Implementação de um Sistema Digital em VHDL

Cronômetro para Jogos de Basquete

ESPECIFICAÇÃO DO CONTADOR

1. **Introdução**: O objetivo deste trabalho é especificar um módulo que deverá ser implementado em hardware pelos alunos, desde a escrita em VHDL do circuito, até a sua prototipação.

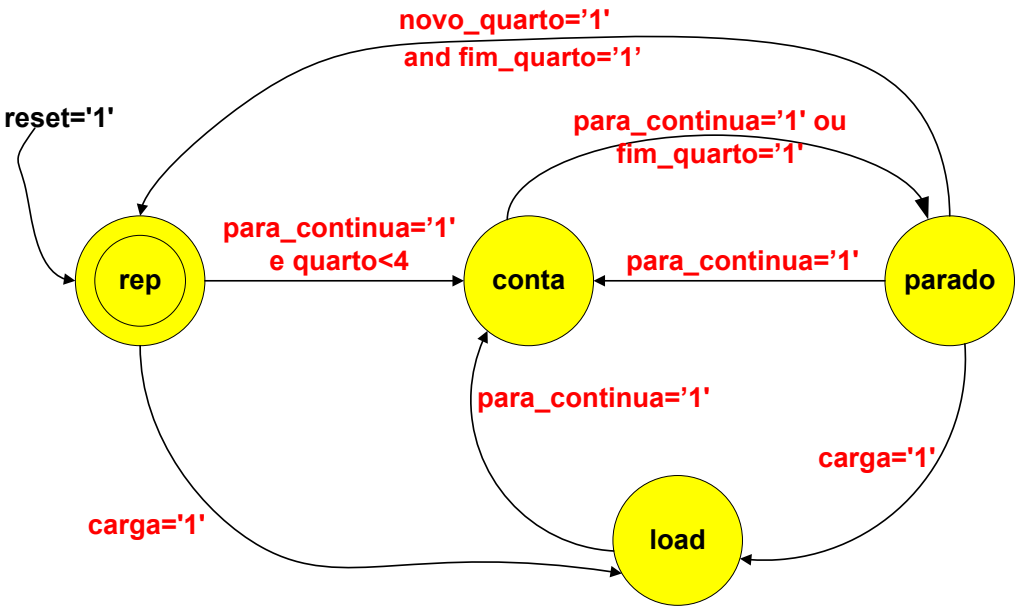
2. **Pressupostos**: O cronômetro deve operar sobre a plataforma Nexys2 disponível em laboratório, com todas as restrições que esta plataforma impõe, o que inclui:
   2.1. Só existe uma fonte de relógio na plataforma, um cristal de 50MHz;
   2.2. Somente podem ser usados os dispositivos de entrada e saída nativos da placa, quais sejam: para saída de dados, existem 4 mostradores de sete segmentos e 8 diodos emissores de luz (LEDs); para entrada de dados existem apenas 8 chaves deslizantes e quatro botões (push Button).

3. **Especificação do Cronômetro**: Para servir a jogos de basquete o cronômetro deve:
   3.1. Mostrar o jogo como sendo composto de 4 **quartos** de exatamente 15 minutos (quartos são numerados de 1 a 4). O tempo que falta para terminar um **quarto** de jogo em curso é visualizável em notação **decimal** com precisão de centésimos de segundo (a conversão de hexadecimal para decimal será feita no arquivo *top,* logo implementar internamente contadores binários). Os contadores de minutos, segundos e centésimos evoluem em ordem **decrescente**. O contador de quartos evolui em ordem **crescente**. Assim, na inicialização o cronômetro deve marcar **15min 00s 00cent**. Ao comandar o início da contagem do tempo do quarto, a primeira mudança no cronômetro ocorre exatamente 1 centésimo de segundo após o início do quarto, com o cronômetro passando para 14min 59s 99cent.
   3.2. Possuir 4 botões de controle:
      - **reset**: reinicializa o contador, com valores **quarto=1, tempo=15min 00s 00cent**. O cronômetro inicia parado.
      - **para_continua**: uma vez pressionado ativa a contagem. Pressionado novamente para imediatamente o cronômetro, congelando o tempo de jogo.
      - **novo_quarto**: estando o jogo parado **e** ao final de um quarto, se pressionado leva para o início do próximo quarto, deixando o cronômetro congelado no tempo de início do quarto, 15min 00s 00cent.
      - **carga**: para considerar eventos especiais durante um jogo, que possam implicar o reinício deste em qualquer contagem de quarto, minutos e segundos, deve haver suporte para isto. Este botão quando acionado com o cronômetro parado (no estado após o reset ou no estado *parado*), lê um conjunto de entradas que especificam novos valores de quarto, minutos e segundos, assumindo que os centésimos reiniciarão em 00.

3.3. Toda vez que um quarto for concluído, o cronômetro automaticamente para. O reinício do mesmo pode ser obtido apertando o botão *novo_quarto*, seguido do botão *para_continua*.

4. **Estratégia de Implementação**: Sugere-se criar uma estrutura com um Bloco de Dados e um Bloco de Controle.
   - O Bloco de Controle conterá a uma máquina de estados.
   - O Bloco de Dados conterá os vários contadores (process) e as atribuições concorrentes necessárias (como detecção de fim de quarto, contagem de algum contador igual a um dado valor, etc).

   Conceitualmente, como sugestão de máquina de controle (não necessariamente os estados precisam ser estes), tem-se:



- *fim_quarto*: sinal interno, gerado por atribuição concorrente aos processos de contagem, indicando que se atingiu o término de um quarto (ou seja, a contagem chegou a 00min 00s 00cent).

5. **Módulos da Implementação:**
   - Um divisor de relógio que gere um relógio interno de 1/100 segundos ➔ ck_1_cent. Para a prototipação deve-se contar 500.000 ciclos de relógio para contabilizar 1 centésimo de segundo. Como as escalas de tempo são muito diferentes, esta contagem de 500.000 ciclos leva muito tempo de simulação. Assim, pode-se alterar a contagem de 500.000 para digamos 50 ou 10, apenas visando acelerar a validação do projeto por simulação (para a prototipação no FPGA o valor 500.000 é obrigatório).
   - Sincronizadores das teclas com o relógio ck_1_cent (ver abaixo).
   - Máquina de controle, controlada pelo relógio ck_1_cent – utilizar os sinais de controle oriundos dos sincronizadores.
   - Atribuição concorrente para detecção de fim_quarto.
   - Um contador de centésimos circular, controlado pelo relógio ck_1_cent – conta de 0 a 99 em ordem decrescente com módulo de contagem igual a 100 (0, 99, 98, …). Sua contagem é habilitada cada vez que o cronômetro está no estado de contagem (conta na máquina de estados acima) e não chegou ao final do quarto.

Importante, deve-se verificar se o estado é de carga, para realizar uma inicialização com valor externo.

- Um contador de segundos circular, controlado pelo relógio ck_1_cent – conta de 0 a 59 em ordem decrescente com módulo de contagem igual a 60 (0, 59, 58, ...). Sua contagem é habilitada cada vez que o cronômetro está no estado de contagem, os centésimos estão com valor zero e não chegou ao final do quarto. Importante, deve-se verificar se o estado é de carga, para realizar uma inicialização com valor externo.
- Um contador de minutos, controlado pelo relógio ck_1_cent – conta de 15 (0xF) a 0 em ordem decrescente (15, 14, 13,...). Sua contagem é habilitada cada vez que o cronômetro está no estado de contagem, os segundos e centésimos estão com valor zero e não chegou ao final do quarto. Importante, deve-se verificar se o estado é de carga, para realizar uma inicialização com valor externo. Notar que quando houver condição de inicialização este contador recebe o valor 15 (0xF), e não 0.
- Um contador de quartos, que conta de 1 a 4, em ordem crescente. No reset seu valor é 0, e estando no estado parado e pressionando-se novo_quarto seu valor é incrementado. Recomenda-se usar três bits (internamente), pois quando o quarto chegar a '100' significa que os 4 quartos já passaram ('00', '01', '10', '11), devendo assim o cronômetro parar e só sair por reset ou carga de valor (ativação do sinal carga).

6. Entidade externa

Utilizar obrigatoriamente esta interface externa (para posterior compatibilidade com o circuito top que será prototipado como topo da hierarquia no FPGA da placa Nexys2):

```
entity crono_bskt is
    Port ( clock :          in  STD_LOGIC;
           reset :          in  STD_LOGIC;

           carga :          in  STD_LOGIC;
           para_continua :  in  STD_LOGIC;
           novo_quarto :    in  STD_LOGIC;

           c_quarto :       in  STD_LOGIC_VECTOR (1 downto 0);
           c_minutos :      in  STD_LOGIC_VECTOR (3 downto 0);
           c_segundos :     in  STD_LOGIC_VECTOR (5 downto 0);

           centesimos :     out  STD_LOGIC_VECTOR (6 downto 0);
           segundos :       out  STD_LOGIC_VECTOR (5 downto 0);
           minutos :        out  STD_LOGIC_VECTOR (3 downto 0);
           quarto :         out  STD_LOGIC_VECTOR (1 downto 0)
         );
end crono_bskt;
```

7. Sincronizadores

Os 3 sinais de controle externos (carga, para_continua, novo_quarto) podem ficar ativos por muitos centésimos de segundo, pois são oriundos de uma tecla pressionada por um ser humano. Para compatibilizar estes eventos "lentos" com a velocidade de

operação do circuito (baseado em um relógio com 20 bilionésimos de segundo de período, ou 20ns), são necessárias adaptações ao circuito, Para isto, pode-se usar o circuito simples dado a seguir, que detecta uma borda. O par entidade-arquitetura representa o circuito simples de sincronização e deve ser incluído no projeto.

```
--------------------------------------------------------------------------------
-- DETECTOR DE BORDA
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
library work;

entity edge_detector is
        port(   clock           : in    std_logic;
                reset           : in    std_logic;
                din             : in    std_logic;              -- Input data
                rising          : out   std_logic               -- Edge detected
            );
end edge_detector;

architecture edge_detector of edge_detector is
        signal din_reg          : std_logic;
begin

   rising <= '1' when din_reg = '0' and din = '1' else '0';

   process (clock,reset)
   -- Este process implementa um flip-flop sensível à borda de
   -- subida do clock
   begin
          if reset='1' then
                 din_reg <= '0';
          elsif clock'event and clock='1' then
                  din_reg <= din;
          end if;
   end process;

end edge_detector;
```

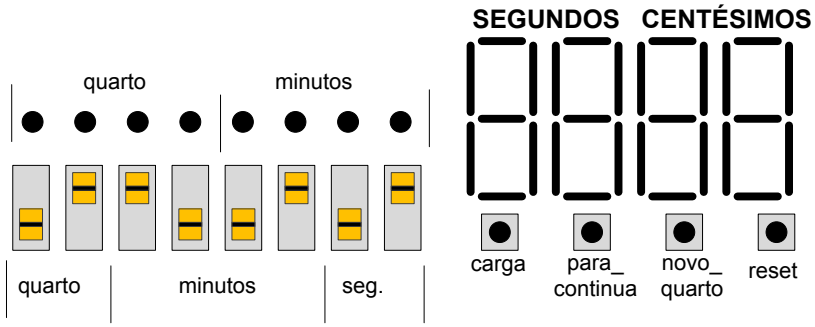
Este módulo amostra no relógio de referência o valor de entrada. Quando o valor armazenado for ‘0’ e valor atual for ‘1’, significa que o valor de entrada fez uma transição de ‘0’ para ‘1’, por isto este módulo é chamado de detector de borda.

7.1. Utilização do sincronizador

```
   a1: entity work.edge_detector  port map(clock=>ck_1_cent, reset=>reset,
                 din=>carga, rising=>carga_int);

   a2: entity work.edge_detector  port map(clock=>ck_1_cent, reset=>reset,
                 din=> para_continua, rising=> para_continua_int);

   a3: entity work.edge_detector  port map(clock=>ck_1_cent, reset=>reset,
                  din=>novo_quarto,   rising=>novo_quarto_int);
```

Estas três instanciações do detector de borda podem ser feitas em qualquer ponto do código. O objetivo delas é gerar um pulso com duração de apenas um ciclo de relógio nos sinais internos a cada tecla apertada por tempo “longo” (centésimos de segundo), produzindo os sinais internos *carga_int, para_continua_int, novo_quarto_int*, sincronizados com o *relógio ck_1_cent* e com duração adequada.

ESPECIFICAÇÃO DO MÓDULO TOP



- 4 leds mais significativos: exibem o quarto, da seguinte forma “0001” quando o quarto for igual a 0, “0010” quando o quarto for 1, e assim por diante.
- 4 leds menos significativos: exibem os minutos, por exemplo, se minutos igual a 9 será exibido “1001”.
- 2 displays de sete-segmentos mais significativos exibem os segundos em decimal (de 59 a 0).
- 2 displays de sete-segmentos menos significativos exibem os centésimos de segundo em decimal (de 99 a 0).
- 2 chaves indicam o quarto que deve ser inicializado. Por exemplo, chave em “00” é quarto 0, “01” quarto 0, “10” quarto 2, “11” quarto 3.
- 4 chaves indicam os minutos que devem ser inicializados,
- 2 chaves indicam os segundos. Codificação: “00” inicializa 0 segundos, “01” inicializa 15 segundos, “10” inicializa 30 segundos e “11” inicializa 45 segundos.
- Os botões são utilizados para controle, na seguinte ordem: carga, para_continua, novo_quarto, reset.

```
entity top_cr_bskt is
    Port ( clock : in  STD_LOGIC;

           --- 4 botões push-buttons    ---------
           carga :          in  STD_LOGIC;
           para_continua :  in  STD_LOGIC;
           novo_quarto :    in  STD_LOGIC;
           reset :          in  STD_LOGIC;

           --- valores de carga - 8 dip-switches ---------
           c_quarto : in  STD_LOGIC_VECTOR (1 downto 0);
           c_minutos : in  STD_LOGIC_VECTOR (3 downto 0);
           c_segundos : in  STD_LOGIC_VECTOR (1 downto 0);

           -- 4 displays de sete-segmentos para segundos e centésimos ----
           DSPL_cent_seg : out  STD_LOGIC_VECTOR (7 downto 0);
           anodo : out  STD_LOGIC_VECTOR (3 downto 0);

           --- 8 leds, para indicar os minutos e quarto --------
           minutos : out  STD_LOGIC_VECTOR (3 downto 0);
           quarto_led : out  STD_LOGIC_VECTOR (3 downto 0));
end top_cr_bskt;
```

Dica para o VHDL

Os valores dos contadores de centésimo e de segundo são valores em hexadecimal que variam de 0 a 99 (0x63 em hexa) e de 0 a 59 (0x3B em hexa), respectivamente. Pode-se usar uma memória ROM que converte o número em hexadecimal para dois dígitos decimais equivalentes cada um representado em quatro bits.

Entre a **`architecture`** e o **`begin`** insere-se a seguinte declaração:

```
type ROM is array (0 to 99) of std_logic_vector (7 downto 0);
constant Conv_to_BCD : ROM:=(
         "00000000", "00000001", "00000010", "00000011", "00000100",
         "00000101", "00000110", "00000111", "00001000", "00001001",
         "00010000", "00010001", "00010010", "00010011", "00010100",
         "00010101", "00010110", "00010111", "00011000", "00011001",
         "00100000", "00100001", "00100010", "00100011", "00100100",
         "00100101", "00100110", "00100111", "00101000", "00101001",
         "00110000", "00110001", "00110010", "00110011", "00110100",
         "00110101", "00110110", "00110111", "00111000", "00111001",
         "01000000", "01000001", "01000010", "01000011", "01000100",
         "01000101", "01000110", "01000111", "01001000", "01001001",
         "01010000", "01010001", "01010010", "01010011", "01010100",
         "01010101", "01010110", "01010111", "01011000", "01011001",
         "01100000", "01100001", "01100010", "01100011", "01100100",
         "01100101", "01100110", "01100111", "01101000", "01101001",
         "01110000", "01110001", "01110010", "01110011", "01110100",
         "01110101", "01110110", "01110111", "01111000", "01111001",
         "10000000", "10000001", "10000010", "10000011", "10000100",
         "10000101", "10000110", "10000111", "10001000", "10001001",
         "10010000", "10010001", "10010010", "10010011", "10010100",
         "10010101", "10010110", "10010111", "10011000", "10011001"
         );
```

E depois no código do top:

segundos_BCD <= Conv_to_BCD(CONV_INTEGER(segundos));

Assume-se aqui que segundos é um sinal do tipo std_logic_vector de 7 bits. A função CONV_INTEGER recebe (no caso) um std_logic_vector como parâmetro e retorna o inteiro equivalente, que é então usado como índice para acesso ao vetor constante de conversão de inteiro para formato BCD (decimal codificado em binário, em inglês *binary coded decimal*) em 8 bits. Por exemplo, suponha que o contador *segundos* contenha **0x0F**. Sua conversão para inteiro (***CONV_INTEGER***) resulta em **15**. Na posição 15 da ROM há "00010101", o que resulta nos dígitos decimais ‘1’ (“0001”) e ‘5’ (“0101”), cada um representado em 4 bits.

LABORG

Parte 5 – Projeto de um circuito digital de média complexidade

Fernando Gehm Moraes

Ney Laert Vilar Calazans

21/abril/2013

Sumário

- Introdução
- Especificação
- Trabalhos intermediários
- Integração

Introdução

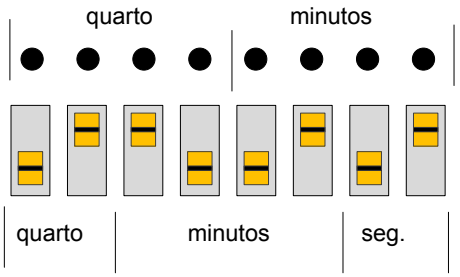
- **O objetivo deste trabalho é especificar um módulo que deverá ser implementado em hardware pelos alunos, desde a escrita em VHDL do circuito, até a sua prototipação**

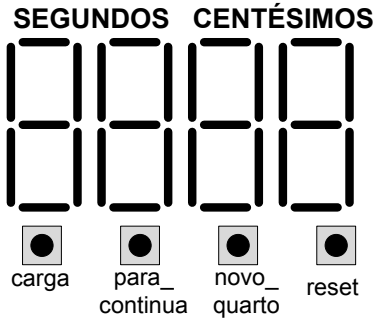
- **Pressupostos:** o cronômetro deve operar sobre a plataforma Nexys2 disponível em laboratório, com todas as restrições que esta plataforma impõe, o que inclui:
  - Só existe uma fonte de relógio na plataforma, um cristal de 50MHz
  - Somente podem ser usados os dispositivos de entrada e saída nativos da placa, quais sejam: para saída de dados, existem 4 mostradores de sete segmentos e 8 diodos emissores de luz (LEDs); para entrada de dados existem apenas 8 chaves deslizantes e quatro botões (push Button)

Sumário

- Introdução
- Especificação
- Trabalhos intermediários
- Integração

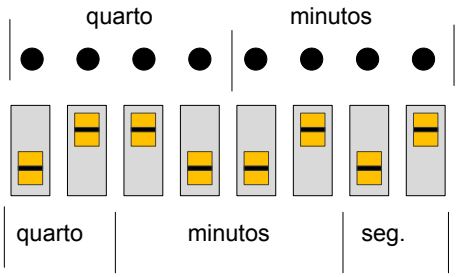
Especificação

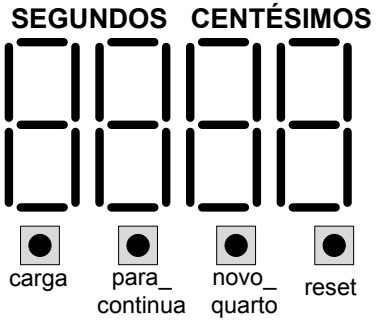




- Mostrar o jogo como sendo composto de 4 quartos de exatamente 15 minutos (quartos são numerados de 1 a 4)
- O tempo que falta para terminar um quarto de jogo em curso é visualizável em notação **decimal** com precisão de **centésimos** de segundo
  (a conversão de hexadecimal para decimal será feita no arquivo *top,* logo implementar contadores binários)
- Os contadores de minutos, segundos e centésimos evoluem em ordem **decrescente**
- O contador de quartos evolui em ordem **crescente**
- Na inicialização o cronômetro deve marcar **1q 15min 00s 00cent**. Ao comandar o início da contagem do tempo do quarto, a primeira mudança no cronômetro ocorre exatamente 1 centésimo de segundo após o início do quarto, com o cronômetro passando para **1q 14min 59s 99cent**

Especificação





- O hardware possui 4 botões de controle:
  - **reset**: reinicializa o contador, com valores **1q 15min 00s 00cent**. O cronômetro inicia parado
  - **para_continua**: uma vez pressionado ativa a contagem, pressionado novamente para imediatamente o cronômetro
  - **novo_quarto**: estando o jogo parado **E** ao final de um quarto, se pressionado leva para o início do próximo quarto, deixando o cronômetro congelado no tempo de início do quarto, 15min 00s 00cent
  - **carga**: quando acionado com o cronômetro **parado** (no estado após o reset ou no estado *parado*), lê um conjunto de entradas que especificam novos valores de quarto, minutos e segundos, assumindo que os centésimos reiniciarão em 00

Especificação

- Toda vez que um quarto for **concluído**, o cronômetro automaticamente para. O reinício do mesmo pode ser obtido apertando o botão *novo_quarto*, seguido do botão *para_continua*

Sumário

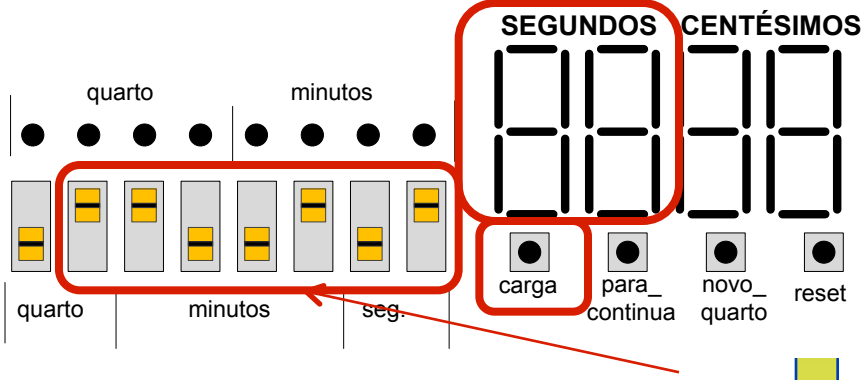
- Introdução
- Especificação
- Trabalhos intermediários
- Integração

Aula 1

1. **Contador para dividir o *clock*, obtendo 1 segundo, fazendo um led qualquer piscar. No arquivo UCF:**
   - **Entradas: clock e reset**
   - **Saída: um led qualquer**

2. **Concluindo a atividade, continuar com as atividades descritas nas próximas lâminas**

Aula 2



1. **Contador de segundo de 99 a 00**
   - **Este contador deve usar o *clock* da aula 1**
   - **Ao se pressionar no valor carga deve-se inicializar o contador com os valores das 7 chaves da direita**
   - **A exibição dos valores de segundo devem usar o display de sete segmentos**

2. **Estrutura do código VHDL**
   - **Divisor com precisão de 1 segundo**
   - **Contador de 59 a 0 com condição de carga externa**
   - **Conversão de hexa para decimal (código na próxima lâmina)**

     segundos_BCD <= Conv_to_BCD(CONV_INTEGER(segundos));
   - **Definição dos dígitos que irão para o driver do display (dois dos dígitos devem ficar apagados)**
   - **Instanciação do driver do display de sete segmentos**

3. **Concluindo a atividade, continuar com as atividades de escrita do cronômetro de basquete**
   - » Esperado ao final da aula 2: contador executando na placa, com precisão de segundo

```
type ROM is array (0 to 99) of std_logic_vector (7 downto 0);
constant Conv_to_BCD : ROM:=(
              "00000000", "00000001", "00000010", "00000011", "00000100",
              "00000101", "00000110", "00000111", "00001000", "00001001",
              "00010000", "00010001", "00010010", "00010011", "00010100",
              "00010101","00010110", "00010111", "00011000", "00011001",
              "00100000", "00100001", "00100010", "00100011", "00100100",
              "00100101", "00100110", "00100111", "00101000", "00101001",
              "00110000", "00110001", "00110010", "00110011", "00110100",
              "00110101", "00110110", "00110111", "00111000", "00111001",
              "01000000", "01000001", "01000010", "01000011", "01000100",
              "01000101", "01000110", "01000111", "01001000", "01001001",
              "01010000", "01010001", "01010010", "01010011", "01010100",
              "01010101", "01010110", "01010111", "01011000", "01011001",
              "01100000", "01100001", "01100010", "01100011", "01100100",
              "01100101", "01100110", "01100111", "01101000", "01101001",
              "01110000", "01110001", "01110010", "01110011", "01110100",
              "01110101", "01110110", "01110111", "01111000", "01111001",
              "10000000", "10000001", "10000010", "10000011", "10000100",
              "10000101", "10000110", "10000111", "10001000", "10001001",
              "10010000", "10010001", "10010010", "10010011", "10010100",
              "10010101", "10010110", "10010111", "10011000", "10011001"
              );
```

Supondo segundos= 0x0F

segundos_BCD <= Conv_to_BCD(CONV_INTEGER(segundos));

O retorno de *Conv_to_BCD* será 00010101 (1 5)

UCF para o clock, reset, leds..... (adaptem os nomes “switch” e “led” para os sinais utilizados na implementação

```
# pinos para as entradas e saídas

NET "clock"          LOC = "b8"   ;
NET "carga"          LOC = "h13"  ;
NET "para_continua"  LOC = "e18"  ;
NET "novo_quarto"    LOC = "d18"  ;
NET "reset"          LOC = "b18"  ;

NET "SWITCH<0>"    LOC = "r17"  ;
NET "SWITCH<1>"    LOC = "n17"  ;
NET "SWITCH<2>"    LOC = "l13"  ;
NET "SWITCH<3>"    LOC = "l14"  ;
NET "SWITCH<4>"    LOC = "k17"  ;
NET "SWITCH<5>"    LOC = "k18"  ;
NET "SWITCH<6>"    LOC = "h18"  ;
NET "SWITCH<7>"    LOC = "g18"  ;

NET "anodo<0>"  LOC = "f17"  ;
NET "anodo<1>"  LOC = "h17"  ;
NET "anodo<2>"  LOC = "c18"  ;
NET "anodo<3>"  LOC = "f15"  ;

NET "DSPL_cent_seg<0>"  LOC = "c17"  ;
NET "DSPL_cent_seg<1>"  LOC = "h14"  ;
NET "DSPL_cent_seg<2>"  LOC = "j17"  ;
NET "DSPL_cent_seg<3>"  LOC = "g14"  ;
NET "DSPL_cent_seg<4>"  LOC = "d16"  ;
NET "DSPL_cent_seg<5>"  LOC = "d17"  ;
NET "DSPL_cent_seg<6>"  LOC = "f18"  ;
NET "DSPL_cent_seg<7>"  LOC = "l18"  ;

NET "LED<0>"  LOC = "j14"  ;
NET "LED<1>"  LOC = "j15"  ;
NET "LED<2>"  LOC = "k15"  ;
NET "LED<3>"  LOC = "k14"  ;
NET "LED<4>"  LOC = "e16"  ;
NET "LED<5>"  LOC = "p16"  ;
NET "LED<6>"  LOC = "e4"  ;
NET "LED<7>"  LOC = "p4"  ;
```

Sumário

- Introdução
- Especificação
- Trabalhos intermediários
- Integração

INTEGRAÇÃO – Sugestão de estrutura de código

1. **divisor de relógio para 1/100 seg. Para a prototipação deve-se contar o número de ciclos de relógio necessários para contabilizar 1 centésimo de segundo. Para a simulação dividir por no máximo 10.**

2. **3 sincronizadores das teclas com o relógio ck_1_cent: carga / para_continua / novo_quarto**

```
a1: entity work.edge_detector port map (clock=>ck_1_cent,
reset=>reset, din=> carga, rising=>carga_int);
```

3. **Máquina de controle, controlada pelo relógio ck_1_cent**
   - » utilizar os sinais de controle oriundos dos sincronizadores

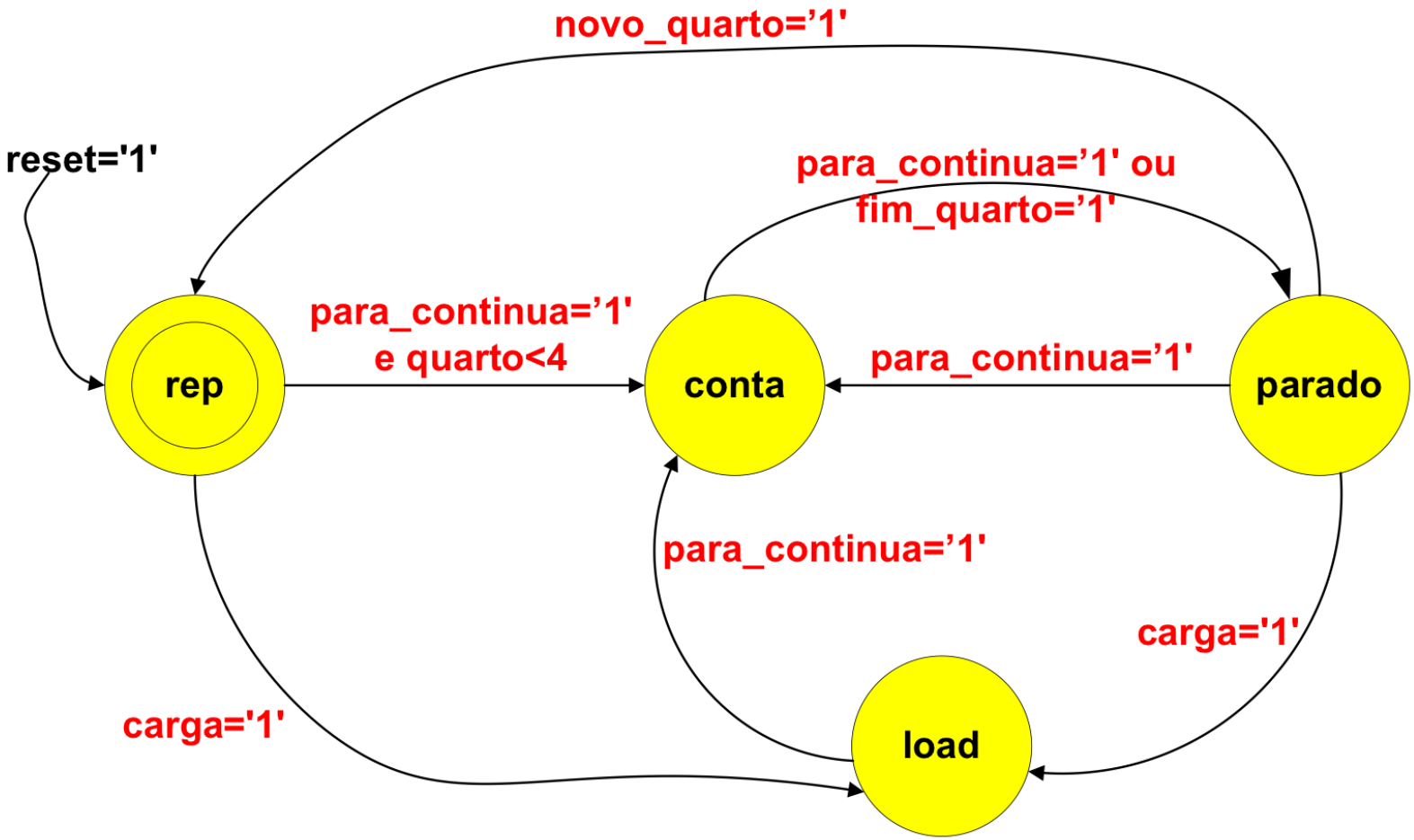
4. **Atribuição concorrente para detecção de fim_quarto.:**

```
fim_quarto <= '1' when minutos_int=0 and segundos_int=0 and
centesimos_int=0  else '0';
```

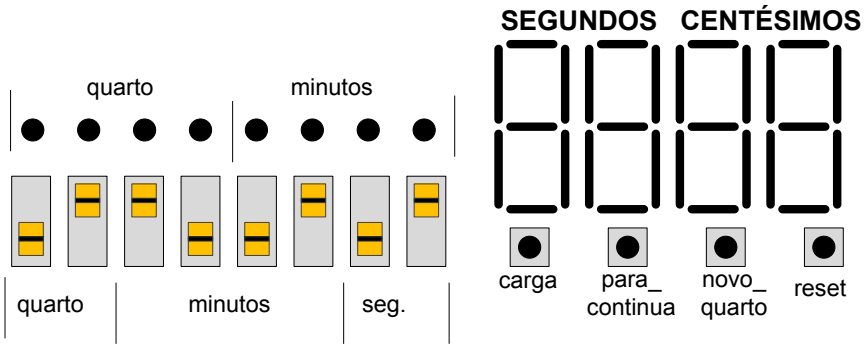
INTEGRAÇÃO

5. contador decrescente de centésimos, controlado pelo relógio ck_1_cent – conta de 99 a 0
   - contagem é habilitada no estado de contagem e não chegou ao final do quarto
   - verificar se o estado é de carga, para realizar uma inicialização com valor externo
6. contador decrescente de segundos, controlado pelo relógio ck_1_cent – conta de 59 a 0 e
   - contagem é habilitada no estado de contagem, os centésimos estão com valor zero e não chegou ao final do quarto
   - verificar se o estado é de carga, para realizar uma inicialização com valor externo
7. contador decrescente de minutos, controlado pelo relógio ck_1_cent – conta de 15 a 0
   - contagem é habilitada cada vez que o cronômetro está no estado de contagem, os segundos e centésimos estão com valor zero e não chegou ao final do quarto
   - verificar se o estado é de carga, para realizar uma inicialização com valor externo
   - no reset ele não vai para zero e sim para 15
8. contador crescente de quartos, que conta de 1 a 4,
   - estando no estado parado e pressionando-se novo_quarto seu valor é incrementado
   - usar três bits, pois quando o quarto='100' significa que os 4 quartos já passaram, devendo assim o cronômetro parar e só sair por reset ou carga de valor externo

MÁQUINA DE ESTADOS



Aula 3

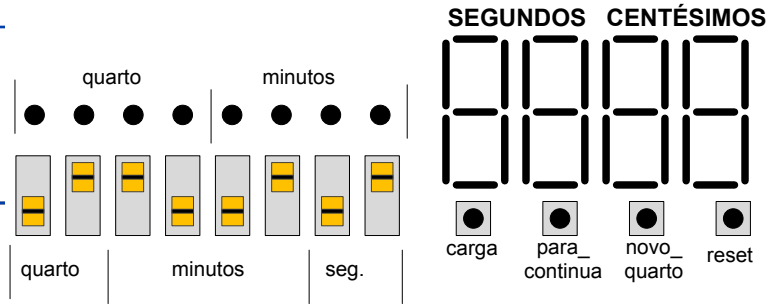


- **Escrita do cronômetro de basquete, e simulação (test bench fornecido)**
- **Lembrar de dividir o clock de referência por no máximo 10**

SEGUIR ESTRITAMENTE ESTA INTERFACE

```
inst_crono: entity work.crono_bskt
      port map ( clock => clock,
                 reset => reset,

                 carga => carga,
                 para_continua => para_continua,
                 novo_quarto => novo_quarto,

                 c_quarto => c_tempo,
                 c_minutos =>  c_minutos,
                 c_segundos => c_segundos,

                 centesimos => centesimos,
                 segundos => segundos,
                 minutos => minutos,
                 quarto => quarto
               );
```

Aula 4



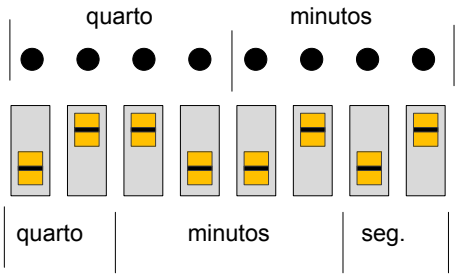
- **Concluir a simulação mostrando ao professor a correta operação do circuito**
- **Desenvolver o circuito *top* e prototipar**

SEGUIR ESTRITAMENTE ESTA INTERFACE

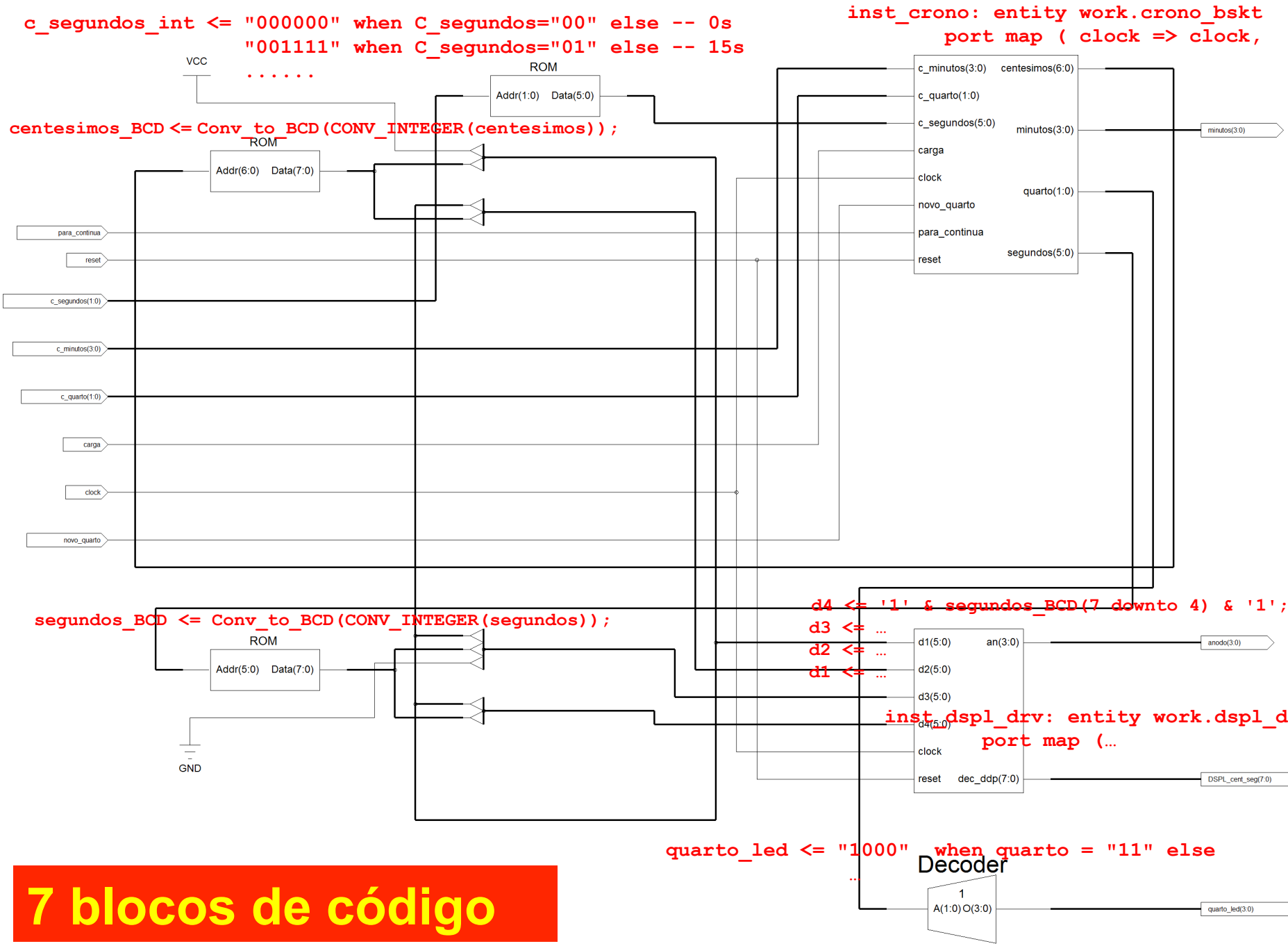
```
entity top_cr_bskt is
    Port ( clock : in  STD_LOGIC;

           --- 4 botões push-buttons     ---------
           carga :           in  STD_LOGIC;
           para_continua :   in  STD_LOGIC;
           novo_quarto :     in  STD_LOGIC;
           reset :           in  STD_LOGIC;

           --- valores de carga – 8 dip-switches ---------
           c_quarto : in  STD_LOGIC_VECTOR (1 downto 0);
           c_minutos : in  STD_LOGIC_VECTOR (3 downto 0);
           c_segundos : in  STD_LOGIC_VECTOR (1 downto 0);

           -- 4 displays de sete-segmentos para segundos e centésimos ----
           DSPL_cent_seg : out  STD_LOGIC_VECTOR (7 downto 0);
           anodo : out  STD_LOGIC_VECTOR (3 downto 0);

           --- 8 leds, para indicar os minutos e tempo --------
           minutos : out  STD_LOGIC_VECTOR (3 downto 0);
           quarto_led : out  STD_LOGIC_VECTOR (3 downto 0));
end top_cr_bskt;

architecture top_cr_bskt of top_cr_bskt is
```

```
begin

inst_crono: entity work.crono_bskt
      port map ( clock => clock,      -- sinais na lâmina anterior --     );

c_segundos_int <= "000000" when C_segundos="00" else -- 0s
                  ...
                  "101101" when C_segundos="11";     -- 45s

 quarto_led <= "1000"  when quarto = "11" else
               ....
               "0001";

 segundos_BCD <= Conv_to_BCD(CONV_INTEGER(segundos));
 centesimos_BCD <= Conv_to_BCD(CONV_INTEGER(centesimos));

 d4 <= '1' & segundos_BCD(7 downto 4) & '1';
 d3 <= '1' & segundos_BCD(3 downto 0) & '0';
 d2 <= '1' & centesimos_BCD(7 downto 4) & '1';
 d1 <= '1' & centesimos_BCD(3 downto 0) & '1';

 inst_dspl_drv: entity work.dspl_drv       port map (....);

end top_cr_bskt;
```

Aula 4 - TOP

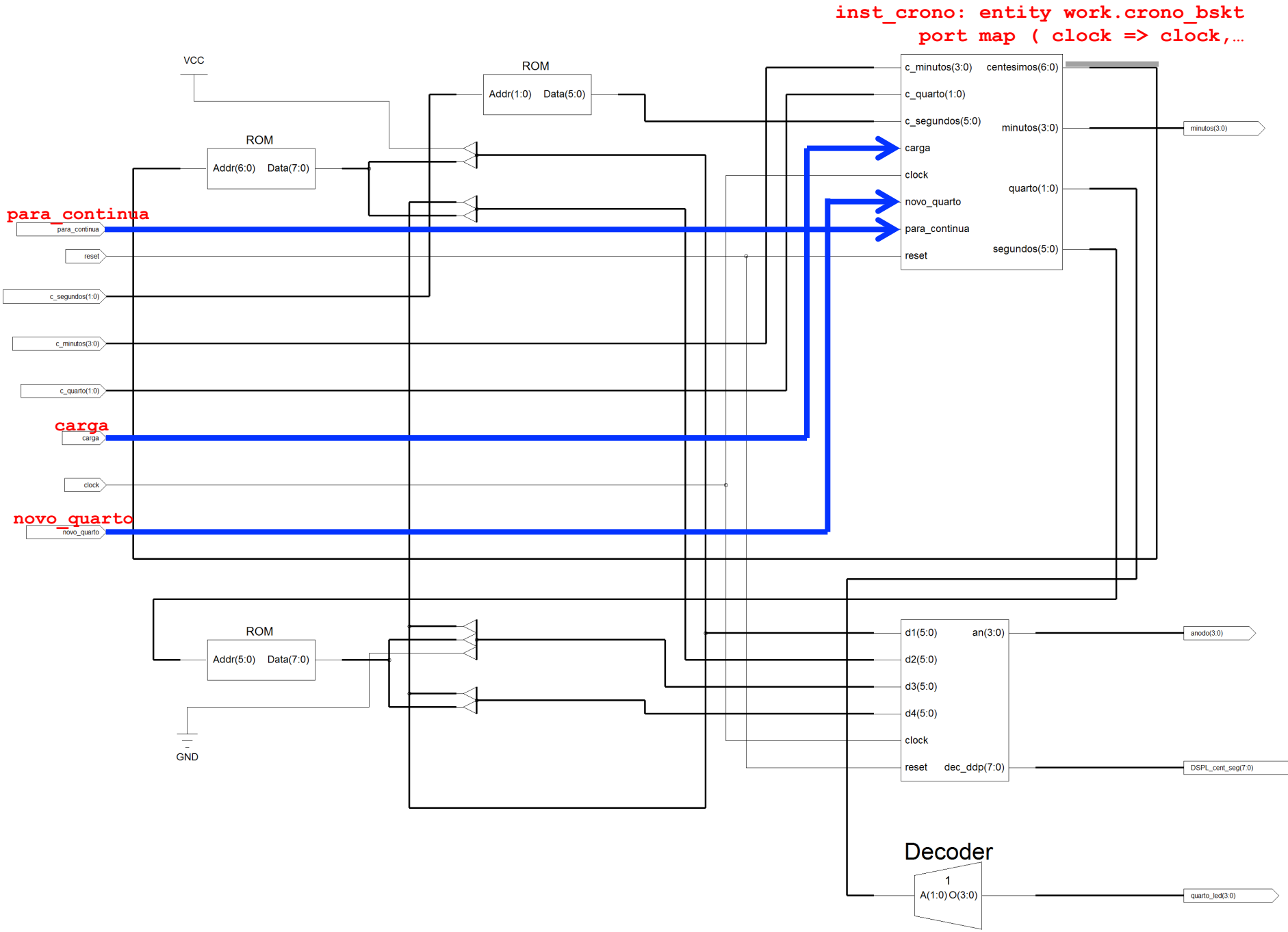



```
entity top_cr_bskt is
    Port ( clock : in  STD_LOGIC;

           --- 4 botões push-buttons    ---------
           carga :           in  STD_LOGIC;
           para_continua :   in  STD_LOGIC;
           novo_quarto :     in  STD_LOGIC;
           reset :           in  STD_LOGIC;

           --- valores de carga - 8 dip-switches ---------
           c_quarto : in  STD_LOGIC_VECTOR (1 downto 0);
           c_minutos : in  STD_LOGIC_VECTOR (3 downto 0);
           c_segundos : in  STD_LOGIC_VECTOR (1 downto 0);

           -- 4 displays de sete-segmentos para segundos e centésimos -
           DSPL_cent_seg : out  STD_LOGIC_VECTOR (7 downto 0);
           anodo : out  STD_LOGIC_VECTOR (3 downto 0);

           --- 8 leds, para indicar os minutos e quarto --------
           minutos : out  STD_LOGIC_VECTOR (3 downto 0);
           quarto_led : out  STD_LOGIC_VECTOR (3 downto 0));
end top_cr_bskt;
```



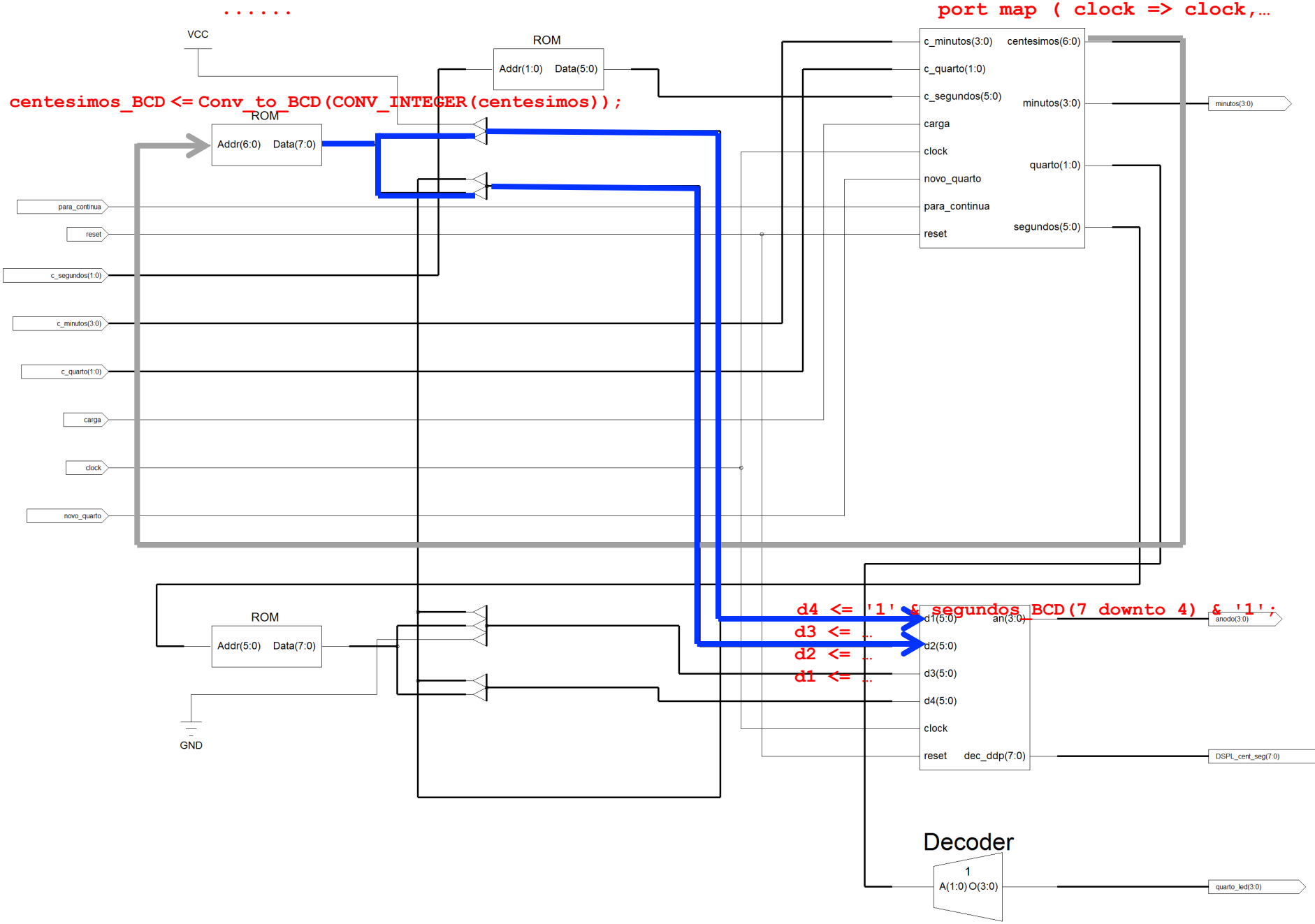
7 blocos de código

Sinais de controle entram direto no circuito cronômetro de basquete



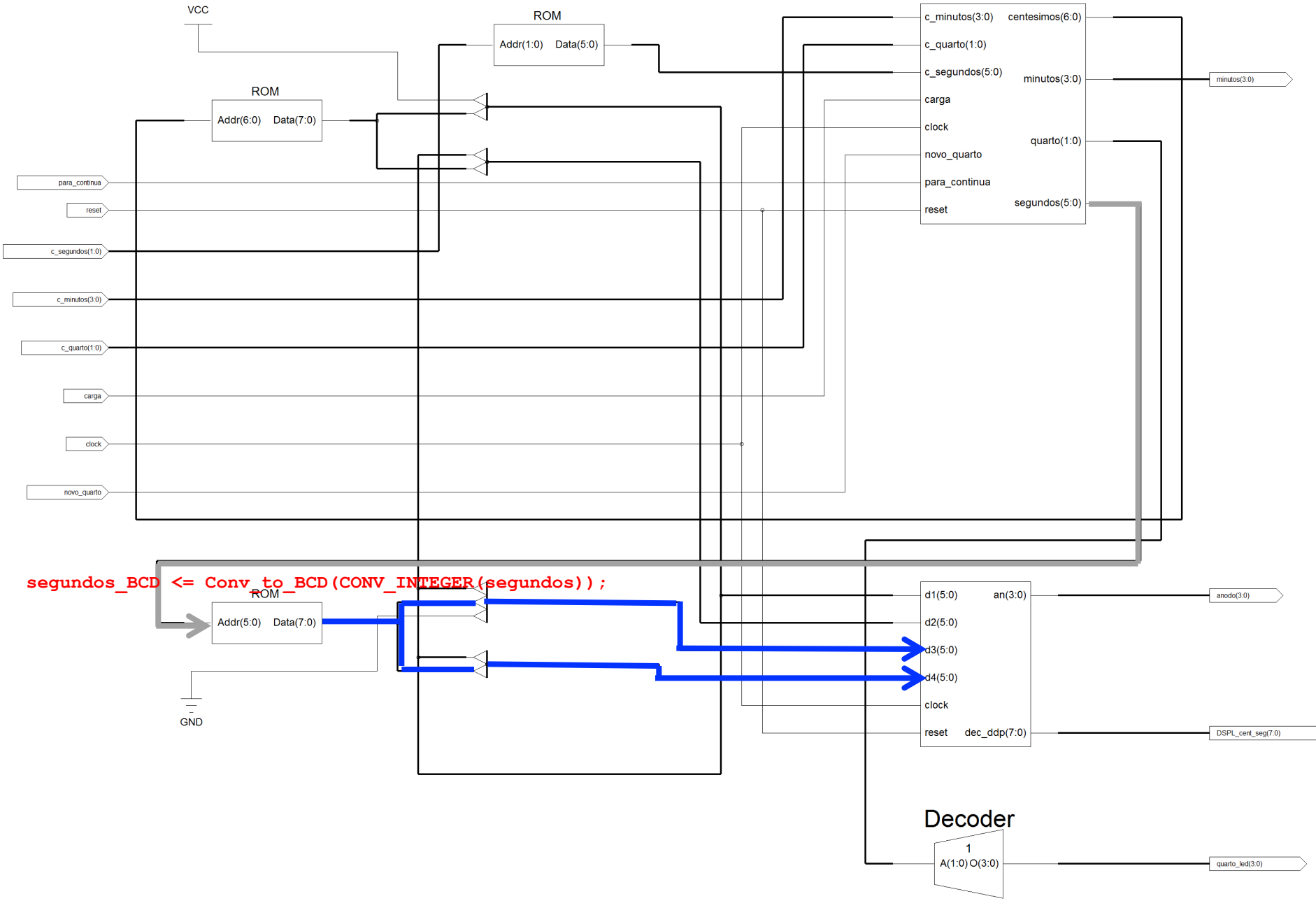
O contador de centésimos (até 99), com 7 bits, entra na rom para gerar dois valores hexadecimais, os quais vão para o *driver* de display de sete segmentos

```
inst_crono: entity work.crono_bskt
  port map ( clock => clock,…
```

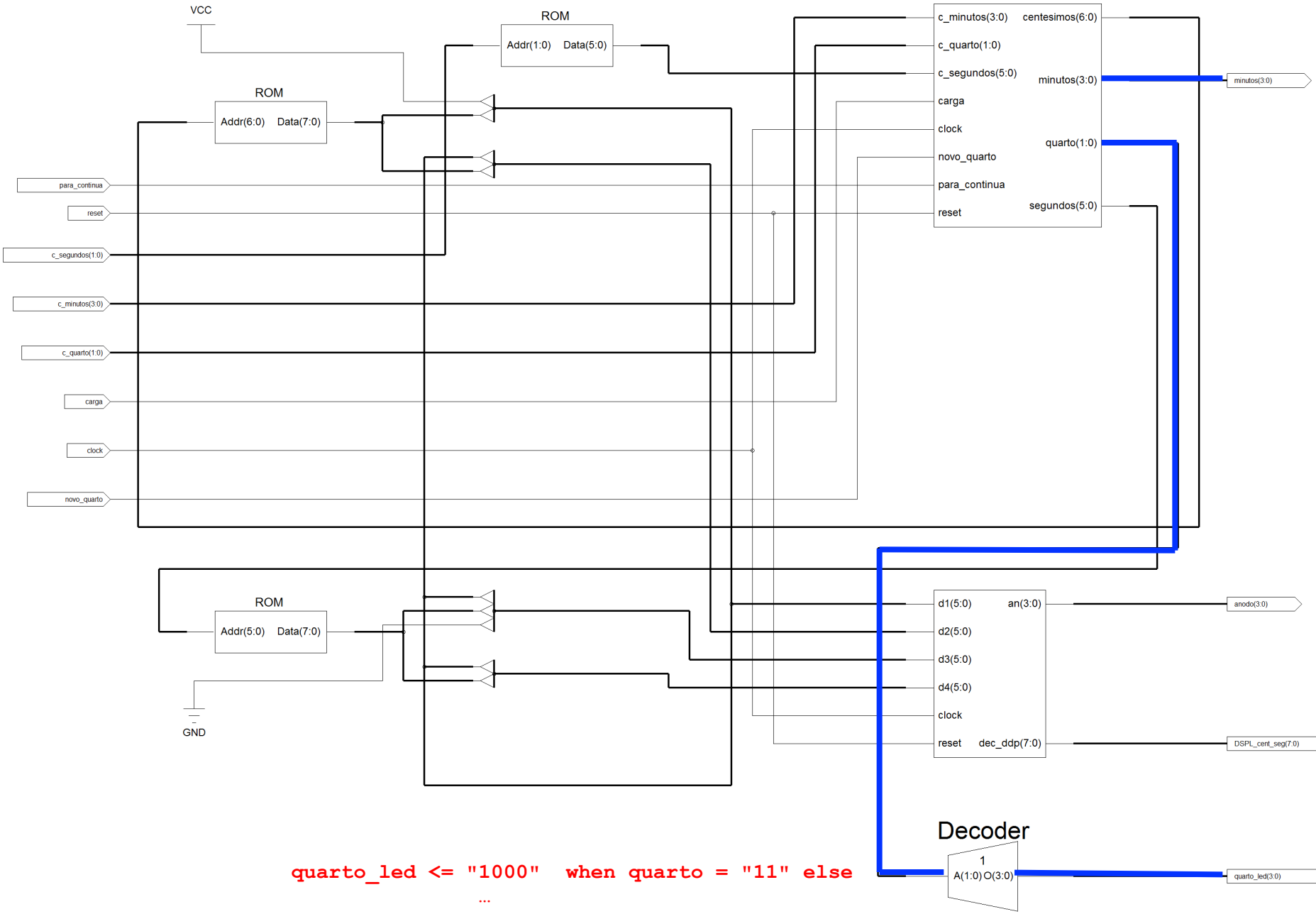


O contador de segundos (até 59), com 6 bits, entra na rom para gera dois valores hexas, os quais vão para o *driver* de display de sete segmentos

```
inst_crono: entity work.crono_bskt
      port map ( clock => clock,
```



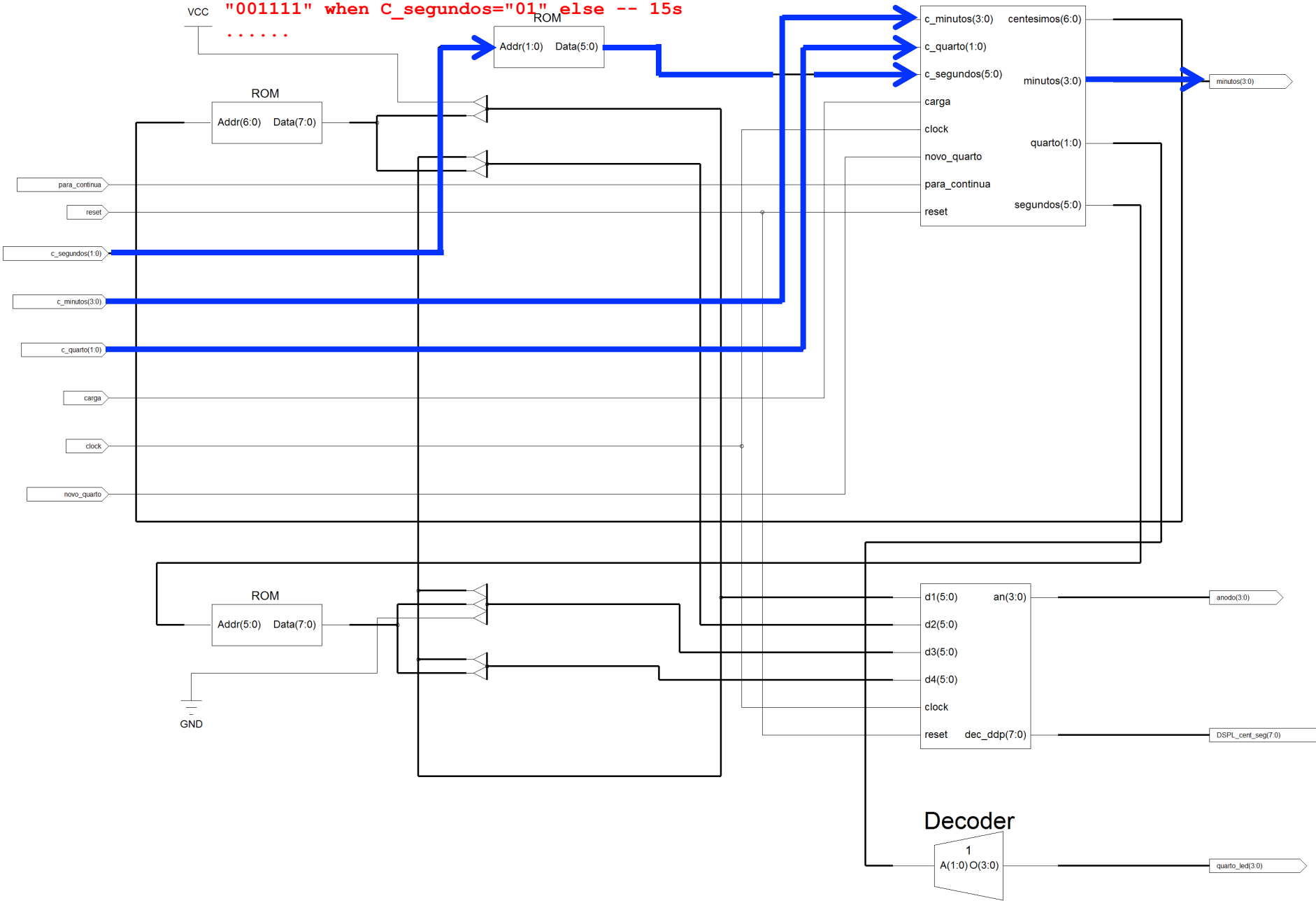
A contagem de minutos vai direto para os leds
A contagem de quarto passa por um decodificador 2→4 para ir para os leds




```
quarto_led <= "1000"  when quarto = "11" else
              ...
```

A carga dos valores externos é direta, exceto pela carga dos segundos, que é codificada

```
c_segundos_int <= "000000" when C_segundos="00" else -- 0s
                  "001111" when C_segundos="01" else -- 15s
                  ......
```



Aula 5

- **TESTAR NA PLACA DE PROTOTIPAÇÃO E MOSTRAR AO PROFESSOR**

- **ENTREGÁVEIS NO MOODLE:**
  - **Arquivo VHDL do contador de basquete**
  - **Arquivo UCF do top**
  - **Arquivo .BIT funcional da implementação**
  - **Arquivo em formato PDF com as formas de onda relevantes para o projeto, explicando os eventos mostrados**
  - **Todos os arquivos em um único ZIP**