Código Concurrente

Código Concurrente

El código VHDL puede ser concurrente o secuencial. Únicamente las sentencias colocadas dentro de un PROCESS, FUNCTION o PROCEDURE (los últimos dos son llamados subprogramas) son ejecutas secuencialmente.

Sin embardo VHDL por inherentemente concurrente, un PROCESS, como un todo, es también concurrente con respecto de cualquier otra sentencia (externa).

Hay tres sentencias puramente concurrentes (que sólo se pueden usar fuera de un PROCESS o subprograma) las cuales son: WHEN, SELECT y GENERATE.

Código Concurrente

El código concurrente esta destinado únicamente para el diseño de circuitos combinacionales.

Nota. Un circuito combinacional es un circuito en las que sus salidas dependen únicamente estado actual de sus entradas.

Nota. Un circuito secuencial es aquel en las que sus salidas dependen de los estados previos del sistema, junto con una señal de reloj que controla la evolución del sistema.

WHEN

Es la sentencia condicional más simple. Su sintaxis es la siguiente

```
assigment_expression WHEN conditions ELSE
    assignment_value WHEN conditions ELSE
    …;
```

Aspectos de WHEN:

- ✓ Acepta múltiples condiciones (expresiones booleanas) que se pueden agrupar usando AND, OR y NOT.
- ✓ No requiere que todos los valores de entrada (casos) sean especificadas. Sin embargo, es buena práctica (recomendación) cubrir todos los casos para evitar la inferencia de Latches por parte del sintetizador. El uso de la palabra reservada OTHERS es útil.

WHEN

Ejemplo.

```
x <= '0' WHEN rst='0' ELSE
     '1' WHEN (a = '0' OR b = '1') ELSE
      '-'; --don't care

y <= "00" WHEN (a AND b)= '1' ELSE
     "11" WHEN (a AND b)= '0' ELSE
     "ZZ"; -- alta impedancia
```

SELECT

SELECT es otra sentencia concurrente. Su definición es

```
WITH identifier SELECT
      assignment_expression WHEN values,
            assignment_value WHEN values,
            …;
```

Aspectos de SELECT:

- ✓ Permite el uso de múltiples valores (en lugar de múltiples condiciones), que se pueden agrupar con " | " (que significa "or") o "TO" (para rango):

```
WHEN value1  |  value 2  |  … -- value1 o value2 o …
WHEN value1 TO value2        -- rango
```

SELECT

- ✓ Requiere que todos los valores de entrada sean cubiertos (tabla de verdad completa). La palabra OTHERS es útil en este caso.

Ejemplo

```
WITH control SELECT
       y <= "000" WHEN 0 | 1,
            "100" WHEN 2 TO 5,
            "Z--" WHEN OTHERS;
```

GENERATE

Es otra sentencia concurrente. En su forma más popular (unconditional GENERATE), es equivalente a la sentencia LOOP. Se utiliza para repetir una sección de código un cierto número de veces.

Uncoditional GENERATE (también llamado FOR-GENERATE) es usado para crear múltiples instancias de una sección de código.

Su sintaxis se pospone hasta la introducción de las sentencias IF y LOOP.

Breve Recordatorio

Para las prácticas siguientes es necesario recordar los siguiente puntos:

- ✓ Los operadores aritméticos son +, - , *, /.
- ✓ Las operaciones aritméticas están disponibles en el paquete *std_logic_arith* de la biblioteca de la ieee.
- ✓ La forma en que el operador trabaja sus operadores está en función de los paquetes *std_logic_unsigned* y *std_logic_signed*.
- ✓ Es deseable usar para los diseños el tipo *std_logic* y *std_logic_vector*.
- ✓ Los vectores de tipo de datos se acceden por medio de (*índice/posición*).
- ✓ En la declaración de un arreglo de tipos de dato la palabra reservada "downto" indica que la posición más significativa esta a la derecha del arreglo y la menos significativa a la izauierda.

Breve Recordatorio

- ✓ En la declaración de un arreglo de tipos de dato la palabra reservada “to” indica que la posición más significativa esta a la izquierda del arreglo y la menos significativa a la derecha.
- ✓ SIGNAL (cable) permite conectar varias unidades de código.

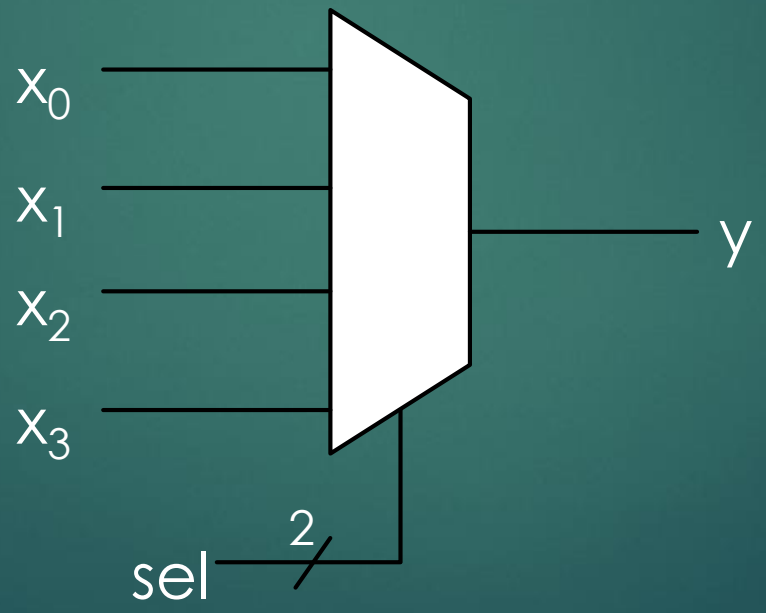
Ejemplo. Arreglos de tipos de datos y acceso a sus elementos.

```
X : std_logic_vector(3 downto 0) :=  "1000"; -- MSB a LSB
Y : std_logic_vector(0 to 3) :=  "0100" -- LSB a MSB

X(0) -- x(0) es igual a 1.
Y(0) -- y(0) es igual a 0.
X(2 downto 1) -- es igual a "00".
```

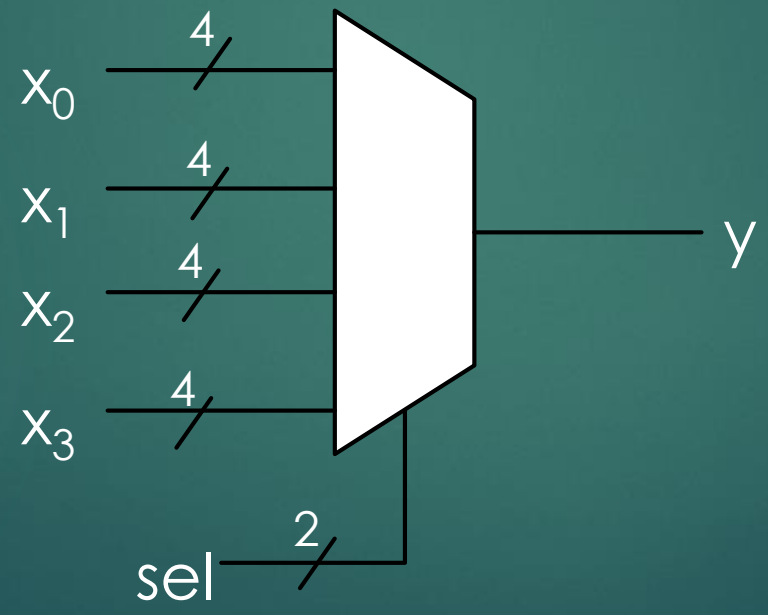
Ejercicios 1

1. Implemente un multiplexor de 4 x 1 (cuatro entradas una salida) usando la ecuación $y = sel'_1 \cdot sel'_0 \cdot x_0 + sel'_1 \cdot sel_0 \cdot x_1 + sel_1 \cdot sel'_0 \cdot x_2 + sel_1 \cdot sel_0 \cdot x_3$, donde ´ indica negación.



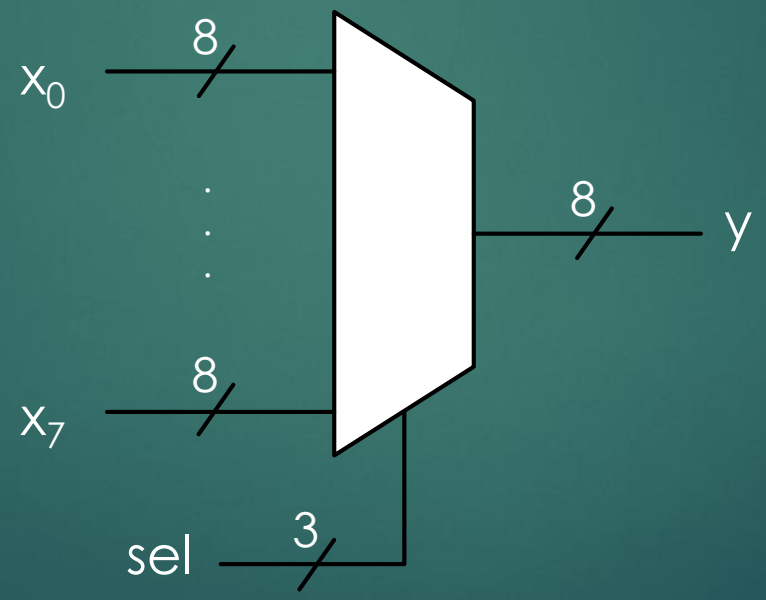
Ejercicios

2. Implemente un multiplexor de 4 x 1 usando la sentencia WHEN, donde cada entrada del multiplexor es de 4 bits.



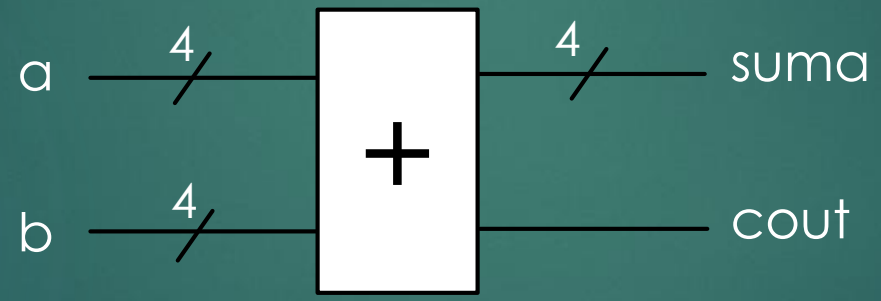
Ejercicios

3. Implemente un multiplexor de 8 x 1 usando la sentencia SELECT, donde cada entrada del multiplexor es de 8 bits.



Ejercicios

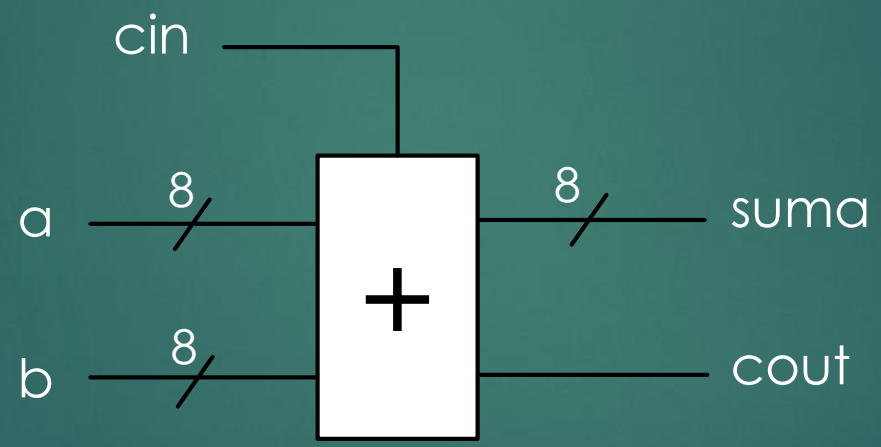
4. Implemente un sumador de 4 bits sin signo y sin acarreo de entrada.



Nota. Use los operadores de suma y concatenación.

Ejercicios

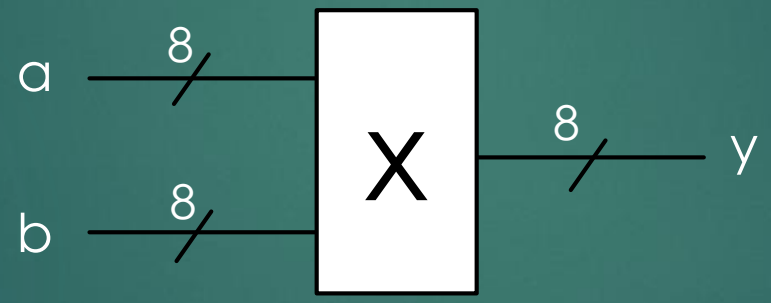
5. Implemente un sumador de 8 bits con signo y acarreo de entrada.



Nota. Use los operadores de suma y concatenación.

Ejercicios

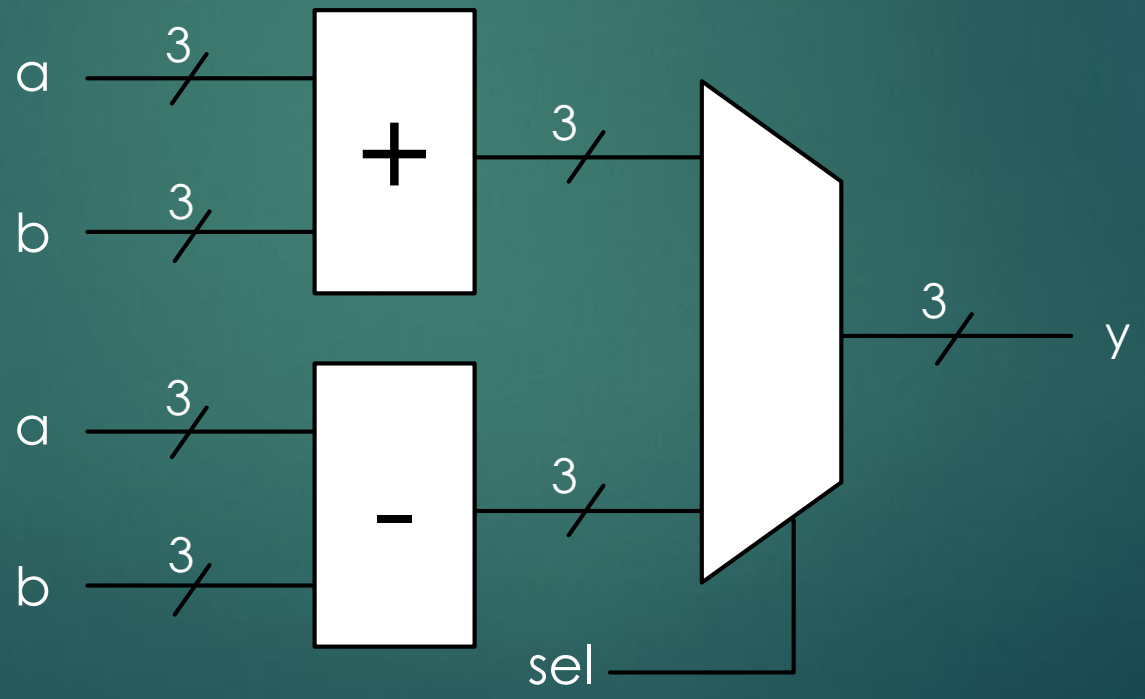
6. Implemente un multiplicador de 8 bits con signo.



Nota. Haga el cambio pertinente para hacer el mutiplicador sin signo.

Ejercicios

7. Implemente el circuito que aparece en la figura.



Ejercicios

8. Implemente el circuito con la siguiente tabla de verdad

Unidad	Instrucción	Operación	opcode
Lógica	Complemento de a	y = NOT a	0000
	Complemento de b	y = NOT b	0001
	AND	y = a AND B	0010
	OR	y = a OR B	0011
	NAND	y = a NAND B	0100
	NOR	y = a NOR B	0101
	XOR	y = a XOR B	0110
	XNOR	y = a XNOR B	0111
Aritmética	Transferir a	y = a	1000
	Transferir b	y = b	1001
	Incrementar a	y = a + 1	1010
	Incrementar b	y = b + 1	1011
	Decrementar a	y = a - 1	1100
	Decrementar b	y = b - 1	1101
	Sumar a y b	y = a + b	1110
	Sumar a, b y cin	y = a +b + cin	1111