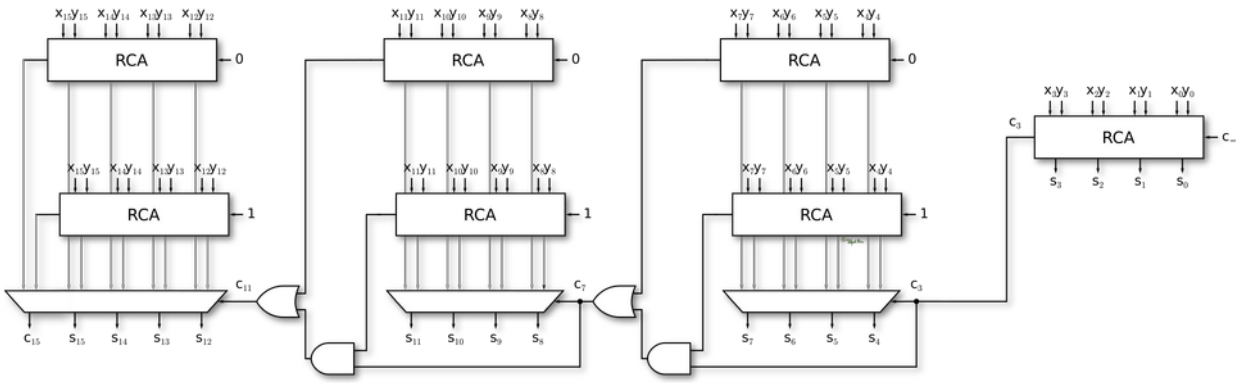
LABORATORIO DE ARQUITECTURA DE COMPUTADORES

SESIÓN 5: selector de acarreo CSLA

Objetivo

El sumador RCA es sencillo pero lento. Por ese motivo se han desarrollado diferentes propuestas que aceleran la operación de suma. Una de ellas es el sumador **selector de acarreo** o CSLA (*Carry-Select Adder*).

Como vemos en la figura siguiente, la idea es trocear los operandos de *n* bits en porciones pequeñas de *k* bits. Con cada una de esas porciones se hacen dos sumas: una con acarreo entrante '0' y otro con acarreo entrante '1' de manera que cuando ya se conocen los acarreos verdaderos se selecciona la porción correcta en los multiplexores. Cada uno de esos bloques de *k* bits se suma en sumadores RCA que siendo pequeños tienen un tiempo de retardo aceptable ($r_{RCA} \approx 2k\, r_g$).



El acarreo se propaga de bloque en bloque RCA por medio de los circuitos anticipadores de acarreo basados en la función G y la función P. Como sabemos, la función G se obtiene como salida de un sumador que tiene acarreo entrante '0' y la función P se obtiene como salida de un sumador que tiene como acarreo entrante '1'.

El retardo de este sumador medido en niveles lógicos (r_g) es:

$$r_{CSL} = \left\lceil 2k + 2\left(\frac{n}{k} - 2\right) + 3 \right\rceil r_g$$

siendo *n* el número de bits de los operandos, *k* el tamaño de los bloques RCA y el último 3 el retardo del multiplexor. Como podemos deducir de la ecuación del retardo, el valor de *k* afecta al primer sumando directamente y al segundo inversamente de manera que hay que llegar a un compromiso que optimice el retardo. Veamos cuál es el mínimo de la función.

$$\frac{\partial r_{CSL}}{\partial k} = 0 \qquad \frac{\partial r_{CSL}}{\partial k} = 2 - 2\frac{n}{k^2} = 0 \qquad \frac{n}{k^2} = 1 \qquad k = \sqrt{n}$$

En esta práctica vamos a modelar un sumador CSLA para 16 bits usando los bloques RCA ya modelados en la sesión anterior instanciados para *k* igual a 4 bits. También utilizaremos el *test bench* programado en la sesión anterior para obtener el retardo del circuito.

Modelo VHDL de un multiplexor 2:1

En esta sesión necesitamos escribir los modelos que den lugar a un multiplexor 2:1 para 4 bits. A continuación podemos encontrar un ejemplo **estructural** de multiplexor 2:1 para 1 bit:

```
ENTITY mux21 IS
  PORT (e1, e2, control: IN BIT; s: OUT BIT);
END mux21;

ARCHITECTURE estructural OF mux21 IS
  COMPONENT inv
    PORT (e: IN BIT; sal: OUT BIT);
  END COMPONENT;
  COMPONENT and2
    PORT (e1, e2: IN BIT; sal: OUT BIT);
  END COMPONENT;
  COMPONENT or2
    PORT (e1, e2: IN BIT; sal: OUT BIT);
  END COMPONENT;

  FOR negado_control: inv USE ENTITY WORK.inv(comportamiento);
  FOR puerta_and_1, puerta_and_2: and2 USE ENTITY WORK.and2(comportamiento);
  FOR puerta_or: or2 USE ENTITY WORK.or2(comportamiento);

  SIGNAL no_control, s_and_1, s_and_2: BIT;

BEGIN
  negado_control: inv PORT MAP (control, no_control);
  puerta_and_1: and2 PORT MAP (no_control, e1, s_and_1);
  puerta_and_2: and2 PORT MAP (control, e2, s_and_2);
  puerta_or: or2 PORT MAP (s_and_1, s_and_2, s);
END estructural;
```

Como vemos, en este diseño el número de niveles lógicos que hemos de atravesar es 3: uno para el inversor, otro para los productos en las puertas AND y finalmente otro para la suma en la OR.

Usando la sentencia GENERATE podremos fácilmente producir el modelo de multiplexor para 4 bits.

Consideraciones de orden práctico

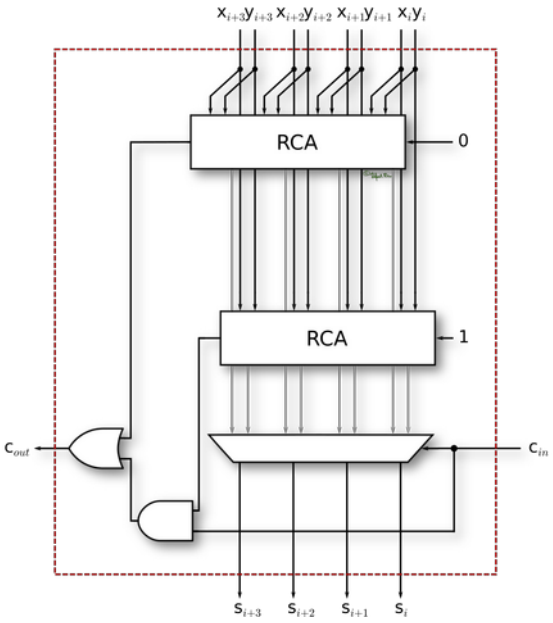
A la hora de realizar la práctica nos vamos a encontrar con ciertos asuntos que requieren tomar una decisión en un sentido o en otro, ambos válidos, pero que es importante pensar con antelación.

En primer lugar, hemos de pensar cómo vamos a **medir los retardos**. ¿En tiempo físico (ns) o en niveles lógicos (r_g)? Si dejamos las puertas lógicas con sus retardos típicos TTL obtendremos retardos en tiempo físico que no serán exactamente los esperados según la expresión del retardo dada más arriba. Si deseamos obtener retardos en niveles lógicos sabemos que VHDL sólo trabaja con tiempo físico.

Un solución para trabajar con niveles lógicos consiste en asignar a todas las puertas un retardo físico de 1ns. En ese caso, el retardo que nos presente la simulación VHDL en ns será equivalente a puertas lógicas (1ns = r_g).

Otro asunto importante es cómo vamos a agrupar los diferentes elementos del sumador en su diseño **jerárquico estructural**. Podemos modelar por un lado selectores de suma y por otro bloques anticipadores o bien podemos integrarlos. Cualquier opción es válida pero la propuesta siguiente es posiblemente la más sencilla de programar usando GENERATE.

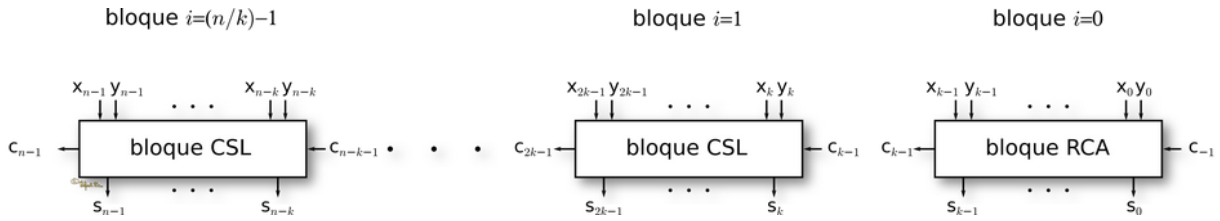
Se recomienda modelar un circuito que contenga los 2 RCA con acarreo '0' y '1' respectivamente más el multiplexor que ofrece la suma verdadera a la salida. Además, el bloque CSLA deberá incluir la anticipación de acarreo a la etapa siguiente.



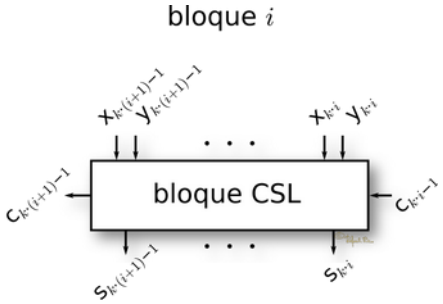
Como vemos, este bloque ofrece hacia fuera los mismos puertos que un sumador RCA, es decir, las entradas de los operandos, la salida del resultado y los acarreos entrante y saliente. Este hecho simplifica la programación y la vista del código.

Quizá la única dificultad que podemos verle es que obtiene el acarreo de salida en el último bloque de una forma diferente al primer esquema mostrado en este guión pero no es ni incorrecto ni menos eficiente.

Finalmente, podemos encontrarnos con alguna dificultad para barrer correctamente los índices usando una sentencia GENERATE. La figura siguiente propone un forma de hacerlo sencilla. Asumimos que los operandos son de tamaño *n* bits y los bloques RCA de tamaño *k* bits. En definitiva, tenemos *n*/*k* bloques desde el *i*=0 hasta el *i*=*n*/*k* – 1.



Es decir, cada bloque procesa la siguiente porción de los operandos de entrada:



Que escrito en VHDL daría lugar a un PORT MAP similar al que se expone a continuación con la definición de componente que se ofrece.

```
COMPONENT bloque_XX_k_bits
  PORT (X, Y: IN BIT_VECTOR (k-1 DOWNTO 0);
        R: OUT BIT_VECTOR (k-1 DOWNTO 0);
        …
        …
        …);
END COMPONENT;


  PORT MAP (X(k*(i+1)-1 DOWNTO k*i),
            Y(k*(i+1)-1 DOWNTO k*i),
            R(k*(i+1)-1 DOWNTO k*i),
            …
            …
            …);
```

Prácticas

A) Escriba el modelo de sumador CSLA de 16 bits.

El bloque CLSA propuesto más arriba conforma los bloques segundo y siguientes, y es sencillo integrarlo en el modelo del sumador utilizando la sentencia GENERATE. El primer bloque será un sencillo sumador RCA.

B) Determine el retardo del sumador utilizando el *test bench* de la sesión anterior.

Pruebe con diferentes implementaciones del sumador completo para encontrar cuál es la que mejor retardo produce.

C) Elabore una memoria con los modelos y los resultados obtenidos.