El objetivo de la práctica 1 es servir de introducción al Verilog con un ejemplo de diseño puramente combinacional. La práctica se dividirá en tres sesiones: dos sesiones de trabajo en el aula para avanzar con el diseño y otra donde se solicitará alguna modificación y se corregirá el resultado.

Al terminar la práctica, habrán construido una pequeña ALU capaz de realizar diferentes operaciones aritméticas en complemento a dos de 4 bits y generar los bits de flags asociados, así como varias operaciones lógicas. Seguiremos un proceso incremental, diseñando los módulos componentes más simples que, combinados, constituirán la ALU. Escribiremos cada módulo en un fichero diferente, cuyo nombre será el mismo de como hayamos denominado al módulo añadiendo la extensión ".v"

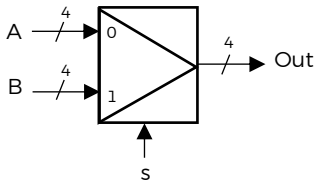
OBJETIVO 1

1.1. Basándonos en los ejemplos vistos en las sesiones tutorizadas, implementar un multiplexor de 4 entradas de un bit con el siguiente prototipo:

```
mux4_1(output reg out, input wire a, b, c, d, input wire [1:0] S);
```

1.2. Realiza ahora un multiplexor de 2 entradas a una salida en el que las entradas son de 4 bits. Como ya sabes, la línea de selección ***s*** determina cuál de las entradas pasa a la salida. El prototipo del módulo es el siguiente:

```
mux2_4(output wire [3:0] Out, input wire [3:0] A, input wire [3:0] B, input wire s);
```

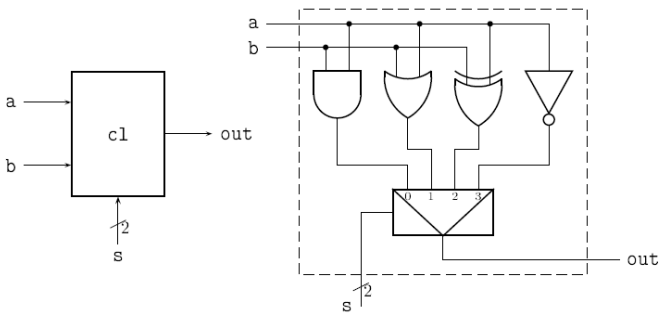


OBJETIVO 2

2.1. Implementar una "celda lógica" con el prototipo siguiente:

```
cl(output wire out, input wire a, b, input wire [1:0] S);
```

Dicha celda lógica calculará sobre los bits **a** y **b** las operaciones lógicas **and**, **or**, **xor** e **inversión** del bit **a** cuando el vector de dos bits **s** vale 00, 01, 10 y 11 respectivamente.



2.2. Implementar una "unidad lógica de 4 bits" de forma estructural, utilizando las celdas lógicas descritas en el objetivo 2.1. El prototipo sería el siguiente:

```
ul4(output wire[3:0] Out, input wire[3:0] A, input wire[3:0] B, input wire [1:0] S);
```

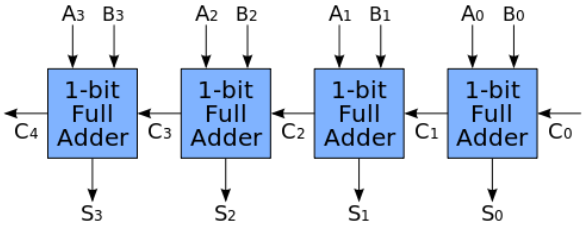
OBJETIVO 3

3.1. Implementar un "Full-Adder" de la manera más simple posible (operador concatenación '**{...}**'), con el prototipo

```
fa(output wire c_out, sum, input wire a, b, c_in);
```

3.2. Implementar un sumador completo de 4 bits de forma estructural utilizando los "Full-Adder" descritos en el objetivo 2.2. El prototipo sería el siguiente:

```
sum4(output wire[3:0] S, output wire c_out, input wire[3:0] A, input wire[3:0] B, input wire c_in);
```



3.3. Realiza una implementación alternativa que no utilice los "Full-Adder" de 1 bit, sino asignación continua y operador de concatenación. Tu profesor te pedirá en la corrección que uses uno u otro.

```
sum4_v2(output wire[3:0] S, output wire c_out, input wire[3:0] A, input wire[3:0] B, input wire c_in);
```

OBJETIVO 4

Es interesante que podamos complementar o no a voluntad, en función de una señal de control **cpl**, de forma que podamos dejar pasar un dato sin modificar o hacer su complemento a uno. Elabora el módulo con el prototipo siguiente:

```
compl1(output wire [3:0] Out, input wire [3:0] Inp, input wire cpl);
```

Si cpl = 1, Out = Cpl1(Inp) y si no, Out = Inp.

OBJETIVO 5

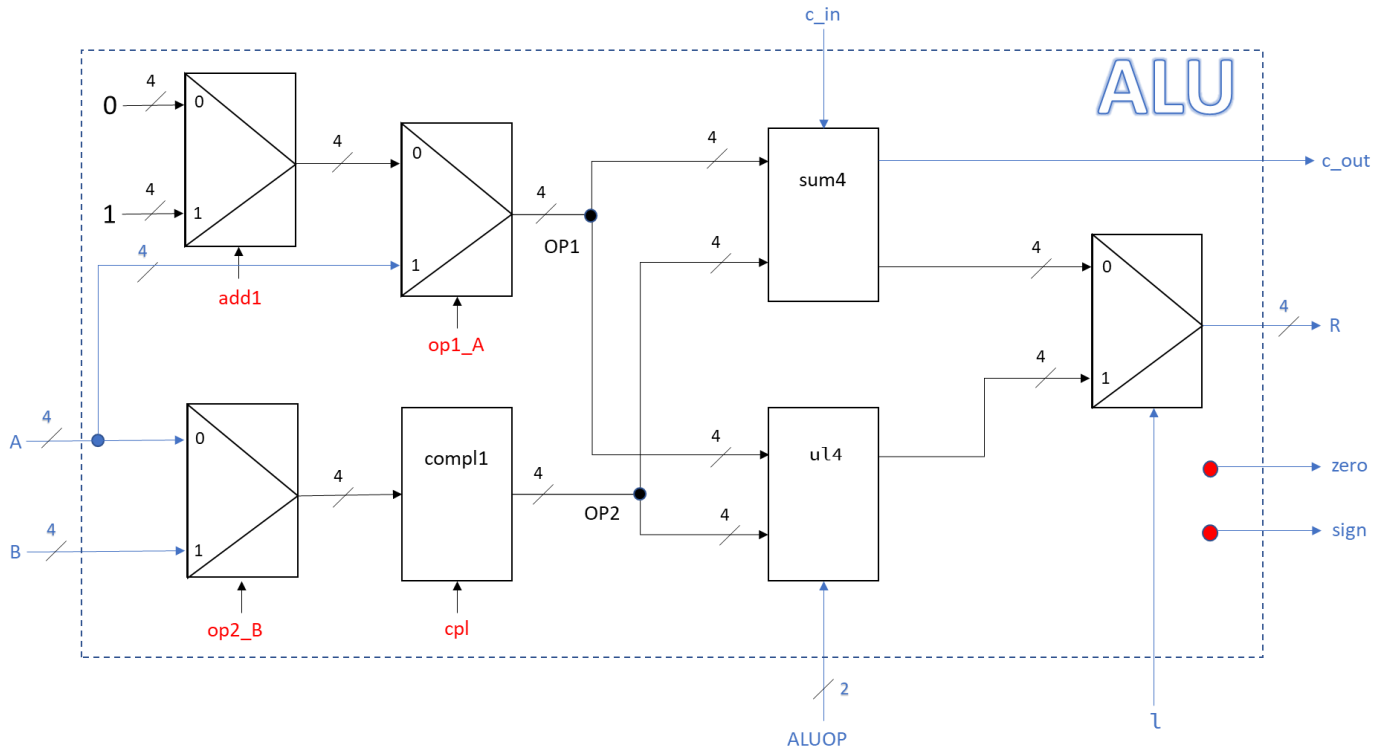
El objetivo de esta fase es completar la ALU. Para ello, es necesario que cuenten con los módulos desarrollados en los objetivos anteriores: fa.v, cl.v, sum4.v, ul4.v, compl1.v, y los multiplexores mux 4 a 1 y mux 2 a 1 (de 4 bits).

Las operaciones lógicas que puede realizar la ALU son las que ya se han implementado en la celda lógica. Las operaciones aritméticas que debemos desarrollar (dados dos operandos **A y B**, vectores de cuatro bits) son: la suma **A+B**, el **incremento de A**, el **complemento a 2 de A** y la **identidad de A** (una operación que pasa el valor de A a la salida sin modificar). La operación de suma se realiza directamente con el módulo **sum4**. Se han denotado las entradas a ese sumador como **OP1** y **OP2**, ambos vectores de cuatro bits. La operación de complemento a dos la podemos lograr a partir de los elementos que tenemos mediante el procedimiento de realizar el complemento a 1 y sumar 1. A su vez, la suma de una unidad la podemos lograr aprovechando la entrada con una constante de valor 1 que se habilita con la señal de control **add1.** Si ponemos dicha entrada a 1, lograríamos sumar una unidad al sumador definido en **sum4**.

Con el módulo de compl1, podemos elaborar la operación del **complemento a 2 de A** simplemente presentando a las entradas **OP1** y **OP2** del sumador los valores adecuados:

Identidad(A) = A. Para lograr esto, **OP1 = 0**, **OP2 = A**
Cpl2(A) = Cpl2(A) = Cpl1(A) + 1. Para lograr esto, **OP1 = 1**, **OP2 = Cpl1(A)**
A + B se obtiene simplemente con **OP1 = A** y **OP2 = B**
A + 1 se obtiene simplemente con **OP1 = 1** y **OP2 = A**

Por tanto, vemos que OP1 puede valer A, 0 ó 1. También vemos que OP2 será A, Cpl1(A), B ó Cpl1(B). Todo esto se logra con la lógica a base de multiplexores que aparece en la parte izquierda de la figura. No olviden que, en todas las operaciones aritméticas, el resultado siempre sumará el valor de c_in.



Para elaborar la ALU seguiremos el diseño de la figura, completando los módulos que creamos necesarios, de forma que el módulo final tenga el prototipo

```
alu(output wire [3:0] R, output wire zero, c_out, sign, input wire [3:0] A, B, input wire c_in,
input wire [1:0] ALUOP, input wire l);
```

y el funcionamiento debe ser el descrito por la siguiente tabla

l	ALUOP	R	zero	carry	sign
0	00	Id(A) + c_in	1 si (R == 0)	1 si Id(A) + c_in tiene acarreo	Bit más significativo de R
0	01	Cpl2(A) + c_in	1 si (R == 0)	1 si Cpl2(A) + c_in tiene acarreo	Bit más significativo de R
0	10	A + B + c_in	1 si (R == 0)	1 si (A+B) + c_in tiene acarreo	Bit más significativo de R
0	11	A + 1 + c_in	1 si (R == 0)	1 si (A+1) + c_in tiene acarreo	Bit más significativo de R
1	00	A and B	1 si (R == 0)	No importa	No importa
1	01	A or B	1 si (R == 0)	No importa	No importa
1	10	A xor B	1 si (R == 0)	No importa	No importa
1	11	Cpl1(A)	1 si (R == 0)	No importa	No importa

Vemos que la entrada **l** indica si la operación es del grupo lógico o aritmético (l =1: operación lógica, l =0: operación aritmética) y que los dos bits de la entrada ALUOP especifican la operación dentro de cada uno de estos grupos.

OBJETIVO 6

Para terminar el diseño y lograr el comportamiento anterior, tenemos que diseñar las funciones lógicas de las señales que controlan los multiplexores, el complementador a uno y el acarreo de entrada (**op1_A**, **op2_B**, **cpl** y **add1**) a partir de las entradas **1**, **ALUOP**[1] y **ALUOP**[0]. Para crearlas, lo más recomendable es escribir las tablas de verdad de cada una. Por último, también es necesario diseñar las funciones lógicas de los flags de acarreo, cero y signo (las dos últimas son muy sencillas, el cero requiere una pequeña expresión lógica de los bits de R). Los bits de acarreo y signo no importan en las operaciones lógicas, así que se pueden dejar conectados igual que en las operaciones aritméticas.

Una vez acabado el diseño, deben comprobar su buen funcionamiento con el testbench que proporcionará el profesor.

OBJETIVO 7

El profesor planteará una ampliación o modificación de la práctica que deben realizar en la última sesión.

ENTREGA Y EVALUACIÓN

Una vez que el profesor les haya corregido la práctica e indicado su calificación, se subirá a la tarea del campus virtual un fichero comprimido que incluya todos los ficheros creados hasta ese momento (independientemente de que se hayan terminado todos o no). En caso de haber intentado implementar la ampliación de la práctica sin éxito, deben subir la práctica básica.

La práctica se evaluará de 0 a 10, con el criterio que describe la siguiente rúbrica:

Nota numérica	¿Cómo se consigue?
0-4	- Presentando una práctica copiada (nota: 0). - Si la parte básica de la práctica no funciona. - Si no responden satisfactoriamente a las preguntas del profesor (aunque la práctica funcione).
5-6	Si la parte básica de la práctica funciona, pero: - No han sabido implementar la ampliación - Y no demuestran una destreza adecuada con verilog
7-10	- Si la parte básica de la práctica funciona. - Si implementan la ampliación con éxito. - Si responden correctamente a las preguntas del profesor La calificación final dependerá de la calidad de la implementación y las respuestas