```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
library STD;
use STD.textio.all;

entity mux_8to1 is
      port (A   : in  bit_vector (7 downto 0);
            Sel : in  bit_vector (2 downto 0);
            F   : out bit);
end entity;

architecture mux_8to1_arch of mux_8to1 is
begin
      F <= (A(0) and not Sel(2) and not Sel(1) and not Sel(0)) or
     (A(1) and not Sel(2) and not Sel(1) and Sel(0)) or
     (A(2) and not Sel(2) and Sel(1) and not Sel(0)) or
     (A(3) and not Sel(2) and Sel(1) and Sel(0)) or
     (A(4) and Sel(2) and not Sel(1) and not Sel(0)) or
     (A(5) and Sel(2) and not Sel(1) and Sel(0)) or
     (A(6) and Sel(2) and Sel(1) and not Sel(0)) or
     (A(7) and Sel(2) and Sel(1) and Sel(0));
end architecture;

entity mux_8to1_TB is
end entity;

architecture mux_8to1_TB_arch of mux_8to1_TB is
component demux_1to8
      port (A   : bit_vector (7 downto 0);
            Sel : in  bit_vector (2 downto 0);
            F   : out bit;
end component;
signal A_TB   : bit_vector (7 downto 0);
signal Sel_TB : bit_vector (2 downto 0);
signal F_TB   : bit;
begin
DUT1: mux_8to1 port map (A => A_TB, Sel => Sel_TB, F => F_TB);
STIMULUS: process
signal i : integer;
file Fout: TEXT open WRITE_MODE is “output_vector.txt”;
file Fin: TEXT open READ_MODE is “input_vector.txt”;
variable current_wline : line;
variable current_rline : line;
begin
write(currentwline, string’(“Input=A, Sel, Output=F”));
writeline(Fout, currentwline);
for i in 0 to 7 loop
readline(Fin, current_line);
read(current_line, A_TB); wait for 10 ns;
write(currentwline, string’(“Sel=”));
write(currentwline, Sel_TB);
write(currentwline, string’(“ F=”));
write(currentwline, F_TB);
writeline(Fout, currentwline)
end loop;
end process;
end architecture;
```

---

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.numeric_std_unsigned.all;

entity rom_4x4_async is
port (address : in std_logic_vector(1 downto 0);
data_out : out std_logic_vector(3 downto 0));
end entity;
```

```
architecture rom_4x4_async_arch of rom_4x4_async is
type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);
constant ROM : ROM_type := (0 => x”D”,
1 => x”2”,
2 => x”F”,
3 => x”4”);
begin
data_out <= ROM(to_integer(unsigned(address)));
end architecture;

entity rom_4x4_async_TB is
end entity;

architecture rom_4x4_async_TB_arch of rom_4x4_async_TB is
component rom_4x4_async
port (address : in std_logic_vector(1 downto 0);
data_out : out std_logic_vector(3 downto 0));
end component;
signal address_TB : std_logic_vector(1 downto 0);
signal data_out_TB : std_logic_vector(3 downto 0);
begin
DUT1: rom_4x4_async port map (address => address_TB,
data_out => data_out_TB);
STIMULUS : process
signal i : integer;
begin
for i in 0 to 3 loop
address_TB = std_logic_vector(to_unsigned(i,2));
report “adress= “ & address_TB’image &
“ data_out= “ & data_out_TB’image;
end loop;
end process;
end architecture;
```