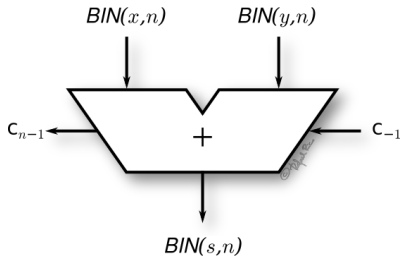
LABORATORIO DE ARQUITECTURA DE COMPUTADORES

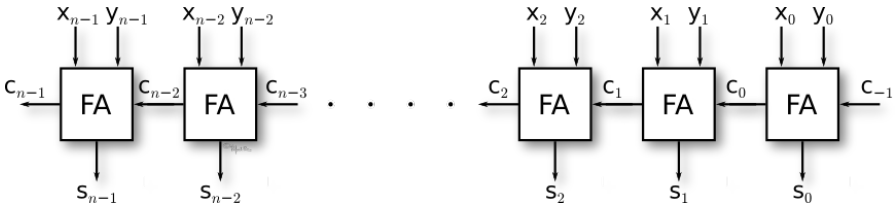
SESIÓN 4: sumador RCA

Objetivo

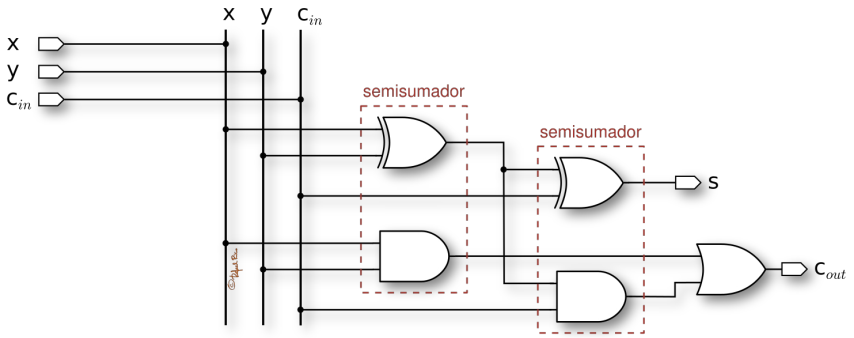
El sumador binario es un operador fundamental ya que es el núcleo de casi cualquier otro operador aritmético que podamos imaginar: sumadores-restadores en diferentes sistemas de representación, comparadores, multiplicadores, divisores, etc.



El sumador binario más sencillo es el RCA (*Ripple-Carry Adder*) o sumador propagador. Está constituido por una batería de sumadores completos que se comunican unos a otros, del menor al mayor peso, los acarreos. La figura siguiente ilustra su disposición y funcionamiento.



En esta práctica vamos a modelar el sumador RCA usando la sentencia GENERATE del VHDL. Como sumador completo primero usaremos el trabajado en la sesión anterior (sumador completo basado en la síntesis suma de productos) y luego modelaremos y usaremos el basado en semisumadores cuyo esquema reproducimos a continuación:



Diseñaremos un *test bench* para el sumador RCA y lo aplicaremos a ambos casos, es decir, cuando el sumador completo sea el derivado como suma de productos y cuando sea el derivado de semisumadores. También probaremos los retardos utilizando diferentes tecnologías.

Sentencia GENERATE en VHDL

En una práctica anterior introdujimos la sentencia GENERATE de VHDL que nos facilita la instanciación de modelos en estructuras repetitivas. Concretamente, vimos que para una batería de puertas AND podíamos escribir lo siguiente:

```
bateria: FOR i IN 0 TO n-1 GENERATE
  puertas: and2 PORT MAP (x(i), y(i), r(i));
END GENERATE;
```

No obstante, a veces dichas estructuras repetitivas son ligeramente diferentes en sus extremos, por ejemplo, en el peso menor o inicial y en el mayor o final. Es el caso del sumador RCA que recibe en su celda de menor peso un acarreo entrante externo (c_{in} o c_{-1}) y que produce en su celda de salida un acarreo saliente (c_{out} o c_{n-1}). Para estos casos, la sentencia GENERATE se puede usar junto con sentencias condicionales.

En el ejemplo siguiente, se ilustra el uso de GENERATE en combinación con sentencias IF para diseñar un RCA cuya primera celda es un semisumador, es decir, no tiene en cuenta ningún posible acarreo entrante.

```
G: FOR i IN 0 TO 7 GENERATE
  lsb: IF i = 0 GENERATE
  primero: medio_sumador PORT MAP (op1(0), op2(0), suma(0), c(0));
  END GENERATE lsb;

  siguientes: IF i > 0 AND i < 7 GENERATE
  intermedios: sumador PORT MAP (op1(i), op2(i), c(i-1), suma(i), c(i));
  END GENERATE siguientes;

  msb: IF i = 7 GENERATE
  ultimo: sumador PORT MAP (op1(i), op2(i), c(i-1), suma(i), cout);
  END GENERATE msb;
END GENERATE G;
```

Test bench

El *test bench* debe cumplir dos funciones: comprobar que las sumas son correctas y medir el retardo del camino crítico.

Para comprobar que el operador realiza su función correctamente podemos convertir el **`BIT_VECTOR`** del resultado a su entero correspondiente (teniendo en cuenta el acarreo) y comprobar su exactitud con la realidad mediante la sentencia ASSERT. A continuación se propone un posible código.

```
      valor:=0;
      FOR k IN vector_suma'RANGE LOOP
        IF vector_suma(k)='1' THEN
          valor:=valor + 2**k;
        END IF;
      END LOOP;
      --tengo que tener en cuenta el acarreo de salida
      IF carry_out='1' THEN
        valor:=valor + 2**TAMANO_PALABRA;
      END IF;

      ASSERT valor = i+j REPORT "resultado incorrecto" SEVERITY ERROR;
```

En el código precedente, las variables `i` y `j` son los valores enteros de los operandos de entrada al sumador.

La sentencia ASSERT también nos permite sacar por pantalla datos del modelo. Es interesante saber hacerlo con objeto de tener herramientas que nos permitan depurar el código. A continuación se presenta un ejemplo:

```
ASSERT false REPORT "i = " & integer'image(i) SEVERITY NOTE;
```

Para medir el retardo, introducimos una combinación de operandos de entrada que sepamos produce al mayor retardo posible ya que ha de conmutar toda la cadena de acarreos intermedia (por ejemplo, los operandos `X="1 ··· 1", Y="0 ··· 0" y cin = '1'` después de `X="0 ··· 0", Y="0 ··· 0" y cin = '0'`). Luego usamos la sentencia WAIT UNTIL para que nos ofrezca el instante preciso en el que conmuta la señal de acarreo y/o la de suma. A continuación se propone un ejemplo.

```
vector_x <= (others=>'0');
vector_y <= (others=>'0');
carry_in <= '0';
WAIT FOR TAMANO_PALABRA*10 ns;
vector_x <= (others=>'1');
carry_in <= '1';
WAIT UNTIL carry_out='1' AND vector_suma(TAMANO_PALABRA-1) = '0';
```

En el ejemplo reproducido más arriba vemos cómo se rellena un **`BIT_VECTOR`** con ceros o unos según interese.

Prácticas

A) Escriba el modelo de sumador RCA de *n* bits.

Utilizando la sentencia GENERATE modele un sumador RCA para *n* bits. Instancie el sumador completo escrito en la sesión anterior (ecuaciones lógicas en forma de suma de productos) que admitía puertas lógicas **con más de 2 entradas**. Utilice los retardos típicos de la familia lógica TTL.

B) Escriba el *test bench* adecuado.

El *test bench* deberá comprobar el correcto funcionamiento del operador RCA y deberá darnos el retardo del mismo introduciendo aquella combinación de señales que deban propagarse por el camino crítico. Suponga que trabajamos con **operandos de 4 bits**. ¿Cómo cambian los retardos si utilizamos la familia lógica CMOS?

C) Escriba el modelo de sumador completo basado en semisumadores.

Utilice el esquema presentado en la sección “Objetivos”.

D) Utilice el *test bench* anteriormente diseñado para comprobar el correcto funcionamiento y medir la nueva longitud del camino crítico.

Intercambie el modelo del sumador completo anterior por el nuevo en el modelo de RCA y compruebe su funcionamiento con el mismo *test bench* del apartado B. ¿Qué retardo tenemos ahora? ¿Cómo cambian los retardos si utilizamos la familia lógica CMOS?

E) Elabore una memoria con los modelos y los resultados

Presente los dos modelos VHDL del sumador completo, los del sumador RCA con cada uno de los anteriores y el *test bench* utilizado. Construya una tabla con los retardos obtenidos para cada sumador y familia lógica. ¿Con qué modelo de sumador completo obtenemos antes el acarreo de salida que el MSB de la suma? ¿Por qué sucede esto?