Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Computação e Sistemas Digitais
PCS 3635 – Laboratório Digital 1
Turma 2 - Professor Edson Toshimi Midorikawa
Bancada A3

Experiência 3

Projeto de uma Unidade de Controle

Nome Completo	**nºUSP**
Igor Pontes Tresolavy	12553646
Thiago Antici de Souza Rodrigues	12551412

São Paulo
2023/01/15

1. Introdução e Objetivos

Esta experiência tem como objetivo o estudo e desenvolvimento de uma unidade de controle em VHDL de um circuito digital simples, e o teste e sintetização do circuito em simulação e uma placa FPGA DE0-CV.

Além disso, com base no fluxo de dados desenvolvido na experiência anterior, objetiva-se desenvolver um circuito digital composto da unidade de controle e fluxo de dados, testando-se cada componente de maneira iterativa e incremental, a fim de atingir-se um circuito cujo funcionamento corresponda às especificações desejadas.

O funcionamento do circuito digital a ser desenvolvido pode ser descrito em texto corrido como a seguir:

> *"O circuito do sistema digital sequencial inclui um conjunto de 16 dados de 4 bits que é armazenado em uma memória interna, cujos endereços são percorridos por meio de um contador interno. Depois do acionamento do sinal reset, o circuito deve aguardar o início de sua operação que é realizado com o acionamento do sinal de entrada iniciar. Depois de iniciar seu funcionamento, o circuito deve armazenar o valor das chaves de entrada (sinal chaves) e depois comparar o conteúdo armazenado das chaves com o respectivo dado da memória e deve indicar o resultado da comparação na saída de depuração db_igual. O conteúdo armazenado das chaves é apresentado na saída de depuração db_chaves. Em seguida, o contador interno deve ser incrementado para posicionar o endereçamento da memória para permitir o acesso ao próximo dado da memória. As saídas de depuração db_contagem e db_memoria indicam, respectivamente, o endereço e o dado armazenado pela memória, ao passo que a saída de depuração db_estado, por sua vez, deve indicar o código do estado vigente da Unidade de Controle em determinado instante do funcionamento do sistema digital. O ciclo de comparação e reposicionamento da memória deve prosseguir até que todos os 16 dados sejam verificados. Ao final da operação, o sinal de saída pronto deve ser acionado por um período de clock. Depois disso, o circuito deve voltar para o estado inicial para aguardar a próxima ativação de iniciar"*

A seguir, explicitam-se as alterações realizadas no fluxo de dados da experiência anterior a fim de atingir os requisitos funcionais da descrição acima, seguidas da especificação da unidade de controle do circuito em VHDL.

Finalmente, as descrições da unidade de controle e do fluxo de dados são acopladas. Ao longo do documento, há extenso uso de simulações e análises de formas de onda para a verificação da corretude lógica das descrições em VHDL.

2. Descrição do Projeto

a. Projeto Lógico do Fluxo de Dados

Apresentam-se, a seguir, diagramas correspondentes ao fluxo de dados da experiência 02 (anterior) e da atual experiência:

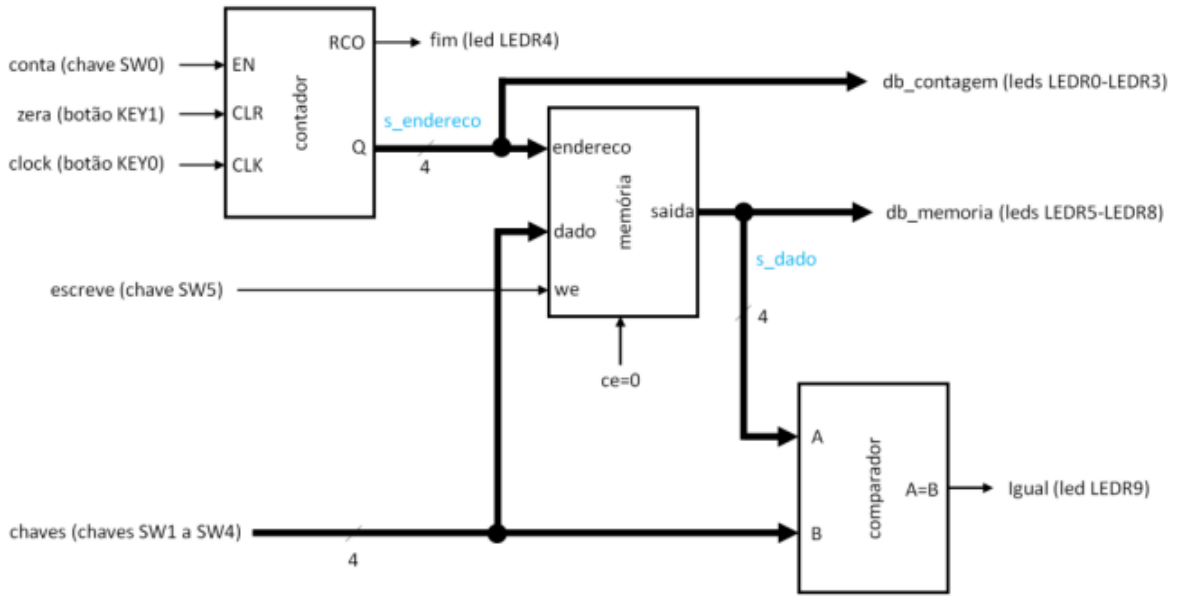


Figura 1 — Fluxo de Dados da experiência 2.

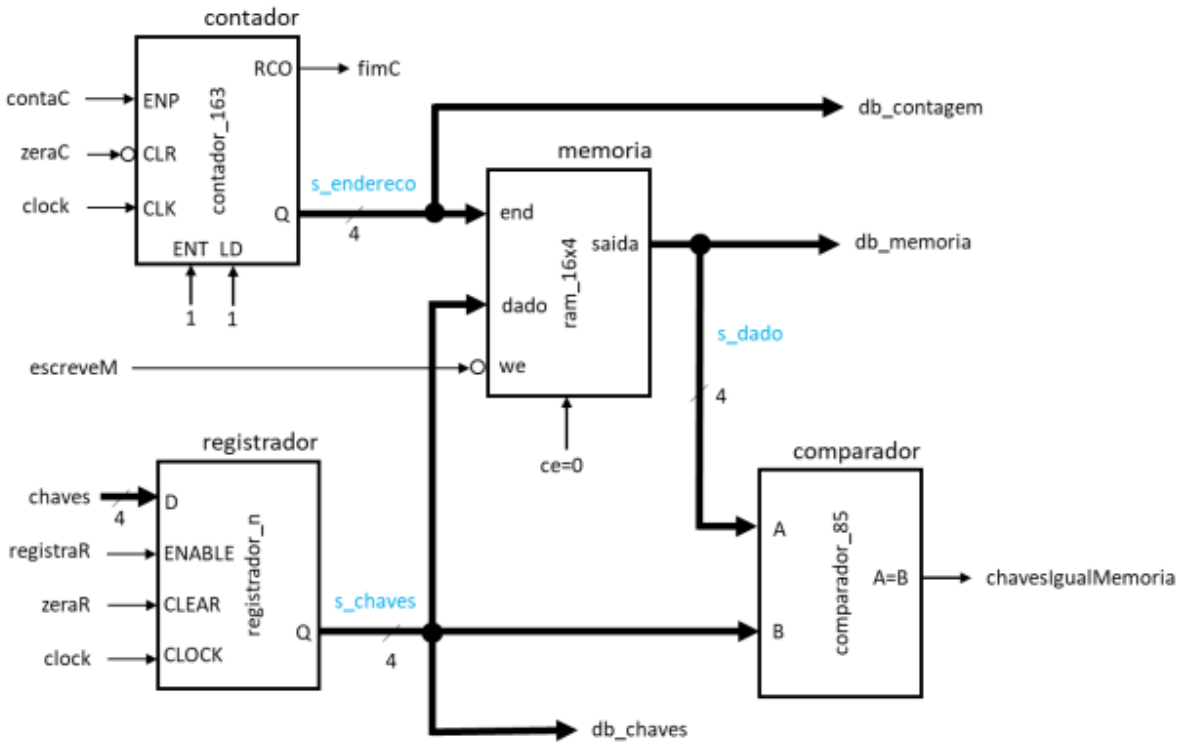


Figura 2 — Fluxo de Dados da experiência atual.

Nota-se que, além do nome de alguns sinais, houve modificação em relação à origem dos dados da entrada *B* do comparador. No fluxo de dados desta experiência, os dados das chaves da placa FPGA são primeiro armazenados no componente *registrador_n* para posteriormente serem comparados ao valor de saída da memória *ram_16x4* — como especificado na descrição introdutória do circuito: "*Depois de iniciar seu funcionamento, o circuito deve armazenar o valor das chaves de entrada [...]*".

O componente *registrador_n* consiste em um elemento de memória de 4 bits síncrono em borda de subida com sinal de *clear* assíncrono. A entidade do componente em VHDL é dada abaixo:

```
36  entity registrador_n is
37      generic (
38          constant N: integer := 8
39      );
40      port (
41          clock  : in  std_logic;
42          clear  : in  std_logic;
43          enable : in  std_logic;
44          D      : in  std_logic_vector (N-1 downto 0);
45          Q      : out std_logic_vector (N-1 downto 0)
46      );
47  end entity registrador_n;
```

Figura 3 — Interface da entidade VHDL do componente *registrador_n*, contido no arquivo *registrador_n.vhd* do material experimental. A constante *N* define a quantidade de bits de armazenamento do registrador. O sinal de *clock* é sensível à borda de subida e, se o enable estiver ativo (nível lógico '1'), carrega o valor da entrada *D* no registrador, cujo valor será refletido na saída *Q*. O *clear* restaura a saída *Q* para '0' assincronamente.

Adicionou-se, então, o componente *registrador_n* ao fluxo de dados da experiência anterior como mostra a figura 2 e alterou-se a interface da entidade como descrito na figura 4. Também alterou-se o conteúdo da memória *ram_16x4* com os dados do arquivo *ram_conteudo_jogada.mif*, disponibilizado junto ao material da experiência. Finalmente, testou-se o componente em simulação seguindo o plano de testes descrito na tabela 1, cuja codificação se encontra presente no arquivo *fluxo_dados_tb.vhd*, também disponível junto ao material:

```
28  entity fluxo_dados is
29      port (
30          clock               : in  std_logic;
31          zeraC               : in  std_logic;
32          contaC              : in  std_logic;
33          escreveM            : in  std_logic;
34          zeraR               : in  std_logic;
35          registraR           : in  std_logic;
36          chaves              : in  std_logic_vector (3 downto 0);
37          chavesIgualMemoria  : out std_logic;
38          fimC                : out std_logic;
39          db_contagem         : out std_logic_vector (3 downto 0);
40          db_memoria          : out std_logic_vector (3 downto 0);
41          db_chaves           : out std_logic_vector (3 downto 0)
42      );
43  end entity;
```

Figura 4 — Interface da entidade VHDL do componente *fluxo_dados*, contido no arquivo *fluxo_dados.vhd* do material experimental. Há uma correspondência direta entre os sinais da entidade e os sinais da figura 2.

#	Operação	Sinais de controle	Resultado Esperado
c.i	Condições iniciais (todas as entradas desativadas)	clock=0, zeraC=0, contaC=0, zeraR=0, registraR=0, chaves=0000	chavesIgualMemoria=0 a=0 fimC=0 db_contagem=0000 db_memoria=0001 db_chaves=0000
1	Zerar contador e registrador (manter outras entradas desativadas)	zeraC=1, zeraR=1, clock ↑	chavesIgualMemoria=0 fimC=0, db_contagem=0000, db_memoria=0001, db_chaves=0000
2	Verificar saídas com chaves=0001 (manter outras entradas desativadas)	zeraC=0, contaC=0, zeraR=0, registraR=0, chaves=0001, clock ↑	chavesIgualMemoria=0 fimC=0, db_contagem=0000, db_memoria=0001, db_chaves=0000
3	Registrar chaves com chaves=0001 (manter outras entradas desativadas)	chaves=0001, registrarR=1, clock ↑	chavesIgualMemoria=1 fimC=0 db_contagem=0000 db_memoria=0001 db_chaves=0001
4	Verificar saídas com chaves=0001 (manter outras entradas desativadas)	chaves=0001, clock ↑	chavesIgualMemoria=1 fimC=0 db_contagem=0000 db_memoria=0001 db_chaves=0001
5	Incrementar contador (manter outras entradas desativadas)	contaC=1, clock ↑	chavesIgualMemoria=0 fimC=0 db_contagem=0001 db_memoria=0010 db_chaves=0001
6	Registrar chaves com chaves=0010 (manter outras entradas desativadas)	chaves=0010 clock ↑	chavesIgualMemoria=1 fimC=0 db_contagem=0001 db_memoria=0010 db_chaves=0010

7	Verificar saídas com chaves=0010 (manter outras entradas desativadas)	chaves=0010 clock ↑	chavesIgualMemoria=1 fimC=0 db_contagem=0001 db_memoria=0010 db_chaves=0010
8	Incrementar contador (manter outras entradas desativadas)	contaC=1, clock ↑	chavesIgualMemoria=0 fimC=0 db_contagem=0010 db_memoria=0100 db_chaves=0010
9	Registrar chaves com chaves=1000 (manter outras entradas desativadas)	chaves=1000 clock ↑	chavesIgualMemoria=0 fimC=0 db_contagem=0010 db_memoria=0100 db_chaves=1000
10	Verificar saídas com chaves=1000 (manter outras entradas desativadas)	chaves=1000 clock ↑	chavesIgualMemoria=0 fimC=0 db_contagem=0010 db_memoria=0100 db_chaves=1000
11	Incrementar contador até final da contagem	contaC=1 clock ↑ (13x)	chavesIgualMemoria=0 fimC=1 db_contagem=1111 db_memoria=0100 db_chaves=1000

Tabela 1 — Plano de Testes para o Circuito do Fluxo de Dados

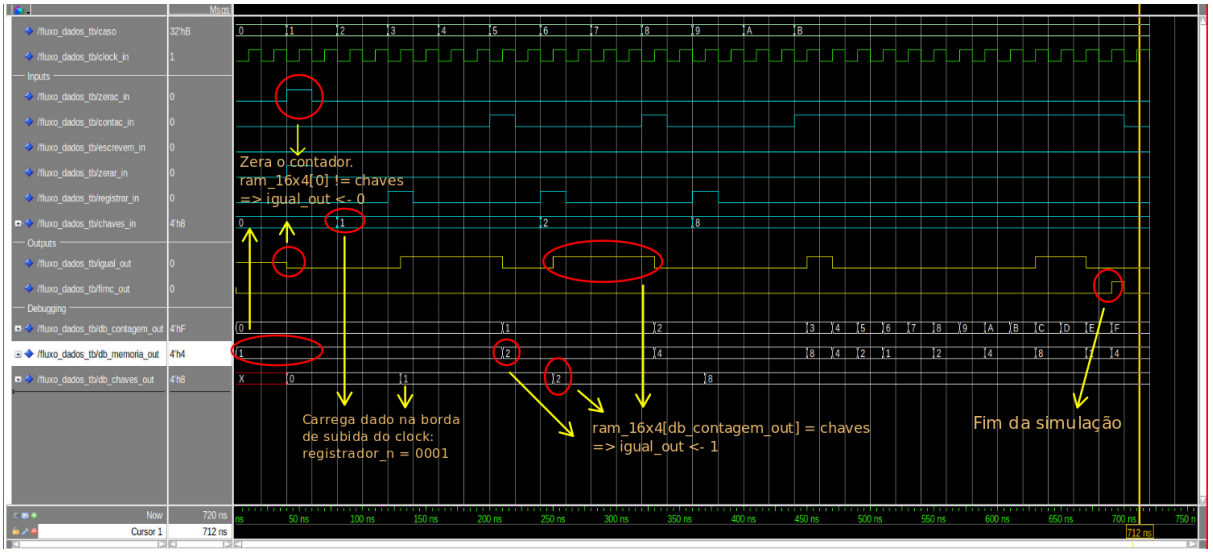


Figura 5 — Formas de onda de simulação do componente *fluxo_dados*.

Dado que a parcela do fluxo de dados correspondente à experiência anterior já foi testada e verificada, assumiu-se que a única fonte de erros possível nesta etapa de teste consiste no *registrador_n* e a interligação dos seus sinais com o resto do fluxo de dados. No entanto, não houve intercorrências durante o teste.

Destaca-se em azul, na tabela 1, alguns sinais em especial. No caso de teste 1, reiniciou-se a contagem do *contador_163* e configurou-se as *chaves* para 0. No teste 2, nota-se que, mesmo com o sinal *chaves* possuindo o mesmo valor que o armazenado no endereço da memória *ram_16x4* correspondente ao valor da contagem *db_contagem*, o comparador *comparador_85* não acusa igualdade nos dados (*chavesIgualMemoria* = ’0’), pois estes não haviam sido carregados no

registrador_n até então. No caso de teste 3 e 4, no entanto, após uma borda de subida de *clock*, constata-se que *chavesIgualMemoria* = *'1'*.

No caso do teste 5, após a mudança no valor do contador, o dado de saída da memória alterou-se e o comparador deixou de acusar igualdade em seus valores de entrada *A* e *B*. O restante dos casos seguem de maneira análoga aos aqui citados, com destaque para o teste 10, no qual confirma-se *chaveIgualMemoria* = '0' para o caso no qual o dado de saída de memória não equivale ao valor carregado pelas chaves no *registrador_n*.

b. Projeto Lógico de uma Unidade de Controle

A partir da descrição em texto corrido do funcionamento desejado do circuito digital (exposta na introdução), pode-se caracterizar o comportamento da unidade de controle responsável por coordenar o fluxo de dados em pseudocódigo e no formato de máquina de estados, como exibido abaixo:

```
Algoritmo: um sistema digital simples
 entradas: iniciar, chaves
 saídas: pronto
 depuração: chaves, contagem, memória, estado, igual, iniciar
 {
    while (verdadeiro) {
        espera acionamento do sinal iniciar
        inicia circuito para condições iniciais
        while (não atingiu o último dado) {
            compara chaves de entrada com dados armazenados e atualiza
saídas
            incrementa contador interno
        }
    ativa saída pronto
}
```

Figura 6 — Pseudocódigo do sistema digital descrito na introdução deste relatório.

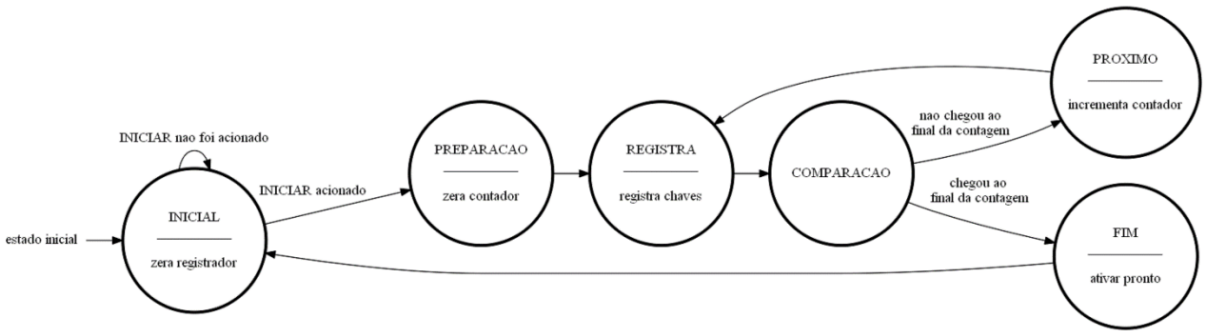


Figura 7 — Máquina de estados de alto nível da unidade de controle do circuito digital descrito na introdução deste relatório (imagem retirada dos materiais da experiência).

Pode-se, por fim, mapear os sinais de controle de alto nível aos sinais de controle reais dos componentes do fluxo de dados da figura 2, como na figura 8 abaixo:

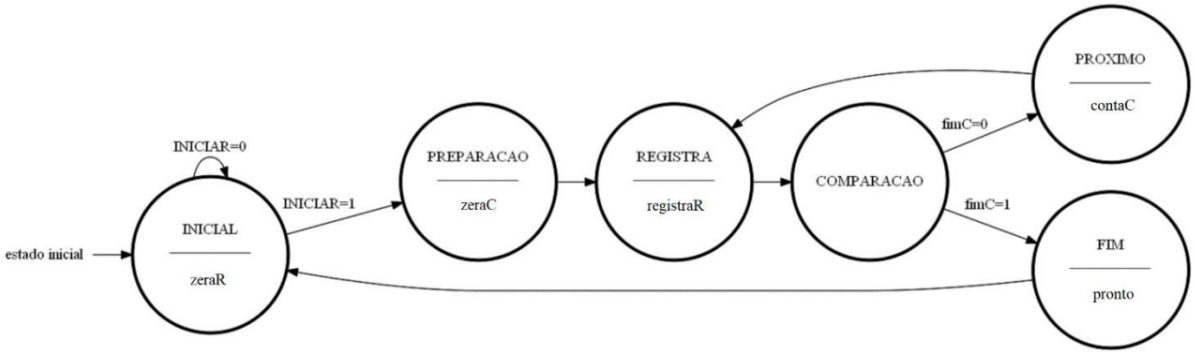


Figura 8 — Máquina de estados de baixo nível de abstração do circuito digital descrito na introdução deste relatório. Nota-se que os sinais de controle do fluxo de dados da figura 2 (*contaC*, *zeraC*, *escreveM*, *registraR* e *zeraR*) estão todos representados, assim como os sinais de condição *fimC* (uma das saídas do fluxo de dados) e *INICIAR* do fluxo de dados. O sinal *pronto* é um sinal de *status* da unidade de controle e indica o fim da contagem do *contador_163*. (imagem retirada dos materiais da experiência).

O arquivo *unidade_controle.vhd* contém uma descrição em VHDL da unidade de controle do circuito digital em desenvolvimento. A descrição pode ser dividida em duas partes: uma combinatória e uma sequencial. A parcela combinatória é responsável pelo cálculo lógico do próximo estado da máquina. A parcela sequencial, por sua vez, realiza a transição do estado atual para o calculado, de maneira síncrona sensível à borda de subida do *clock*.

Seguiu-se, então, com a análise estática e modificação da descrição VHDL, comparando-se a descrição funcional do circuito com o código escrito, como exposto a seguir. O código-fonte completo da unidade de controle (*unidade_controle.vhd*) se encontra no apêndice a.

A figura 9 apresenta a implementação da funcionalidade descrita na introdução deste relatório: *“Depois do acionamento do sinal reset, o circuito deve aguardar o início de sua operação que é realizado com o acionamento do sinal de entrada iniciar”.*

```
42      -- memoria de estado
43      process (clock,reset)
44      begin
45          if reset='1' then
46              Eatual <= inicial;
47          elsif clock'event and clock = '1' then
48              Eatual <= Eprox;
49          end if;
50      end process;
```

Figura 9 — Descrição comportamental da parcela sequencial do circuito. *Eatual* corresponde ao estado atual da máquina, enquanto *Eprox*, ao próximo estado, calculado pelo circuito combinatório. Nota-se que o sinal *reset*, que não foi representado na máquina de estados da figura 8, é assíncrono; *i.e.* retorna a máquina ao estado inicial independente do estado atual. Uma borda de subida de *clock* causa a transição de estados, caso *reset*=’0’.

Na figura 10, é possível identificar as dependências das transições de estados de acordo com os sinais de condição identificados na figura 8.

```
53      Eprox <=
54          inicial     when  Eatual=inicial and iniciar='0' else
55          preparacao  when  Eatual=inicial and iniciar='1' else
56          registra    when  Eatual=preparacao else
57          comparacao  when  Eatual=registra else
58          proximo     when  Eatual=comparacao and fimC='0' else
59          fim         when  Eatual=comparacao and fimC='1' else
60          registra    when  Eatual=proximo else
61          inicial     when  Eatual=fim else
62          inicial;
```

Figura 10 — Descrição da lógica combinatória para cálculo do próximo estado. É possível identificar e relacionar diretamente os sinais de condição da descrição com os representados nas arestas de transição da máquina de estados da figura 8.

```
64      -- logica de saída (maquina de Moore)
65      with Eatual select
66          zeraC <=      '1' when preparacao,
67                        '0' when others;
68
69      with Eatual select
70          zeraR <=      '1' when inicial | preparacao,
71                        '0' when others;
72
73      with Eatual select
74          registraR <=  '1' when registra,
75                        '0' when others;
76
77      with Eatual select
78          contaC <=     '1' when proximo,
79                        '0' when others;
80
81      with Eatual select
82          pronto <=     '1' when fim,
83                        '0' when others;
```

Figura 11 — Descrição dos sinais de controle associados a cada estado (a máquina descrita é de Moore). Analogamente ao caso da figura 9, há uma relação direta entre os valores dos sinais da descrição e os sinais representados dentro de cada estado da figura 8.

Destaca-se a presença de inconsistências relativas aos estados *preparacao* e *proximo*. No diagrama de transição de estados disponibilizado junto ao material da experiência, o estado *preparacao* não possui *zeraR* como um de seus sinais ativos. Na máquina de estados descrita em VHDL, no entanto, constata-se a presença do sinal tanto no estado *inicial*, quanto no estado *preparacao*. Todavia, concluiu-se que há maior corretude na descrição VHDL do que no diagrama de transição de estados, dado que o *registrador_n* deve armazenar o conteúdo das chaves somente no estado *registra*. O estado *proximo*, por sua vez, deveria possuir a transição para o estado *fim*, dado que o sinal *fimC* só é igual a ‘1’ quando *contaC* também o é, o que nunca ocorrerá no estado *comparacao*. Modificou-se, então, a máquina de estados e a codificação para atender à esses requisitos:

```
52      -- logica de proximo estado
53      Eprox <=
54          inicial     when  Eatual=inicial and iniciar='0' else
55          preparacao  when  Eatual=inicial and iniciar='1' else
56          registra    when  Eatual=preparacao else
57          comparacao  when  Eatual=registra else
58          proximo     when  Eatual=comparacao else
59          fim         when  Eatual=proximo and fimC='1' else
60          --fim         when  Eatual=comparacao and fimC='1' else
61          registra    when  Eatual=proximo and fimC='0' else
62          inicial     when  Eatual=fim else
63          inicial;
```

Figura 12 — Descrição da lógica combinatória para cálculo do próximo estado após modificação.

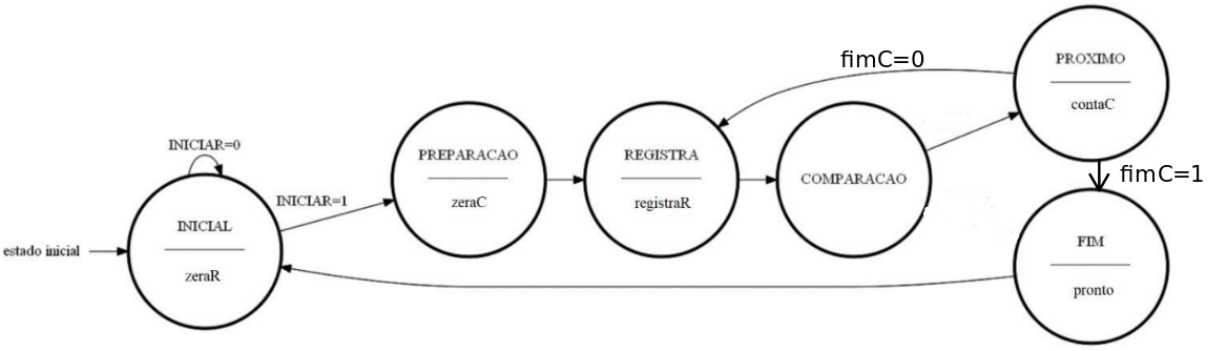


Figura 13 — Máquina de estados de baixo nível de abstração após modificação (transições dependentes do sinal *reset* omitidas).

Finalmente, para fins de depuração, mapeou-se cada estado para um valor binário, como a seguir:

```
85      -- saida de depuracao (db_estado)
86      with Eatual select
87          db_estado <= "0000" when inicial,     -- 0
88                       "0001" when preparacao,  -- 1
89                       "0100" when registra,    -- 4
90                       "0101" when comparacao,  -- 5
91                       "0110" when proximo,     -- 6
92                       "1100" when fim,         -- C
93                       "1111" when others;      -- F
```

Figura 14 — Mapeamento dos estados para valores em binário, usados para a depuração do código.

Considerou-se, portanto, a partir da análise estática, que a codificação da máquina de estados está coerente com a descrição da funcionalidade do circuito.

c. Projeto do Sistema Digital

Tendo-se aferido a descrição lógica do circuito, pode-se prosseguir para o acoplamento do fluxo de dados à unidade de controle. Utilizou-se a figura 2 como base para a interligação dos sinais, ligando-se o sinal *escreveM* ao nível lógico '0' (não há escrita na memória nesta experiência). Abaixo está representado a entidade *circuito_exp3*, contida no arquivo *circuito_exp3.vhd*. O código-fonte completo do componente se encontra no apêndice b.

```
16  entity circuito_exp3 is
17      port (
18          clock       : in std_logic;
19          reset       : in std_logic;
20          iniciar     : in std_logic;
21          chaves      : in std_logic_vector (3 downto 0);
22          pronto      : out std_logic;
23          db_igual    : out std_logic;
24          db_iniciar  : out std_logic;
25          db_contagem : out std_logic_vector (6 downto 0);
26          db_memoria  : out std_logic_vector (6 downto 0);
27          db_chaves   : out std_logic_vector (6 downto 0);
28          db_estado   : out std_logic_vector (6 downto 0)
29      );
30  end entity;
```

Figura 15 — Interface da entidade VHDL do componente *circuito_exp3*, contido no arquivo *circuito_exp3.vhd*. Há uma correlação direta entre os sinais listados acima e os representados na figura 2. Ligou-se os sinais de depuração à saídas de *displays* de sete segmentos (os mesmos da experiência anterior), cujas entradas ligaram-se , respectivamente, aos sinais *chavesIgualMemoria* (fluxo de dados), *iniciar*, *db_contagem* (fluxo de dados), *db_memoria* (fluxo de dados), *db_chaves* (fluxo de dados), *db_estado* (unidade de controle).

Testou-se, então, o componente *circuito_exp3* de acordo com os casos de teste descritos na tabela 2. Nota-se que modificou-se o penúltimo caso de teste a fim de garantir a transição para o estado *fim* levando em conta as alterações realizadas na unidade de controle descritas na seção anterior.

#	Operação	Sinais de controle	Resultado Esperado
c.i.	Condições iniciais	clock=0 reset=0 iniciar=0 chaves=0000	pronto=0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0000 db_estado=0000
1	"Resetar" circuito e observar a saída da memória	reset=1 clock ↑	pronto=0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0000

			db_estado=0000
2	Acionar sinal de clock 5 vezes com iniciar=0	reset=0 iniciar=0 clock ↑ (5x)	(permanece no estado inicial) pronto=0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0000 db_estado=0000
3	Ajustar chaves para 0100, ativar iniciar=1 e acionar clock 1x	chaves=0100 iniciar=1 clock ↑	(muda para estado preparação) pronto=0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0000 db_estado=0000 → 0001
4	Mantém chaves em 0100 e acionar clock 1x	chaves=0100 iniciar=0 clock ↑	(muda para estado registra) pronto=0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0000 db_estado= 0001 → 0100
5	Mantém chaves em 0100 e acionar clock 1x	chaves=0100 iniciar=0 clock ↑	(muda para estado comparação) pronto = 0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0100 db_estado = 0100 → 0101
6	Mantém chaves em 0100 e acionar clock 1x	chaves=0100 clock ↑	(muda para estado próximo) pronto=0 db_igual=0 db_contagem=0000 db_memoria=0001 db_chaves=0100 db_estado=0100 →0110
7	Mantém chaves em 0100 e acionar clock 3x	chaves=0100 clock ↑ (3x)	(passa pelos estados registra, comparação e próximo) pronto=0 db_igual=0 db_contagem=0001 db_memoria=0010 db_chaves=0100
8	Mantém chaves em 0100 e acionar clock 3x	chaves=0100 clock ↑ (3x)	(passa pelos estados registra, comparação e próximo) pronto=0 db_igual=1 db_contagem=0010 db_memoria=0100 db_chaves=0100
9	Mantém chaves em 0100 e acionar clock 9x	chaves=0100 clock ↑ (9x)	(passa 3x pelos estados registra, comparação e próximo) pronto=0 db_igual varia 0-1-0 db_contagem varia 0011-0100-0101 db_memoria varia 1000-0100-0010 db_chaves=0100
10	Ajustar chaves para 0001 e acionar clock 6x	chaves=0001 clock ↑ (6x)	pronto=0 db_igual= 0 → 1 db_contagem= 0110 → 0111 db_memoria= 0001 db_chaves= 0000 → 0001 db_estado= 3x(0110 →0100→0101) →0110
11	Ajustar chaves para 0010 e acionar clock 6x	chaves=0010 clock ↑ (6x)	pronto=0 db_igual= 1 → 0 → 1 db_contagem= 0111 → 1000 → 1001 db_memoria= 0001 → 0010 db_chaves= 0001 → 0010 db_estado= 3x(0110 →0100→0101) →0110
12	Ajustar chaves para 0100 e acionar clock 6x	chaves=0100 clock ↑ (6x)	pronto=0 db_igual= 1 → 0 → 1

			db_contagem= 1001 → 1010 → 1011 db_memoria= 0010 → 0100 db_chaves= 0010 → 0100 db_estado= 3x(0110 →0100→0101) →0110
13	Ajustar chaves para 1000 e acionar clock 6x	chaves=1000 clock ↑ (6x)	pronto=0 db_igual= 1 → 0 → 1 db_contagem= 1011 → 1100 → 1101 db_memoria= 0100 → 1000 db_chaves= 0100 → 1000 db_estado= 3x(0110 →0100→0101) →0110
14	Ajustar chaves para 0000 e acionar clock 3x	chaves=0000 clock ↑ (3x)	pronto=0 db_igual= 1 → 0 db_contagem= 1101 → 1110 db_memoria= 1000 → 0001 db_chaves= 1000 → 0000 db_estado= 0110 →0100→0101 →0110
15	Mantém chaves em 0000 e acionar clock 4x	chaves=0000 clock ↑ (4x)	(passa pelo estado fim) pronto = 0 → 1 db_igual= 0 db_contagem= 1110 → 1111 → 0000 db_memoria= 0001 → 0100 → 0001 db_chaves= 0000 db_estado= 0110 →0100→0101 →0110 →1100
16	Mantém chaves em 0000 e acionar clock	chaves=0000 clock ↑	(termina no estado inicial) pronto = 1 → 0 db_igual= 0 db_contagem= 0000 db_memoria= 0001 db_chaves= 0000 db_estado= 1100 → 0000

Tabela 2 — Plano de Teste para o Circuito da Experiência.

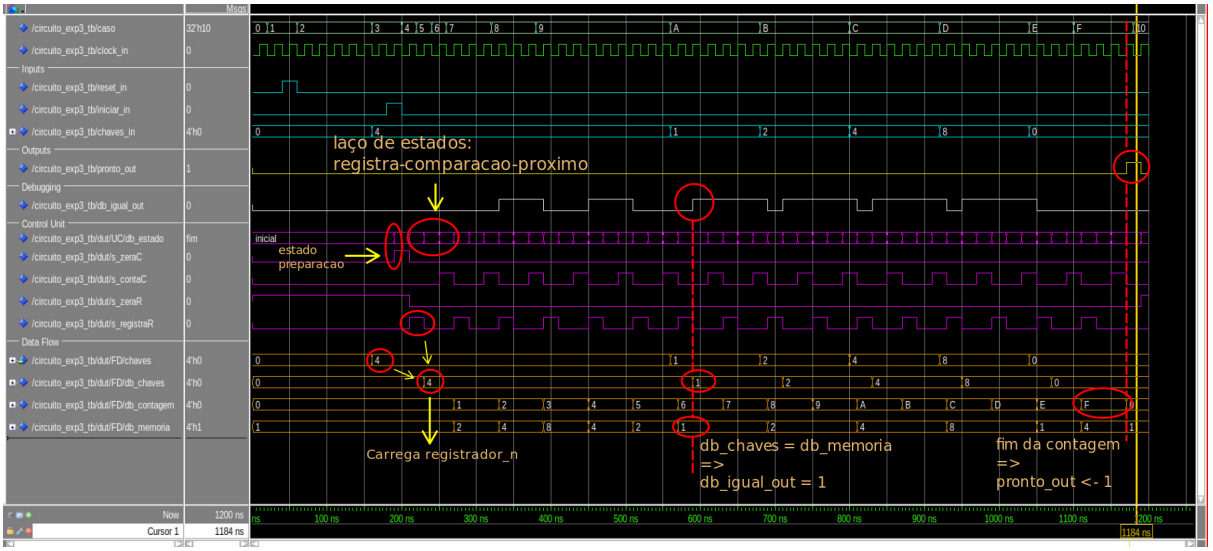


Figura 16 — Formas de onda de simulação do componente *circuito_exp3*.

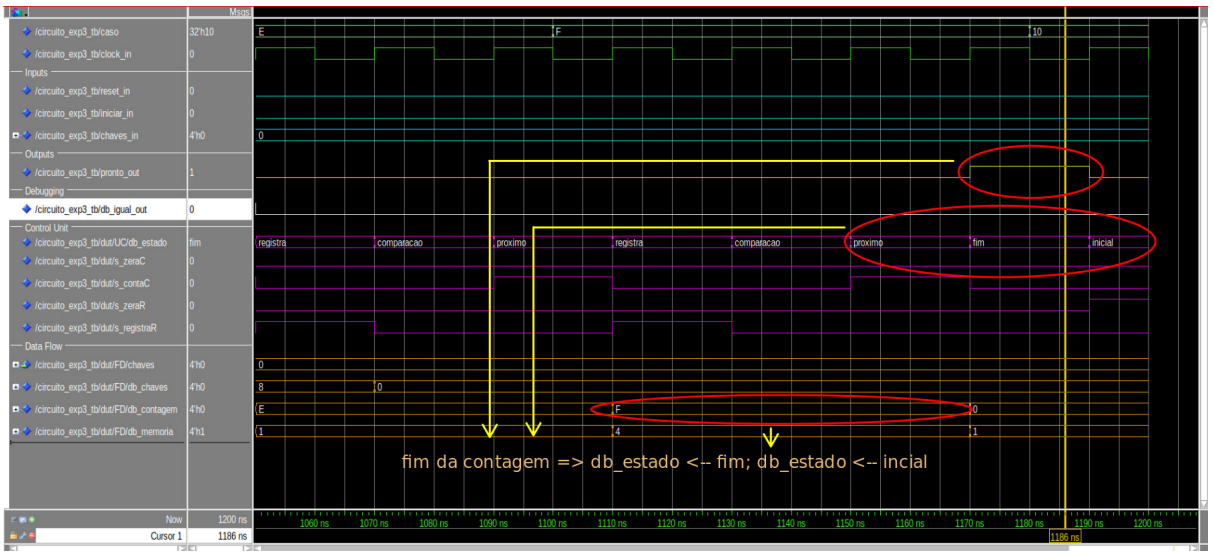


Figura 17 — Formas de onda de simulação detalhadas dos casos finais de teste do componente *circuito_exp3*.

Dadas as modificações citadas anteriormente, o plano de testes seguiu como esperado. No caso de teste 1 reinicia-se o *contador_163* e o *registrador_n*. Nos casos subsequentes, percorre-se a memória e compara-se sua saída com os valores das chaves armazenados no *registrador_n*, verificando-se sempre o *status* do sinal *db_igual*, que acusa igualdade entre os sinais citados. Finalmente, ao fim da contagem, certifica-se a mudança do estado *proximo* para o estado *fim*, e deste para o estado *inicial*.

Abaixo encontra-se um diagrama *RTL* do circuito desenvolvido:

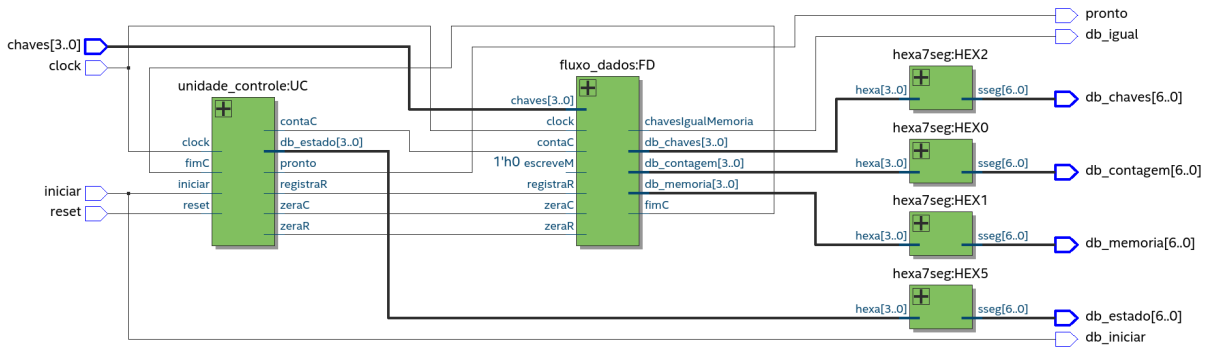


Figura 18 — visão *RTL* (*Register Transfer Level*) do circuito digital simulado.

3. Síntese do circuito

Após a aferição dos componentes e do circuito completo, prosseguiu-se com a designação de pinos da entidade topo (*circuito_exp3*) como apresentado na tabela 4. Antes, alterou-se a interface VHDL entidade *circuito_exp3* a fim de adicionar-se os sinais de depuração *db_zeraC, db_contaC, db_fimC, db_zeraR, db_registraR*.

```vhdl
entity circuito_exp3 is
    port (
        clock       : in std_logic;
        reset       : in std_logic;
        iniciar     : in std_logic;
        chaves      : in std_logic_vector (3 downto 0);
        pronto      : out std_logic;
        db_igual    : out std_logic;
        db_iniciar  : out std_logic;
        db_contagem : out std_logic_vector (6 downto 0);
        db_memoria  : out std_logic_vector (6 downto 0);
        db_chaves   : out std_logic_vector (6 downto 0);
        db_estado   : out std_logic_vector (6 downto 0);
        db_zeraC      : out std_logic;
        db_contaC   : out std_logic;
        db_fimC   : out std_logic;
        db_zeraR    : out std_logic;
        db_registraR: out std_logic
    );
end entity;
```

Figura 19 — Interface da entidade VHDL do componente *circuito_exp3*, contido no arquivo *circuito_exp3.vhd*, com sinais de depuração adicionais: *db_zeraC, db_contaC, db_fimC, db_zeraR, db_registraR*, ligados nos respectivos sinais sem o prefixo *db_* do fluxo de dados.

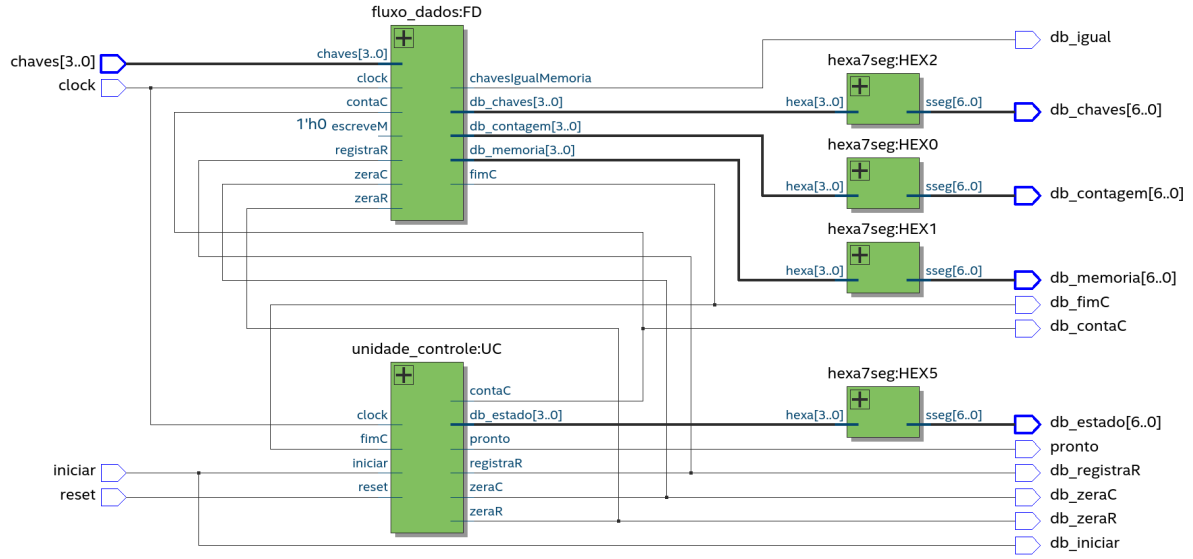
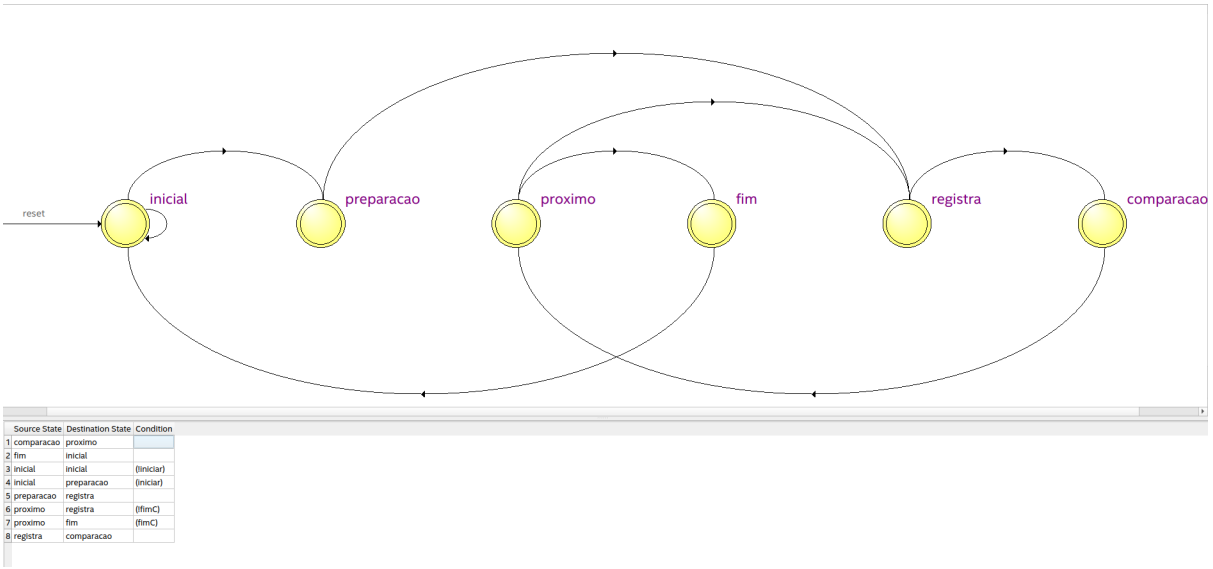


Figura 20 — Visão *Register Transfer Level* do circuito final a ser sintetizado e programado na FPGA.



	Source State	Destination State	Condition
1	comparacao	proximo	
2	fim	inicial	
3	inicial	inicial	(!iniciar)
4	inicial	preparacao	(iniciar)
5	preparacao	registra	
6	proximo	registra	(!fimC)
7	proximo	fim	(fimC)
8	registra	comparacao	

Figura 21 — Máquina de estados obtidas pela funcionalidade *State Machine Viewer*, do software Quartus. Essa máquina equivale à descrita na figura 13.

Sinal	**Pino na placa DE0-CV**	**Pino no FPGA**	**Analog Discovery**
CLOCK	GPIO_0_D0	PIN_N16	StaticIO – Button 0/1
RESET	GPIO_0_D1	PIN_B16	StaticIO – Button 0/1
INICIAR	chave SW0	PIN_U13	
CHAVES(0)	chave SW1	PIN_V13	
CHAVES(1)	chave SW2	PIN_T13	
CHAVES(2)	chave SW3	PIN_T12	
CHAVES(3)	chave SW4	PIN_AA15	
PRONTO	led LEDR0	PIN_AA2	
DB_IGUAL	led LEDR1	PIN_AA1	
DB_INICIAR	led LEDR2	PIN_W2	
DB_ZERAC	led LEDR4	PIN_N2	
DB_CONTAC	led LEDR5	PIN_N1	
DB_FIMC	led LEDR6	PIN_U2	
DB_ZERAR	led LEDR8	PIN_L2	
DB_REGISTRAR	led LEDR9	PIN_L1	
DB_CONTAGEM	display HEX0	PIN_U21 PIN_V21 PIN_W22 PIN_W21 PIN_Y22 PIN_Y21 PIN_AA22	
DB_MEMORIA	display HEX1	PIN_AA20	

		PIN_AB20 PIN_AA19 PIN_AA18 PIN_AB18 PIN_AA17 PIN_U22	
DB_CHAVES	display HEX2	PIN_Y19 PIN_AB17 PIN_AA10 PIN_Y14 PIN_V14 PIN_AB22 PIN_AB21	
DB_ESTADO	display HEX5	PIN_N9 PIN_M8 PIN_T14 PIN_P14 PIN_C1 PIN_C2 PIN_W19	

Tabela 3 — Designação de pinos da placa FPGA DE0-CV.

4. Apêndice

a. Código-fonte *unidade_controle.vhd*

```
1. ------------------------------------------------------------------
2. -- Arquivo   : unidade_controle.vhd
3. -- Projeto   : Experiencia 3 - Projeto de uma unidade de controle
4. ------------------------------------------------------------------
5. -- Descricao : unidade de controle
6. --
7. --             1) codificação VHDL (maquina de Moore)
8. --
9. --             2) definicao de valores da saida de depuracao
10. --                db_estado
11. --
12. ------------------------------------------------------------------
13. -- Revisoes  :
14. --     Data        Versao  Autor             Descricao
15. --     20/01/2022  1.0     Edson Midorikawa  versao inicial
16. --     22/01/2023  1.1     Edson Midorikawa  revisao
17. ------------------------------------------------------------------
18. --
19. library ieee;
20. use ieee.std_logic_1164.all;
21.
22. entity unidade_controle is
23.     port (
24.         clock     : in  std_logic;
25.         reset     : in  std_logic;
26.         iniciar   : in  std_logic;
27.         fimC      : in  std_logic;
28.         zeraC     : out std_logic;
29.         contaC    : out std_logic;
30.         zeraR     : out std_logic;
31.         registraR : out std_logic;
32.         pronto    : out std_logic;
33.         db_estado : out std_logic_vector(3 downto 0)
34.     );
35. end entity;
36.
37. architecture fsm of unidade_controle is
38.     type t_estado is (inicial, preparacao, registra, comparacao, proximo, fim);
39.     signal Eatual, Eprox: t_estado;
40. begin
41.
42.     -- memoria de estado
43.     process (clock,reset)
```

```
44.     begin
45.         if reset='1' then
46.             Eatual <= inicial;
47.         elsif clock'event and clock = '1' then
48.             Eatual <= Eprox;
49.         end if;
50.     end process;
51.
52.     -- logica de proximo estado
53.     Eprox <=
54.         inicial     when  Eatual=inicial and iniciar='0' else
55.         preparacao  when  Eatual=inicial and iniciar='1' else
56.         registra    when  Eatual=preparacao else
57.         comparacao  when  Eatual=registra else
58.         proximo     when  Eatual=comparacao and fimC='0' else
59.         fim         when  Eatual=comparacao and fimC='1' else
60.         registra    when  Eatual=proximo else
61.         inicial     when  Eatual=fim else
62.         inicial;
63.
64.     -- logica de saída (maquina de Moore)
65.     with Eatual select
66.         zeraC <=      '1' when preparacao,
67.                       '0' when others;
68.
69.     with Eatual select
70.         zeraR <=      '1' when inicial | preparacao,
71.                       '0' when others;
72.
73.     with Eatual select
74.         registraR <=  '1' when registra,
75.                       '0' when others;
76.
77.     with Eatual select
78.         contaC <=     '1' when proximo,
79.                       '0' when others;
80.
81.     with Eatual select
82.         pronto <=     '1' when fim,
83.                       '0' when others;
84.
85.     -- saida de depuracao (db_estado)
86.     with Eatual select
87.         db_estado <= "0000" when inicial,     -- 0
88.                      "0001" when preparacao,  -- 1
```

```
                    "0100" when registra,     -- 4
                    "0101" when comparacao,   -- 5
                    "0110" when proximo,      -- 6
                    "1100" when fim,          -- C
                    "1111" when others;       -- F

end architecture fsm;
```

b. Código-fonte *circuito_exp3.vhd*

```
-------------------------------------------------------------------
-- Arquivo   : circuito_exp3.vhd
-- Projeto   : Experiencia 3 - Projeto de uma unidade de controle
-------------------------------------------------------------------
-- Descricao : implementacao em vhdl da entidade topo do projeto
-------------------------------------------------------------------
-- Revisoes  :
--     Data        Versao  Autor                         Descricao
--     28/01/2022  1.0     Thiago Souza                  versao inicial
--      29/01/2022  1.1     Igor Pontes Tresolavy         versao inicial
-------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity circuito_exp3 is
    port (
        clock       : in std_logic;
        reset       : in std_logic;
        iniciar     : in std_logic;
        chaves      : in std_logic_vector (3 downto 0);
        pronto      : out std_logic;
        db_igual    : out std_logic;
        db_iniciar  : out std_logic;
        db_contagem : out std_logic_vector (6 downto 0);
        db_memoria  : out std_logic_vector (6 downto 0);
        db_chaves   : out std_logic_vector (6 downto 0);
        db_estado   : out std_logic_vector (6 downto 0)
    );
end entity;

architecture toplevel of circuito_exp3 is

    component unidade_controle
        port (
          clock     : in  std_logic;
```

```
37.           reset     : in  std_logic;
38.           iniciar   : in  std_logic;
39.           fimC      : in  std_logic;
40.           zeraC     : out std_logic;
41.           contaC    : out std_logic;
42.           zeraR     : out std_logic;
43.           registraR : out std_logic;
44.           pronto    : out std_logic;
45.           db_estado : out std_logic_vector(3 downto 0)
46.       );
47.   end component;
48.
49.   component fluxo_dados is
50.       port (
51.           clock               : in  std_logic;
52.           zeraC               : in  std_logic;
53.           contaC              : in  std_logic;
54.           escreveM            : in  std_logic;
55.           zeraR               : in  std_logic;
56.           registraR           : in  std_logic;
57.           chaves              : in  std_logic_vector (3 downto 0);
58.           chavesIgualMemoria  : out std_logic;
59.           fimC                : out std_logic;
60.           db_contagem         : out std_logic_vector (3 downto 0);
61.           db_memoria          : out std_logic_vector (3 downto 0);
62.           db_chaves           : out std_logic_vector (3 downto 0)
63.       );
64.   end component;
65.
66.   component hexa7seg is
67.       port (
68.           hexa : in  std_logic_vector(3 downto 0);
69.           sseg : out std_logic_vector(6 downto 0)
70.       );
71.   end component hexa7seg;
72.
73.   signal s_zeraC : std_logic;
74.   signal s_contaC : std_logic;
75.   signal s_zeraR : std_logic;
76.   signal s_registraR : std_logic;
77.   signal s_fimC : std_logic;
78.
79.   signal s_db_contagem : std_logic_vector(3 downto 0);
80.   signal s_db_memoria : std_logic_vector(3 downto 0);
81.   signal s_db_chaves : std_logic_vector(3 downto 0);
```

```
82.     signal s_db_estado : std_logic_vector(3 downto 0);
83.
84. begin
85.
86.     FD: fluxo_dados
87.         port map (
88.             clock => clock,
89.             zeraC => s_zeraC,
90.             contaC => s_contaC,
91.             escreveM => '0',
92.             zeraR => s_zeraR,
93.             registraR => s_registraR,
94.             chaves => chaves,
95.             chavesIgualMemoria => db_igual,
96.             fimC => s_fimC,
97.             db_contagem => s_db_contagem,
98.             db_memoria => s_db_memoria,
99.             db_chaves => s_db_chaves
100.        );
101.
102.     UC: unidade_controle
103.         port map (
104.             clock => clock,
105.             reset => reset,
106.             iniciar => iniciar,
107.             fimC => s_fimC,
108.             zeraC => s_zeraC,
109.             contaC => s_contaC,
110.             zeraR => s_zeraR,
111.             registraR => s_registraR,
112.             pronto => pronto,
113.             db_estado => s_db_estado
114.         );
115.
116.     HEX2: hexa7seg
117.         port map (
118.             hexa => s_db_chaves,
119.             sseg => db_chaves
120.         );
121.
122.     HEX0: hexa7seg
123.         port map (
124.             hexa => s_db_contagem,
125.             sseg => db_contagem
126.         );
```

```
127.
128.     HEX1: hexa7seg
129.         port map (
130.             hexa => s_db_memoria,
131.             sseg => db_memoria
132.         );
133.
134.     HEX5: hexa7seg
135.         port map (
136.             hexa => s_db_estado,
137.             sseg => db_estado
138.         );
139.
140.     db_iniciar <= iniciar;
141.
142. end toplevel;
143.
```