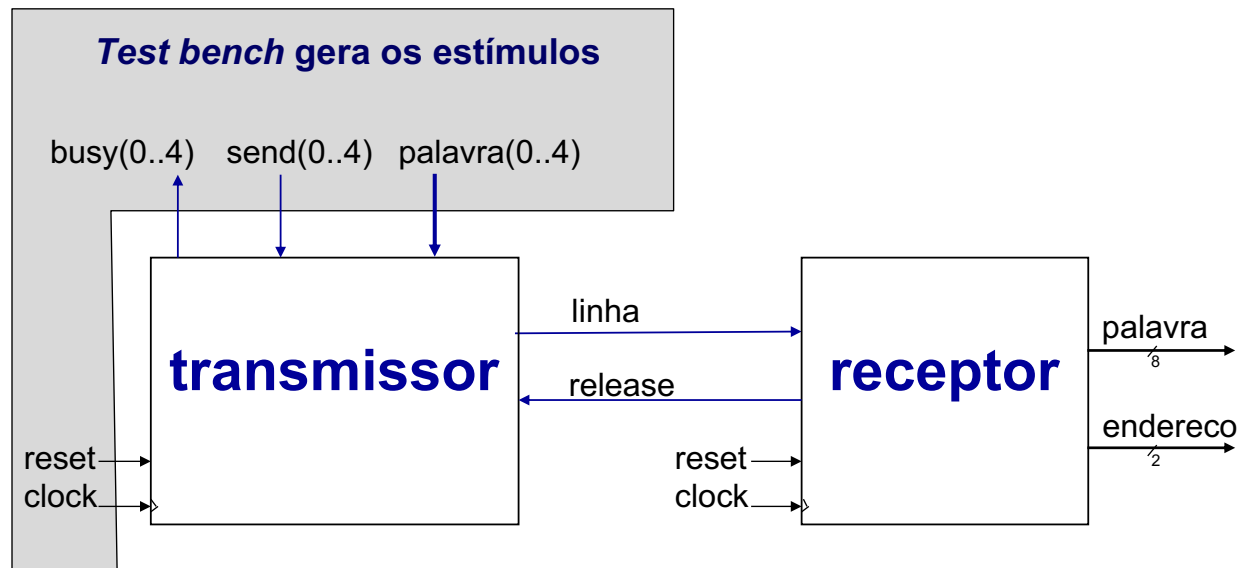


# TP6 – TXRX

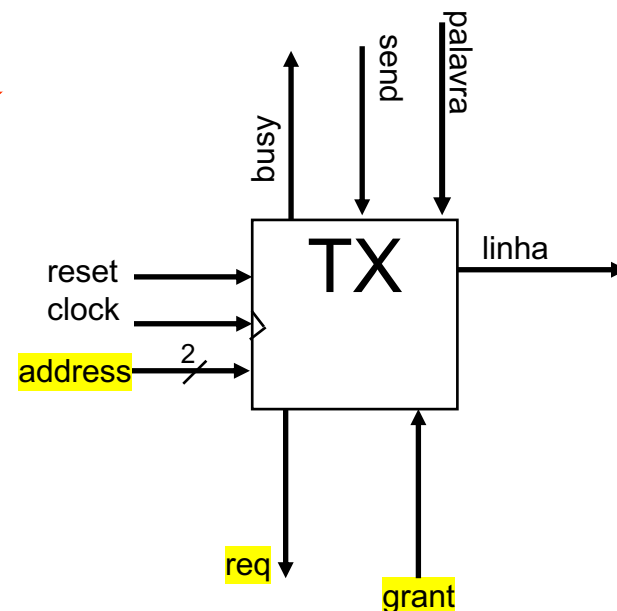
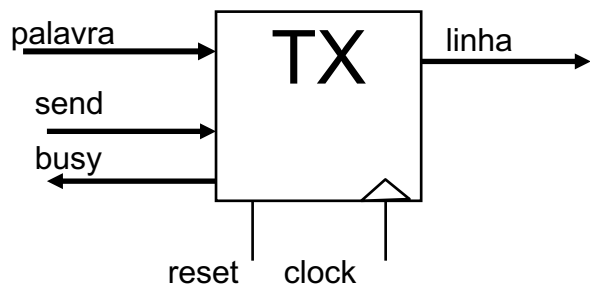
Este trabalho visa consolidar os conceitos de sistemas digitais, explorando os conceitos:

- 1 – Hierarquia de módulos
- 2 – Máquina de estados (FSM)
- 3 – Projeto RTL (registradores controlados por FSM)

O trabalho consiste de 4 módulos de transmissão, controlados por um árbitro (bloco *transmissor*) e um módulo de recepção de dados (bloco *receptor*)



# Parte 1 – Modificação do TX



Linha em repouso: '1'

Linha em repouso: 'H'

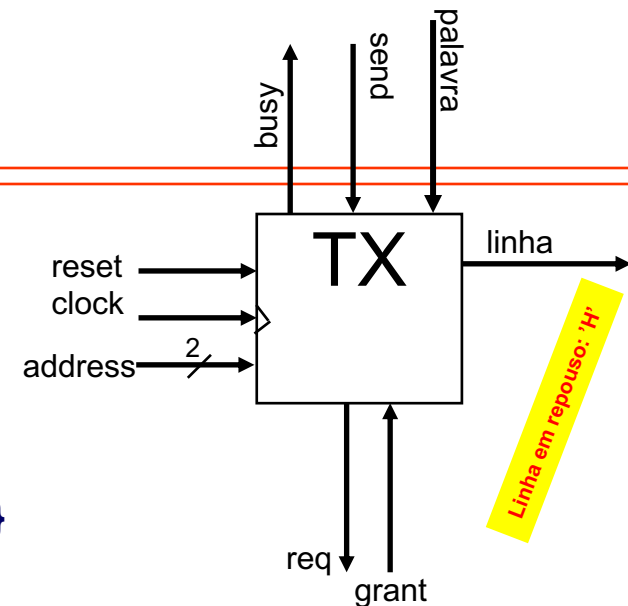
O novo protocolo do transmissor é:

1. *test bench* disponibiliza nova **palavra** e ativa o sinal **send** por um ciclo
2. o TX ao detectar um **send**, ativa o sinal de **req** para o árbitro, e aguarda que o sinal de **grant** suba
3. uma vez o **grant** ativado, o TX envia:  
{ start bit, address(0), address(1), palavra(0)...palavra(7), stop bit }
4. Ao enviar o stop bit desce o sinal **busy**

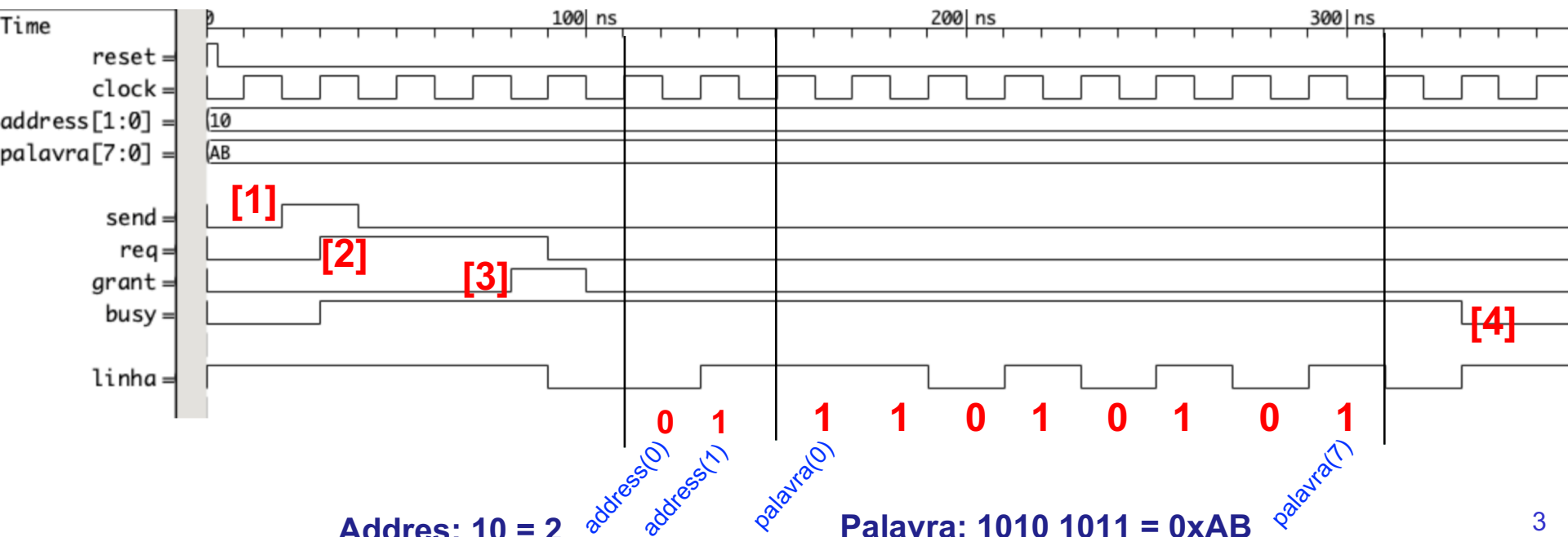
Requer novos estados na máquina de estados original

# Parte 1 – Modificação do TX

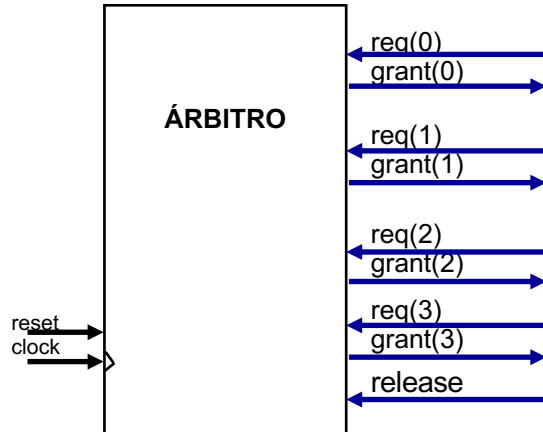
1. test bench disponibiliza nova **palavra** e ativa o sinal **send** por um ciclo
2. o TX ao detectar um **send**, ativa o sinal de **req** para o árbitro, e aguarda que o sinal de **grant** suba
3. uma vez o **grant** ativado, o TX envia:  
{ start bit, address(0), address(1), palavra(0)...palavra(7), stop bit}
4. Ao enviar o stop bit desce o sinal **busy**



***tb\_novo\_tx.vhd* disponibilizado para teste**



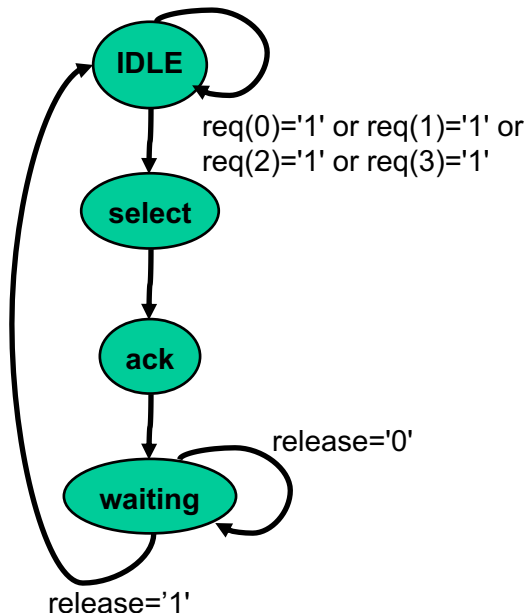
# Parte 2 – Modificação do Árbitro



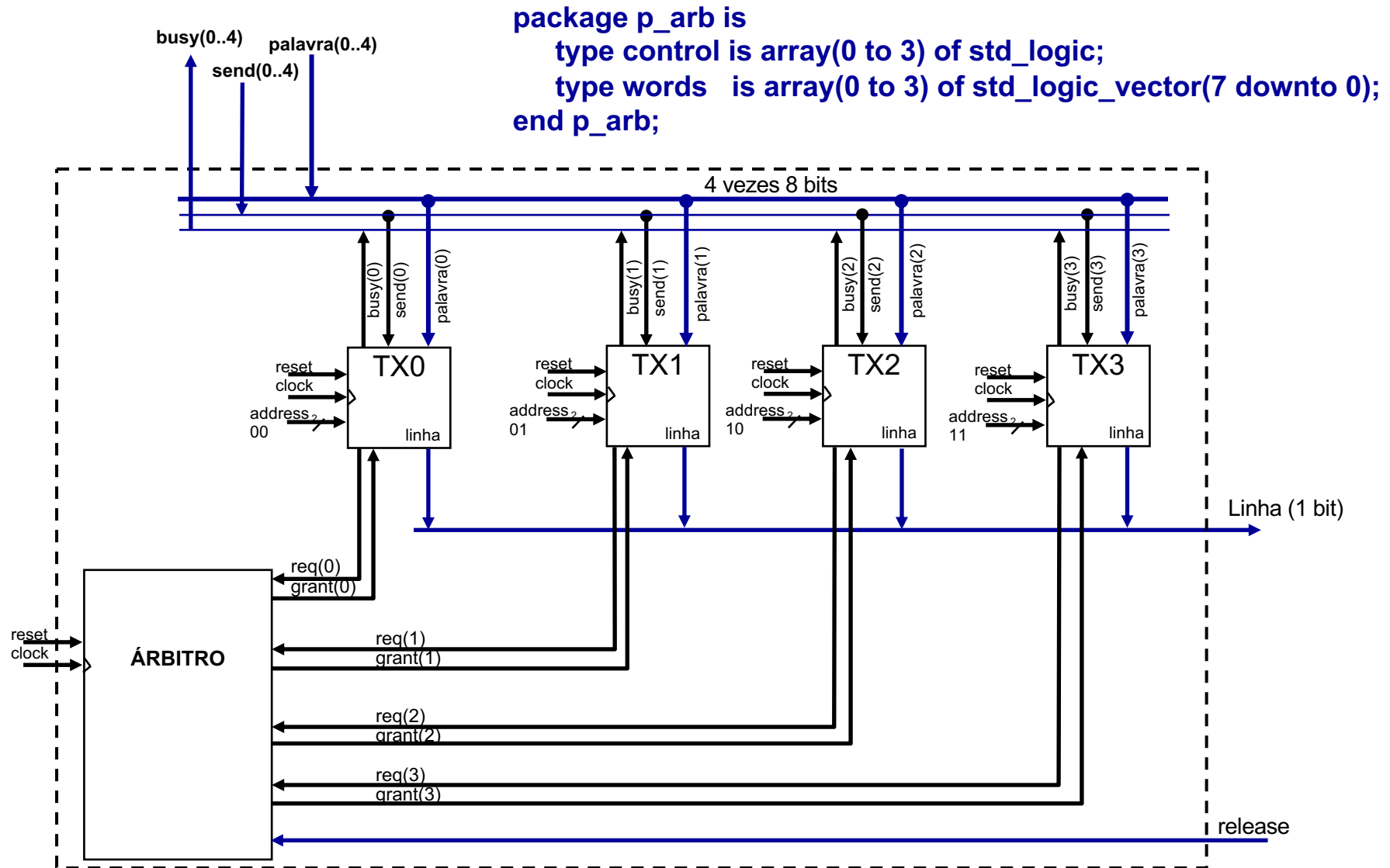
A única modificação a realizar no árbitro é relativa ao sinal de **release**

Árbitro original: 4 sinais de **release**, um para cada componente ligado ao par req/grant

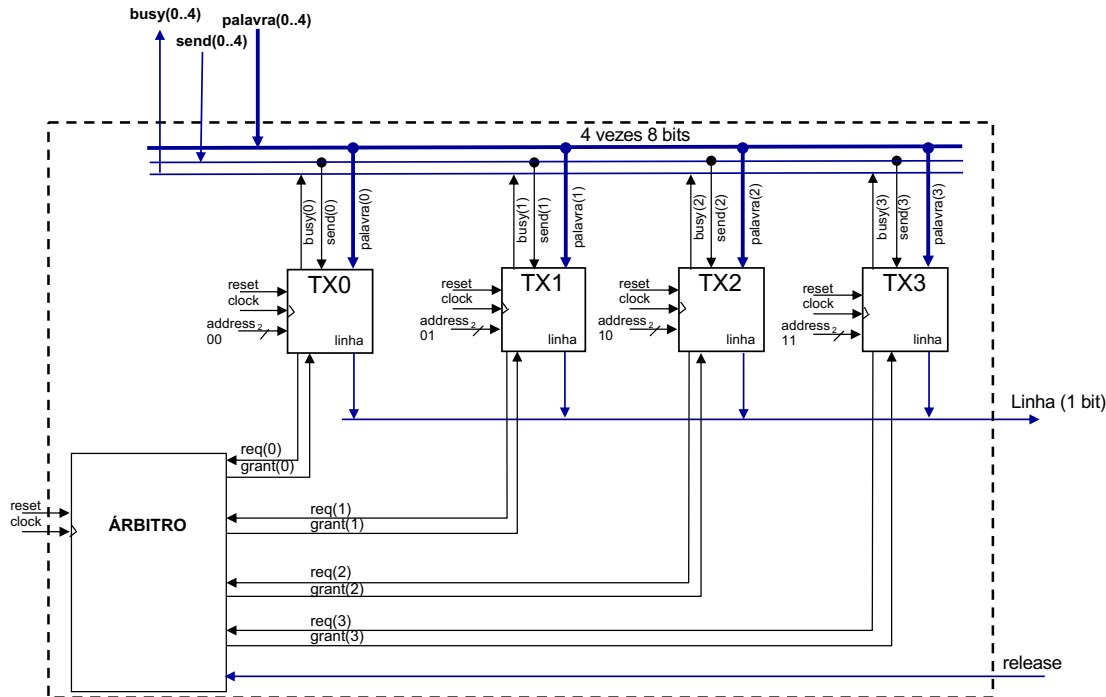
Árbitro modificado para TP6: o **release** é um fio de entrada (std\_logic), que libera a próxima arbitragem



# Parte 3 – Integração TXs - árbitro



# Parte 3 – Integração TXs - árbitro



```
entity transmissores is
  port (
    clock:    in std_logic;
    reset:    in std_logic;

    palavra:  in words;
    send:     in control;
    busy:     out control;

    release:  in std_logic;
    linha:    out std_logic
  );
end transmissores;
```

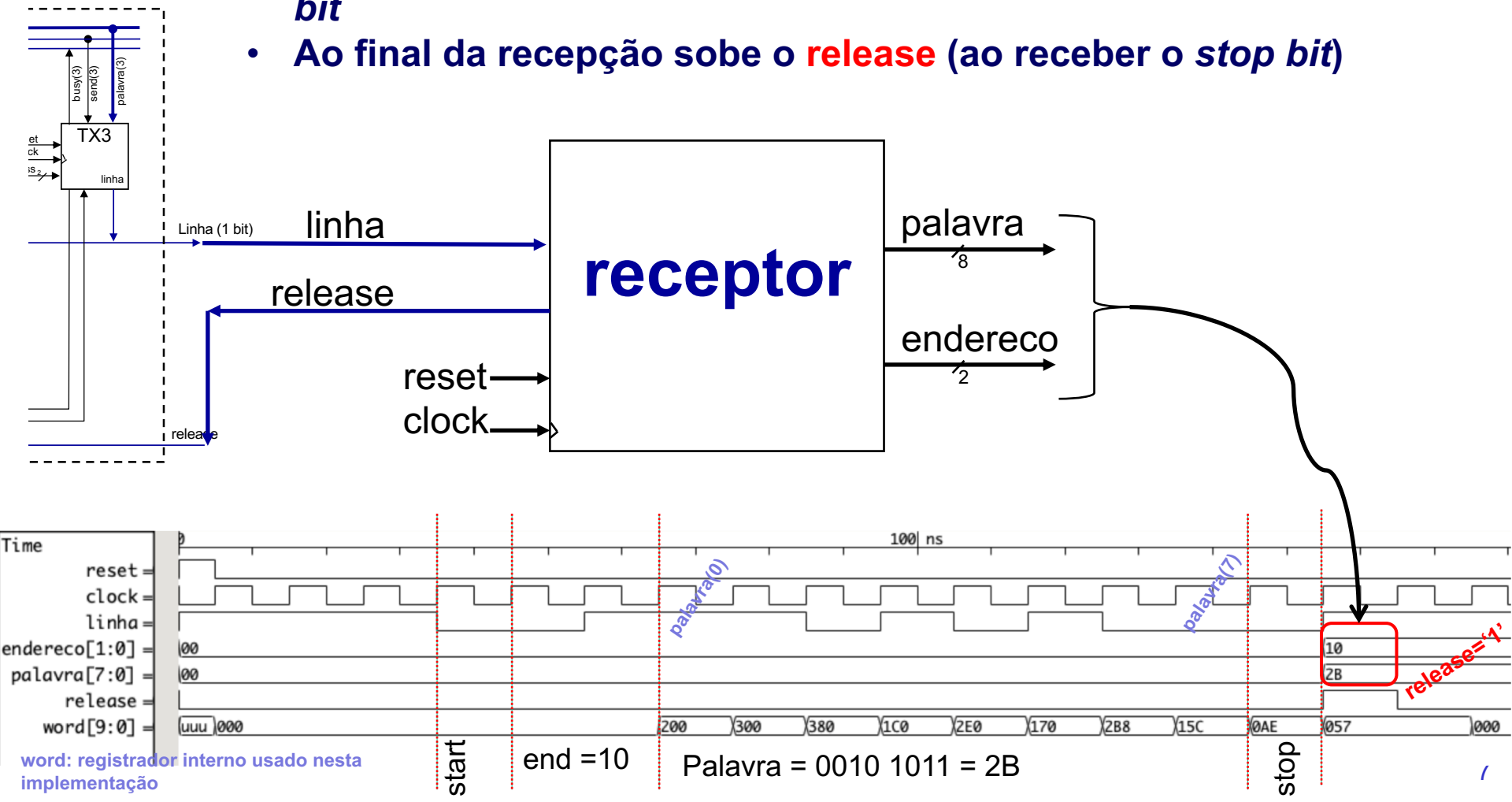
O módulo transmissores integra 4 instâncias do TX e uma instância do árbitro

Sinais internos: *req* e *grant*, do tipo **control**

```
package p_arb is
  type control is array(0 to 3) of std_logic;
  type words   is array(0 to 3) of std_logic_vector(7 downto 0);
end p_arb;
```

# Parte 4 – Receptor de dados

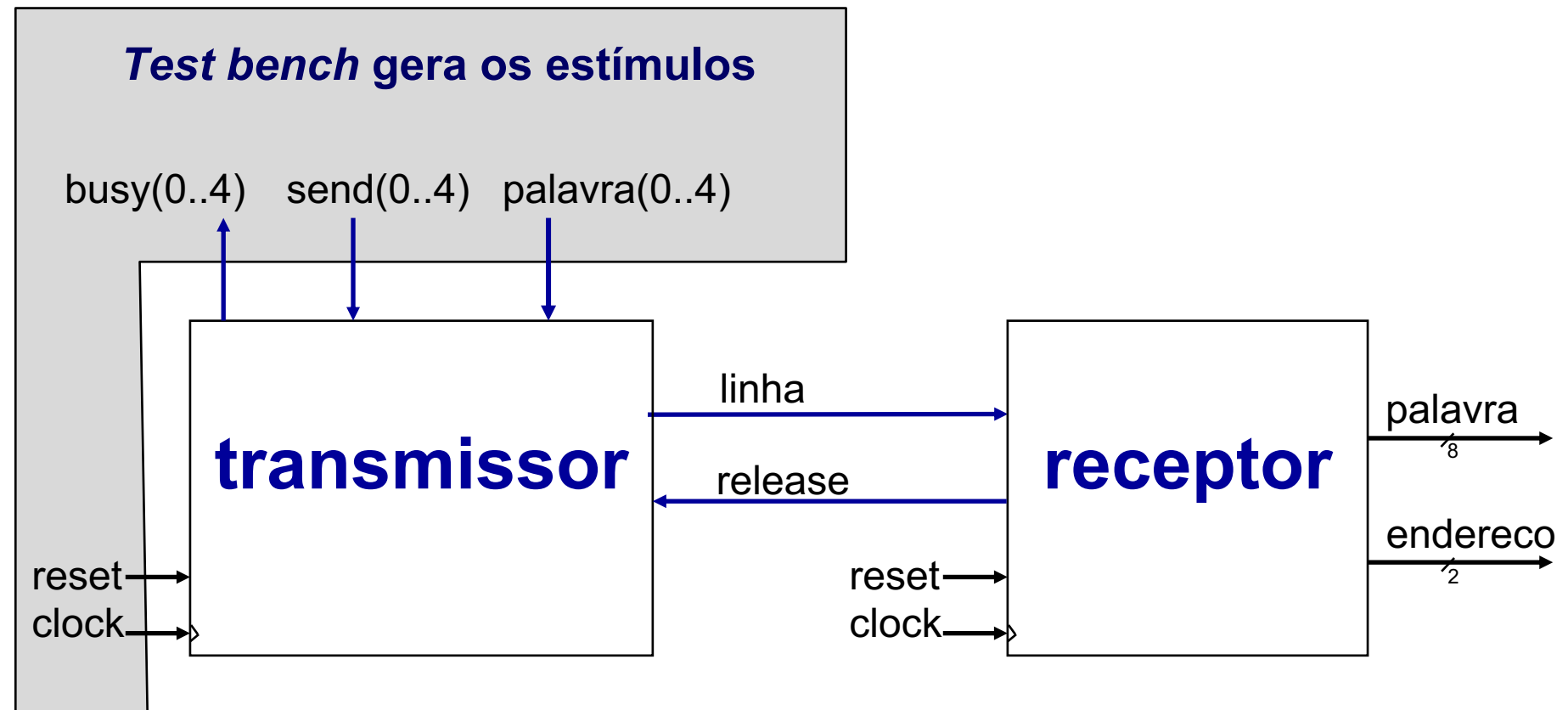
- Lê o dado da linha, a partir do *start bit*, colocando nos registradores de saída o valor do periférico que gerou o dado (**endereço**) e o conteúdo do dado (**palavra**), ao receber o *stop bit*
- Ao final da recepção sobe o **release** (ao receber o *stop bit*)



# Test bench

O test bench gera os estímulos para o transmissor (cuidando os busy), e na saída do receptor verifica-se os resultados

Este arquivo é fornecido





# TP6 – testbench de referência

```
entity tb is
end tb;

architecture tb of tb is
  signal .....
  type test_record is record
    word : words;
    sendt : control;
  end record;

  type padroes is array(natural range <>) of test_record;
  constant padrao_de_teste : padroes := (
    (word => (x"00", x"00", x"2B", x"00"), sendt=>('0', '0', '1', '0')), -- '2' pede
    (word => (x"00", x"1A", x"00", x"22"), sendt=>('0', '1', '0', '1')), -- dois requests
    simultaneos '1' e '3'
    (word => (x"00", x"3C", x"00", x"00"), sendt=>('0', '1', '0', '0')), -- 1 pede
    (word => (x"4D", x"5E", x"6F", x"8B"), sendt=>('1', '1', '1', '1')), -- todos pede
    (word => (x"33", x"00", x"00", x"00"), sendt=>('1', '0', '0', '0')), -- 0 pede
    (word => (x"00", x"00", x"AA", x"DD"), sendt=>('0', '0', '1', '1')), -- 2 e 3 pedem
    (word => (x"8B", x"00", x"00", x"00"), sendt=>('1', '0', '0', '0'))); -- 0 pede
begin
```

**Pedidos de transmissão,  
havendo disputa pela linha**

```
txs : entity work.transmissores
  port map ( clock    => clock, reset=> reset,
            palavra   => palavra,
            send      => send,
            busy      => busy,
            release    => release,
            linha     => linha
          );
```

```
rx: entity work.receptor
  port map( clock    => clock, reset => reset,
            palavra  => saida,
            endereco => endereco,
            linha    => linha,
            release  => release
          );
```

```
reset <= '1', '0' after 5 ns;
clock <= not clock after 5 ns;
```

**Instanciação dos  
transmissores e  
receptor**

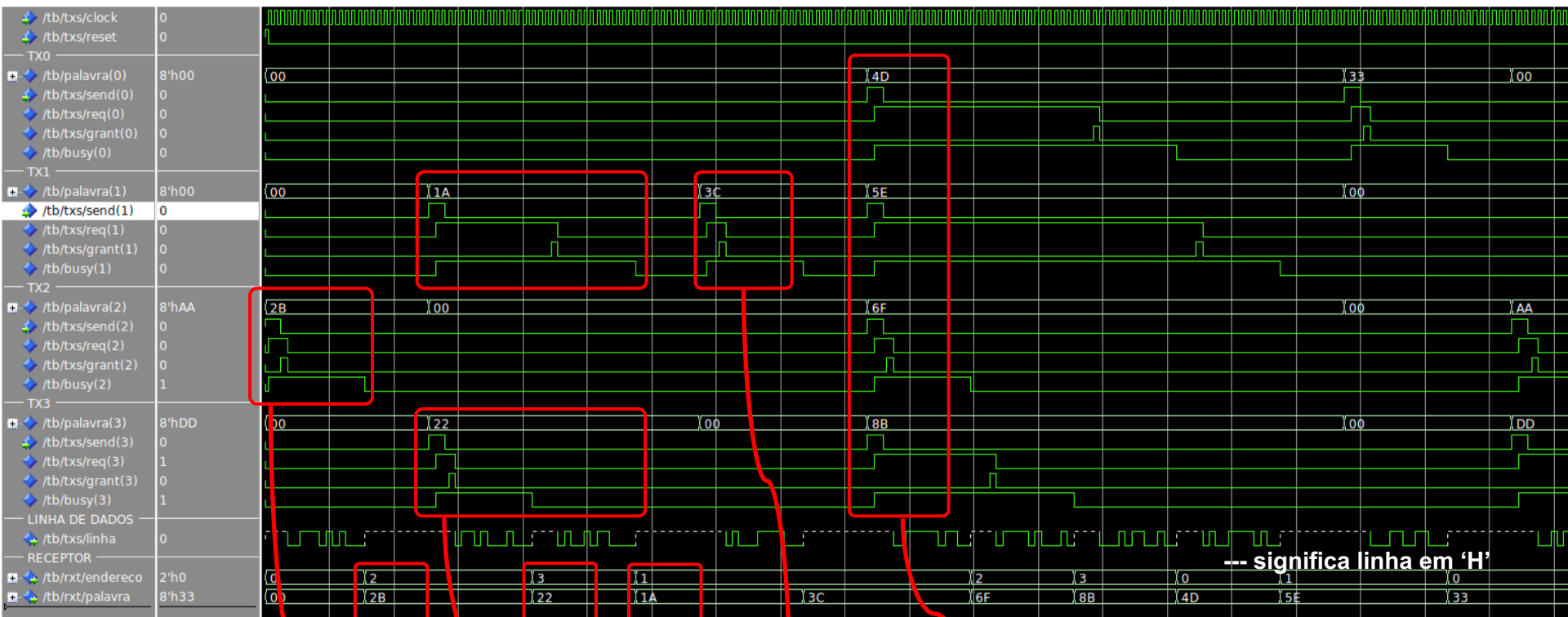
```
test: process
begin
  for i in 0 to padrao_de_teste'high loop
    palavra <= padrao_de_teste(i).word;
    send    <= padrao_de_teste(i).sendt;
    wait for 25 ns;
    -- depois de fazer o send já desce o mesmo
    send    <= ('0', '0', '0', '0');
    wait for 25 ns;
    ---- aguardo todos os transmissores estarem livres
    wait until busy(0)='0' and busy(1)='0' and busy(2)='0' and busy(3)='0';
    wait for 100 ns;
  end loop;
end process;

end tb;
```

**Faz a solicitação de acordo com os  
padrões de teste, e aguarda que todos  
os *busy* estejam em '0'**

# TP6 - simulação

```
constant padrao_de_teste : padroes := (
  (word => (x"00", x"00", x"2B", x"00"), sendt=>('0', '0', '1', '0')), -- '2' pede
  (word => (x"00", x"1A", x"00", x"22"), sendt=>('0', '1', '0', '1')), -- dois requests
  simultaneos '1' e '3'
  (word => (x"00", x"3C", x"00", x"00"), sendt=>('0', '1', '0', '0')), -- 1 pede
  (word => (x"4D", x"5E", x"6F", x"8B"), sendt=>('1', '1', '1', '1')), -- todos pede
  (word => (x"33", x"00", x"00", x"00"), sendt=>('1', '0', '0', '0')), -- 0 pede
  (word => (x"00", x"00", x"AA", x"DD"), sendt=>('0', '0', '1', '1')), -- 2 e 3 pedem
  (word => (x"8B", x"00", x"00", x"00"), sendt=>('1', '0', '0', '0'))); -- 0 pede
```



Tx2 pede para enviar (observar que durante a transmissão o busy fica alto)

recepção correta

**DISPUTA** entre o Tx1 e Tx3  
Tx3 ganha e depois Tx1 é atendido (devido à arbitragem)

Tx1 pede sozinho agora


**Todos pedem ao mesmo tempo.**

Como o Tx1 foi o último a ser atendido, atendo na sequência o 2, 3, 0, 1; mostrando que a arbitragem esta correta

# TP6 – avaliação

- Postar no Moodle um arquivo zip com:
  - Projeto que não ‘compila’ e que não mostre as formas de onda não será avaliado!
  - 2.0 – correta modificação do transmissor
  - 2.0 – correta modificação do árbitro
  - 3.0 – correto desenvolvimento do módulo *transmissores*
  - 3.0 – correto desenvolvimento do módulo *receptor*

**Entrega: até dia 06/12 as 23:59**

 arbitro\_tp6.vhd


modificar conforme visto em aula

 pkg\_tp6.vhd

fornecido

 receptor.vhd


desenvolver conforme especificação

 sim.do

fornecido

 tb.vhd


fornecido

 transmissores.vhd

desenvolver conforme especificação

 tx\_tp6.vhd

modificar conforme visto em aula

 wave.do

fornecido