2.2.4 Demultiplexador

O inverso de um multiplexador (MUX) é um demultiplexador (DEMUX). Neste circuito combinacional o sinal presente em sua única linha de entrada é comutado para uma de suas 2^S saídas, de acordo com s linhas de controle (variáveis de seleção). Se o valor binário nas linhas de controle for x, a saída x é selecionada.

Conforme ilustra a Figura 2.23, o MUX e o DEMUX selecionam o caminho dos dados. Uma determinada saída do demultiplexador corresponderá a uma determinada entrada do multiplexador.

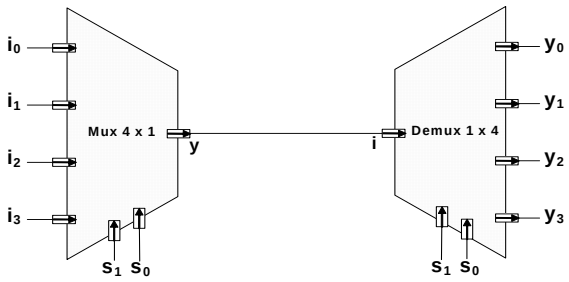
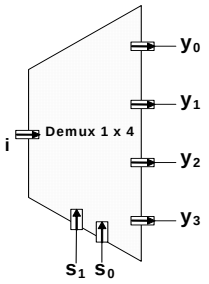


Figura 2.23 – Bloco diagrama de um multiplexador 4x1 conectado a um demultiplexador 1x4.

No multiplexador define-se apenas qual entrada passará seu valor lógico a sua única saída. Um multiplexador pode ser usado para selecionar uma de n fontes de dados que deve ser transmitida através de um único fio. Na outra ponta, um demultiplexador pode ser usado para selecionar de um único fio para um de n destinos. A função de um demultiplexador é o inverso de um multiplexador, e vice versa.

Um demultiplexador de 1xn saídas possui uma única entrada de dados e s entradas de seleção para selecionar uma das ($n = 2^S$) saídas de dados (Figura 2.24).



Entradas			Saídas			
i	S1	S0	y3	y2	y1	y0
x	0	0	0	0	0	x
x	0	1	0	0	x	0
x	1	0	0	x	0	0
x	1	1	x	0	0	0

Figura 2.24 – Bloco diagrama e tabela verdade de um demultiplexador 1x4.

O demultiplexador 1x4 (Figura 2.24 e Figura 2.25) é também chamado decodificador, contudo nesta configuração o circuito possui a função de um distribuidor de dados e pode ser visto como um dispositivo similar a uma chave rotativa unidirecional.

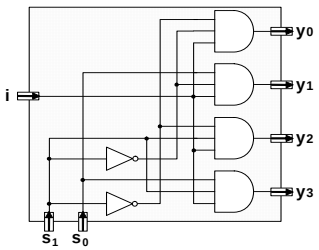


Figura 2.25 – Circuito do demultiplexador 1x4 em nível de portas lógicas.

Uma possível descrição do demultiplexador 1x4 para implementar o circuito da Figura 2.25 é desenvolvida com as declarações WHEN/ELSE. Na seqüência é apresentado código VHDL correspondente, transcrito como segue:

```
-----------------------------------------------
-- Circuito: demultiplexador 1x4:(demux1_1x4.vhd)
--                   s  Selecao da entrada
--                    i  Entrada
--                    y  Saídas,  y(3:0)
-- Utilizacao das declarações de (WHEN/ELSE)
-----------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------
ENTITY demux1x4 IS
      PORT ( s   : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
             i   : IN STD_LOGIC;
             y   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END demux1x4;
-------------------------------------------
 ARCHITECTURE demux1_1x4 OF demux1x4 IS
     BEGIN
           y <=    "0001" WHEN s ="00" And i = '1' ELSE
                   "0010" WHEN s ="01" And i = '1' ELSE
                   "0100" WHEN s ="10" And i = '1' ELSE
                   "1000" WHEN s ="11" And i = '1' ELSE
                   "0000";
END demux1_1x4;
```

A descrição do *testbench* (testbench6a) para validação do demultiplexador 1x4 é desenvolvida a partir do exemplo anterior, para validação multiplexador 4x1 (testbench6). No código VHDL foi acrescentado o componente demultiplexador (`demux1x4`) que é interconectado à saída do multiplexador que servira de estímulo na seqüência selecionada pelo sinal seletor (`tb_sel`). O sinal seletor gerado pelo processo estímulo do *testbench* é conectado simultaneamente às entradas (`sel`) do multiplexador (`mux4x1`) e (`s`) do demultiplexador (`demux1x4`) conforme código VHDL transcrito como segue:

```
-- ****************************************
-- Testbench para simulação Funcional dos
-- Circuito: multiplexador 4x1:(mux1_4x1.vhd)
--           sel (1:0) Selecao da entrada
--                     a Entrada, sel = 00
--                     b Entrada, sel = 01
--                     c Entrada, sel = 10
--                     d Entrada, sel = 11
--                     y Saída (WHEN/ELSE)
-- Circuito: demultiplexador 1x4:(demux1_1x4.vhd)
--                   s  Selecao da entrada
--                    i  Entrada
--                    y  Saídas,  y(3:0)
-- Utilizacao das declarações de (WHEN/ELSE)
-- ****************************************
ENTITY testbench6a IS END;
-----------------------------------------------
-- Testbench para mux1_4x1.vhd
-- Validação sincrona (clk1 e clk2)
-----------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
USE std.textio.ALL;

ARCHITECTURE tb_mux1_4x1 OF testbench6a IS
-----------------------------------
-- Declaração do componente mux2x1
```

```
-----------------------------------
component mux4x1
    PORT ( sel         : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
           a, b, c, d  : IN STD_LOGIC;
           y           : OUT STD_LOGIC);
end component;

component demux1x4
    PORT (  s   : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
           i   : IN STD_LOGIC;
           y   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
end component;

constant T: time := 5 ns;       -- periodo para o clk_1
signal clk_1, clk_2         : std_logic;
signal tb_c, tb_d, tb_yi    : std_logic;
signal tb_sel               : STD_LOGIC_VECTOR (1 DOWNTO 0);

Begin

mux1: mux4x1 PORT MAP ( sel => tb_sel, a => clk_1, b => clk_2,
                        c => tb_c, d => tb_d, y => tb_yi);

demux1: demux1x4  PORT MAP ( s => tb_sel, i => tb_yi, y => open);

clk1: process   -- clk_1 Generator
begin
    clk_1 <= '0', '1' after T/2, '0' after T;
    wait for T;
end process;

estimulo: PROCESS
   begin
      tb_sel <= "00"; clk_2 <= '0';
      tb_c<= '0'; tb_d <= '1';
      wait for 11 ns; tb_sel <= tb_sel + '1';
      loop
          clk_2 <= '1';
          WAIT FOR 2 ns;
          clk_2 <= '0';
          WAIT FOR 8 ns;       -- clock.
          tb_sel <= tb_sel + '1';
          if tb_sel = "01" then tb_c <= '1'; end if;
          if tb_sel = "10" then tb_d <= '0'; end if;
        end loop;
    end PROCESS estimulo;
end tb_mux1_4x1;
```

Os resultados obtidos na saída do demultiplexador (`demux1x4`) devem ser evidentemente idênticos aos da entrada do multiplexador (`mux4x1`) anteriormente descrito. A Figura 2.26 ilustra os dois sinais de relógio (`clk_1, clk_2`) gerados para estímulos das entradas (`a, b`), os sinais (`tb_c, tb_d`) de estímulo das entradas (`c, d`) do MUX. O sinal (`tb_yi`) utilizado para interconectar a saída `y` do MUX na entrada `i` do DEMUX, também estão apresentados, bem como o sinal `s` que comuta os sinais de entrada do MUX e simultaneamente os de saída do DEMUX.

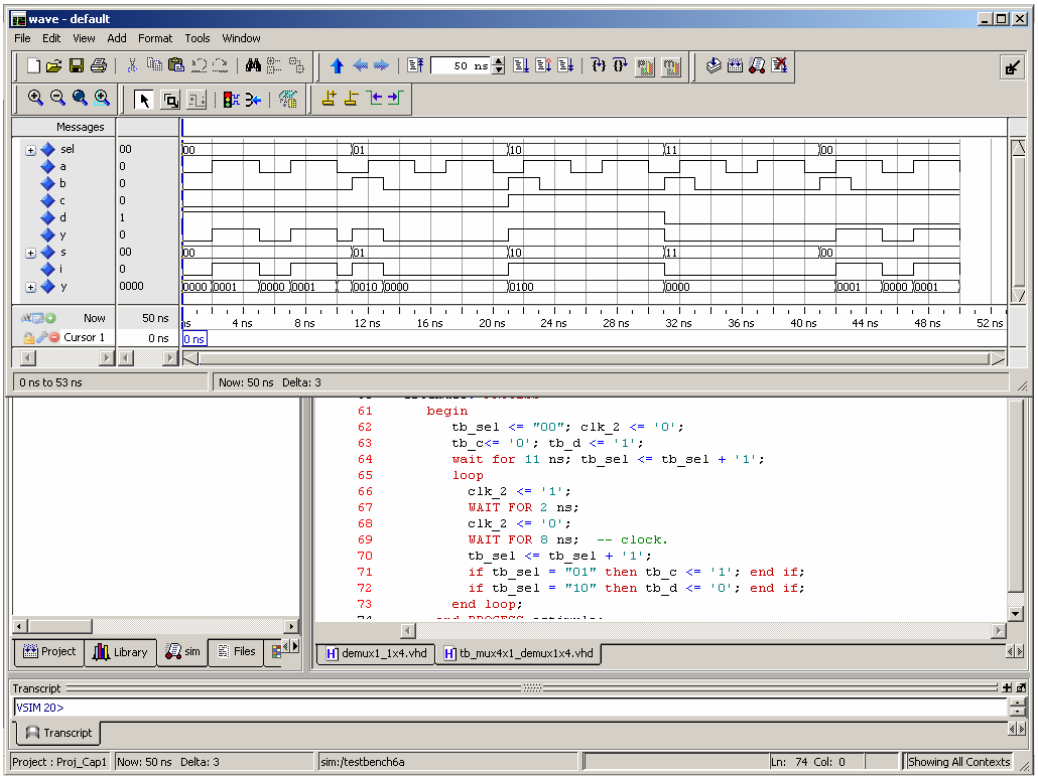


Figura 2.26 - Resultados da simulação do multiplexador 4x1 conectado ao demultiplexador 1x4.

Na Figura 2.27 apresenta-se um detalhamento da simulação do multiplexador 4x1 conectado ao demultiplexador 1x4. As entradas `sel` (do MUX) e `s` (do DEMUX) recebem os mesmo estímulos, para que haja um sincronismo das entradas com as saídas, isto é, i0 seja reproduzido em y0, i1 seja reproduzido em y1, e assim sucessivamente (Figura 2.26). Para que fosse possível utilizar descrições anteriores adotou-se a, b, c e d como entradas do MUX (correspondentes à y0, y1, y2 e y3 respectivamente). As saídas do DEMUX permanecem conforme as definidas em sua *entity* que são y0, y1, y2 e y3. À medida que sel é incrementada, e conseqüentemente s, a cada 11 ns (wait for 11 ns; tb_sel <= tb_sel + '1';) uma das quatro entradas do MUX é selecionada e a mesma é reproduzida na saída correspondente do DEMUX.

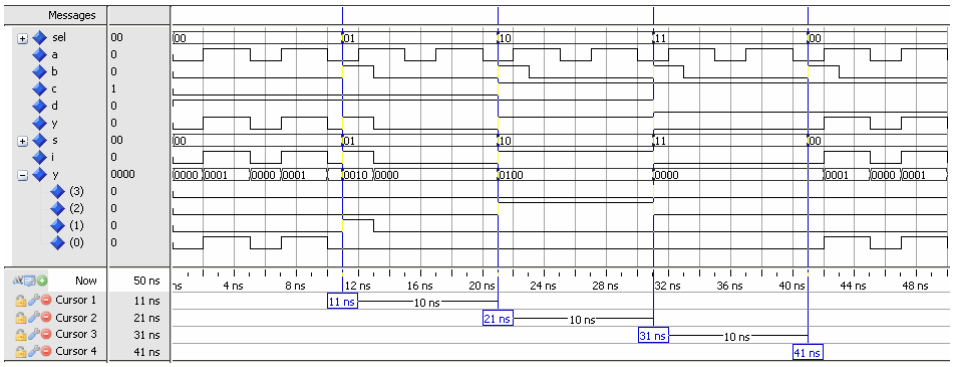


Figura 2.27 – Detalhamento um ciclo de seleção completo MUX/DEMUX.

Observa-se que nesta mesma Figura a saída `y` do DEMUX está expandida detalhando cada uma das saídas (`y(3)`, `y(2)`, `y(1)` e `y(0)`). Com este detalhamento foi possível observar que em 31 ns (Figura 2.28) há um *glitch*. Para expandir um sinal composto de **n** bits (vetor) basta utilizar o botão esquerdo do mouse apontando no ícone "+" ao lado do sinal.

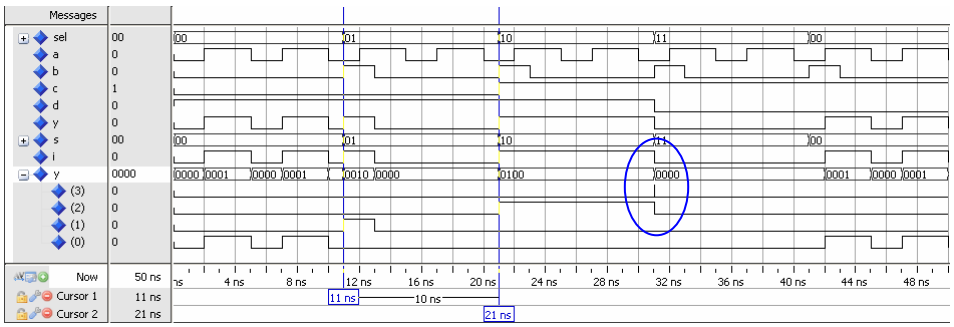


Figura 2.28 – *Glitch* observado na saída y(3) a partir da expansão do vetor de saída y na tela de simulação.

Os conteúdos abordados até aqui permitem uma boa noção do que define as bases do VHDL. Pode-se agora concentrar os projetos no código em si. O código VHDL pode ser combinacional (paralelo) ou seqüencial (concorrente). A classe combinacional está sendo estudada em detalhes neste volume (Volume 1), enquanto que a seqüencial será vista mais profundamente na continuação desta obra (Volume 2).

Esta divisão é muito importante para permitir uma melhor compreensão das declarações que estão destinadas a cada classe de código, bem como as conseqüências de usar-se uma ou outra. Uma comparação da classe combinacional versus seqüencial já está sendo gradativamente introduzida nesta Seção para demonstrar as fundamentais diferenças entre lógica combinacional e lógica seqüencial em nível de descrição de hardware.

O tipo especial de declaração, denominada IF, já vem sendo utilizada nas descrições de *testbenches*. Esta é empregada para especificar condições (sinais de estímulo) desejadas para desenvolver a seqüência dos testes necessários para a validação funcional dos componentes sob teste.

Para concluir esta Seção, discutem-se alguns códigos concorrentes, ou seja, estudam-se as declarações que só podem ser utilizadas fora de PROCESS, FUNCTION, ou PROCEDURES. Elas são as declarações da classe combinacional WHEN e GENERATE. Além destas, outras atribuições que utilizam apenas operadores (lógica, aritmética, etc.) podem naturalmente ser empregadas para criar circuitos combinacionais.