Trabalho Prático 3 – TP3 (EM DUPLAS)
Cronômetro de Xadrez

1 Orientações Gerais

Entrega do TP3	09/NOVEMBRO/23 (QUI) – 23h59

Este trabalho consiste nas seguintes tarefas:

- **[2,0 pontos]** Projetar a máquina de estados (FSM) que controla o circuito proposto, *relogio_xadrez*, desenhando-a e explicando o funcionamento da mesma. Este projeto é a **parte 1 do relatório** a ser entregue. Significa apresentar de forma textual a descrição da FSM, com o seu desenho, e explicação do funcionamento do circuito.
- **[1,5 pontos]** Modelar corretamente o módulo *temporizador*. Este módulo deve ser instanciado (*port map*) duas vezes no *relogio_xadrez*.
- **[3,5 pontos]** Desenvolver a modelagem do circuito *relogio_xadrez* em linguagem VHDL. No material de apoio ao trabalho há o arquivo *relogio_xadrez.vhd*, que deve receber o código desenvolvido. Os pontos serão divididos da seguinte maneira:
  - [0,4 pontos] correta modelagem da entidade
  - [0,4 pontos] correta modelagem dos sinais internos à arquitetura
  - [0,4 pontos] instanciação correta dos 2 módulos *temporizadores*
  - [1,9 pontos] correta modelagem da máquina de estados finita (FSM)
  - [0,4 pontos] correta modelagem das atribuições das saídas e demais sinais que dependem do estado atual.
- **[3,0 pontos]** Apresentar no relatório **um cenário** de jogo, diferente do apresentado, contendo:
  - [1,0 ponto] modelagem correta de uma nova partida (no arquivo *tb.vhd*), explicando os tempos de cada jogador, como no exemplo fornecido no material de apoio.
  - [2,0 pontos] apresentar e explicar as formas de onda. A apresentação do jogo é **parte 2 do relatório** a ser entregue. Usar como modelo para o relatório a figura apresentada no final deste documento.

▶ **Entregar:** um arquivo em formato ZIP contendo **7 (sete)** arquivos:
1. *dec_counter.vhd (arquivo fornecido, **não** alterar)*
2. *temporizador.vhd*
3. *relogio_xadrez.vhd*
4. *tb.vhd* (**alterado** em relação ao arquivo fornecido para contemplar a parte 2 do relatório)
5. *script* de simulação *sim.do* (fornecido, mas precisa adequar o tempo de simulação ao jogo apresentado. No *sim.do*, na linha 16, o tempo de simulação é de 26 µs: run 26 µs)
6. arquivo *wave.do* (fornecido)
7. relatório em formato PDF

▶ **O cenário de teste só será avaliado se o resultado corresponder ao resultado da simulação, isto é, o código deverá ser compilado e o simulador gerar as formas de onda presentes no relatório.**
▶ **Plágio é impossível, pois cada grupo necessariamente fará cenários de teste diferentes.**
▶ **Plágio em qualquer parte das entregas zera a nota para os grupos em que se detectar esta grave infração.**

2 Descrição do Módulo Cronômetro de Xadrez

Em campeonatos, o xadrez é jogado usando um relógio. A razão para isso é limitar o tempo máximo de jogo e evitar que ele dure para sempre. Um relógio de xadrez é composto por dois cronômetros e dois botões, que controlam a contagem dos cronômetros.

O **objetivo** desse trabalho é o projeto de um relógio de xadrez, utilizando conceitos de projeto de circuitos digitais, em particular máquina de estados finita e registradores.

Figura 1 – Exemplo de cronômetro para controlar o tempo de jogos de xadrez.

O funcionamento básico desse tipo de relógio é definido como:

1. Um tempo máximo de jogo para cada jogador é definido, e os dois cronômetros são ajustados para esse valor (quando *load=1* na Figura 2, *contj1* e *contj2* são ajustados para o valor de *init_time*). Os contadores contam de forma decrescente, em **decimal**.
2. Um jogador inicia sua jogada e seu cronômetro começa a regredir. Por exemplo, o jogador 1 inicia o jogo e seu cronômetro começa a regredir (*j1=1*);
3. Assim que este jogador terminar sua jogada, seu cronômetro deve parar de regredir e é a vez do próximo jogador. Para isto, o jogador que iniciou o jogo pressiona o botão para interromper a contagem e habilitar a regressão do cronômetro de seu adversário (*j1=1*);
4. Após o jogador seguinte pressiona o seu botão ao final da jogada (*j2=1*) Esse processo segue até que um dos cronômetros chegue ao ponto 0 (zero).

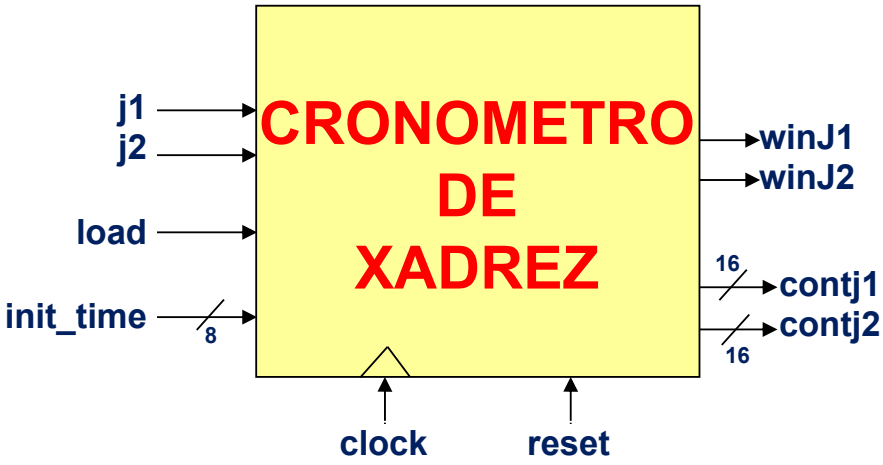


Figura 2 – Interface externa do cronômetro de xadrez.

Exemplo de uma partida:

1. Define-se partida de 20 minutos
2. **Jogador 1** começa a partida 10 ciclos depois - *j1=1*

3. Jogador 1 termina sua jogada após 120 ciclos (2 minutos) - *j1=1*. Observar contJ1=18m00s
4. Jogador 2 termina sua jogada após 120 ciclos (2 minutos) - *j2=1*. Observar contJ2=18m00s

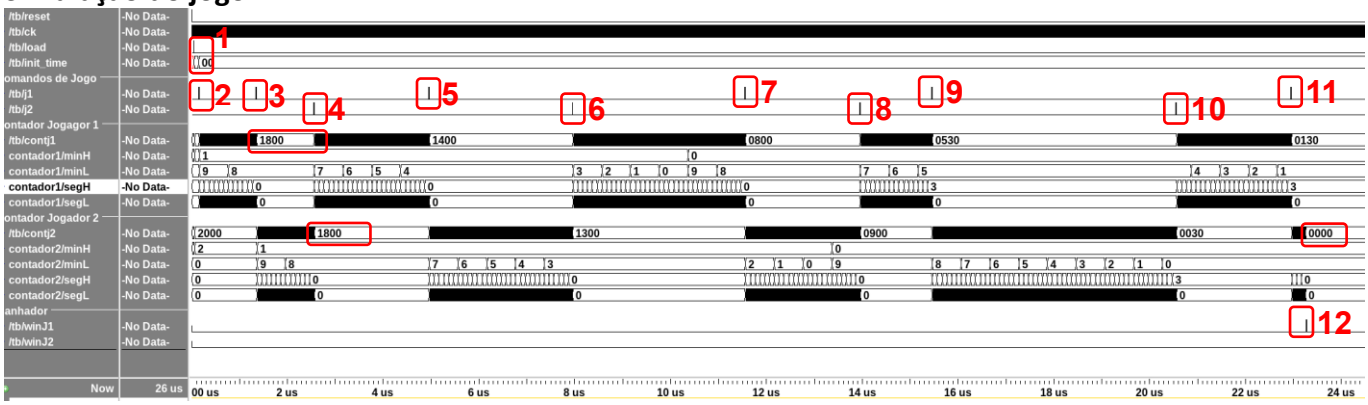
5. Jogador 1 termina sua jogada após 240 ciclos (4 minutos) - *j1=1*. Observar contJ1=14m00s
6. Jogador 2 termina sua jogada após 300 ciclos (5 minutos) - *j2=1*. Observar contJ2=13m00s

7. Jogador 1 termina sua jogada após 360 ciclos (6 minutos) - *j1=1*. Observar contJ1=08m00s
8. Jogador 2 termina sua jogada após 240 ciclos (4 minutos) - *j2=1*. Observar contJ2=09m00s

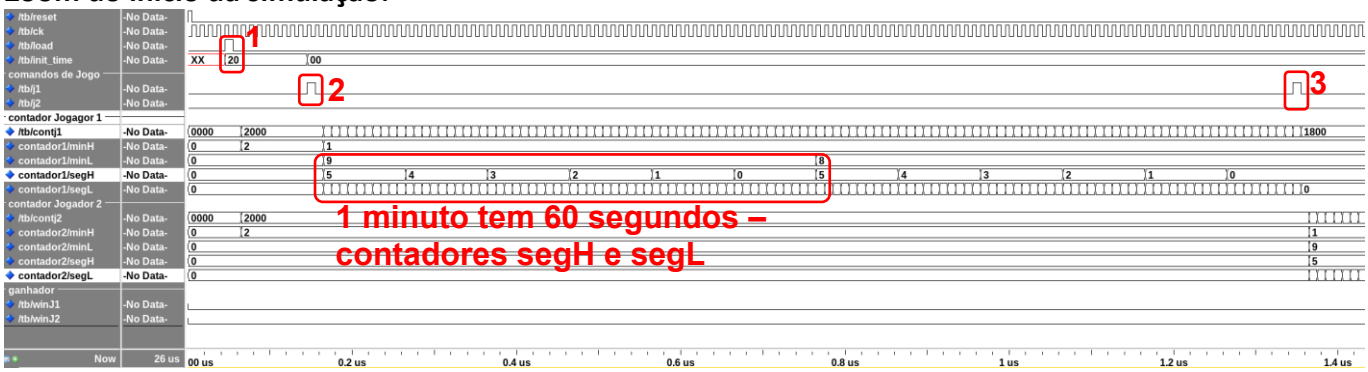
9. Jogador 1 termina sua jogada após 150 ciclos (2min 30 seg). - *j1=1* Observar contJ1=05m30s
10. Jogador 2 termina sua jogada após 510 ciclos (8 min 30 seg). - *j2=1* Observar contJ2=00m30s

11. Jogador 1 termina sua jogada após 240 ciclos (4 min) - *j1=1*. Observar contJ1=01m30s

Simulação do jogo:



Zoom do início da simulação:



3 Estrutura do Código

1) **dec_counter.vhd** – módulo que realiza a contagem regressiva, se *en*==1, a partir de um valor inicial (*first_value*) até chegar em zero. Após chegar em zero, a regressiva recomeça a partir do valor limite (*limit*). O valor inicial é carregado através do sinal de controle *load.*

Possui a seguinte interface:

```
entity dec_counter is
    port( clock, reset, load, en : in std_logic;
          first_value : in  std_logic_vector(3 downto 0);
          limit       : in  std_logic_vector(3 downto 0);
          cont        : out std_logic_vector(3 downto 0)
    );
end dec_counter;
```

Este módulo já se encontra completo, entretanto, compreender seu funcionamento é imprescindível para continuar o desenvolvimento.

2) **temporizador.vhd** – É um timer decrescente. Consiste em um contador que conta de um determinado valor inicial (em minutos), até parar em "00:00" (zero minutos e zero segundos).

Cada 4 bits da saída do contador correspondem a um dígito decimal, ou seja, a saída possui um total de 16 bits:

- *minH* (dezena do minuto)
- *minL* (unidade do minuto)
- *segH* (dezena do segundo, conta de 5 a 0)
- *segL* (unidade do segundo)

```
entity temporizador is
    port( clock, reset, load, en : in std_logic;
          init_time : in  std_logic_vector(7 downto 0);
          cont      : out std_logic_vector(15 downto 0)
      );
end temporizador;
```

Observe:

1. este circuito deve parar a contagem quando o cronômetro zerar.
2. a inicialização é feita apenas em minutos (8 bits, duas partes de 4 bits), mas a contagem é em minutos e segundos (16 bits, quatro partes de 4 bits).

Exemplo: para inicializarmos o cronômetro com **13** minutos o valor de entrada (*init_time*) deve ser x"13" (minH="0001" e minL="0011"). Os segundo são sempre inicializados em zero.
Onde:

- **load** – se for '1' indica para inicializar os minutos com o valor definido em **init_time**, e os segundos em 00
- **en** – habilita a contagem decrescente
- **init_time** – valor de inicialização dos minutos (8 bits)
- **cont** – valor de saída do temporizador (16 bits)

Sugere-se o desenvolvimento do *temporizador* utilizando a lógica ilustrada abaixo:

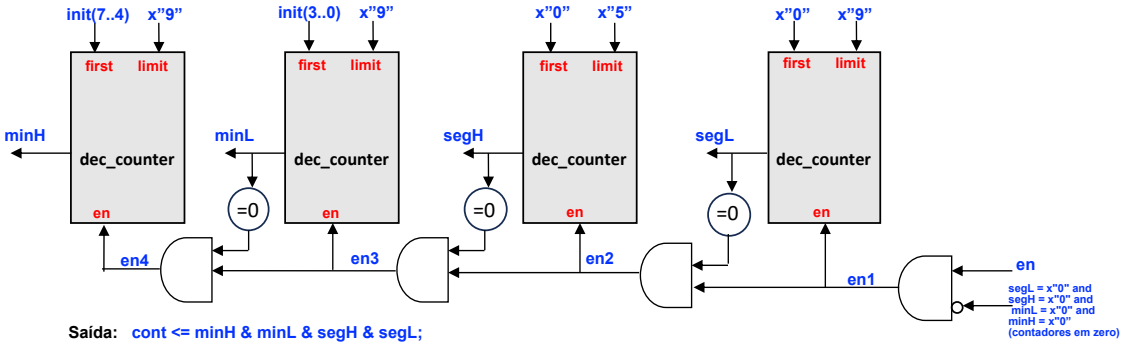


Figura 2 – Sugestão de implementação do contador decrescente.

A Figura 2 mostra que:

- para se decrementar *segL* deve-se ter o sinal *en* habilitado e todos os dígitos não devem ser zero (para travar a contagem em "00:00");
- para se decrementar *segH* deve-se ter habilitação de contagem (en1=1) e o dígito anterior (*segL*) ter chegado a zero (por exemplo ..., 21, 2**0**, **19**);
- a mesma lógica se aplica aos demais bits de contagem.

Note que no módulo *dec_counter* devemos informar o valor de inicialização (*first_value),* onde apenas inicializamos os minutos. Também devemos informar o valor máximo do contador. Os contadores *minH*, *minL* e *segL* decrementam de 9 a 0. Já o contador *segH* decrementa de 5 a 0 (um minuto tem 59 segundos).

Observar que esta é uma **sugestão de lógica**. Diferentes implementações podem ser feitas.

```
architecture a1 of temporizador is
    signal segL, segH, minL, minH : std_logic_vector(3 downto 0);
    signal en1, en2, en3, en4: std_logic;
begin

   en1 <= ...;
   en2 <= ...;
   en3 <= ...;
   en4 <= ...;

   sL : entity work.dec_counter port map ( ... );
   sH : entity work.dec_counter port map ( ... );
   mL : entity work.dec_counter port map ( ... );
   mH : entity work.dec_counter port map ( ... );

   cont <= minH & minL & segH & segL;
end a1;
```

3) relogio_xadrez.vhd – desenvolver conforme a operação especificação realizada na Seção 2 deste documento**.**

A Figura 3 mostra uma sugestão para a estrutura interna do módulo cronômetro de xadrez. Este módulo conterá 3 partes internas:

a. FSM de controle
b. instanciação dos 2 módulos temporizadores, *contador 1* e *contado 2*
c. atribuições das saídas e demais sinais que dependem do estado atual.

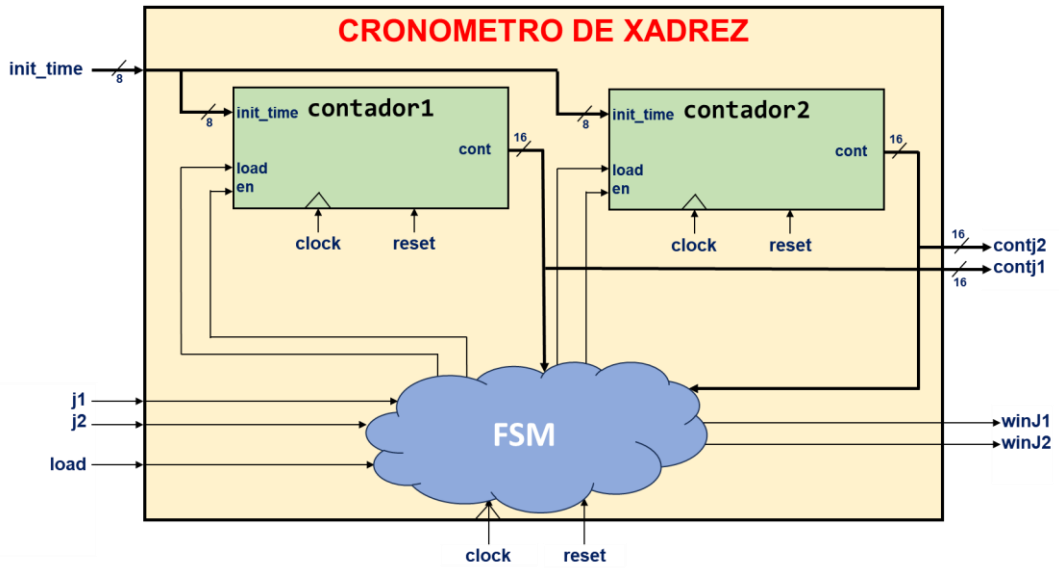


Figura 3 – Sugestão de implementação do cronômetro de xadrez.

4) **tb.vhd**. Permite definir uma partida. Para isto alterar o *testebench*, na parte que descreve a partida:

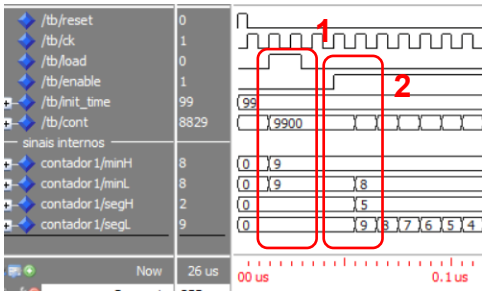
```
constant padrao_de_teste: padroes := (
    (t =>   4,  load=>'1', init=>x"20", j1=>'0', j2 =>'0'),  -- partida de 20 minutos
    (t =>  10,  load=>'0', init=>x"00", j1=>'1', j2 =>'0'),  -- jogador 1 começa a partida 10 ciclos depois
    (t => 120,  load=>'0', init=>x"00", j1=>'1', j2 =>'0'),  -- jogador 1 joga por 120 ciclos (2 min)
    (t => 120,  load=>'0', init=>x"00", j1=>'0', j2 =>'1'),  -- jogador 2 joga por 120 ciclos (2 min)
    (t => 240,  load=>'0', init=>x"00", j1=>'1', j2 =>'0'),  -- jogador 1 joga por 240 ciclos (4 min)   (6)
    (t => 300,  load=>'0', init=>x"00", j1=>'0', j2 =>'1'),  -- jogador 2 joga por 300 ciclos (5 min)   (7)
    (t => 360,  load=>'0', init=>x"00", j1=>'1', j2 =>'0'),  -- jogador 1 joga por 360 ciclos (6 min)  (12)
    (t => 240,  load=>'0', init=>x"00", j1=>'0', j2 =>'1'),  -- jogador 2 joga por 240 ciclos (4 min)  (12)
    (t => 150 ,  load=>'0', init=>x"00", j1=>'1', j2 =>'0'), -- jogador 1 joga por 60 ciclos (2'30'' min)  (14'30)
    (t => 510,  load=>'0', init=>x"00", j1=>'0', j2 =>'1'),  -- jogador 2 joga por 510 ciclos (8'30' min)  (19'30)
    (t => 240,  load=>'0', init=>x"00", j1=>'1', j2 =>'0'),  -- jogador 1 joga por 240 ciclos (4 min)  (18'30)
    (t => 10000,  load=>'0', init=>x"00", j1=>'0', j2 =>'0') );  -- último comando – coloca todos os valores em zero
```

5) **wave.do**. Script para as formas de onda.

6) **sim.do**. Script de simulação.

4 Validação do *temporizador*

Para a validação apenas do temporizador, há no material de apoio 3 arquivos: **tb_temp.vhd**, **wave_temp.do, sim_temp.do**. O *script* **sim_temp.do** permite simular o temporizador (para simular: *do sim_temp.do*). Esta simulação inicializa o temporizador com x"**99**" (99 minutos) – evento destacado com número 1 na simulação, e depois ativa o sinal de habilitação – evento destacado com número 2 na simulação. Com isto o contador deve contar por 5.940 ciclos (99 * 60), com *clock* de 10 ns, correspondendo a uma simulação de 59,4 µs.



Exemplo de contagem, de 90’10’’, 90’09’’ .... 90’00’’, 89’59’’ ... 89’’50’, 89’’49’’