

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**  
**CENTRO TECNOLÓGICO**  
**DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**  
**CURSO DE CIÊNCIAS DA COMPUTAÇÃO**  
**INE5406 - SISTEMAS DIGITAIS**

**PROJETO PRÁTICO DE SISTEMAS DIGITAIS:**

**Sistema digital que leia uma matriz  $N \times N$  e informe se ela é um quadrado mágico, em que a soma de cada linha, coluna ou diagonal é a mesma**

Área: Sistemas Digitais

Equipe:

Emanuelle Maria Botega Foscarini

Willian Kruscinski

Florianópolis

Novembro 2020

Equipe:

Emanuelle Maria Botega Foscarini

Willian Kruscinski

PROJETO PRÁTICO DE SISTEMAS DIGITAIS:

**Sistema digital que leia uma matriz  $N \times N$  e informe se ela é um quadrado mágico - em que a soma de cada linha, coluna ou diagonal é a mesma**

Trabalho da disciplina “INE5406 - Sistemas Digitais” apresentado ao Curso de Ciências da Computação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina.

Professor: Rafael Luiz Cancian, Dr. Eng.

Florianópolis Novembro/2020

# Lista de Figuras

- 2.1 Interface do sistema digital proposto.
- 2.1 FSMD
- 2.2 Circuito para atribuição do registrador cont4 recebendo 0
- 2.3 Circuito para atribuição do registrador cont10 recebendo 0
- 2.4 Circuito para atribuição do registrador magica recebendo 0
- 2.5 Circuito para atribuição do registrador end recebendo 0
- 2.6 Circuito para atribuição do registrador end recebendo 2
- 2.7 Circuito para atribuição do registrador conta\_loop recebendo 0
- 2.8 Circuito para atribuição do registrador acumulador recebendo 0
- 2.9 Circuito para atribuição do registrador end com soma
- 2.10 Circuito para atribuição do registrador cont10 com soma  $\text{conta10} += 1$
- 2.11 Circuito para atribuição do registrador conta\_loop com soma  $\text{conta\_loop} += 1$
- 2.12 Circuito para atribuição do registrador cont4 com soma  $\text{cont4} += 1$

# Sumário

<b>Introdução</b>	<b>5</b>
<b>Projeto do Sistema</b>	<b>6</b>
Identificação das Entradas e Saídas	6
Descrição e Captura do Comportamento e código	8
Registradores de dados	12
Operações	12
Fluxo de Controle	14
Dependência entre Dados	14
Projeto do Bloco Operativo	15
Circuitos COMBINACIONAIS para implementação das operações	15
Circuitos SEQUENCIAIS para implementação das operações	16
Elementos do bloco operativo	18
Projeto do Bloco de Controle	24
FSMD do bloco de controle	24
Diagrama de transição de estados	25
Diagrama de saídas	26
<b>Desenvolvimento</b>	<b>27</b>
Desenvolvimento do bloco operativo	27
Registrador (registrador)	27
Multiplexador 4:1 (mux_41)	27
Somador (adderNbits)	28
Comparador (comparador)	30
Bloco Operativo	31
Bloco Controle	34
Quadrado Mágico (topo)	38
Controle dos estados (Pkg_quadrado_magico_BC_Estado)	39
Memória RAM	40
Área	41
<b>Testes</b>	<b>42</b>
Registrador	42
Comparador	42



# 1. Introdução

Este projeto prático de sistemas digitais visa desenvolver um sistema digital que leia uma matriz  $N \times N$ , calculando a soma de suas linhas, colunas e diagonais, e analisando se ela é um quadrado mágico. Esse sistema digital será descrito em VHDL, sintetizado, simulando e prototipando em FPGA da Altera. Para a representação de números em ponto flutuantes será utilizado o padrão IEEE 754.

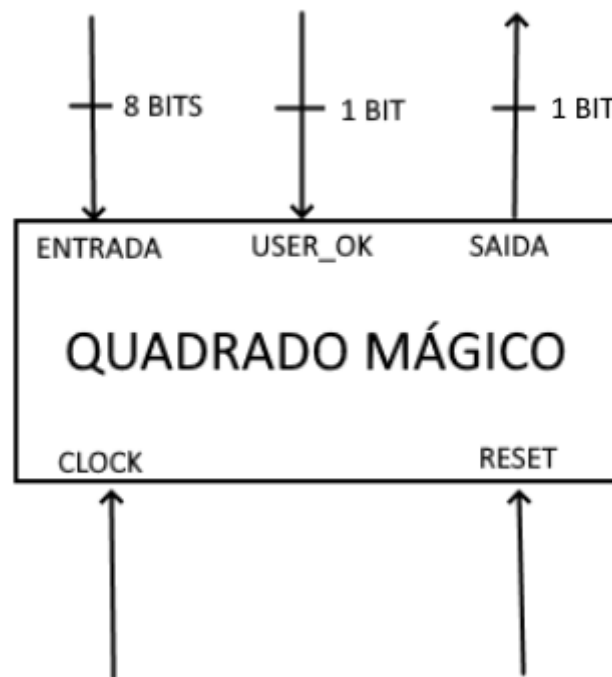
No sistema digital proposto o “usuário” poderá informar os dados de cada posição da matriz quadrada  $N \times N$  com um  $N$  previamente proposto, e igual a 3. Quando o cálculo da soma de todas as linhas, colunas e diagonais termina, é verificado se a matriz de entrada é um quadrado mágico, ou seja, essas somas resultam em valores equivalentes, e assim o resultado é apresentado e sinalizado.

## 2. Projeto do Sistema

O projeto do sistema digital inicia com a identificação das entradas e saídas e descrição e captura do comportamento do sistema, o que é realizado nas seções seguintes.

### 2.1 Identificação das Entradas e Saídas

Todas as entradas e saídas possuem 8 bits, por padronização, excluindo o USER\_OK que terá apenas um bit representado pelo Switch.



**Figura 2.1:** Interface do sistema digital proposto.

1) Entradas:

- a) O circuito contará com uma entrada que receberá 9 valores colocados pelo usuário, um por vez, que são armazenados.
- b) Além dessa, constará com a entrada chamado USER\_OK que o usuário irá ativar ao término de inserção de um dado para seguir ao próximo, que em código seria o 'init'.

2) Saída: Permite ao sistema digital externalizar seu estado atual:

- a) “Não mágico”: representando por ‘0’;
- b) “Mágico”: representando por ‘1’.

Além dos sinais descritos, o sistema digital proposto possui ainda um sinal de entrada clock, que corresponde ao sinal de relógio para sincronismo e o sinal de entrada reset, que corresponde a um sinal de reset assíncrono.



## 2.2 Descrição e Captura do Comportamento e código

Essa entrada de dados pode então ser executada pelo algoritmo computacional 2.1. Nele, já é inserido os dados nas linhas e colunas de uma matriz 3x3. As entradas só serão colocadas após o “init == 1” para que seja possível o usuário coloca-las uma por vez.

```
1  #ALGORITMO ATUALIZADO
2  #inicio do algoritmo
3  init = input('0/1: ')
4  if init == '1':
5      cont4      = 0
6      cont10     = 0
7      magica     = 0
8      acumulador = 0
9      end        = 0
10     conta_loop = 0
11     mem = [0]*9
12     #recebe entradas
13     while cont10 < 9:
14         mem[end] = int(input())
15         end = end + 1
16         cont10 = cont10 + 1
17     end        = 0
18     cont4 = 0
```

**Algoritmo 2.1:** Algoritmo de entrada de dados pelo usuário, inserindo-os na matriz.

Fonte: Própria

Após isso, verificamos a diagonal principal desta matriz, apresentada no algoritmo 2.2. É optado por comparar a soma de cada linha, coluna e diagonal secundária com a diagonal principal.

```
19      #calcula a diagonal principal e atribui ao parametro magica
20      while cont4 < 3:
21          magica = magica + mem[end]
22          end    = end + 4
23          cont4  = cont4 + 1
24      cont4 = 0
25      end    = 2
```

**Algoritmo 2.2:** Inserção dos dados com posições iguais, na diagonal principal.

Assim, o funcionamento completo do sistema digital proposto pode ser descrito pelo algoritmo 2.3. Nesse algoritmo, podemos verificar que foram equiparados e testados todos os outros valores de soma das linhas, colunas e diagonal secundária, com a soma dos valores da diagonal principal, se caso um dos resultados dessas somas já for diferente do resultado dessa diagonal, o código resulta em não ser um quadrado mágico.

E assim é sendo testado e comparado:

- 1- Diagonal principal com diagonal secundária, caso a soma seja igual o algoritmo segue;
- 2- Diagonal principal com as linhas, caso a soma seja igual o algoritmo segue;
- 3- Diagonal principal com as colunas, resultado.

```

26     #calcula a diagonal secundaria e compara ao parametro magica
27     while cont4 < 3:
28         acumulador = acumulador + mem[end]
29         cont4 = cont4 + 1
30         end = end + 2
31     if acumulador != magica:
32         print('Não magico') #representado por 0011
33         exit()
34     end = 0
35     acumulador = 0
36     cont4 = 0
37     #calcula as linhas e compara ao parametro magica
38     while conta_loop < 3:
39         while cont4 < 3:
40             acumulador = acumulador + mem[end]
41             cont4 = cont4 + 1
42             end = end + 1
43         if cont4 == 3 and acumulador != magica:
44             print('Não magico') #representado por 0011
45             exit()
46         cont4 = 0
47         conta_loop = conta_loop + 1
48         acumulador = 0
49     cont4 = 0
50     conta_loop = 0
51     acumulador = 0
52     #soma as colunas e compara ao parametro magica
53     while conta_loop < 3:
54         end = conta_loop
55         while cont4 < 3:
56             acumulador = acumulador + mem[end]
57             cont4 = cont4 + 1
58             end = end + 3
59         if cont4 == 3 and acumulador != magica:
60             print('Não magico') #representado por 0011
61             exit()
62         cont4 = 0
63         conta_loop = conta_loop + 1
64         if conta_loop == 3 and acumulador == magica:
65             print('É magico') #representado por 1111
66         acumulador = 0
67     fim do algoritmo

```

**Algoritmo 2.3:** Descrição completa do funcionamento do sistema digital para cálculo do quadrado mágico.

Visando obter plena confiança no algoritmo que descreve o comportamento do sistema, o mesmo foi implementado em linguagem de programação Python, visto ser uma linguagem já de fácil compreensão e pleno conhecimento pelos autores do trabalho.

Analisando o algoritmo 2.3 podemos verificar se há a necessidade da existência de registradores de dados para armazenar variáveis internas, podemos extrair as operações aritméticas, lógicas, relacionais e de transferência que são necessárias, bem como o fluxo de controle de execução desse comportamento e ainda dependências entre dados que nos permitem identificar as operações que precisam ser sequenciadas e as operações que podem ser realizadas em paralelo.

Portanto, com base neste algoritmo, podemos identificar os elementos abaixo, que farão parte do projeto do bloco de controle e do bloco operativo deste sistema digital.

### **Registradores de dados**

Os registradores necessários podem ser extraídos observando as variáveis internas. Nesse sistema digital, são necessários pelo menos os seguintes registradores:

(1)	<b>registrador(reset/carga):</b>	<b>cont4</b>
(2)	<b>registrador(reset/carga):</b>	<b>cont10</b>
(3)	<b>registrador(reset/carga):</b>	<b>magica</b>
(4)	<b>registrador(reset/carga):</b>	<b>acumulador</b>
(5)	<b>registrador(reset/carga):</b>	<b>end</b>
(6)	<b>registrador(reset/carga):</b>	<b>conta_loop</b>

### **Operações**

As operações aritméticas, lógicas, relacionais e de transferência podem ser extraídas observando diretamente as operações realizadas pelo algoritmo e também seus operandos. Nesse sistema digital são necessárias as seguintes operações, ordenado por variável tipo do sinal (comando ou status) e depois por variável:

Comandos:

(7)	<b>init</b>	<b>= input()</b>	<b>==&gt; cmdSetInit</b>
(8)	<b>cont4</b>	<b>= 0</b>	<b>==&gt; cmdResetCont4</b>
(9)	<b>cont10</b>	<b>= 0</b>	<b>==&gt; cmdResetCont10</b>
(10)	<b>magica</b>	<b>= 0</b>	<b>==&gt; cmdResetMagica</b>
(11)	<b>acumulador</b>	<b>= 0</b>	<b>==&gt; cmdResetAcumulador</b>
(12)	<b>end</b>	<b>= 0</b>	<b>==&gt; cmdResetEnd</b>
(13)	<b>conta_loop</b>	<b>= 0</b>	<b>==&gt; cmdResetContaLoop</b>
(14)	<b>output(não magico)</b>		<b>==&gt; cmdSelectOutput=0</b>
(15)	<b>output(magico)</b>		<b>==&gt; cmdSelectOutput=1</b>

Status:

<b>-- user_ok</b>	<b>= 1</b>	<b>==&gt; sttGo</b>
<b>-- cont10</b>	<b>&lt; 10</b>	<b>==&gt; sttCont10</b>
<b>-- cont4</b>	<b>&lt; 3</b>	<b>==&gt; sttCont4</b>
<b>-- acumulador</b>	<b>== magica</b>	<b>==&gt; sttMatch</b>
<b>-- conta_loop</b>	<b>== 3</b>	<b>==&gt; sttConta_loop</b>

As operações listadas, após as devidas decisões de projeto sobre compartilhamento ou não de recursos, serão associadas a elementos de hardware no bloco operativo.

### **Fluxo de Controle**

O fluxo de controle pode ser extraído a partir das próprias estruturas de controle de fluxo do algoritmo (como if-the-else, switch-case, for, do-while, sub-rotinas, etc), além do controle de início de término de algumas operações que podem levar de um pulso de clock. Assim, o fluxo de controle principal do sistema consiste em:

- (1) **Inicialização variáveis:** Necessária em praticamente todos os sistemas digitais e explícita nas linhas 2, 6, 7 ,8, 9, 10 e 11.
- (2) **Espera pelo comando:** Necessário para o controle da linha 4, em que o sistema fica “parado” até que um comando tenha sido enviado.
- (3) **Controle de final de laço:** Necessário para o controle das linhas 13, 30, 37, 38 e 39, em que o fluxo de execução pode voltar ou continuar, conforme a condição especificada.

O fluxo de controle é função do bloco de controle, que pode ser implementado como uma FSM ou outra alternativa (microcódigo, ROM, etc). Ressalta-se que o fluxo de controle pode requerer sinais de status providos do bloco operativo que forneçam informações sobre a “direção” desse fluxo.

### **Dependência entre Dados**

A dependência entre dados exige uma análise mais cuidadosa do algoritmo. Em geral, a natureza sequencial de um algoritmo já ajuda a identificar quais dados dependem de outros, devido à ordem de execução dos comandos.

- (1) A única variável que possui uma relação de dependência é a ‘magica’ que, depende dos resultados dos estados anteriores para setar um valor.

Com todos esses elementos extraídos diretamente a partir do algoritmo que descreve o comportamento do sistema digital, podemos capturar o comportamento do sistema digital em uma máquina de estados de alto nível (FSMD). A figura 2.2 apresenta a FSMD do sistema digital proposto.

## 2.3 Projeto do Bloco Operativo

A partir dos registradores de dados e das operações identificados com o algoritmo do comportamento do sistema, iniciamos o projeto do bloco operativo.

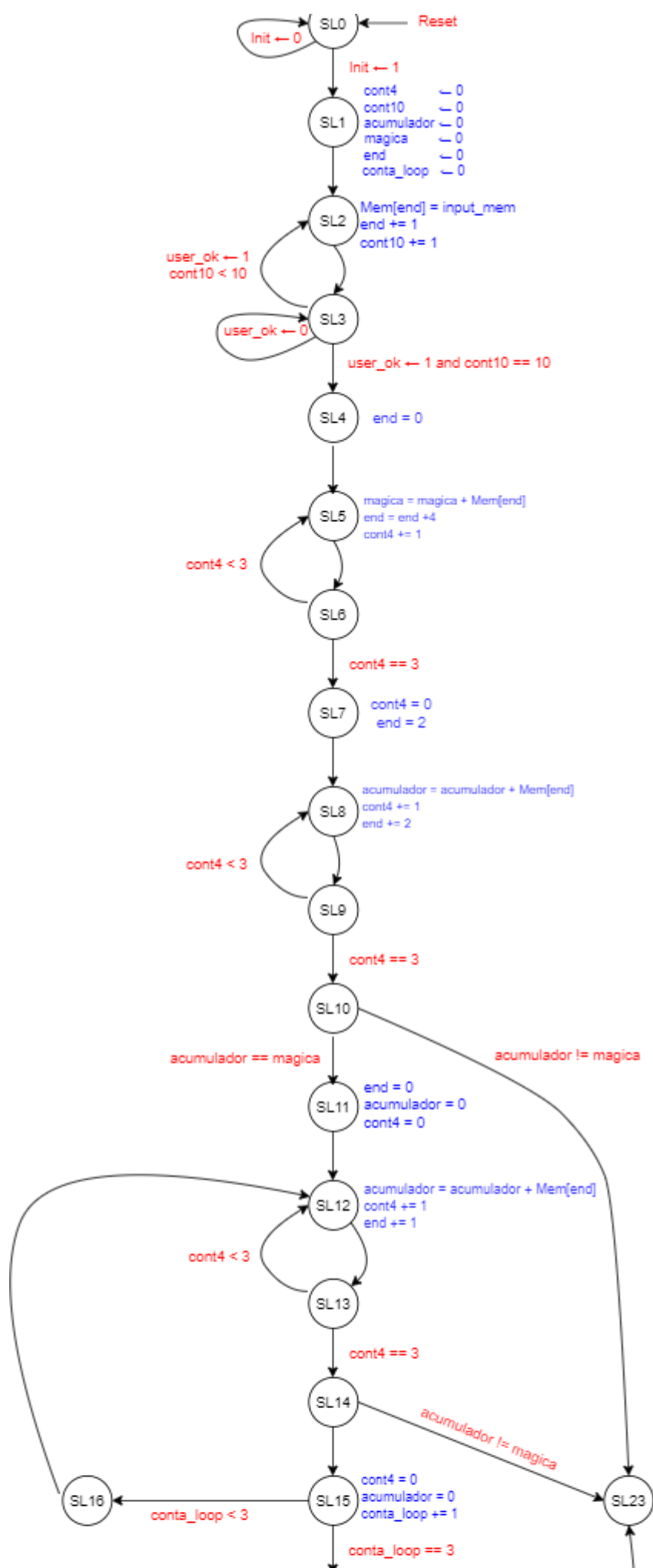
### Circuitos COMBINACIONAIS para implementação das operações

- -- incrementador:             $\text{cont10} = \text{cont10} + 1$              $\Rightarrow \text{cmdSetCont10}$
- -- comparador menor:     $\text{cont10} < 10$              $\Rightarrow \text{sttCont10}$
- -- somador:             $\text{end} = \text{end} + 4$              $\Rightarrow \text{cmdSetMuxEndSoma}$
- -- somador:             $\text{end} = \text{end} + 3$              $\Rightarrow \text{cmdSetMuxEndSoma}$
- -- somador:             $\text{end} = \text{end} + 2$              $\Rightarrow \text{cmdSetMuxEndSoma}$
- -- somador:             $\text{end} = \text{end} + 1$              $\Rightarrow \text{cmdSetMuxEndSoma}$
- -- somador:             $\text{magica} = \text{magica} + \text{Mem}[\text{end}]$              $\Rightarrow \text{cmdSetMagica}$
- -- comparador menor:     $\text{cont4} < 3$              $\Rightarrow \text{sttCont4}$
- -- incrementador:         $\text{cont4} = \text{cont4} + 1$              $\Rightarrow \text{cmdSetCont4}$
- -- comparador igualdade:  $\text{acumulador} = \text{magica}$              $\Rightarrow \text{sttMatch}$
- -- somador:             $\text{acumulador} + \text{Mem}[\text{end}]$              $\Rightarrow \text{cmdSetAcumulador}$
- -- incrementador:         $\text{conta\_loop} = \text{conta\_loop} + 1$              $\Rightarrow \text{cmdSetConta\_Loop}$
- -- comparador menor:     $\text{conta\_loop} < 3$              $\Rightarrow \text{sttConta\_loop}$
- -- multiplexador:         $\text{seleciona end} = \text{conta\_loop ou end} = 2$   
 $\Rightarrow \text{cmdSetMuxEnd}$
- -- multiplexador:        selecionar output             $\Rightarrow \text{sttSaida}$
- -- multiplexador:        selecionar end             $\Rightarrow \text{cmdSetMuxEndSoma}$



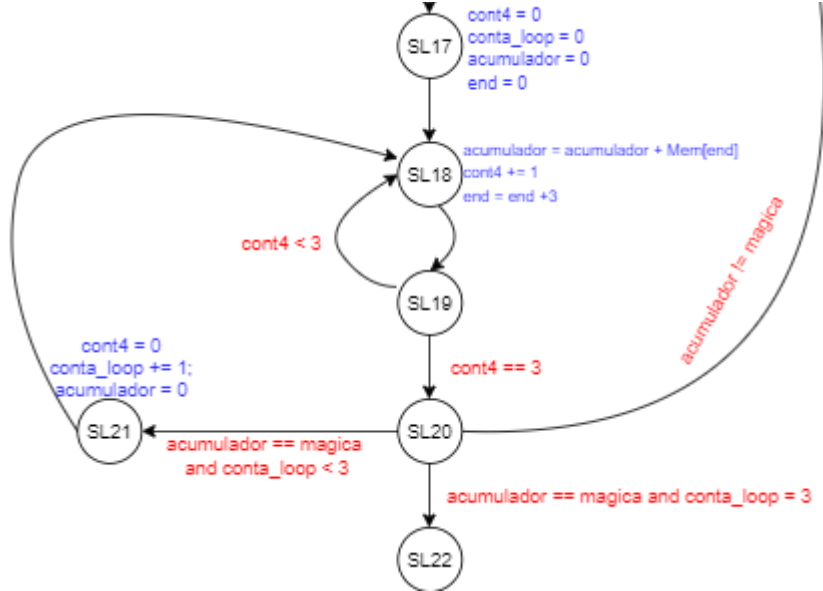
### **Circuitos SEQUENCIAIS para implementação das operações**

- -- registrador(reset):        cont4 = 0                ==> cmdResetCont4
- -- registrador(carga):        cont4 += 1            ==> cmdSetCont4
- -- registrador(reset):        cont10 = 0           ==> cmdResetCont10
- -- registrador(carga):        cont10 += 1           ==> cmdSetCont10
- -- registrador(reset):        magica = 0            ==> cmdResetMagica
- -- registrador(carga):        magica += Mem[end] ==> cmdSetMagica
- -- registrador(reset):        acumulador = 0        ==> cmdResetAcumulador
- -- registrador(carga):        acumulador += Mem[end] ==> cmdSetAcumulador
- -- registrador(reset):        conta\_loop = 0        ==> cmdResetConta\_Loop
- -- registrador(carga):        conta\_loop += 1        ==> cmdSetConta\_Loop
- -- MemoriaRam
- -- (write): mem[end]int(input())                ==> cmdSetMemEnd
- -- (read): mem[end]                                ==> sttMemEnd



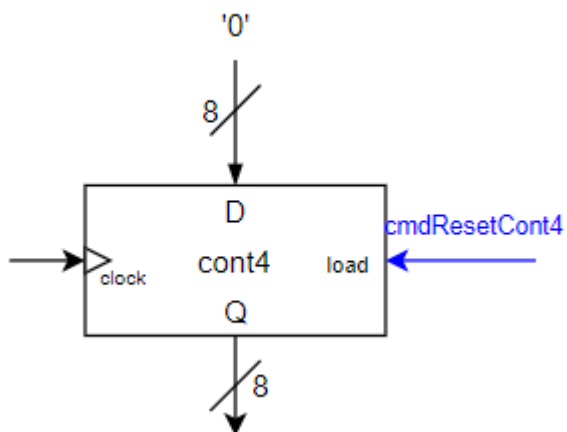
**Figura 2.2:** FSMD do sistema digital – O comportamento do sistema digital, previamente representado em algoritmo, foi capturado numa FSM de alto nível, que já considera os registradores, operações, fluxo de controle e dependências identificados.

Fonte: Própria



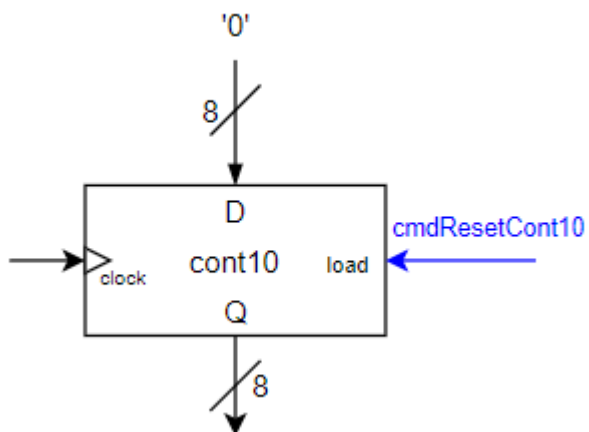
## Elementos do bloco operativo

1. `cont4 = 0`



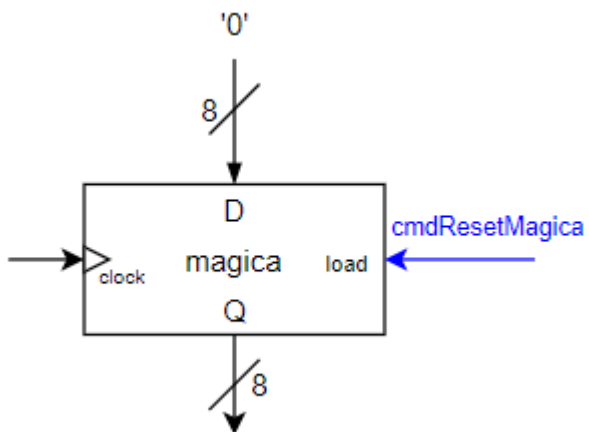
**Figura 2.2:** Circuito para atribuição do registrador `cont4` recebendo 0

2. `cont10 = 0`; (linha 49)



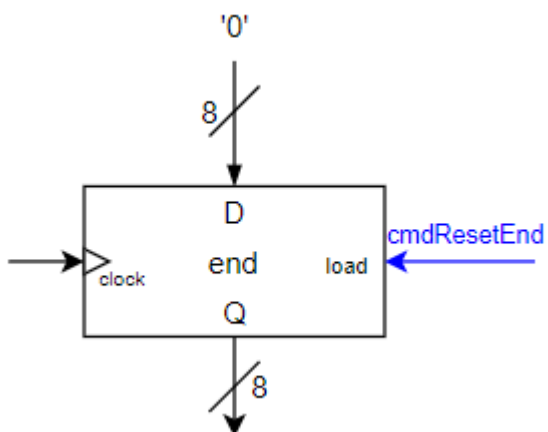
**Figura 2.3:** Circuito para atribuição do registrador `cont10` recebendo 0

3.  $\text{magica} = 0$



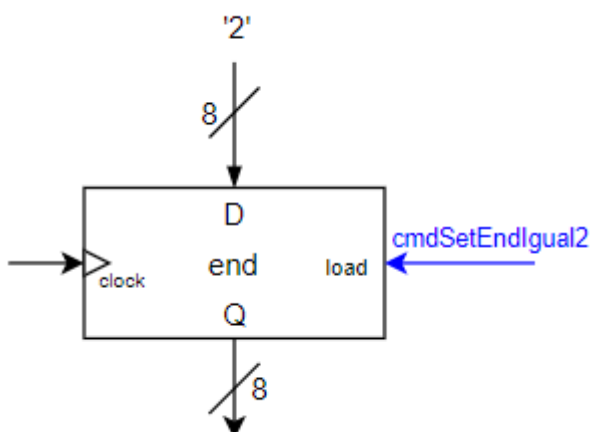
**Figura 2.4:** Circuito para atribuição do registrador *magica* recebendo 0

4.  $\text{end} = 0$



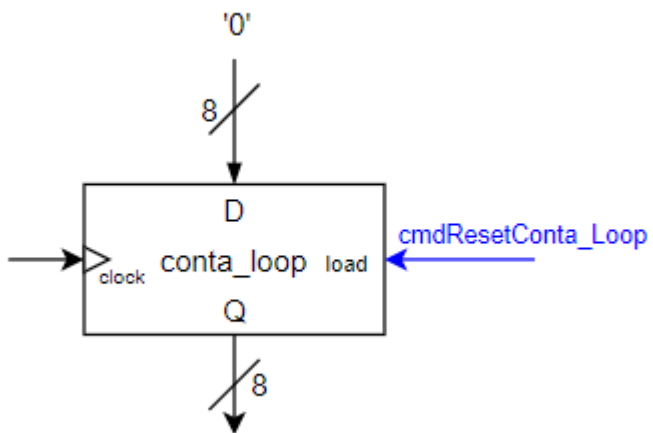
**Figura 2.5:** Circuito para atribuição do registrador *end* recebendo 0

5.  $\text{end} = 2$



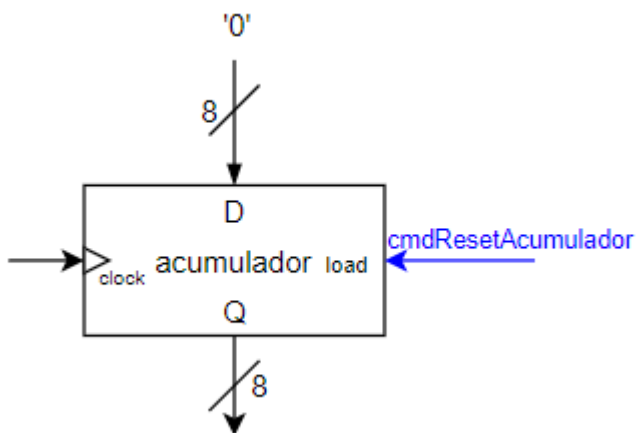
**Figura 2.6:** Circuito para atribuição do registrador *end* recebendo 2

6. `conta_loop = 0`



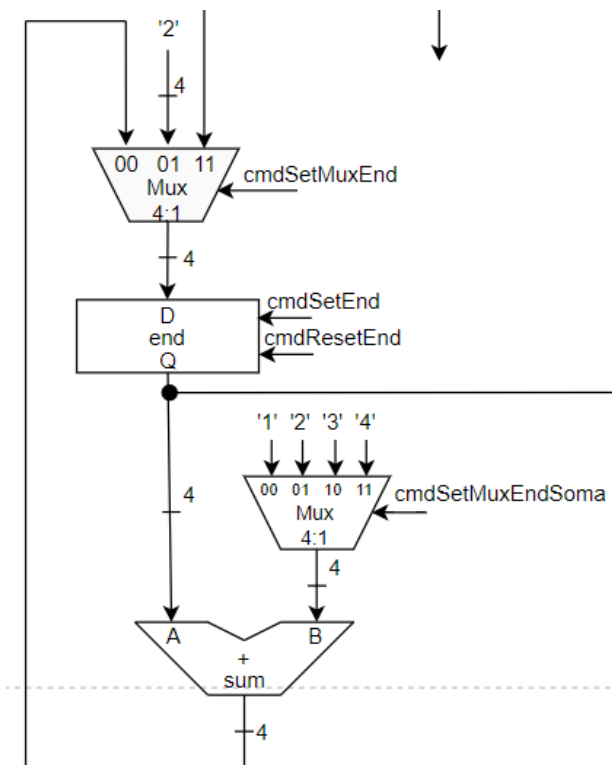
**Figura 2.7:** Circuito para atribuição do registrador `conta_loop` recebendo 0

7. `acumulador = 0`



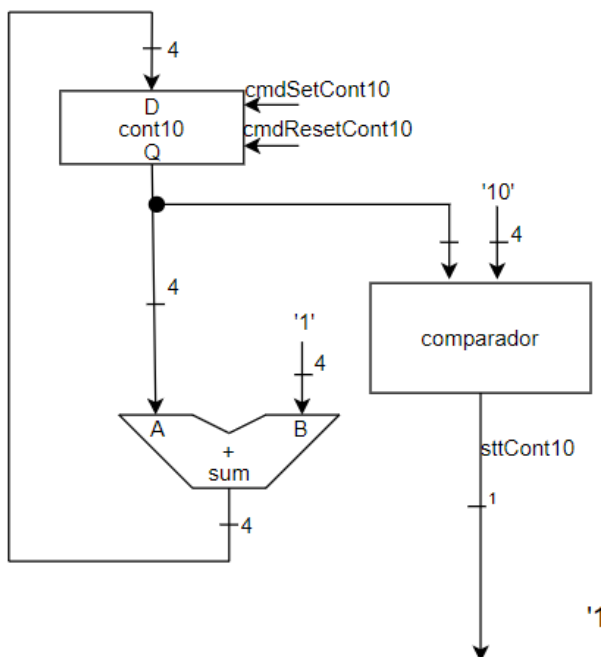
**Figura 2.8:** Circuito para atribuição do registrador `acumulador` recebendo 0

8.  $end += 1$ ;  $end += 2$ ;  $end += 3$  e  $end += 4$



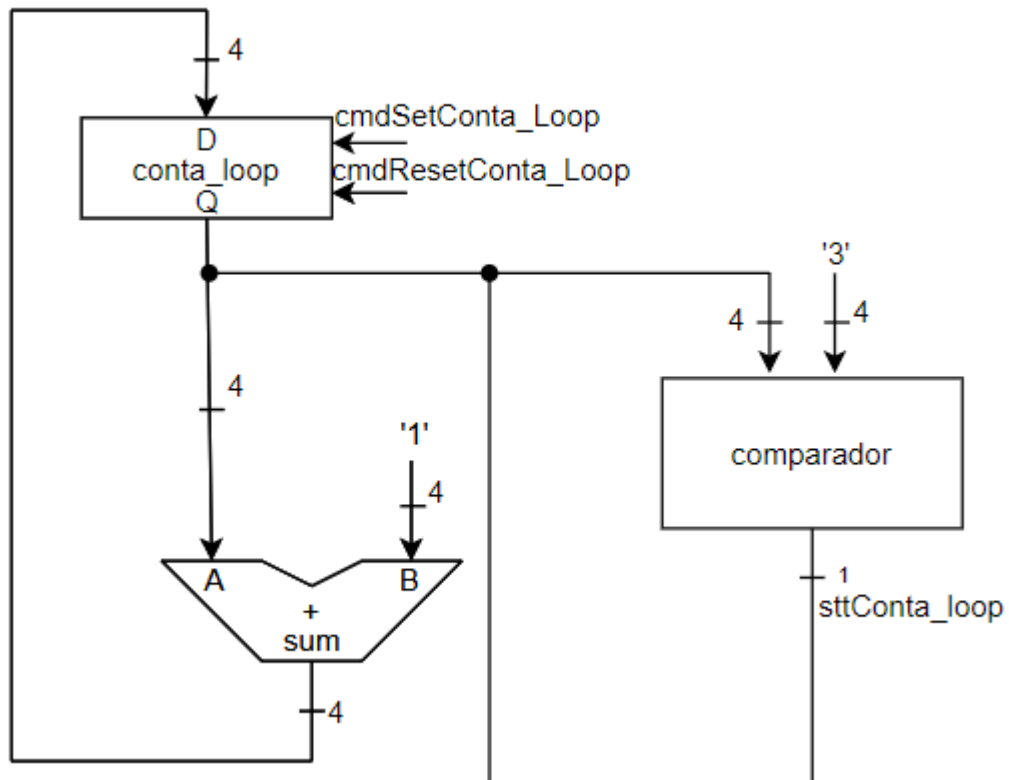
**Figura 2.9:** Circuito para atribuição do registrador end com soma.

9.  $cont10 += 1$



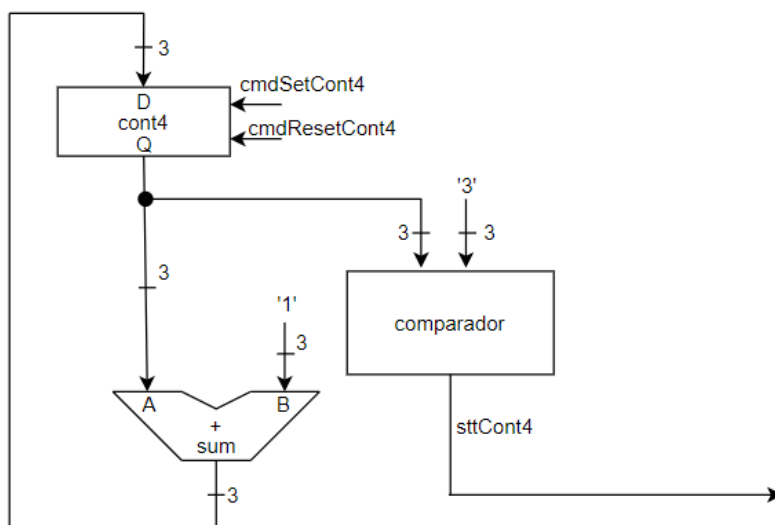
**Figura 2.10:** Circuito para atribuição do registrador cont10 com soma conta10 += 1.

10.  $\text{conta\_loop} += 1$

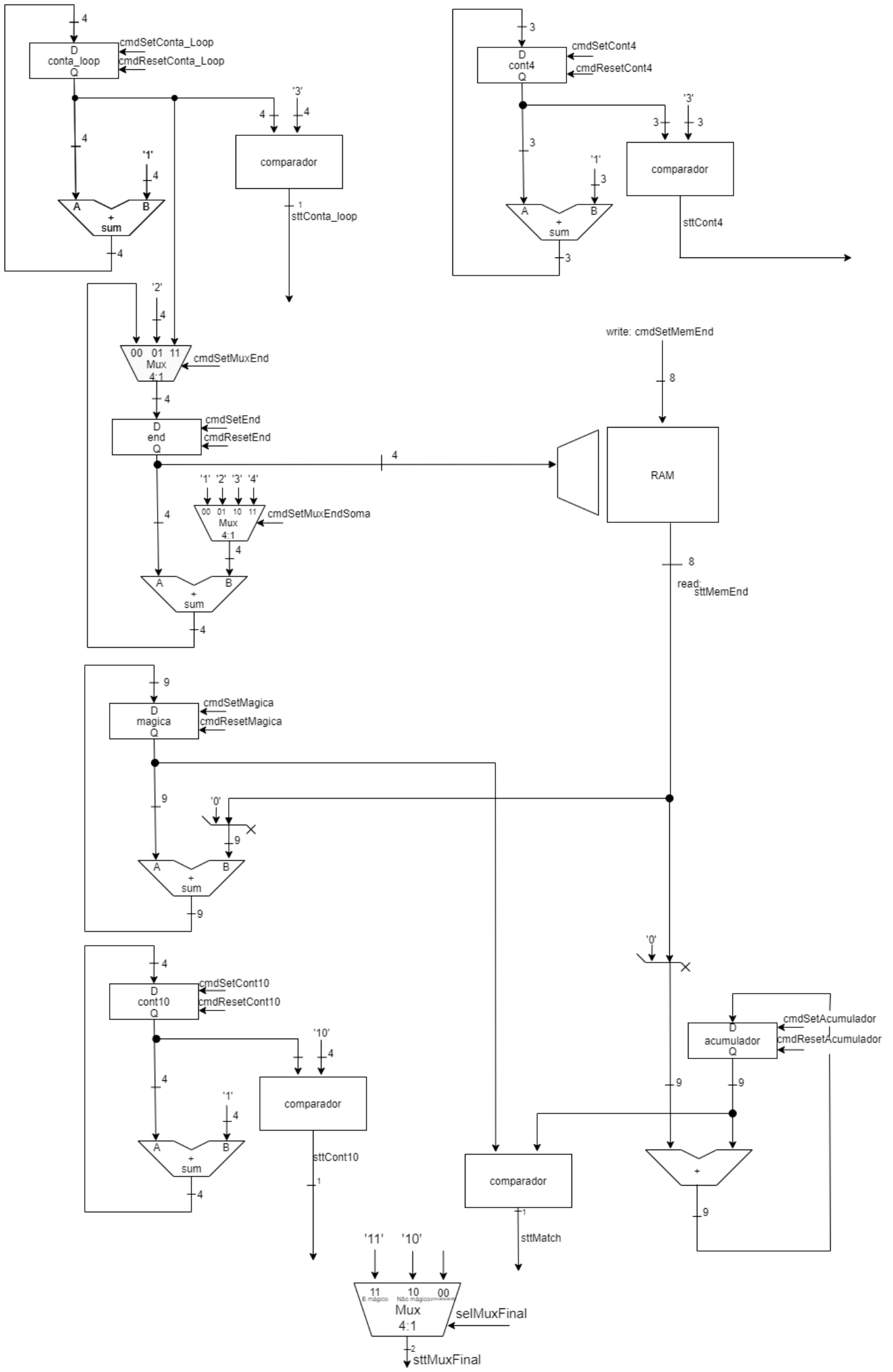


**Figura 2.11:** Circuito para atribuição do registrador `conta_loop` com soma  $\text{conta\_loop} += 1$ .

11.  $\text{cont4} += 1$



**Figura 2.11:** Circuito para atribuição do registrador `cont4` com soma  $\text{cont4} += 1$ .





## 2.4 Projeto do Bloco de Controle

### FSMD do bloco de controle

```
-- //arquivo anexo com FSMD na forma de diagrama
-- SL0: inicia o algoritmo
-- SL1: cont4 = 0; cont10 = 0; magica = 0; acumulador = 0; end = 0; conta_loop = 0
-- SL2: while cont10 < 10: { Mem[end] = input_mem; end += 1; cont10 += 1
-- SL3: }
-- SL4: end = 0
-- SL5: while cont4 < 3: { magica = magica + Mem[end]; end = end +4; cont4 += 1
-- SL6: }
-- SL7: cont4 = 0; end = 2
-- SL8: while cont4 < 3: { acumulador = acumulador + Mem[end]; cont4 += 1; end = end +2
-- SL9: }
-- SL10:
-- SL11: end = 0; acumulador = 0; cont4 = 0
-- SL12: while conta_loop <3: {
-- SL13:     while cont4 < 3: { acumulador = acumulador + Mem[end]; cont4 += 1; end += 1
-- SL14:
-- SL15: } cont4 = 0; acumulador = 0; conta_loop += 1
-- SL16: }
-- SL17: cont4 = 0; conta_loop = 0; acumulador = 0
-- SL18: while conta_loop < 3: { end = conta_loop
-- SL19:     while cont4 < 3: { acumulador = acumulador + Mem[end]; cont4 += 1; end = end +3
-- SL20: }
-- SL21: cont4 = 0; conta_loop += 1; acumulador = 0
-- SL22: }
-- SL23: output
-- SL24: output
```

## Diagrama de transição de estados

```
-- SL0:    --- init == 1/SL1
-- SL1:    --- SL2
-- SL2:    --- SL3
-- SL3:    --- user_ok == 1 and cont10<10/SL2 ; user_ok == 1 and & cont10 == 10/SL4
-- SL4:    --- SL5
-- SL5:    --- SL6
-- SL6:    --- cont4 < 3/SL5 ; cont4 == 3/SL7
-- SL7:    --- SL8
-- SL8:    --- SL9
-- SL9:    --- cont4 <3/SL10 ; cont4 == 3/SL10
-- SL10:   --- acumulador != magica/SL23 ; acumulador == magica/SL11
-- SL11:   --- SL12
-- SL12:   --- SL13
-- SL13:   --- cont4 <3/SL12 ; cont4 = 3/SL14
-- SL14:   --- acumulador == magica/SL15 ; acumulador != magica/SL23
-- SL15:   --- conta_loop <3/SL16 ; conta_loop == 3/SL17
-- SL16:   --- SL12
-- SL17:   --- SL18
-- SL18:   --- SL19
-- SL19:   --- cont4 <3/SL18 ; cont4 == 3/SL20
-- SL20:   --- conta_loop <3; acumulador == magica/SL21 ; acumulador != magica/SL23 ;
acumulador == magica and conta_loop == 3/SL22
-- SL21:   --- SL18
-- SL22:   --- SL24
-- SL23:   --- SL24
-- SL24:   --- fim
```

## Diagrama de saídas

```
-- SL0:    ---
-- SL1:    --- cmdResetCont4 ; cmdResetCont10 ; cmdResetAcumulador ; cmdResetMagica ;
cmdResetEnd ; cmdResetContaLoop
-- SL2:    --- cmdSetMemEnd;cmdSetMuxEndSoma; cmdSetCont10
-- SL3:    --- sttCont10
-- SL4:    --- cmdResetEnd
-- SL5:    --- cmdSetMagica ; cmdSetMuxEndSoma ; cmdSetCont4
-- SL6:    --- sttCont4
-- SL7:    --- cmdResetCont4 ; cmdSetMuxEnd
-- SL8:    --- cmdSetAcumulador ; cmdSetCont4 ; cmdSetMuxEndSoma
-- SL9:    --- sttCont4
-- SL10:   --- sttMatch
-- SL11:   --- cmdResetEnd ; cmdResetAcumulador ; cmdResetCont4
-- SL12:   --- cmdSetAcumulador ; cmdSetCont4 ; cmdSetMuxEndSoma
-- SL13:   --- sttCont4
-- SL14:   --- sttMatch
-- SL15:   --- cmdResetCont4 ; cmdResetAcumulador; sttConta_loop; cmdSetConta_Loop
-- SL16:   ---
-- SL17:   --- cmdResetEnd ; cmdResetCont4 ; cmdResetConta_Loop ; cmdResetAcumulador
-- SL18:   --- cmdSetAcumulador ; cmdSetCont4 ; cmdSetMuxEndSoma
-- SL19:   --- sttCont4
-- SL20:   --- cmdSetConta_loop ; sttMatch ; sttConta_loop
-- SL21:   --- cmdResetAcumulador ; cmdSetConta_loop ; cmdResetCont4
-- SL22:   ---
-- SL23:   --- sttSaida
-- SL24:   --- fim
```

### 3. Desenvolvimento

#### 3.1. Desenvolvimento do bloco operativo

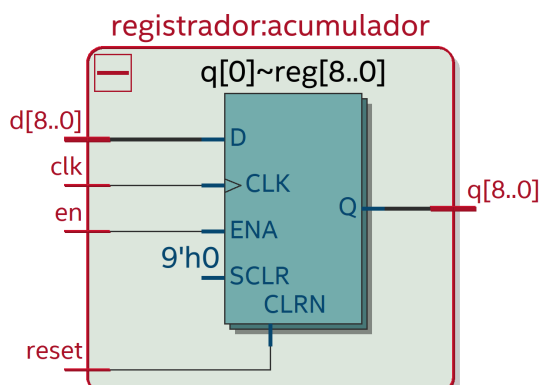
##### Registrador (registrador)

Representando: cont4, cont10, end, magica, acumulador, conta\_loop

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity registrador is
generic (n: natural);
port
(
    d    : in std_logic_vector(n-1 downto 0);
    en   : in std_logic;
    reset : in std_logic;
    clk  : in std_logic;
    q    : out std_logic_vector(n-1 downto 0)
);
end registrador;

architecture behave of registrador is
BEGIN
    process(clk, reset)
    begin
        if reset = '1' then
            q <= (others => '0');
        elsif rising_edge(clk) then
            if en = '1' then
                q <= d;
            end if;
        end if;
    end process;
END behave;
```



Multiplexador 4:1 (mux\_41)

```

library ieee;
use ieee.std_logic_1164.all;

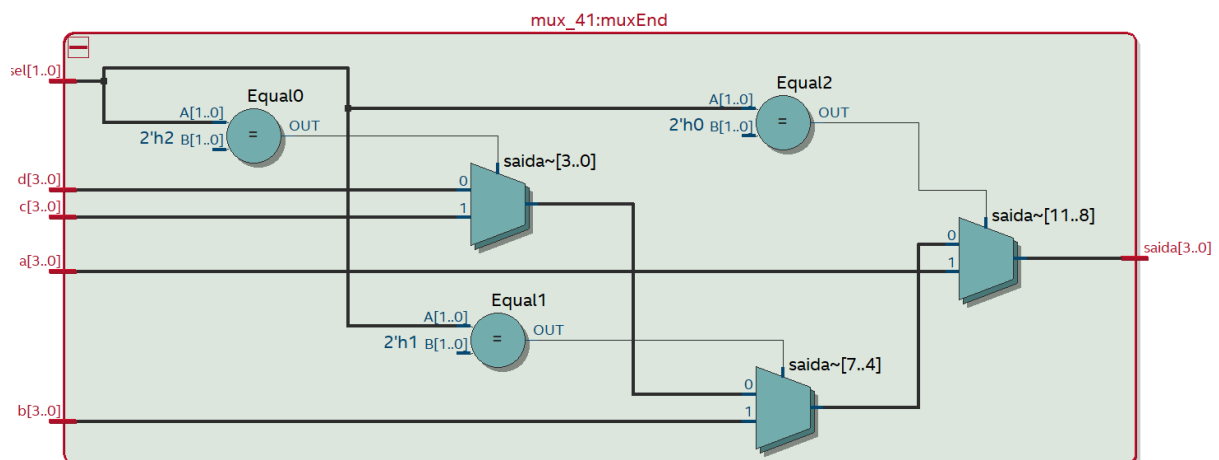
entity mux_41 is
generic(n: natural);
port (a: in std_logic_vector(n-1 downto 0);
      b: in std_logic_vector(n-1 downto 0);
      c: in std_logic_vector(n-1 downto 0);
      d: in std_logic_vector(n-1 downto 0);
      sel: in std_logic_vector(1 downto 0);
      saida: out std_logic_vector(n-1 downto 0)
);
end entity;

architecture mux_bhv of mux_41 is
begin
    saida <= a when sel = "00" else
            b when sel = "01" else
            c when sel = "10" else
            d;

end architecture;

```

Wave - Default		Msgs			
/testefinal/n	4	4			
/testefinal/a	1111	1111			
/testefinal/b	1110	1110			
/testefinal/c	1100	1100			
/testefinal/d	1000	1000			
/testefinal/sel	11	00	01	10	11
/testefinal/saida	1000	1111	1110	1100	1000



**Somador (adderNbits)**

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-----

entity adderNbits is

generic(n: natural);
port( A: in std_logic_vector(n-1 downto 0);
      B: in std_logic_vector(n-1 downto 0);
      carry: out std_logic;
      sum: out std_logic_vector(n-1 downto 0)
);

end adderNbits;

-----

architecture behv of adderNbits is

signal result: std_logic_vector(n downto 0);

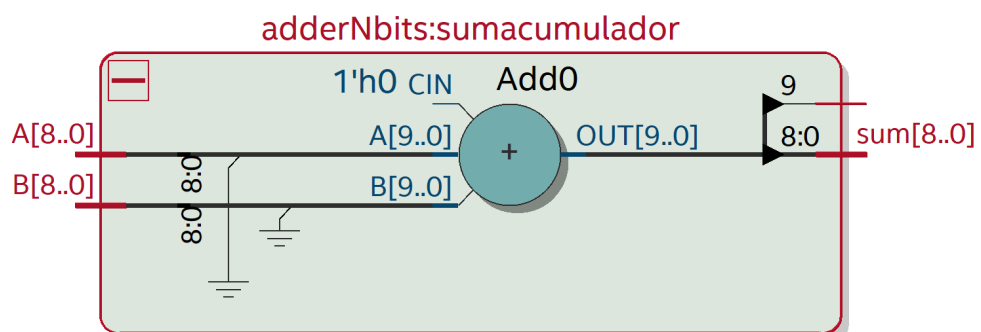
begin

    result <= ('0' & A)+('0' & B);
    sum <= result(n-1 downto 0);
    carry <= result(n);

end behv;

```

Wave - Default		Msgs		
	/testefinal/n	4	4	
	/testefinal/A	1111	0010	1111
	/testefinal/B	1111	0001	1111
	/testefinal/carry	1		
	/testefinal/sum	1110	0011	1110
	/testefinal/result	11110	00011	11110



## Comparador (comparador)

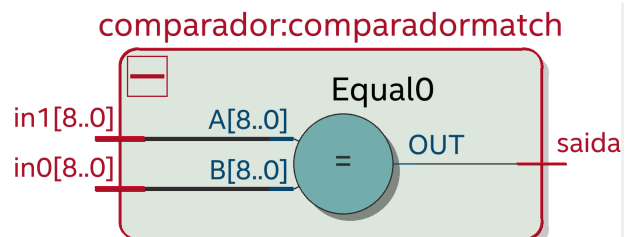
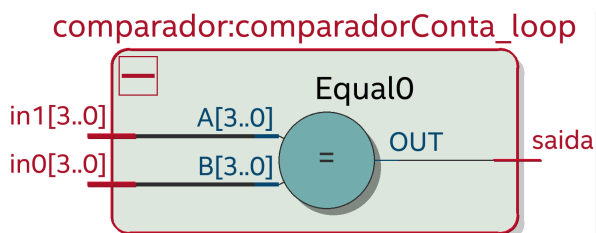
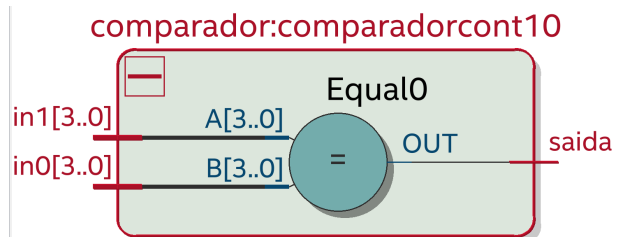
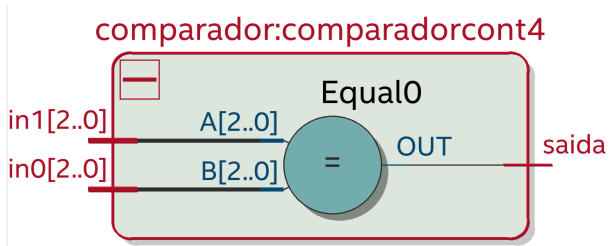
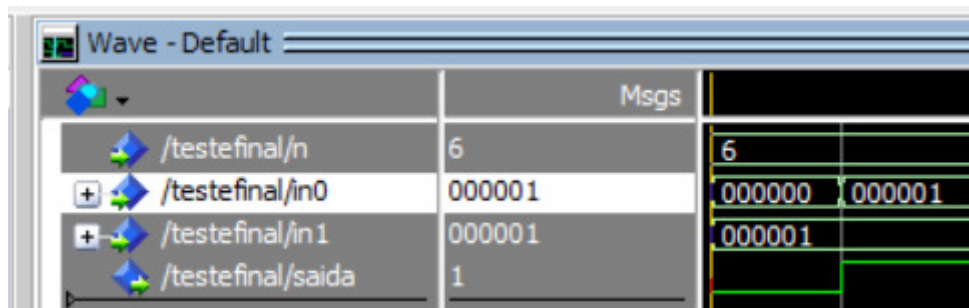
```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;

entity comparador is
generic(n: natural);
port(
    in0, in1: in std_logic_vector(n-1 downto 0);
    saida: out std_logic
);
end entity;

architecture arch1 of comparador is
begin
    saida <= '1' when in0 = in1 else '0';
end architecture;

```



## Bloco Operativo

```
library ieee;
use ieee.std_logic_1164.all;

entity BlocoOperativo is
  generic(
    datawidth: positive := 8;
    addresswidth: positive := 4
  );
  port (
    -- control inputs
    clk : in std_logic;
    reset_req: in std_logic;
    -- data inputs
    testeAddress: out STD_LOGIC_VECTOR (addresswidth-1 DOWNTO 0);
    writedata: IN STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0);
    -- data outputs
    readdata : OUT STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0);
    -- commands from OperativeBlock
    cmdSetEnd, cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
    cmdSetCont10, cmdSetMagica, cmdSetAcumulador, cmdSetCont4, cmdResetConta_loop,
    cmdSetConta_loop: in std_logic;
    cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd: in std_logic_vector(1 downto 0);
    -- status to OperativeBlock
    sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida: out std_logic;
    cmdSetMemEnd: in std_logic_vector (7 downto 0)
  );
end entity;

architecture structuralDatapath of BlocoOperativo is

  component registrador is
    generic(n: natural);
    port(
      d : in std_logic_vector(n-1 downto 0);
      en : in std_logic;
      reset : in std_logic;
      clk : in std_logic;
      q : out std_logic_vector(n-1 downto 0)
    );
  end component;

  component mux_41 is
    generic(n: natural);
    port(a: in std_logic_vector(n-1 downto 0);
      b: in std_logic_vector(n-1 downto 0);
      c: in std_logic_vector(n-1 downto 0);
      d: in std_logic_vector(n-1 downto 0);
      sel: in std_logic_vector(1 downto 0);
      saida: out std_logic_vector(n-1 downto 0)
    );
  end component;

  component adderNbits is
    generic(n: natural);
    port(A: in std_logic_vector(n-1 downto 0);
      B: in std_logic_vector(n-1 downto 0);
      carry: out std_logic;
      sum: out std_logic_vector(n-1 downto 0)
    );
  end component;

  component comparador is
    generic(n: natural);
    port(in0, in1: in std_logic_vector(n-1 downto 0);
      saida: out std_logic
    );
  end component;

  component memoriaRam is
    generic(
      datawidth: positive := 8;
      addresswidth: positive := 4
    );
    port(
      address : IN STD_LOGIC_VECTOR (addresswidth-1 DOWNTO 0);
      clock : IN STD_LOGIC := '1';
      data : IN STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0);
      wren : IN STD_LOGIC;
      q : OUT STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0)
    );
  end component;

  signal carry: std_logic;
  signal D4, Q4: std_logic_vector(2 downto 0);
  signal Dloop, Qloop, D10, Q10, Dend, Qend, Qmuxendsoma, D00: std_logic_vector(3 downto 0);
  signal Dmagica, Qmagica, Dacum, Qacum: std_logic_vector(8 downto 0);
  signal sttMemEnd: std_logic_vector(7 downto 0);
  signal selMuxFinal, sttMuxFinal: std_logic_vector(1 downto 0);
  signal saidaCont10: std_logic;

begin
  testeAddress <= Qend;
  RAM: memoriaRAM
    generic map (datawidth, addresswidth)--
    port map(address=>Qend, clock=>clk, data=>cmdSetMemEnd, wren=> not(saidaCont10), q=>sttMemEnd);
```



```

Cont4: registrador
  generic map (n => 3)
  port map (D4, cmdSetCont4, cmdResetCont4, clk, Q4);

Conta_loop: registrador
  generic map(n => 4)
  port map(Dloop, cmdSetConta_loop, cmdResetConta_loop, clk, Qloop);

Cont10: registrador
  generic map(n => 4)
  port map(D10, cmdSetCont10, cmdResetCont10, clk, Q10);
ende: registrador -- o end virou ende pois é uma palavra reservada somente em vhd1
  generic map(n => 4)
  port map(Dend, cmdSetEnd, cmdResetEnd, clk, Qend);
magica: registrador
  generic map(n => 9)
  port map(Dmagica, cmdSetMagica, cmdResetMagica, clk, Qmagica);
acumulador: registrador
  generic map(n => 9)
  port map(Dacum, cmdSetAcumulador, cmdResetAcumulador, clk, Qacum);

comparadorConta_loop: comparador
  generic map(n => 4)
  port map(Qloop, "0011", sttConta_loop);

comparadorcont10: comparador
  generic map(n => 4)
  port map(Q10, "1010", saidaCont10);

comparadormatch: comparador
  generic map(n => 9)
  port map(Qacum, Qmagica, sttMatch);
comparadorcont4: comparador
  generic map(n => 3)
  port map(Q4, "011", sttCont4);
sumCont10: adderNbits
  generic map(n => 4)
  port map(Q10, "0001", carry, D10);
sumCont4: adderNbits
  generic map(n => 3)
  port map(Q4, "001", carry, D4);
sumConta_loop: adderNbits
  generic map(n => 4)
  port map(Qloop, "0001", carry, Dloop);
sumende: adderNbits
  generic map(n => 4)
  port map(Qend, Qmuxendsoma, carry, D00); --seria D10?

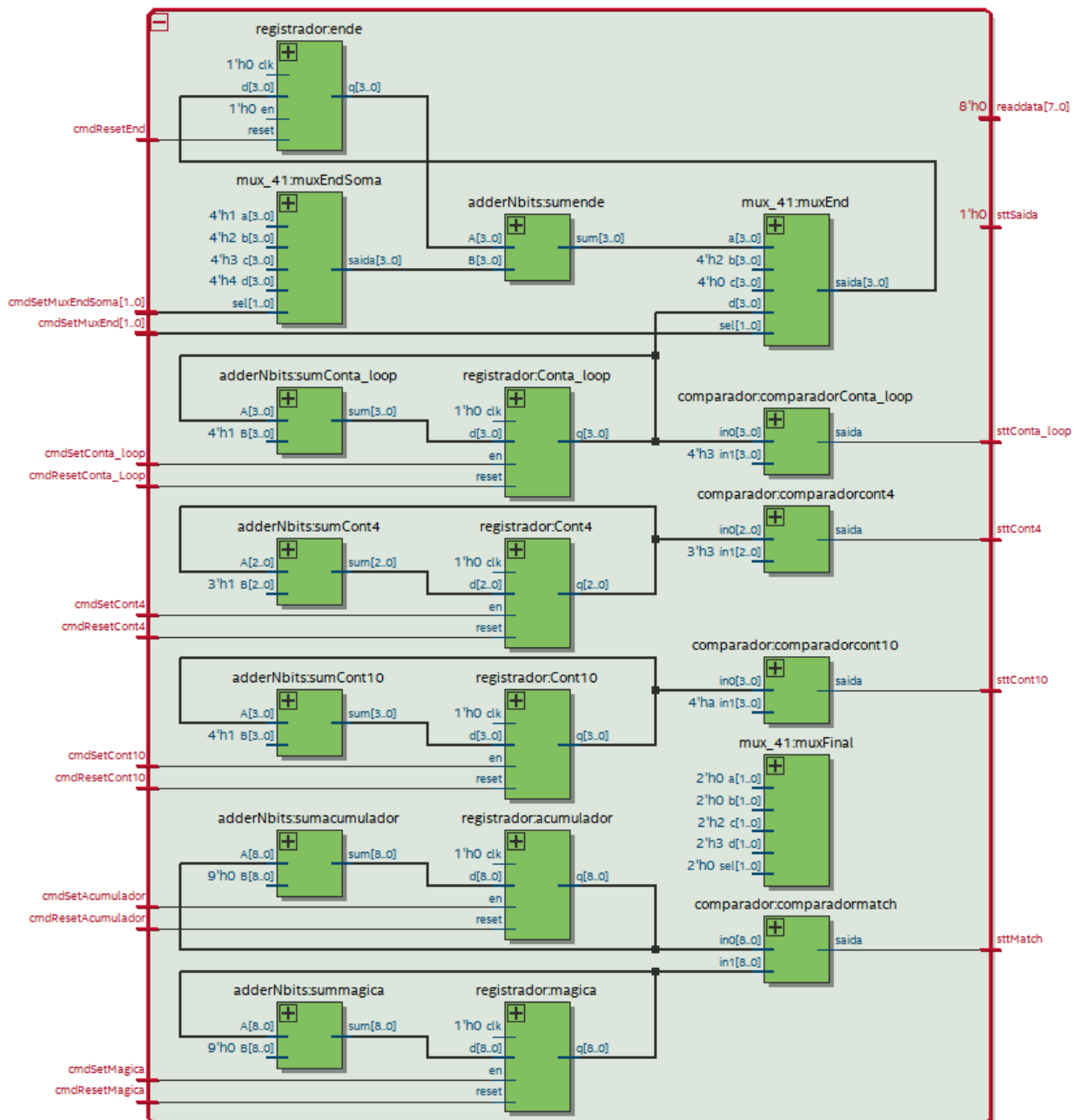
```

```

summagica: adderNbits
  generic map(n => 9)
  port map(Qmagica, '0' & sttMemEnd, carry, Dmagica);
sumacumulador: adderNbits
  generic map(n => 9)
  port map(Qacum, '0' & sttMemEnd, carry, Dacum);
muxFinal: mux_41
  generic map(n => 2)
  port map("00", "00", "10", "11", selMuxFinal, sttMuxFinal);
muxEnd: mux_41
  generic map(n => 4)
  port map(D00, "0010", "0000", Qloop, cmdSetMuxEnd, Dend);
muxEndSoma: mux_41
  generic map(n => 4)
  port map("0001", "0010", "0011", "0100", cmdSetMuxEndSoma, Qmuxendsoma);

```

end architecture;



## Bloco Controle

```
library ieee;
use ieee.std_logic_1164.all;
use work.Pkg_quadrado_magico_BC_Estado.all;

entity BlocoControle is
port (
    clk: in std_logic;
    reset_req: in std_logic;
    sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida: in std_logic;
    cmdSetEnd, cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
    cmdSetMemEnd2, cmdSetCont10, cmdSetMagica, cmdSetCont4, cmdSetAcumulador, cmdResetConta_Loop,
    cmdSetConta_loop: out std_logic;
    cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd: out std_logic_vector(1 downto 0);
    testEstadoAtual: out Estado
);
end entity;

architecture behaviouralFSM of BlocoControle is
    signal estadoAtual, proximoEstado: Estado;
begin
    testEstadoAtual <= estadoAtual;

    process(estadoAtual, sttCont10, sttCont4, sttConta_loop, sttMatch) is
    begin
        proximoEstado <= estadoAtual;
        case estadoAtual is

            when SL0 =>
                proximoEstado <= SL1;
            when SL1 =>
                proximoEstado <= SL2;

            when SL2 =>
                proximoEstado <= SL3;

            when SL3 =>
                if sttCont10 = '0' then
                    proximoEstado <= SL2;
                else
                    proximoEstado <= SL4;
                end if;

            when SL5 =>
                proximoEstado <= SL6;
            when SL6 =>
                if sttCont4 = '0' then
                    proximoEstado <= SL5;
                elsif sttCont4 = '1' then
                    proximoEstado <= SL7;
                end if;
            when SL7 =>
                proximoEstado <= SL8;
            when SL8 =>
                proximoEstado <= SL9;
            when SL9 =>
                if sttCont4 = '0' then
                    proximoEstado <= SL8;
                elsif sttCont4 = '1' then
                    proximoEstado <= SL10;
                end if;
            when SL10 =>
                if sttMatch = '0' then
                    proximoEstado <= SL23;
                elsif sttMatch = '1' then
                    proximoEstado <= SL11;
                end if;
            when SL11 =>
                proximoEstado <= SL12;
            when SL12 =>
                proximoEstado <= SL13;
            when SL13 =>
                if sttCont4 = '0' then
                    proximoEstado <= SL12;
                elsif sttCont4 = '1' then
                    proximoEstado <= SL14;
                end if;
            when SL14 =>
                if sttMatch = '0' then
                    proximoEstado <= SL23;
                elsif sttMatch = '1' then
                    proximoEstado <= SL15;
                end if;
        end case;
    end process;
end architecture;
```

```

when SL15 =>
    if sttConta_loop = '0' then
        proximoEstado <= SL16;
    elsif sttConta_loop = '1' then
        proximoEstado <= SL17;
    end if;
when SL16 =>
    proximoEstado <= SL12;
when SL17 =>
    proximoEstado <= SL18;
when SL18 =>
    proximoEstado <= SL19;
when SL19 =>
    if sttCont4 = '0' then
        proximoEstado <= SL18;
    elsif sttCont4 = '1' then
        proximoEstado <= SL20;
    end if;
when SL20 =>
    if sttConta_loop = '0' and sttMatch = '1' then
        proximoEstado <= SL21;
    elsif sttMatch = '0' then
        proximoEstado <= SL23;
    elsif sttConta_loop = '1' and sttMatch = '1' then
        proximoEstado <= SL22;
    end if;
when SL21 =>
    proximoEstado <= SL18;
when SL22 =>
    proximoEstado <= SL24;
when SL23 =>
    proximoEstado <= SL24;
when SL24 =>
    proximoEstado <= SL0;
end case;
end process;

process(clk, reset_req) is
begin
    if reset_req = '1' then
        estadoAtual <= SL0;
    elsif rising_edge(clk) then
        estadoAtual <= proximoEstado;
    end if;
end process;

```

```

process(estadoAtual) is
begin

    cmdSetEnd          <= '0';
    cmdSetMemEnd2      <= '0';
    cmdSetMuxEndSoma   <= "00";
    cmdSetCont10       <= '0';
    cmdResetCont4      <= '0';
    cmdSetCont4        <= '0';
    cmdResetCont10     <= '0';
    cmdResetAcumulador <= '0';
    cmdSetAcumulador   <= '0';
    cmdResetMagica     <= '0';
    cmdResetContaLoop  <= '0';
    cmdSetConta_loop   <= '0';
    cmdResetEnd        <= '0';
    cmdSelectEnd       <= "00";
    cmdMuxFinal        <= "00";

```

```

case estadoAtual is
when SL2 =>
    cmdSetEnd          <= '1';
    cmdSetMemEnd2      <= '1';
    cmdSetMuxEndSoma   <= "11";
    cmdSetCont10       <= '1';

when SL4 =>
    cmdResetEnd        <= '1';
    cmdSetEnd          <= '1';

when SL5 =>
    cmdSetMagica       <= '1';
    cmdSetMuxEndSoma   <= "11";
    cmdSetCont4        <= '1';

when SL7 =>
    cmdResetCont4      <= '1';
    cmdSetMuxEnd       <= "11";
    cmdSetEnd          <= '1';

when SL8 =>
    cmdSetAcumulador   <= '1';
    cmdSetCont4        <= '1';
    cmdSetMuxEndSoma   <= "11";
    cmdSetEnd          <= '1';

```

```

when SL11 =>
    cmdResetEnd        <= '1';
    cmdResetAcumulador <= '1';
    cmdResetCont4      <= '1';
    cmdSetEnd          <= '1';

when SL12 =>
    cmdSetAcumulador   <= '1';
    cmdSetCont4        <= '1';
    cmdSetMuxEndSoma   <= "11";
    cmdSetEnd          <= '1';

when SL15 =>
    cmdResetCont4      <= '1';
    cmdResetAcumulador <= '1';
    cmdSetConta_Loop   <= '1';
    cmdSetEnd          <= '1';

when SL17 =>
    cmdResetEnd        <= '1';
    cmdResetCont4      <= '1';
    cmdResetConta_Loop <= '1';
    cmdResetAcumulador <= '1';

when SL18 =>
    cmdSetAcumulador   <= '1';
    cmdSetCont4        <= '1';
    cmdSetMuxEndSoma   <= "11";
    cmdSetEnd          <= '1';

when SL20 =>
    cmdSetConta_loop   <= '1';

when SL21 =>
    cmdResetAcumulador <= '1';

when SL22 =>
    cmdMuxFinal        <= "11";

when SL23 =>
    cmdMuxFinal        <= "10";

when others =>
    null;

```

```

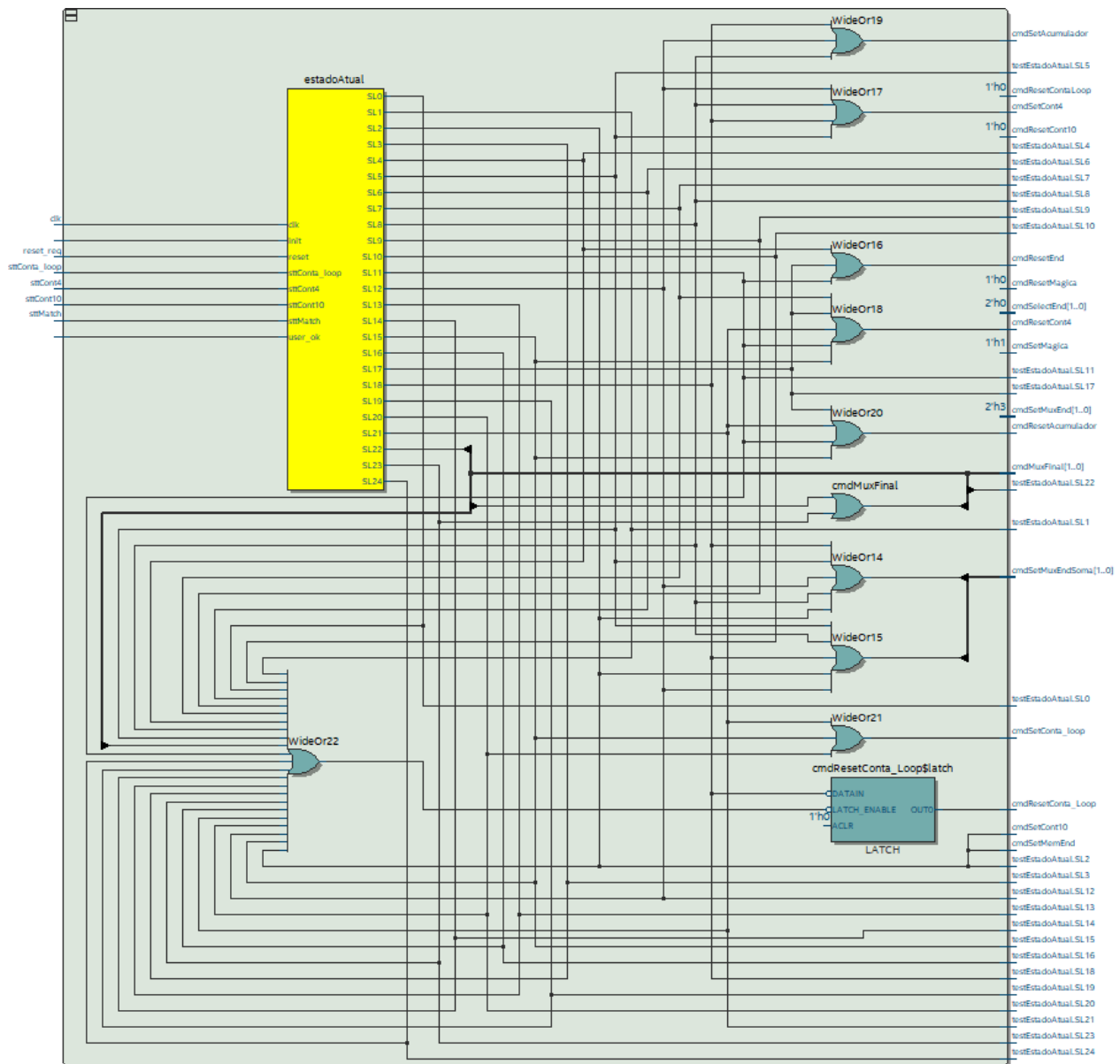
    end case;
end process;

```

```

end architecture;

```



## Quadrado Mágico (topo)

```
library ieee;
use ieee.std_logic_1164.all;
use work.Pkg_quadrado_magico_BC_Estado.all;

entity QuadradoMagico is
    generic(
        dataWidth: positive := 8;
        addressWidth: positive := 4
    );
    port(
        clk: in std_logic;
        reset_req: in std_logic;
        chipselect: in std_logic;
        readd: in std_logic;
        writee: in std_logic;
        testeAddress: out std_logic_vector(addressWidth-1 downto 0);
        writedata: in std_logic_vector(dataWidth-1 downto 0);
        interrupt: out std_logic;
        readdata: out std_logic_vector(dataWidth-1 downto 0);
        testEstadoAtual: out Estado
    );
end entity;

architecture structural of QuadradoMagico is

    component BlocoControle is
        port (
            clk: in std_logic;
            reset_req: in std_logic;
            sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida: in std_logic;
            cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
            cmdSetMemEnd2, cmdSetCont10, cmdSetMagica, cmdSetCont4, cmdSetAcumulador, cmdResetConta_Loop,
            cmdSetConta_loop: out std_logic;
            cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd: out std_logic_vector(1 downto 0);
            testEstadoAtual: out Estado
        );
    end component;

    component BlocoOperativo is
        generic(
            dataWidth: positive := 8;
            addressWidth: positive := 4
        );
        port (
            -- control inputs
            clk: in std_logic;
            reset_req: in std_logic;
            -- data inputs
            testeAddress: out std_logic_vector(addressWidth-1 downto 0);
            writedata: in std_logic_vector(dataWidth-1 downto 0);
            -- data outputs
            readdata: out std_logic_vector(dataWidth-1 downto 0);
            -- commands from OperativeBlock
            cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
            cmdSetCont10, cmdSetMagica, cmdSetAcumulador, cmdSetCont4, cmdResetConta_Loop,
            cmdSetConta_loop: in std_logic;
            cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd: in std_logic_vector(1 downto 0);
            -- status to OperativeBlock
            sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida: out std_logic;
            cmdSetMemEnd: in std_logic_vector(7 downto 0)
        );
    end component;

    -- comandos
    signal cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
        cmdSetMemEnd2, cmdSetCont10, cmdSetMagica, cmdSetCont4, cmdSetAcumulador, cmdResetConta_Loop,
        cmdSetConta_loop: std_logic;
    signal cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd: std_logic_vector(1 downto 0);
    -- status
    signal sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida: std_logic;
    signal cmdSetMemEnd: std_logic_vector(7 downto 0);

begin
    Controle: BlocoControle
        port map ( clk, reset_req,
            sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida,
            cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
            cmdSetMemEnd2, cmdSetCont10, cmdSetMagica, cmdSetCont4, cmdSetAcumulador, cmdResetConta_Loop,
            cmdSetConta_loop, cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd,
            testEstadoAtual);

    Operativo: BlocoOperativo
        generic map (dataWidth, addressWidth)
        port map (clk, reset_req,
            testeAddress, writedata,
            readdata,
            cmdResetCont4, cmdResetCont10, cmdResetAcumulador, cmdResetMagica, cmdResetEnd, cmdResetContaLoop,
            cmdSetCont10, cmdSetMagica, cmdSetAcumulador, cmdSetCont4, cmdResetConta_Loop,
            cmdSetConta_loop, cmdSelectEnd, cmdMuxFinal, cmdSetMuxEndSoma, cmdSetMuxEnd,
            sttCont10, sttCont4, sttMatch, sttConta_loop, sttSaida, cmdSetMemEnd
        );
end architecture;
```

### Controle dos estados (Pkg\_quadrado\_magico\_BC\_Estado)

```
library ieee;
use ieee.std_logic_1164.all;

package Pkg_quadrado_magico_BC_Estado is
    type Estado is (
        SL0, SL1, SL2, SL3, SL4, SL5, SL6, SL7, SL8,
        SL9, SL10, SL11, SL12, SL13, SL14, SL15, SL16,
        SL17, SL18, SL19, SL20, SL21, SL22, SL23, SL24
    );
end package;
```



## Memória RAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY memoriaRAM IS
    generic(
        datawidth: positive := 8;
        addresswidth: positive := 8
    );
    PORT(
        address      : IN STD_LOGIC_VECTOR (addresswidth-1 DOWNTO 0);
        clock        : IN STD_LOGIC := '1';
        data         : IN STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0);
        wren         : IN STD_LOGIC ;
        q            : OUT STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0)
    );
END memoriaRAM;

ARCHITECTURE SYN OF memoriaram IS


    SIGNAL sub_wire0 : STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0);

BEGIN
    q      <= sub_wire0(datawidth-1 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        intended_device_family => "MAX 10",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 2**addresswidth,
        operation_mode => "SINGLE_PORT",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "UNREGISTERED",
        power_up_uninitialized => "FALSE",
        read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
        widthad_a => addresswidth,
        width_a => datawidth,
        width_byteena_a => 1
    )
    PORT MAP (
        address_a => address,
        clock0 => clock,
        data_a => data,
        wren_a => wren,
        q_a => sub_wire0
    );

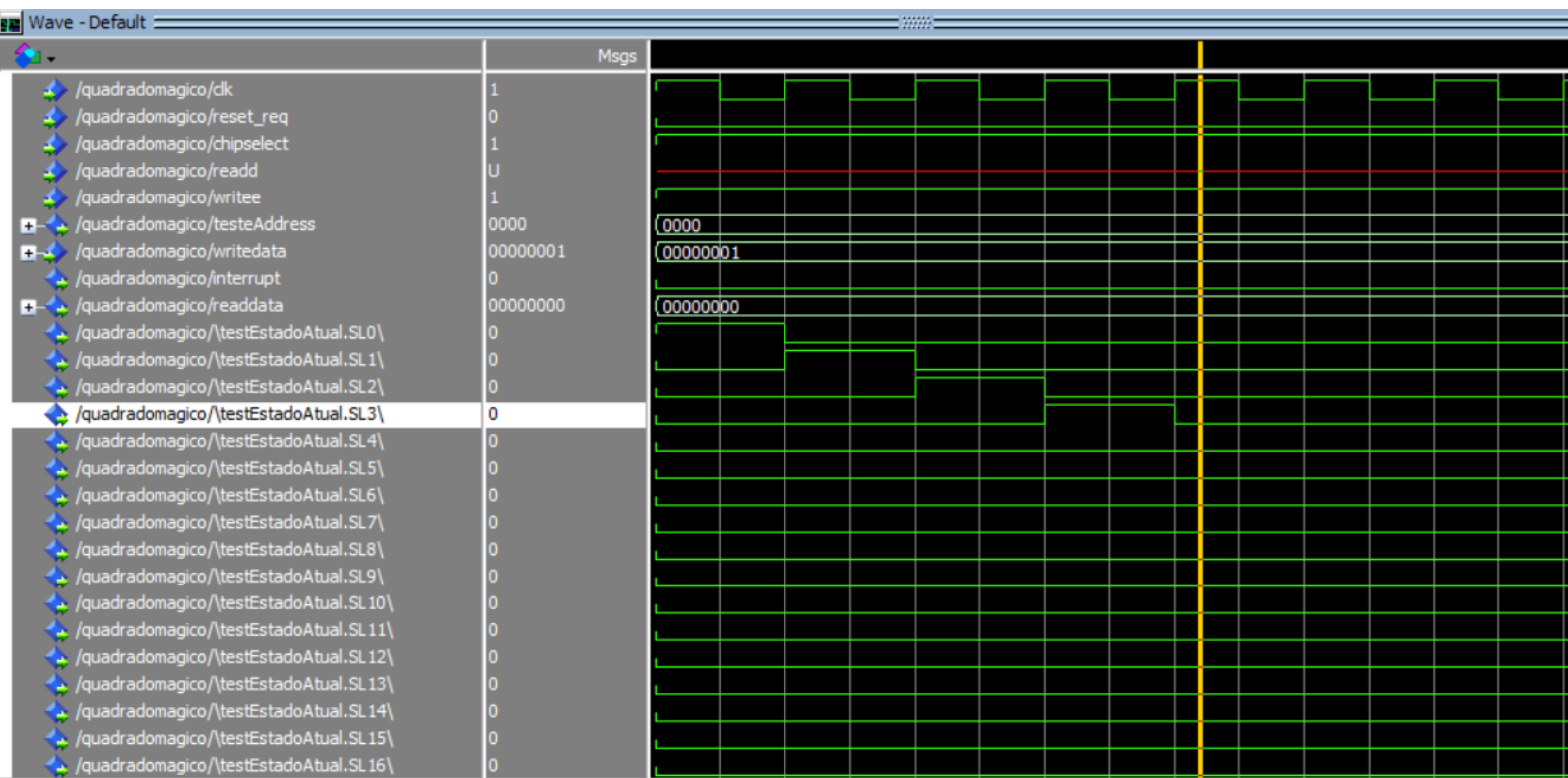
END SYN;
```

## Área

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Thu Dec 03 13:54:37 2020
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	QuadradoMagico
Top-level Entity Name	QuadradoMagico
Family	Cyclone IV E
Device	EP4CE55F29C6
Timing Models	Final
Total logic elements	57 / 55,856 ( < 1 % )
Total registers	44
Total pins	51 / 375 ( 14 % )
Total virtual pins	0
Total memory bits	16 / 2,396,160 ( < 1 % )
Embedded Multiplier 9-bit elements	0 / 308 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

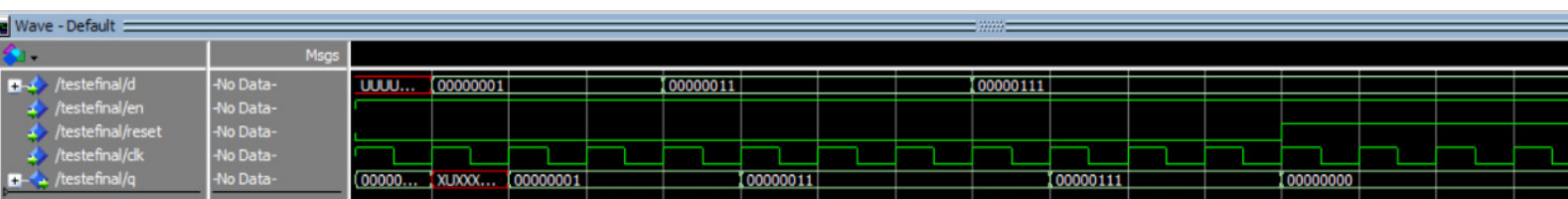
## 4. Testes

É necessário informar antes da apresentação dos testes do projeto que, ao tentar fazer a simulação encontramos problema na transição dos estados. Tentamos a qualquer custo resolver, porém o código (que está apresentado atualizado no tópico anterior) parece estar correto, visto que a State Machine está de acordo com a proposta e a compilação é executada sem mais problemas. Pedimos desculpas pela falta dos testes dos blocos.



O previsto era que seguisse depois do estado 3, após a verificação do loop, para os demais estados seguintes. Acreditamos que possa ter sido algo na memória por conta do testeAddress não estar mudando suas posições (a memória não estar lendo o endereço), ou até mesmo um erro de lógica na variável testEstadoAtual.

### Registrador



Registrador de 8bits com entrada 'd', enable 'en', reset, clock e saída 'q'. Utilizado em end, conta\_loop, conta10, magica, acumulador e cont4.

## Comparador

Wave - Default		Msgs									
/testefinal/in0	1111	UUUU		0000				1111			
/testefinal/in1	1010	UUUU		0000			1111			1010	
/testefinal/saida	0										

Compara duas entradas, 'in0' e 'in1', caso forem iguais retorna "saida" = 1, caso contrário "saida" = 0. Utilizado em conta\_loop, cont10, cont4 e match.

## Somador

Wave - Default		Msgs									
/testefinal/A	0001	0001					1111			0001	
/testefinal/B	1111	0001		0011			1111				
/testefinal/carry	1										
/testefinal/sum	0000	0010		0100			1110			0000	

Soma as entradas 'A' e 'B' e retorna a saída 'sum', vale ressaltar que o somador possui um carry. Utilizado em magica, acumulador, cont4, cont10, conta\_loop e end.

## Multiplexador 4:1

Wave - Default		Msgs									
/testefinal/a	0000	0000									
/testefinal/b	0010	0010									
/testefinal/c	0100	0100									
/testefinal/d	1000	1000									
/testefinal/sel	11	UU		00		01		10		11	
/testefinal/saida	1000			0000		0010		0100		1000	

Multiplexador com entradas 'a', 'b', 'c' e 'd', saída como 'saida' e um seletor 'sel'. Ele é utilizado no MuxEnd, MuxEndSoma e MuxFinal.