El objetivo de la práctica 1 es servir de introducción al Verilog con un ejemplo de diseño puramente combinacional. La práctica se dividirá en dos sesiones: una sesión 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 sumas y restas 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

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);
```

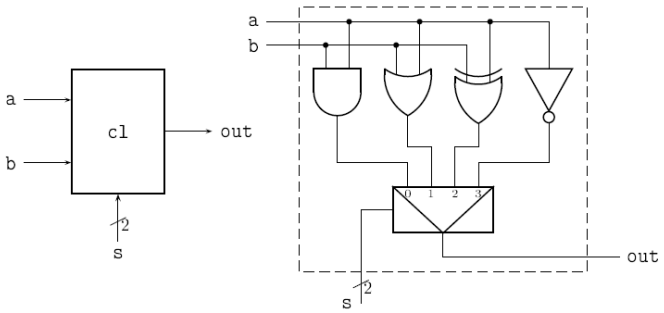
A continuación, implementen un testbench (mux4_1_tb) para comprobar que su módulo funciona correctamente.

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.

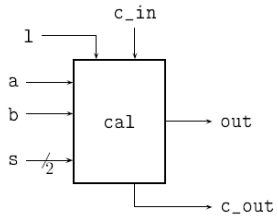


2.2. Implementar un "Full-Adder" de la manera más simple posible (operador concatenación '**{…}**'), con el prototipo

```
fa(output wire cout, sum, input wire a, b, cin);
```

2.3. Unir las dos implementaciones en una "celda arimético-lógica" mediante un multiplexor que conecte sus salidas controlado por una entrada '**l**' (de "Lógica") de forma que cuando esta señal esté a 1 se tenga la salida de la celda lógica y a 0 se obtenga la salida del "full-adder". El prototipo sería el siguiente:

```
cal(output wire out, c_out, input wire a, b, l, c_in, input wire [1:0] s);
```



OBJETIVO 3

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] Sal, input wire [3:0] Ent, input wire cpl);
```

Si cpl = 1, Sal = Cpl1(Ent) y si no, Sal =Ent.

OBJETIVO 4

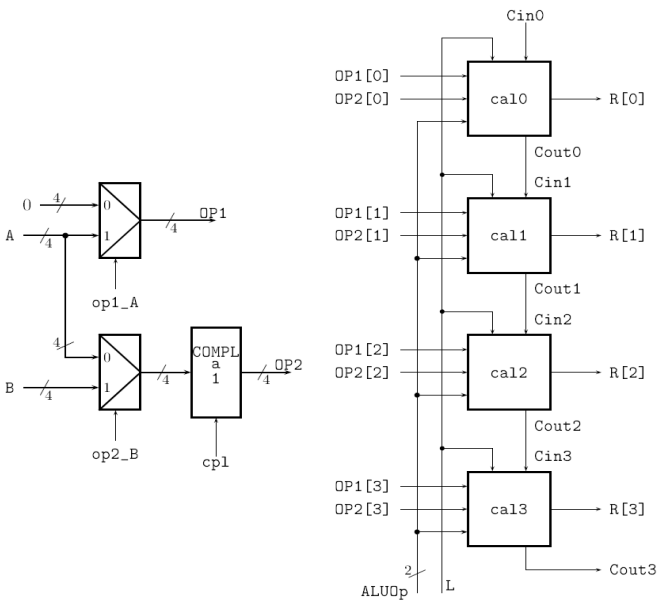
El objetivo de esta fase es completar la ALU. Para ello, es necesario que cuenten con los módulos desarrollados en los objetivos anteriores: fa.v, cl.v, cal.v y compl1.v. Otros módulos necesarios, que le facilitará el profesor son: mux2_4.v, mux2_1.v.

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**, la resta **A-B**, el **complemento a 2 de A** y el **complemento a 2 de B**. La operación de suma la elaboramos simplemente instanciando cuatro elementos **cal** y conectando los acarreos de salida de un elemento al acarreo de entrada del siguiente, como aparece en la parte derecha de la figura de más abajo. 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 de acarreo que nos ha quedado **Cin_0** Si ponemos dicha entrada a 1, lograríamos sumar una unidad en el elemento **cal** de la posición menos significativa.

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

A - B = A + (-B) = A + Cpl2(B) = A + Cpl1(B) + 1. Para lograr esto, **OP1 = A**, **OP2 = Cpl1(B)**, **Cin_0=1**
Cpl2(A) = 0 + Cpl2(A) = 0 + Cpl1(A) + 1. Para lograr esto, **OP1 = 0**, **OP2 = Cpl1(A)**, **Cin_0=1**
Cpl2(B) = 0 + Cpl2(B) = 0 + Cpl1(B) + 1. Para lograr esto, **OP1 = 0**, **OP2 = Cpl1(B)**, **Cin_0=1**
A + B se obtiene simplemente con **OP1 = A**, **OP2 = B**, **Cin_0=0**

Por tanto, vemos que OP1 será o bien A, o bien 0. 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.



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, carry, sign, input wire [3:0] A, B, 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	A + B	1 si (R = 0)	1 si (A+B) tiene acarreo	Bit más significativo de R
0	01	A - B	1 si (R = 0)	1 si (A-B) tiene acarreo	Bit más significativo de R
0	10	Cpl2(A)	1 si (R = 0)	1 si Cpl2(A) tiene acarreo	Bit más significativo de R
0	11	Cpl2(B)	1 si (R = 0)	1 si Cpl2(B) 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: op. lógica, L=0: op. Aritmética) y que los dos bits de la entrada Op especifican la operación dentro de cada uno de estos grupos.

OBJETIVO 5

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 Cin0) a partir de las entradas L, AluOp[1] y AluOp[0]. Puede ayudar a crearlas el escribir las tablas de verdad de cada una. Por último, también es necesario diseñar las funciones lógicas de los flags de cero, acarreo 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 6

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.
5-6	Si la parte básica de la práctica funciona, pero: - No han sabido implementar la ampliación - Y han demostrado pobres conocimientos de verilog
7-8	Si la parte básica de la práctica funciona, pero: - No han sabido implementar la ampliación - O fallan en alguna de las preguntas planteadas por el profesor
9-10	- Si la parte básica de la práctica funciona. - Si implementan la ampliación con éxito. - Si responden correctamente a las preguntas del profesor