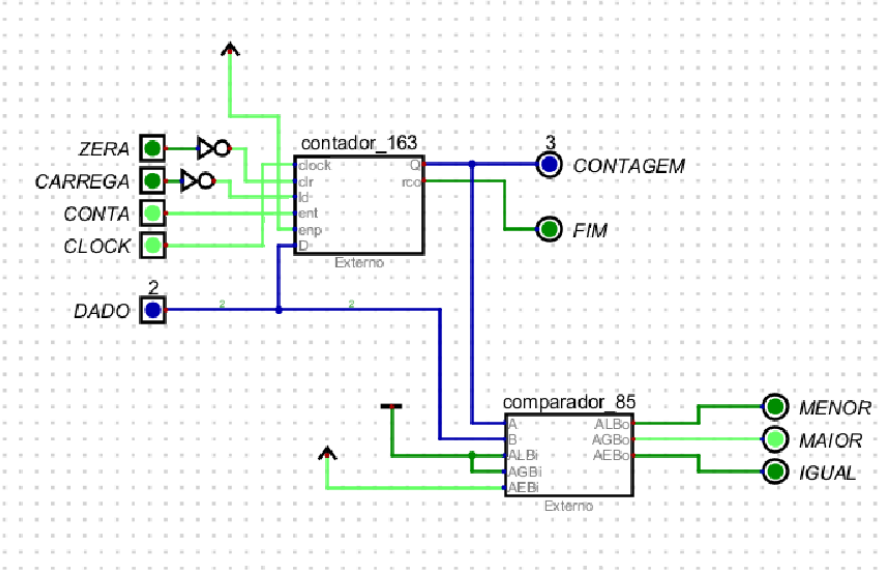
- 2. - Resultado Observado: Esperado



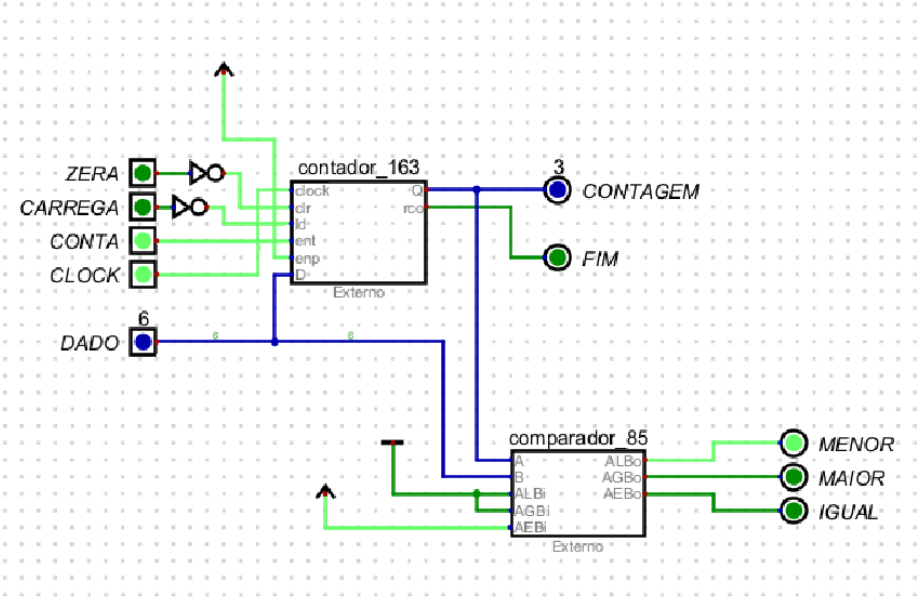
- 3. - Resultado Observado: Esperado



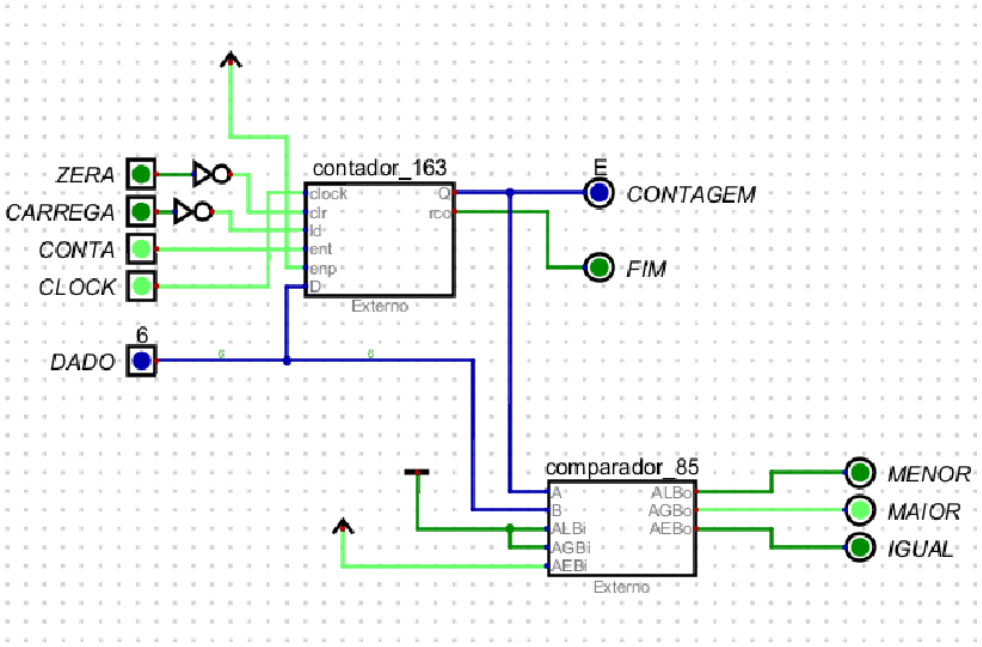
- 4. - Resultado Observado: Esperado



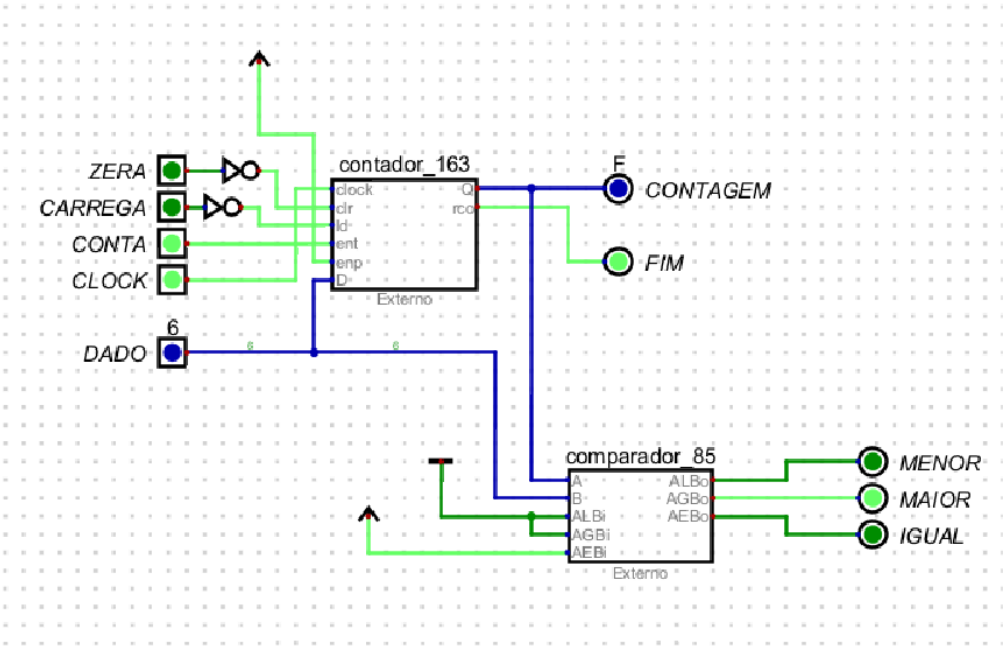
- 5. - Resultado Observado: Esperado



- 6. - Resultado Observado: Esperado



- 7. - Resultado Observado: Esperado



7 Projeto do Desafio da Experiência

7.1 Descrição do Desafio

O desafio consiste em mostrar a saída do contador num display de 7 segmentos, através do mapeamento das saídas do contador. Para isso, foi necessário fazer uma modificação no fluxo de dados.

7.2 Descrição do Projeto Lógico

7.2.1 Pinagem da Placa FPGA

A pinagem foi, novamente, realizada com base nos dados da apostila:

Sinal	Pino na Placa DE0-CV	Pino na FPGA
CLOCK	botão KEY0	PIN_U7
ZERA	chave SW0	PIN_U13
CONTA	chave SW1	PIN_V13
CARREGA	chave SW2	PIN_T13
CHAVES(0)	chave SW3	PIN_T12
CHAVES(1)	chave SW4	PIN_AA15
CHAVES(2)	chave SW5	PIN_AB15
CHAVES(3)	chave SW6	PIN_AA14
DB_CONTAGEM	display HEX0	consultar
MENOR	Led LEDR5	PIN_N1
IGUAL	Led LEDR6	PIN_U2
MAIOR	Led LEDR7	PIN_U1
FIM	Led LEDR9	PIN_L1

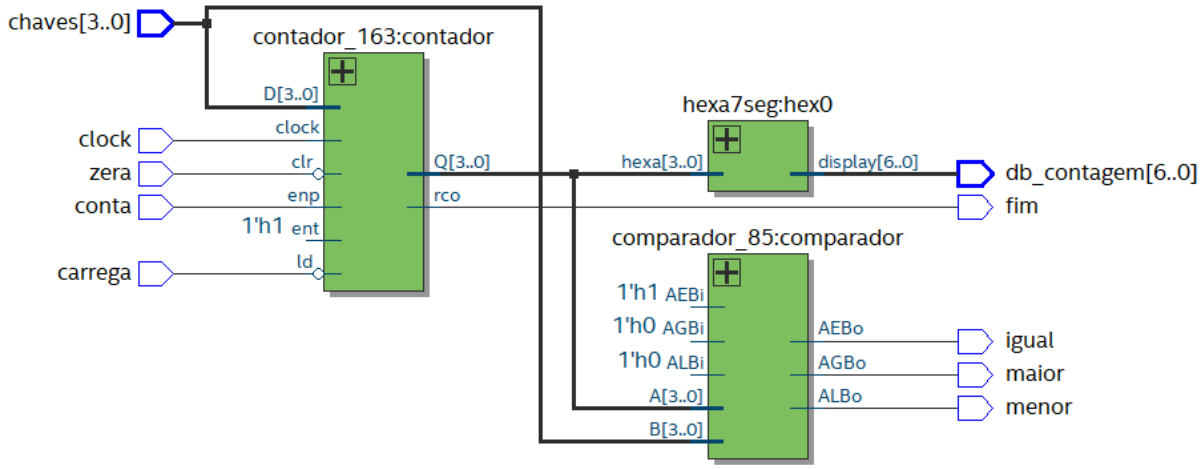
Desta vez, no entanto, foi necessário consultar a pinagem do display em outro documento, também disponibilizado no moodle:

Table 3-5 Pin Assignment of 7-segment Displays

Signal Name	*FPGA Pin No.*	*Description*
HEX00	**PIN_U21**	**Seven Segment Digit 0[0]**
HEX01	**PIN_V21**	**Seven Segment Digit 0[1]**
HEX02	**PIN_W22**	**Seven Segment Digit 0[2]**
HEX03	**PIN_W21**	**Seven Segment Digit 0[3]**
HEX04	**PIN_Y22**	**Seven Segment Digit 0[4]**
HEX05	**PIN_Y21**	**Seven Segment Digit 0[5]**
HEX06	**PIN_AA22**	**Seven Segment Digit 0[6]**

7.2.2 Alterações do Fluxo de Dados

Comparando a atividade com o desafio, a única parte que sofre mudança é o fluxo de dados, uma vez que os códigos em verilog do comparador e do contador permanecem os mesmos. Essa alteração se dá na adição do display que recebe a saída do contador e indica visualmente o número atual, ao invés de mostrar em bits. A imagem a seguir (retirada do Quartus pelo RTLViewer) mostra esse novo módulo:



Além disso, db_contagem agora tem 7 bits ao invés de 4. Isso ocorre porque o display utilizado possui 7 segmentos.

7.3 Verificação e Validação do Desafio

7.3.1 Cenário de Teste

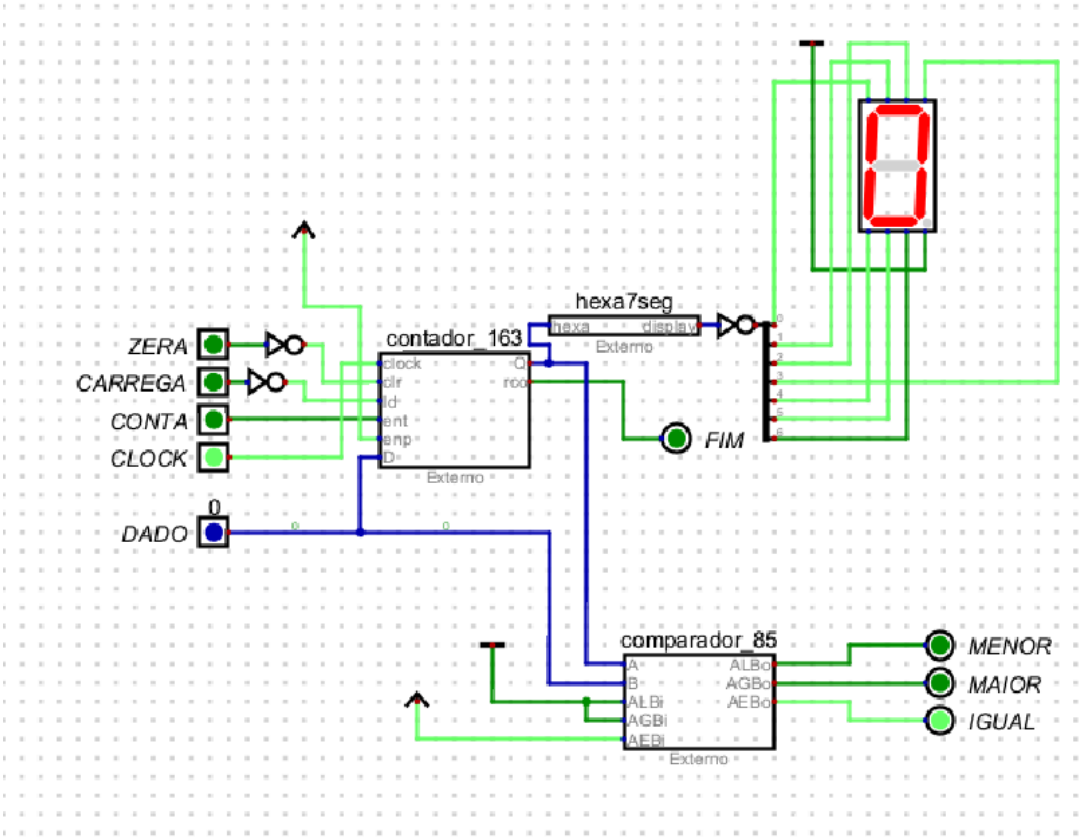
Foram feitos testes no Digital, seguindo novamente a tabela da apostila.

#	Caso de teste	Entradas de Dados e Sinais de controle	Resultado esperado	Resultado observado
c.i.	*Condições iniciais*	clock=1 zera=0 carrega=0 conta=0 chaves=0000	contagem=0, fim=0, maior=0, menor=0, igual=1	
1	*Zerar contador e observar a saída da contagem*	zera=1 clock ↑	contagem=0, fim=0, maior=0, menor=0, igual=1	
2	*Ajustar chaves para 0001*	chaves=0001	contagem=0, fim=0, maior=0, menor=1, igual=0	
3	*Incrementar contador e chaves=0001*	conta=1 clock ↑	contagem=1, fim=0, maior=0, menor=0, igual=1	
4	*Incrementar contador para 3 e ajustar chaves para 0010*	conta=1 clock ↑ (2x) chaves=0010	contagem=3, fim=0, maior=1, menor=0, igual=0	
5	*Ajustar chaves para 0110*	chaves=0110	contagem=3, fim=0, maior=0, menor=1, igual=0	
6	*Incrementar contador até 1110*	conta=1 clock ↑ (11x)	contagem=14, fim=0, maior=1, menor=0, igual=0	
7	*Incrementar contador*	conta=1 clock ↑	contagem=15, fim=1, maior=1, menor=0, igual=0	

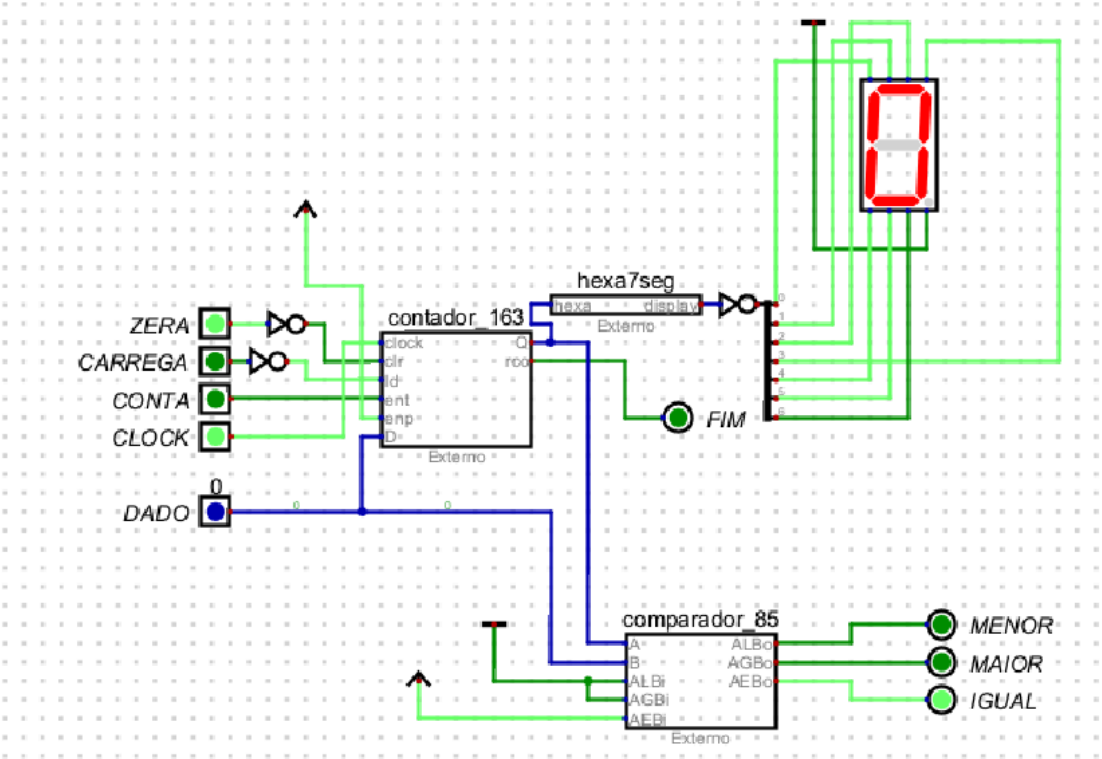
Inicialmente, notou-se que o display mostrava as saídas de maneira invertida, ou seja, os segmentos que deviam estar acesos estavam desligados e vice-versa. Para resolver isso, foi adicionado uma porta not (de 7 bits) para resolver o problema. Além disso, a última entrada do display foi aterrada porque representa o ponto do display, que não fará diferença nesse experimento.

Durante os testes, todas os resultados foram conforme esperado, comprovando seu funcionamento.

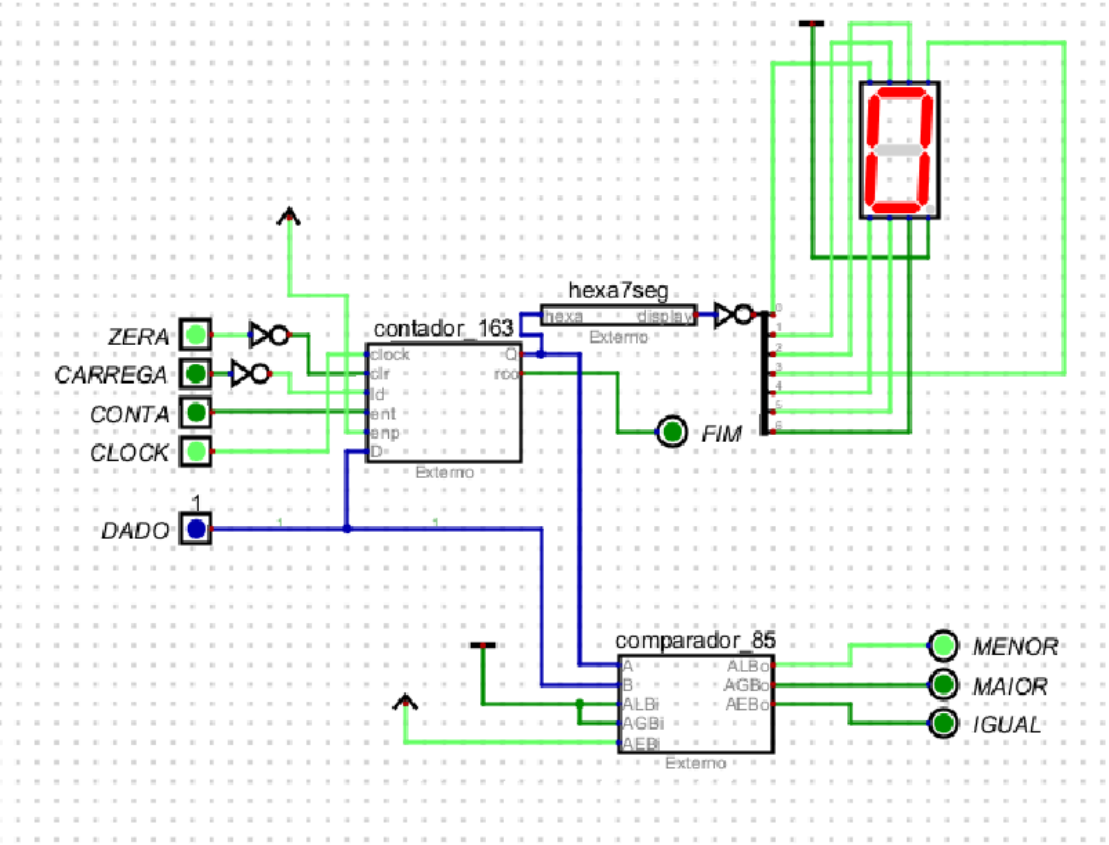
c.i. - Resultado Observado: Esperado



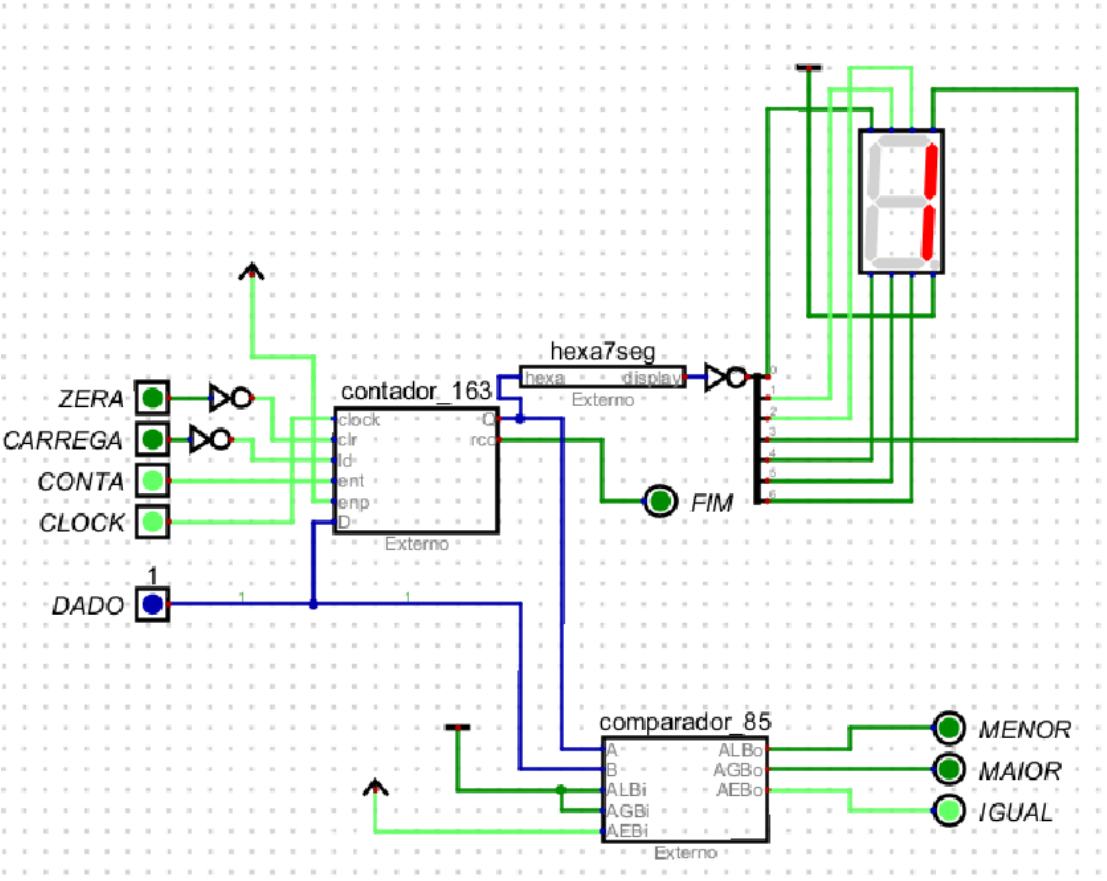
1. - Resultado Observado: Esperado



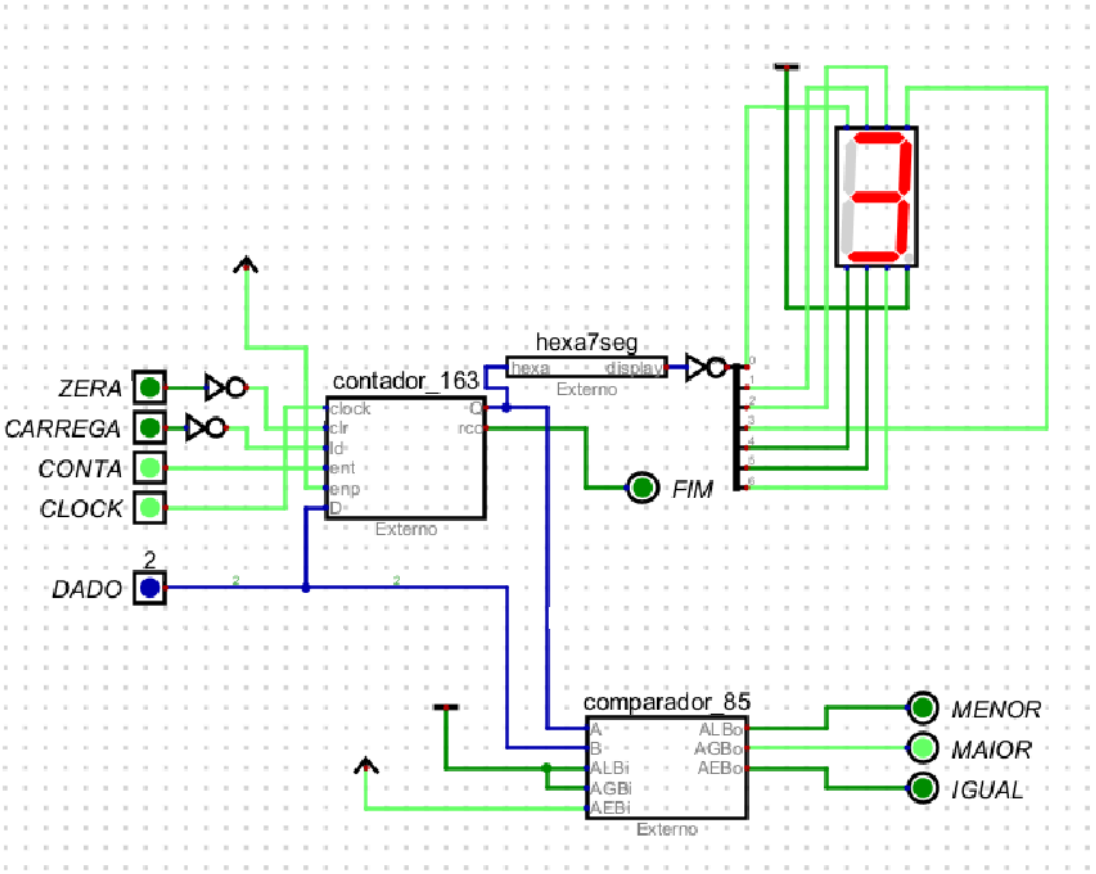
2. - Resultado Observado: Esperado



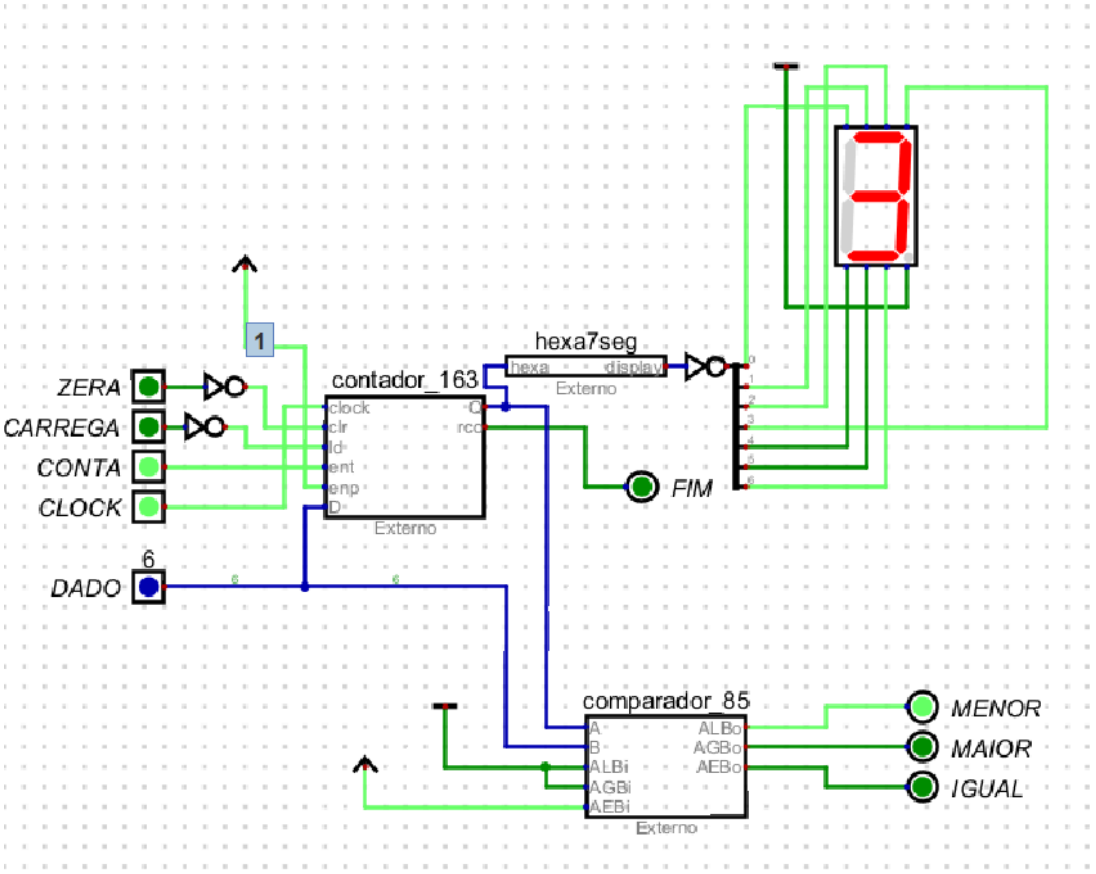
3. - Resultado Observado: Esperado



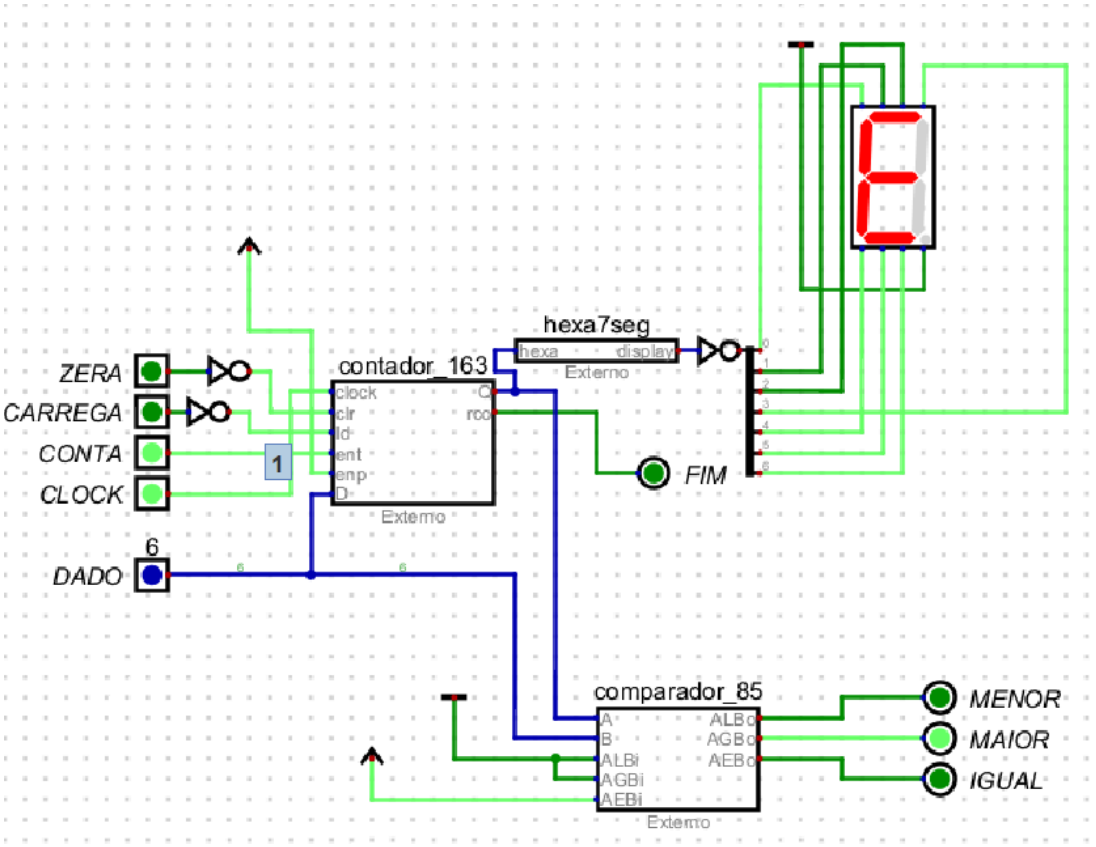
4. - Resultado Observado: Esperado



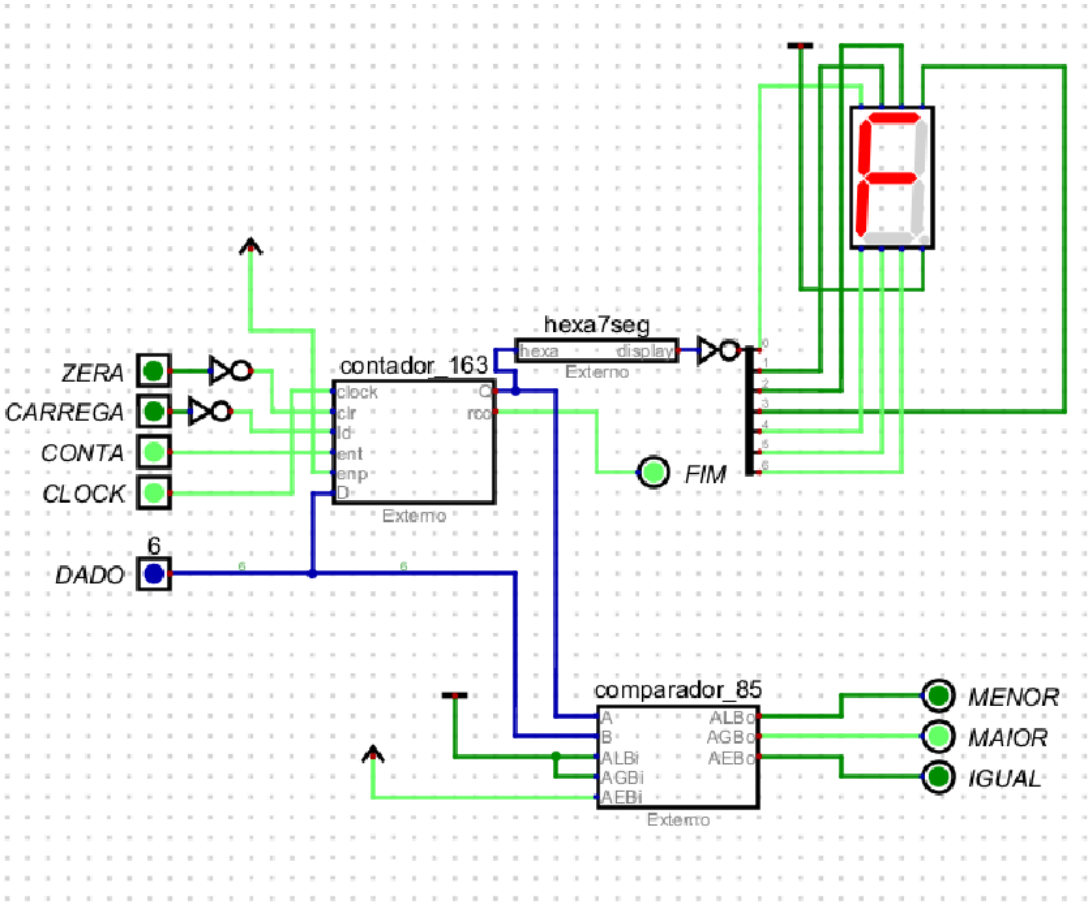
5. - Resultado Observado: Esperado



6. - Resultado Observado: Esperado



7. - Resultado Observado: Esperado



8 Conclusões

O planejamento demonstrou a robustez do fluxo de dados e a funcionalidade dos módulos contador_163 e comparador_85. Simulações no Digital confirmaram o comportamento esperado dos projetos Verilog. Ajustes preventivos na polaridade de controle (clr e ld) foram feitos para compatibilidade com a arquitetura TTL 74xx, minimizando falhas na implementação física.

A simulação no Digital confirmou que o comparador_85 opera conforme a análise lógica do Capítulo 3, com as saídas de magnitude (ALBo e AGBo) dependendo da diferença entre A e B, e os sinais de cascata atuando apenas na igualdade. Isso valida a implementação aritmética de comparação de magnitude. A próxima etapa é a síntese e teste físico na FPGA em laboratório. Este teste validará a lógica e a integração com o display, requisito central do desafio. Embora as simulações indiquem alta preparação, o sucesso final será atestado apenas após a programação e verificação do funcionamento em tempo real na FPGA.