Sistemas Digitales (66.17)

Práctica 2 - VHDL

1. Crear un paquete de utilidades al cual se le vayan agregando todas las funciones, procedimientos y declaración de tipos, constantes, etc., que se creen en los puntos siguientes. Declarar una constante M sólo visible dentro del paquete, y otra J visible para todos los usuarios del paquete.

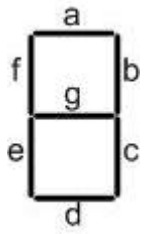
2. Crear otro paquete en el que se deberá incluir la declaración de los componentes utilizados en el transcurso del curso.

3. Implementar la AND genérica de la práctica anterior como una función.

4. Implementar el *shift register* de la práctica anterior utilizando *slices*.

5. Implementar un *procedure* que invierta una señal (MSB ... LSB ➔ LSB ... MSB)

6. Implementar una función que realice la conversión BCD a 7 segmentos de una señal de 4 bits (valores de 0 a 9).



7. Determinar si el siguiente código es correcto. En caso de no serlo, indicar el tipo de error y encontrar la manera de corregirlo, usando siempre la misma sentencia “case”.

```
SIGNAL address : STD_LOGIC_VECTOR(14 downto 0);
...
CASE address IS
WHEN "000010000000000" TO "000010001111111" => ...
...
```

8. Dibujar el diagrama de tiempos de las siguientes asignaciones de señales, incluyendo los retardos d, e indicando las transacciones.

a)
```
ARCHITECTURE ej1a_arq OF ej1a IS
     SIGNAL a, b, c: BIT := ‘0’;
BEGIN
     a <= ‘1’;
     b <= NOT a;
     c <= NOT b;
END ej1a_arq;
```

b)
```
ARCHITECTURE ej1b_arq of ej1b IS
     SIGNAL  a, b, c: bit := '0';
BEGIN
     a <= '1' after 2 ns;
     b <= not a after 3 ns;
     c <= not b after 4 ns;
END ej1b_arq;
```

9. Dibujar el diagrama de tiempos de las siguientes asignaciones de señales.

a)
```
ARCHITECTURE ej2a_arq OF ej2a IS
     SIGNAL x: STD_LOGIC := ‘Z’;
BEGIN
     PROCESS
     BEGIN
           x <= ‘1’ AFTER 5 ns;
           x <= ‘0’ AFTER 8 ns;
           x <= ‘1’ AFTER 6 ns;
           WAIT;
     END PROCESS;
END ej2a_arq;
```

b)
```
ARCHITECTURE ej2b_arq OF ej2b IS
     SIGNAL x: STD_LOGIC := ‘Z’;
BEGIN
     PROCESS
     BEGIN
           x <= TRANSPORT ‘0’ AFTER 5 ns;
           x <= ‘0’ AFTER 3 ns;
           x <= ‘1’ AFTER 11 ns;
           WAIT;
     END PROCESS;
END ej2b_arq;
```

- Si la asignación de la primera señal fuera inercial, habría algún cambio en el diagrama de tiempos?
- Si la asignación de la tercera señal fuera por transporte, habría algún cambio en el diagrama de tiempos?

En caso de que la respuesta sea afirmativa redibujar dicho diagrama.

c)
```
ARCHITECTURE ej2c_arq OF ej2c IS
     SIGNAL x: STD_LOGIC := ‘Z’;
BEGIN
     PROCESS
```

```
        BEGIN
              x <= TRANSPORT ‘1’ AFTER 5 ns;
              x <= TRANSPORT ‘0’ AFTER 8 ns;
              x <= TRANSPORT ‘1’ AFTER 6 ns;
              WAIT;
        END PROCESS;
    END ej2c_arq;
```

d)
```
    ARCHITECTURE ej2d_arq OF ej2d IS
        SIGNAL x: STD_LOGIC := ‘Z’;
    BEGIN
        PROCESS
        BEGIN
              x <= ‘1’ AFTER 5 ns, ‘0’ AFTER 10 ns, ‘1’ AFTER 20 ns;
              x <= ‘0’ AFTER 12 ns, ‘1’ AFTER 16 ns, ‘0’ AFTER 25 ns;
              WAIT;
        END PROCESS;
    END ej2d_arq;
```

e)
```
    ARCHITECTURE ej2e_arq OF ej2e IS
        SIGNAL x: STD_LOGIC := ‘Z’;
    BEGIN
        PROCESS
        BEGIN
              x <= ‘1’ AFTER 5 ns, ‘0’ AFTER 10 ns, ‘1’ AFTER 20 ns;
              x <= ‘0’ AFTER 12 ns, ‘1’ AFTER 16 ns, ‘0’ AFTER 25 ns;
              x <= ‘1’ AFTER 20 ns;
              WAIT;
        END PROCESS;
    END ej2e_arq;
```

f)
```
    ARCHITECTURE ej2f_arq OF ej2f IS
        CONSTANT N: INTEGER := 4;
        CONSTANT TA: BIT_VECTOR (0 to N-1) := "1010";
        CONSTANT TB: BIT_VECTOR(0 to N-1) := "1001";
        SIGNAL a, b, c: BIT := '0';
    BEGIN
        c <= a XOR b AFTER 10 ns;
        PROCESS
        BEGIN
              FOR i IN 0 TO N-1 LOOP
                    a <= TA(i);
                    b <= TB(i);
                    WAIT ON c FOR 20 ns;
              END LOOP;
              WAIT;
        END PROCESS;
    END ej2f_arq;
```

g)
```
    ARCHITECTURE ej2g_arq OF ej2g IS
        SIGNAL a: BIT:= ‘0’;
        SIGNAL b: BIT:= ‘1’;
```

```
BEGIN
     PROCESS
     BEGIN
           WAIT FOR 10ns;
           a <= '1';

           WAIT ON b;
           a <= '0';
     END PROCESS;

     PROCESS
     BEGIN
           WAIT UNTIL a='1';
           WAIT FOR 20ns;
           b <= '0';
     END PROCESS;
END ej2g_arq;
```

10. Crear una señal clock de 20ns de período. Luego, mediante otra asignación, filtrarla (fig 3.a). Finalmente, con el mismo criterio, lograr un desplazamiento (fig 3.b).

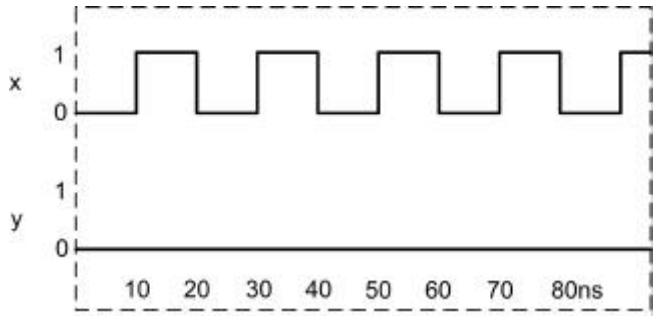


fig. 3.a

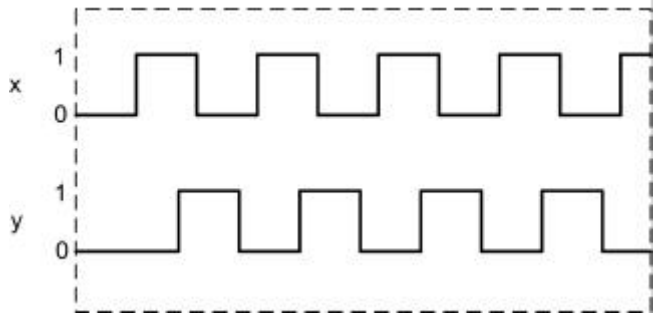


fig. 3.b

11. Implementar una función de resolución que resuelva la asignación de múltiples señales por medio de una OR.

12. Implementar en VHDL, por comportamiento, una compuerta AND genérica (utilizar la sentencia *Generic*).

13. Implementar un registro de desplazamiento (*shift register*) genérico (utilizar la sentencia *Generate*).

14. Implementar un multiplexor de 4 entradas utilizando la sentencia *With-Select* .

15. Implementar un contador de N bits por comportamiento usando las funciones de la librería IEEE.

16. En la implementación del contador de la práctica 1, qué se modificaría en el caso de utilizar un puerto inout?.

17. Implementar en VHDL un full-adder en modo estructural y luego convertirlo en un sumador genérico de N bits utilizando la sentencia *generate*. Los operandos deben estar definidos de la siguiente manera:

    A: in std_logic_vector(N-1 downto 0);
    B: in std_logic_vector(N-1 downto 0);

    Luego de haber corroborado el funcionamiento del sumador modificar la definición de los operandos de la siguiente manera:

    A: in std_logic_vector(0 to N-1);
    B: in std_logic_vector(0 to N-1);

    El resultado obtenido será el mismo que en el caso anterior? Explicar.

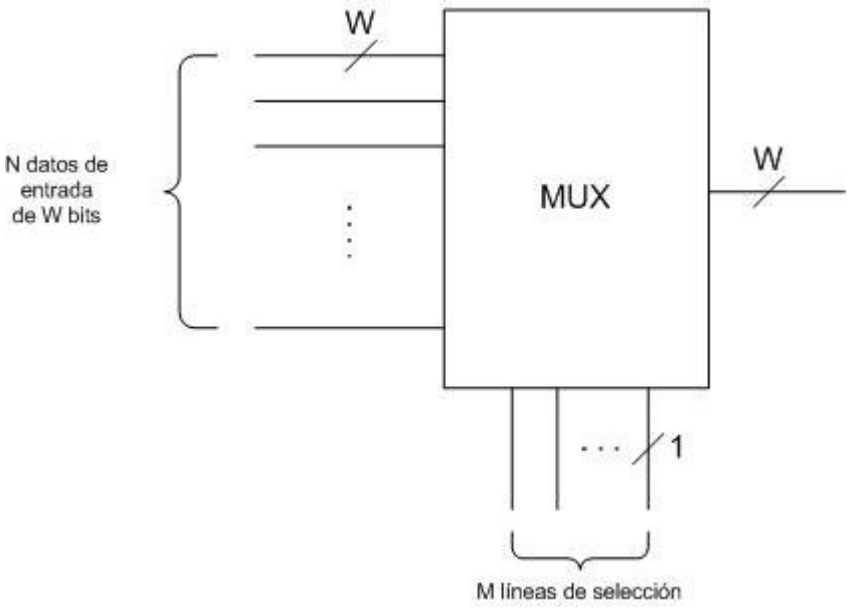
18. Convertir el anterior sumador en un sumador-restador.

19. Implementar en VHDL el sumador-restador anterior utilizando las funciones de la librería IEEE.

20. Implementar en VHDL una memoria ROM.

21. Implementar en VHDL una memoria RAM.

22. Implementar un multiplexor genérico de acuerdo a la figura siguiente.
    Aclaración: se debe crear un tipo WORD_VECTOR (*array de rango genérico de std_logic_vector(W-1 downto 0)*).



23. Implementar un *Procedure* que convierta un número binario (de N bits) a entero, utilizando atributos (utilizar los predefinidos para arreglos). El encabezado del procedure es el siguiente:

    Procedure bin2int (bin: in bit_vector; int: out integer)

    Ayuda: los atributos predefinidos para arreglos son:
    'LEFT, 'RIGHT, 'HIGH, 'LOW, 'RANGE, 'REVERSE_RANGE, 'LENGTH

24. Implementar un *Procedure* que convierta un número entero a un binario (de N bits), utilizando atributos (utilizar los predefinidos para arreglos). El encabezado del procedure es el siguiente:

    Procedure int2bin (int: in integer; bin: out bit_vector)

Nota: en todos los puntos se deberá implementar una entidad de simulación y simular el componente creado.