# LABORATORIO DE ARQUITECTURA DE COMPUTADORES

## SESIÓN 3: síntesis e implementación del sumador completo

### Objetivo

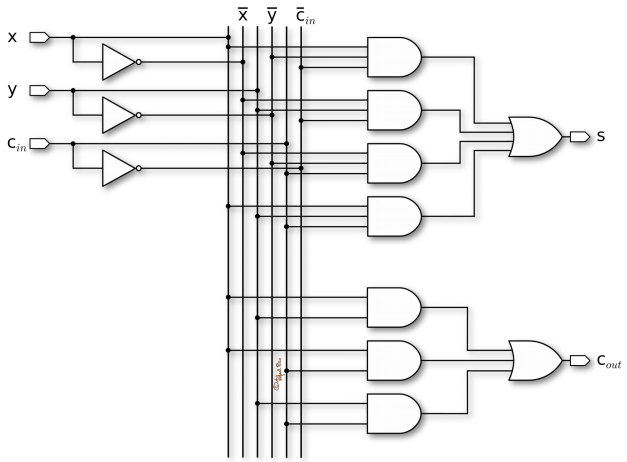
Un sumador completo es un operador aritmético de tamaño 1 bit que realiza la suma del operando de entrada `x` de 1 bit con el operando de entrada `y` de 1 bit teniendo en cuenta un acarreo de entrada `cin` y proporciona como salida el bit de suma `s` y el bit de acarreo de salida $c_{out}$. La especificación se puede formular a través de su tabla de verdad:

| x | y | $c_{in}$ | s | $c_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Dada la especificación, llevamos a cabo la **síntesis** utilizando los diagramas de Karnaugh y el álgebra de Boole. Una posible solución sería la que expresa las ecuaciones lógicas de los bits de salida como sumas de productos (primera forma canónica o *minterms*).

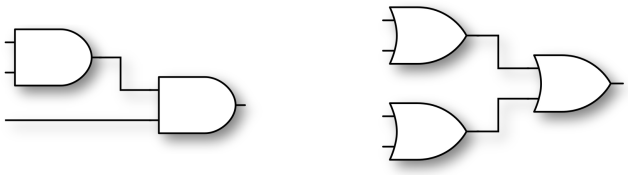
$$s = \overline{x} \cdot y \cdot \overline{c}_{in} + x \cdot \overline{y} \cdot \overline{c}_{in} + \overline{x} \cdot \overline{y} \cdot c_{in} + x \cdot y \cdot c_{in}$$
$$c_{out} = x \cdot y + y \cdot c_{in} + x \cdot c_{in}$$

A continuación, el esquema que corresponde a estas ecuaciones:




Observamos que algunas de las puertas involucradas tienen más de 2 entradas. En concreto, tenemos puertas `AND` de 3 entradas y `OR` de 3 y 4 entradas. Esta realidad ha de ser tenida en cuenta de cara a la **implementación**. Podemos afirmar que la síntesis en papel "lo aguanta todo" pero al llegar a la práctica, la implementación puede encontrarse dificultades inopinadas.

Si la implementación no proporciona puertas de más de 2 entradas es necesario obtenerlas a partir de diseños en cascada tal y como se muestra a continuación:



Estos diseños son absolutamente funcionales pero implican un aumento de la longitud del camino crítico del operador, es decir, las señales deben atravesar más bloques de conmutación de los que pensábamos.

En esta práctica vamos a modelar el sumador completo dos veces. La primera suponiendo que disponemos de puertas de más de 2 entradas que trabajan con el mismo retardo que las de 2 según la tabla de la sesión 0 (retardos asimilables a la familia TTL). Y la segunda construyendo puertas de más de 2 entradas mediante circuitos en cascada.

Diseñaremos un *test bench* único para ambos modelos de sumador completo y determinaremos la longitud del camino crítico en cada caso mediante la inspección del cronograma con `gtkwave`.

## Modelos VHDL de puertas de más de 2 entradas

Presentamos a continuación la puerta AND de 3 entradas (`and3.vhd`) y la OR de 4 (`or4.vhd`) cuyos comportamientos se ajustan al retardo ideal:

```
--puerta and 3 entradas

ENTITY and3 IS
  GENERIC (retardo: TIME:= 2 ns);
  PORT (e1, e2, e3: IN BIT; sal: OUT BIT);
END and3;

ARCHITECTURE comportamiento OF and3 IS
BEGIN
  PROCESS (e1, e2, e3)
  BEGIN
    sal <= e1 AND e2 AND e3 AFTER retardo;
  END PROCESS;
END comportamiento;
```

```
--puerta or 4 entradas

ENTITY or4 IS
  GENERIC (retardo: TIME:= 2 ns);
  PORT (e1, e2, e3, e4: IN BIT; sal: OUT BIT);
END or4;

ARCHITECTURE comportamiento OF or4 IS
BEGIN
  PROCESS (e1, e2, e3, e4)
  BEGIN
    sal <= e1 OR e2 OR e3 OR e4 AFTER retardo;
  END PROCESS;
END comportamiento;
```

Ambas puertas se pueden usar sin problemas cuando requerimos menos entradas. En el caso de la AND, las entradas no usadas se ponen a '1' mientras que en el caso de la OR se ponen a '0'.

## *Test bench* y sentencia `WAIT UNTIL`

Este *test bench* debe servirnos para medir la longitud del camino crítico. Para ello inyectaremos las ocho combinaciones diferentes de valores de entrada y esperaremos hasta que salgan los valores correctos con la

sentencia `WAIT UNTIL r = <valor> and c1 = <valor>`. El tiempo de retardo hasta que se obtienen las señales correctas a la salida depende de la combinación de entrada y de cómo hayan quedado las señales internas tras la operación anterior. No obstante tras la batería de pruebas tendremos unos resultados bastante fidedignos del retardo del operador.

Es **IMPORTANTE** saber que la sentencia `WAIT UNTIL` espera algún cambio en las señales asociadas. Si no se produce ningún cambio no pasa a la siguiente sentencia como si nunca se hubiera producido la combinación indicada. Es por ello que la sucesión de entradas deber ser tal que las sucesivas salidas siempre ofrezcan algún cambio con respecto a la precedentes. En el ejemplo siguiente, una entrada produce `r = '1' and c1 = '1'` mientras que la siguiente `r = '0' and c1 = '0'`.

```
--nueva entrada
    a <= '1';
    b <= '1';
    c0 <= '1';
    marca <= '1';
--espero a los resultados
    WAIT UNTIL r = '1' and c1 = '1';
    marca <= '0';
    WAIT FOR 5 ns;

--nueva entrada
    a <= '0';
    b <= '0';
    c0 <= '0';
    marca <= '1';
--espero a los resultados
    WAIT UNTIL r = '0' and c1 = '0';
    marca <= '0';
    WAIT FOR 5 ns;
```

El ejemplo reproducido más arriba, utiliza una señal auxiliar (`marca`) para que señale claramente en el cronograma el tiempo de retardo de la operación. Así, cuando se inyectan los valores de entrada se pone a '1' (`marca <= '1';`) y cuando aparecen los valores correctos a la salida se pone a '0' (`marca <= '0';`). De esta forma, cuando analicemos el cronograma bastará con buscar los flancos y medir los intervalos.

## Prácticas

### A) Escriba el modelo de sumador completo sin restricciones por número de entradas.

Utilizando la OR de 4 entradas y la AND de 3 escriba el modelo VHDL pedido. Cuando use menos entradas de las disponibles acople la señal de entrada al valor adecuado para asegurar la funcionalidad correcta.

### B) Escriba el *test bench* adecuado.

El *test bench* deberá comprobar el correcto funcionamiento del operador de suma y además deberá permitir cuantificar la longitud del camino crítico. Si es necesario puede crear alguna señal adicional que le facilite la tarea.

### C) Vuelva a escribir el modelo de sumador completo asumiendo las restricciones por número de entradas.

Ahora no podrá usar puertas de más de dos entradas y siempre que lo necesite deberá incluir una cascada de puertas para completar la función.

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

Intercambiando el modelo del sumador completo anterior por el nuevo, el mismo *test bench* del apartado B nos dará la nueva longitud del camino crítico.

### E) Elabore una memoria con los modelos y los resultados

Presente los dos modelos VHDL del sumador completo y el *test bench* utilizado. Incluya las imágenes de los dos cronogramas con la ejecución del *test* para cada caso. ¿Cuál es la longitud del camino crítico para cada caso? Discuta si los resultados obtenidos se ajustan a lo previsible a partir de un análisis del circuito.