LISTA DE EXERCÍCIOS DE VHDL

ATUALIZADO POR: FERNANDO MORAES

02/JULHO/2004

1. Desenhe um diagrama de esquemáticos que tenha funcionalidade equivalente ao código VHDL abaixo. Não se esqueça de desenhar os limites da entidade, identificando entradas e saídas do circuito. Justifique com palavras seu desenho.

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

entity P1_1_001 is
port( a,b,c,d : in std_logic_vector(5 downto 0);
      reset,clock, ch1, pm : in std_logic;
      s : out std_logic_vector(5 downto 0));
end P1_1_001;

architecture P1_1_001 of P1_1_001 is
signal pm1,pm2,res : std_logic_vector(5 downto 0);

begin

process (clock, reset)
begin
              if reset='1' then s <= (others=>'0');
              elsif clock'event and clock='1' then s <= res;
              end if;
end process;

res <= pm1+pm2 when pm='1' else pm1-pm2;
pm1 <= a when ch1='0' else b;
pm2 <= c when ch1='0' else d;

end P1_1_001;
```

2. Desenhe um diagrama de esquemáticos que tenha funcionalidade equivalente ao código VHDL abaixo.

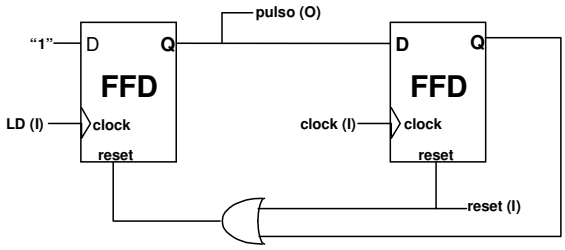
```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_unsigned.all;

entity periferico is
       port( i0, i1, i2, i3, clock, reset : in std_logic;
             saida: out std_logic_vector(3 downto 0) );
end periferico;

architecture arch1 of periferico is
   signal si, s3, s2, s1, s0 : std_logic;
   signal cont : std_logic_vector(1 downto 0);
begin
       saida <= s3 & s2 & s1 & s0;

       process(reset, clock)
       begin
              if reset='1' then
                     s3<='0';    s2<='0';   s1<='0';   s0<='0';
              elsif clock'event and clock='0' then
                     s3 <= si;   s2 <= s3;  s1 <= s2;  s0 <= s1;
              end if;
       end process;

       process(reset, clock)
       begin
              if reset='1' then
                     cont <= "00";
              elsif clock'event and clock='1' then
                     cont <= cont + 1;
```

```
            end if;
        end process;

        si <=   i0 when cont="00" else        i1 when cont="01" else
                i2 when cont="10" else        i3 when cont="11" ;

end arch1;
```

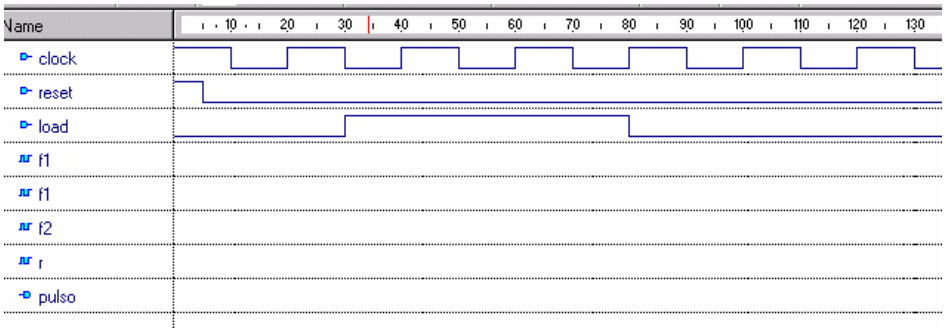
3. Implemente em VHDL um contador DECIMAL (0-9), que tenha como entradas *chip enable*, *clock* e *reset*. Como saída deve ter um bit *ov*, que indica que chegou a 9 e voltou a 0, e 4 bits para o valor da contagem.

4. Utilizando o contador da questão anterior, implemente um módulo contador decimal de 5 dígitos. As entradas são *reset*, *enable* e *clock*. As saídas são 5 dígitos, cada um representado por um barramento de 4 bits, denominados disp1, disp2, disp3, disp4, disp5.

5. A partir do diagrama de esquemáticos abaixo, gere uma descrição VHDL com a mesma funcionalidade, sob a forma de um par entidade-arquitetura. Não se esqueça de definir as entradas e saídas corretamente. Para facilitar, a saída está marcada com (O) e entradas com (I). Todos os fios são simples, e não barramentos. Caso necessite, defina sinais internos e anote-os no esquemático.

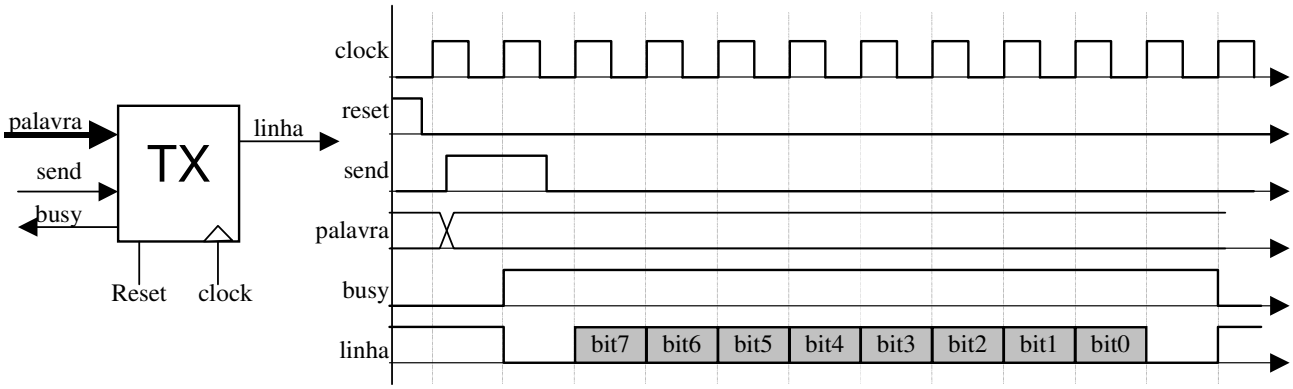


6. Para o circuito acima escreva o test_bench para as seguintes formas de onda:



7. Desenhe as formas de onda do circuito (ou seja, faça o papel do simulador):

8. Implemente um transmissor de dados seriais, em VHDL. DICA: USE MÁQUINA DE ESTADOS. Não é necessário implementar o test_bench:



9. Considere o circuito "Receptor de Dados", ilustrado na Figura 1. A operação deste circuito é a seguinte:

i/ O circuito recebe dados do mundo externo através do barramento 'DIN' (largura igual a 32 bits) e os armazena em um *buffer* interno (de 8 posições). O sinal 'tx' avisa que tem dado e caso o circuito possa armazenar o dado, o armazenamento é sinalizado por 'ack_tx', conforme o laço representado pelos estados 'R0' e 'R1' da máquina de controle.

ii/ Uma vez que o *buffer* do "Receptor de Dados" esteja cheio (todas as 8 posições preenchidas) é feito pedido de envio de dados a um circuito mestre através do sinal 'REQ', conforme o estado 'A1' da máquina de controle.

iii/ Ao receber a confirmação do circuito mestre o "Receptor de Dados" permanece no estado 'A2' enquanto o conteúdo do barramento 'address' for diferente de FFFFH. Enquanto o barramento 'address' for diferente de FFFFH o conteúdo da posição do buffer cujo endereço está especificado em 'address' é colocado em 'data'. Quando o barramento 'address' for igual a FFFFH o circuito volta para o estado 'rst'.

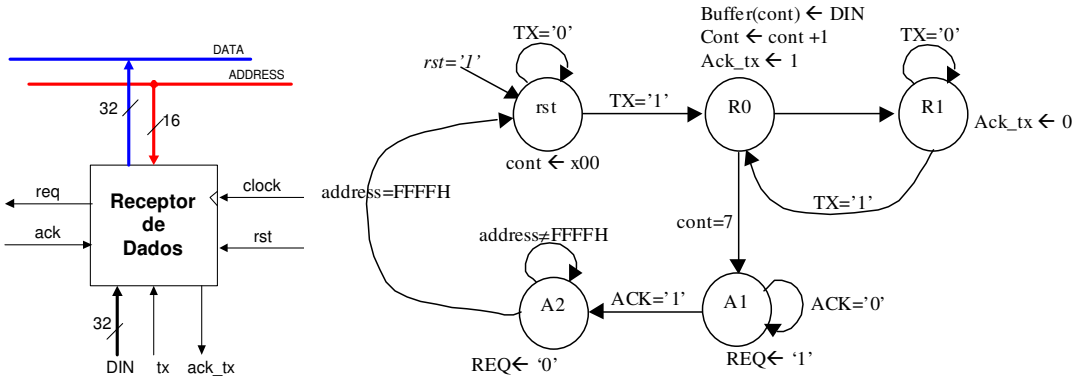


Figura 1 - Interface externa do receptor de dados e principais componentes internos (bloco de dados).

Pede-se: implemente o circuito "Receptor de Dados" em VHDL, em um único par entidade arquitetura.

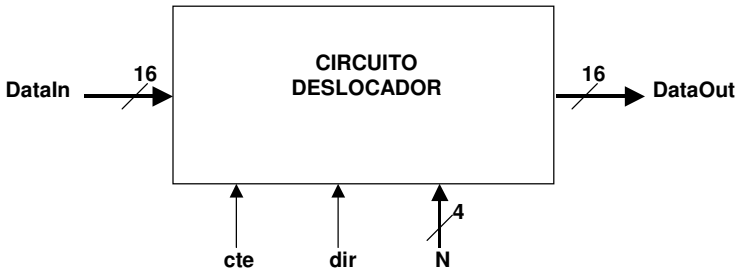
DICAS:

- O *buffer* de 8 posições é muito semelhante ao banco de registradores da MR1, sendo o sinal de habilitação de escrita neste banco ativado no estado 'R0'. Utilizar comando *for generate*.
- Assuma dado o seguinte componente para a criação do *buffer* de 8 posições:

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity registrador is
      port( ck,rst,ce:in std_logic;
            D:in  std_logic_vector(31 downto 0);
            Q:out std_logic_vector(31 downto 0));
end registrador;

architecture registrador of registrador is
   begin
```

```
        process (ck, rst)
          begin
              if rst = '1' then
                    Q <= (others => '0');
              elsif ck'event and ck = '1' then
                    if ce = '1' then  Q <= D; end if;
              end if;
        end process;
end registrador;
```

10. No processador R8 temos quatro instruções para deslocamento: SL0, SL1, SR0, SR1. Estas instruções deslocam um bit para a esquerda ou um bit para a direita por vez. Considere o circuito "***deslocador***" com a seguinte interface externa:



Este circuito recebe como entrada:

- ***DataIn***: palavra de 16 bits;
- ***cte***: constante, que pode assumir os valores '0' ou '1';
- ***dir***: caso seja '0' o circuito realiza deslocamento para a direita e '1' para a esquerda;
- ***N***: indica o número de bits a deslocar.

O circuito produz como resultado a palavra deslocada 'N' bits.

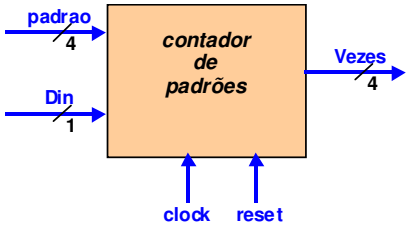
Exemplo:
{ DataIn="1011 0001 0110 1111", cte='0', dir='1', N="0101" } ➔ DataOut="0010 1101 111**0 0000**"

Pede-se:

a) Implemente um módulo de hardware **combinacional** (par entidade arquitetura) deste circuito, em linguagem de descrição de hardware VHDL.
b) Implemente um test_bech para este circuito e mostre a execução de várias situações. Comentar os resultados obtidos.
c) O custo de módulo em termos de área seria muito alto? Por quê?

11. Implemente um circuito em VHDL para contar quantas vezes um padrão qualquer de 4 bits ocorre em um fluxo de dados serial.

Este circuito tem como entradas: (1) *clock*; **(2)** *reset*; **(3)** *Din*, **correspondendo ao valor serial de entrada (1 bit); (4)** *padrao*, **valor fixo de entrada de 4 bits, que deve ser utilizado para comparação. O circuito informa na saída** *Vezes* **(de 4 bits) o número de ocorrências do padrão no fluxo de entrada desde a última ocorrência do reset. Não é necessário considerar padrões superpostos, ou seja, após detectar um padrão, aguarda-se quatro novos bits na entrada serial antes de se procurar detectar outro padrão. Ver abaixo exemplos de comportamento e uma saída da simulação correta do circuito,** Utilizar máquina de estados para o controle deste circuito e desenhar a máquina.



Interface com o mundo externo do circuito *contador de padrões*.

Exemplo 1:

Padrão: 1000

Seqüência: 001001000**100**100**11**

Vezes = 2 (após receber o último bit)

Exemplo 2:

Padrão: 1111

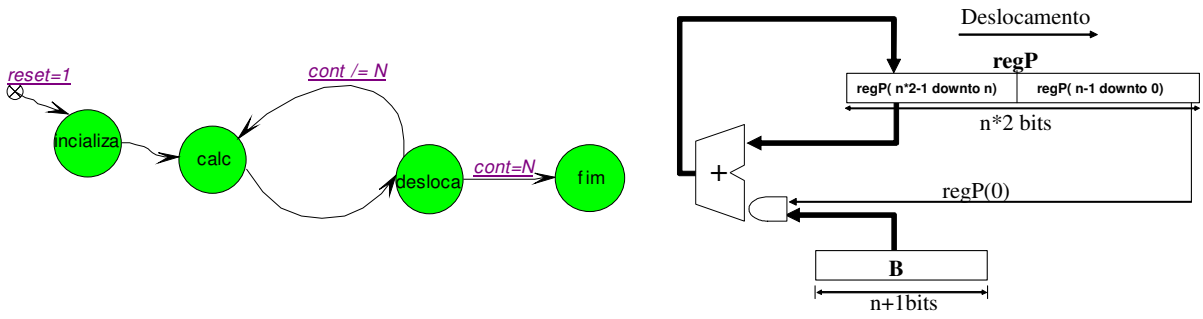
Seqüência: 111111111111**111 (15 1's)**

Vezes = 3 (após receber o último bit)

12. Implemente o testbench para o circuito comparador, utilizando a seguinte estrutura para a geração do dado de entrada.

```
type rom is array (0 to 55) of std_logic;
 constant stream : rom :=
( '0','1','0','0','0','1','0','1','0','1','0','1','1','1','0','1','0','1','0','0','1','0','1','1','1','1','0','1',
  '1','1','1','1','1','1','1','1','0','1','0','1','1','0','0','1','1','1','0','1','0','1','0','1','1','1','0','0');
```

13. Considere o circuito multiplicador abaixo, composto por um conjunto de registradores e uma máquina de estados de controle :



Supor que o circuito multiplique dois números de 4 bits, resultando em um produto de 8 bits. No exemplo abaixo estaremos multiplicando a entrada **M1**=0101 (5) pela entrada **M2**=1101 (13), e o resultado final após o processamento estará contido em **regP**=0100 0001 (65), e aparecerá após o final do processamento (identificado pelo valor **CONT** =4) em **produto**, que é a saída do circuito. O circuito opera como descrito na tabela abaixo. O funcionamento da máquina de estados é o seguinte:

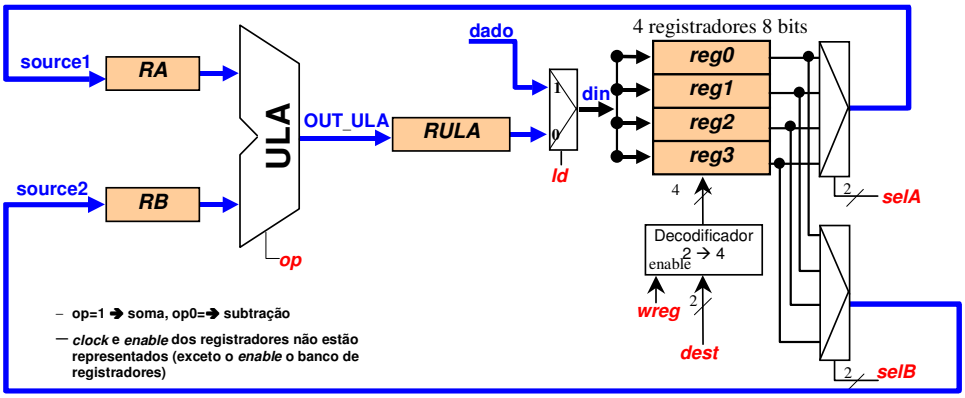
No estado *inicializa*: carrega a parte baixa de **regP** com **M1**, a parte baixa de **regB** com **M2** e zera o contador **CONT**.
CONT não é mostrado, mas obviamente é um elemento seqüencial que deve ser implementado.
No estado *calc*: armazena na parte alta de **regP** o conteúdo da saída do somador, se **regP(0)** for igual a 1.
No estado *desloca*: desloca **regP** um bit para a direita, entrando zero no bit mais significativo.
No estado *fim*: **endop** recebe '1' e **produto** recebe **regP**.

Não esquecer que às entradas e saídas do circuito descritas acima devem ser acrescentados os tradicionais sinais de entrada **clock** e **reset**, para comandar a operação. Além disto, **endop** é uma saída que indica quando **produto** contém o resultado final da operação do multiplicador, segundo a tabela abaixo.

Clock	EA	regP	regB	soma	cont	endop	produto
1	inicializa	00000 0101	0 1101	01101	0	0	0000 0000
2	calc	01101 0101	0 1101	11010	0	0	0000 0000
3	desloca	00110 1010	0 1101	00110	1	0	0000 0000
4	calc	00110 1010	0 1101	00110	1	0	0000 0000
5	desloca	00011 0101	0 1101	10000	2	0	0000 0000
6	calc	10000 0101	0 1101	11101	2	0	0000 0000
7	desloca	01000 0010	0 1101	01000	3	0	0000 0000
8	calc	01000 0010	0 1101	01000	3	0	0000 0000
9	desloca	00100 0001	0 1101	10001	4	1	00100 0001
10	fim	00100 0001	0 1101	10001	4	1	00100 0001

PEDE-SE: implemente o circuito multiplicador acima em VHDL, usando a máquina de estados dada. Mostre a entidade e a arquitetura de sua solução completa.

14. (4 PONTOS) A figura abaixo ilustra o esquemático de um pipeline de 3 estágios. Todos os registradores são de 8 bits, sendo o banco de registradores composto por 4 registradores.



A função de cada estágio é:

- primeiro estágio: lê dois registradores do banco de registradores, especificados pelos endereços ***selA*** e ***selB***, armazenando-os em RA e RB.
- segundo estágio: armazena em RULA o resultado da operação aritmética determinada pelo comando ***op***. A ULA executa apenas 2 operações: soma ou subtração
- terceiro estágio: armazena o conteúdo de RULA ou dado externo, em função do comando ***ld***, no registrador especificado por ***dest*** no banco de registradores, desde que ***wreg*** seja igual a 1.

Implemente o circuito pipeline do esquemático acima em VHDL, utilizando o código abaixo como referência. Completar apenas as partes relativas aos sinais e arquitetura.

```
library IEEE;
use IEEE.Std_Logic_1164.all;
package pipe is
   type microinstrucao is record
               selA, selB, dest : std_logic_vector(1 downto 0);
               op, wreg, ld : std_logic;
   end record;
end pipe;

library IEEE;
use IEEE.Std_Logic_1164.all;   use IEEE.Std_Logic_unsigned.all;  use work.pipe.all;

entity pipeline is
    port( ck, reset : in std_logic;
          uins : in microinstrucao;
          dado : in std_logic_vector(7 downto 0);
          R0, R1, R2, R3 : out std_logic_vector(7 downto 0)
        );
end pipeline;

architecture a1 of pipeline is

      component reg8clear is   -- supor o registrador de 8 bits já descrito no código
          port( clock,reset,ce : in std_logic;
             D : in std_logic_vector(7 downto 0);
             Q : out std_logic_vector(7 downto 0));
      end component reg8clear;

       INSERIR AQUI OS SINAIS NECESSÁRIOS AO FUNCIONAMENTO DO PIPELINE
begin

       DESCREVER AQUI O PIPELINE EXPOSTO NO ESQUEMÁTICO
end a1;
```

15. Com relação ao texto VHDL abaixo faça o seguinte:

(a) Desenhe um diagrama que tenha funcionalidade de hardware equivalente ao trecho VHDL abaixo. É permitido o uso de esquemáticos e/ou diagramas de transição de estados na representação, de forma pura ou mista. Como a entity

VHDL não está especificada, não se preocupe com a interface do módulo, mas não deixe de dar nomes a cada fio que eventualmente esteja representado, de acordo com o código VHDL. Todos os sinais não declarados na arquitetura são da entidade, com a direção (in ou out) definida pelo contexto.

(b) Sugira o que pode ser este módulo, em termos de estrutura e/ou funcionalidade.

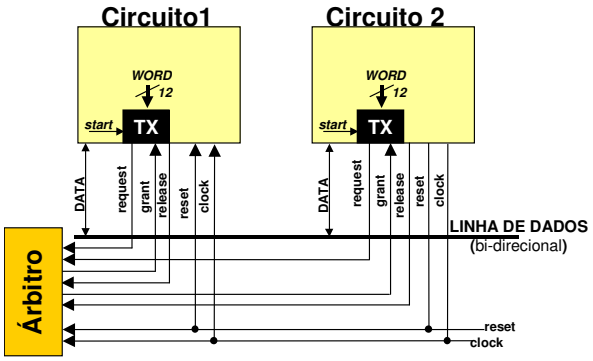
(c) Diga se as cláusulas others dos comandos case são úteis ou não.

(d) Dos três processos que definem a arquitetura, quais são combinacionais puros e quais correspondem a hardware seqüencial?

```
1.  architecture memctlr of memctlr is
2.  type ctler_states is (idle,
3.                        read_setup,
4.                        read_capture,
5.                        write_setup,
6.                        write_capture
7.                        );
8.  signal pr_st_memctl   : ctler_states;
9.  signal nxt_st_memctl  : ctler_states;
10. signal ld_datain      : std_logic;
11. begin
12.
13. process (clock, reset)
14. begin
15.     if reset='1' then
16.         pr_st_memctl <= idle;
17.     elsif clock'event and clock = '1' then
18.         pr_st_memctl <= nxt_st_memctl;
19.     end if;
20. end process;
21.
22. process(pr_st_memctl)
23. begin
24.     case pr_st_memctl is
25.         when idle =>
26.             ce1s_m_n <='1';
27.             oe_m_n <='1';
28.             we_m_n <='1';
29.             busy <= '0';
30.             ld_datain <= '0';
31.         when read_setup =>
32.             ce1s_m_n <='0';
33.             oe_m_n <='0';
34.             we_m_n <='1';
35.             busy <= '1';
36.             ld_datain <= '0';
37.         when read_capture =>
38.             ce1s_m_n <='0';
39.             oe_m_n <='0';
40.             we_m_n <='1';
41.             busy <= '1';
42.             ld_datain <='1';
43.         when write_setup =>
44.             ce1s_m_n <='0';
45.             oe_m_n <='1';
46.             we_m_n <='0';
47.             busy <= '1';
48.             ld_datain <= '0';
49.         when write_capture =>
50.             ce1s_m_n <='0';
51.             oe_m_n <='1';
52.             we_m_n <='0';
53.             busy <= '1';
54.             ld_datain <= '0';
55.         when others => null;
56.     end case;
57. end process;
```

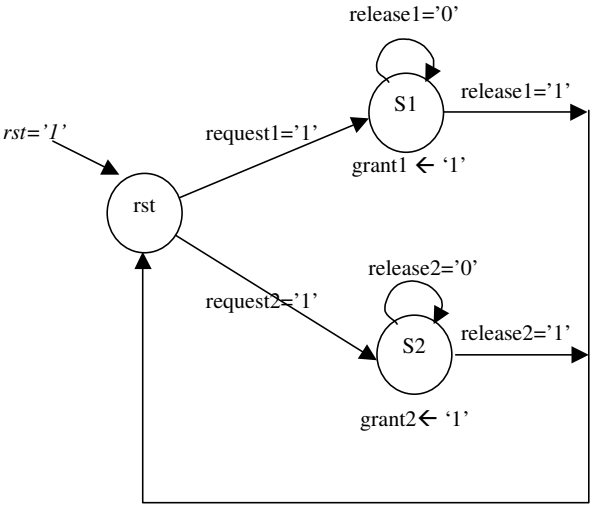
```
58. process(pr_st_memctl, req, we)
59. begin
60.    case pr_st_memctl is
61.       when idle =>
62.          if (req = '0') then
63.             nxt_st_memctl <= idle;
64.          elsif (req='1' and we='0') then
65.             nxt_st_memctl <= read_setup;
66.          else
67.             nxt_st_memctl <= write_setup;
68.          end if;
69.
70.       when read_setup =>
71.          nxt_st_memctl <= read_capture;
72.
73.       when read_capture =>
74.          nxt_st_memctl <= idle;
75.
76.       when write_setup =>
77.          nxt_st_memctl <= write_capture;
78.
79.       when write_capture =>
80.          nxt_st_memctl <= idle;
81.
82.       when others => null;
83.   end case;
84. end process;
```

Considere o circuito ao lado, composto por 3 módulos: árbitro, circuito 1, circuito 2. A função do árbitro é permitir que um dado circuito escreva na "*linha de dados*". Considere que a linha de dados contenha apenas 1 bit (serial) e que um pacote de dados contenha 12 bits: os 4 primeiros contém o endereço do módulo ao qual se destina um pacote de dados e os demais 8 bits os dados.



16. Projeto do árbitro

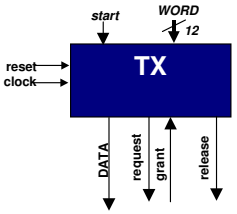
Pode-se projetar um árbitro muito simples, composto por uma máquina de estados para tratar as requisições (*request)*. O árbitro espera que um periférico solicite envio de dados, ativando o sinal de *grant*, e ficando neste estado até que o periférico envie um sinal '1' em *release* liberando o periférico, como abaixo.



Pede-se:

a) Implemente o módulo do árbitro em linguagem de descrição de hardware VHDL.

17. Projeto da máquina de transmissão.



Quando um módulo (circuto *x*) quer enviar um dado para outro circuito (circuito *y*):

a) Máquina de estados de transmissão (TX) fica aguardando um pedido de transmissão por parte do circuito (*start*) (este é o estado em que a máquina entra no momento de reset);
b) TX pede para o árbitro permissão para enviar dado (*request*=1) e armazena o dado de entrada *word* em um registrador de 12 bits;
c) TX fica aguardando que o árbitro dê permissão (*grant*=1);
d) TX remove o *request* e envia serialmente os 12 bits do pacote. Após os 12 bits enviados é enviado um pulso de *release*, liberando o árbitro.
e) TX volta a aguardar um pedido de transmissão do circuito (*start*).

Pede-se:

b) Desenhe, e explique, a máquina de estados que representa o circuito de transmissão;
c) Implemente o módulo de transmissão em linguagem de descrição de hardware VHDL.

18. Descreva em VHDL o circuito *top* que contenha dois módulos TX e um árbitro. A interface externa do circuito serão os sinais clock, reset, word1, start1, word2, start2.

19. Faça um teste bench, onde ocorra a solicitação simultânea de envio de duas palavras.