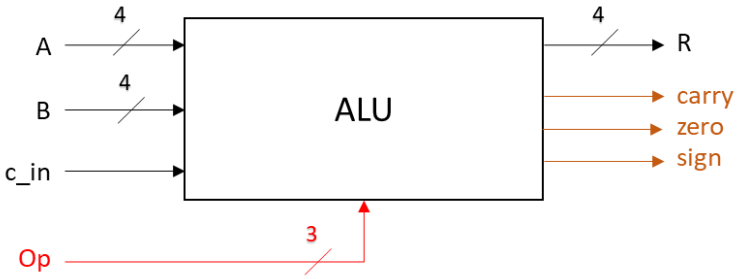
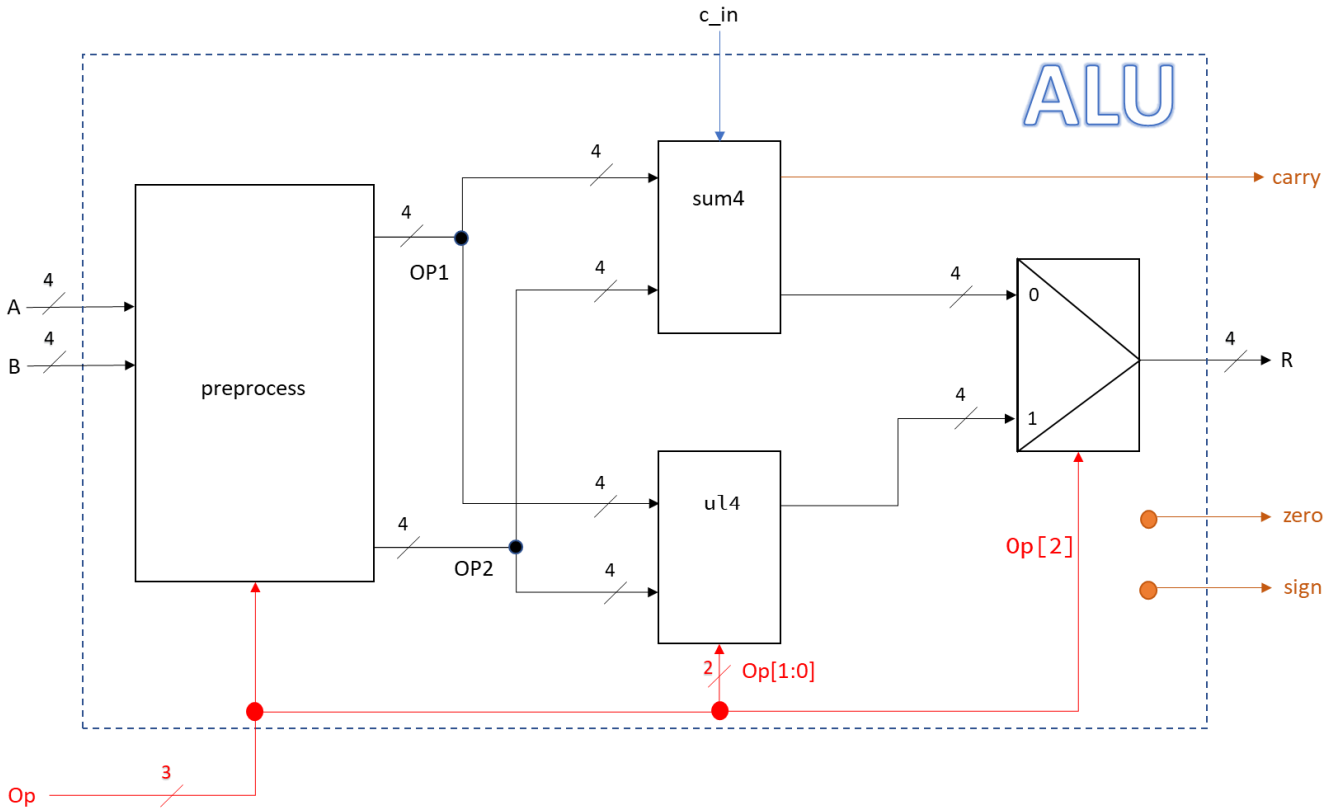
El objetivo de la práctica 1 es servir de introducción a Verilog con un ejemplo de diseño puramente combinacional. La práctica se dividirá en tres sesiones: dos sesiones de trabajo en el aula para avanzar con el diseño y otra donde se solicitará alguna modificación y se corregirá el resultado.

El objetivo general es construir una pequeña ALU capaz de realizar diferentes operaciones aritméticas y lógicas, empleando operandos de 4 bits.



La ALU tiene dos entradas de 4 bits (operandos **A** y **B**) y una salida de 4 bits (**R**), junto con un acarreo de entrada (**c_in**). Además, es capaz de generar los *flags* asociados a las operaciones (**carry**, **zero** y **sign**). La operación a realizar se controla con la línea de 3 bits **Op**, por lo que dispondremos de 8 operaciones diferentes que se detallarán más adelante. Las líneas **c_in** y **carry** se pueden usar para aplicar la técnica del *bit slicing* y construir una ALU mayor

Internamente, la ALU se compone de cuatro grandes bloques: un sumador de 4 bits, una unidad lógica de 4 bits, un módulo de preprocesamiento que permite modificar el valor de los operandos **A** y **B** para poder realizar operaciones diferentes, y un multiplexor que permite seleccionar entre una salida aritmética o lógica.



Para realizar el diseño, seguiremos un proceso incremental, creando módulos para los 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. UNIDAD LÓGICA

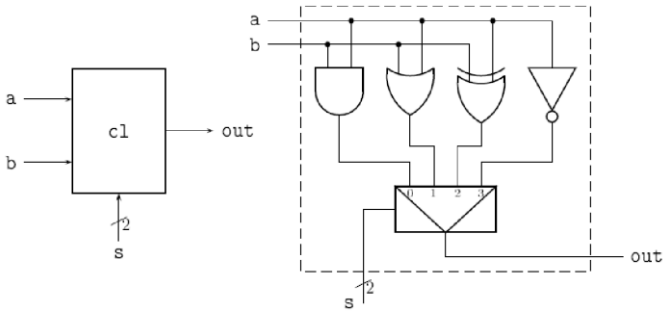
1.1. Basándonos en los ejemplos vistos en las sesiones tutorizadas, implementar un multiplexor de 4 entradas y una salida de un bit con el siguiente prototipo:

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

1.2. 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.



1.3. Implementar una "unidad lógica de 4 bits" de forma estructural, utilizando las celdas lógicas descritas en el objetivo 1.2. 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);
```

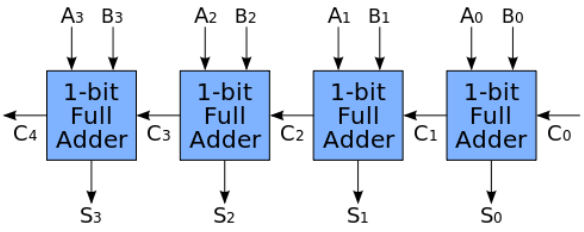
OBJETIVO 2

2.1. Implementar un "Full-Adder" usando el operador concatenación '**{…}**', con el prototipo

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

2.2. Implementar un sumador completo de 4 bits de forma estructural utilizando los "Full-Adder" descritos en el objetivo anterior. 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);
```



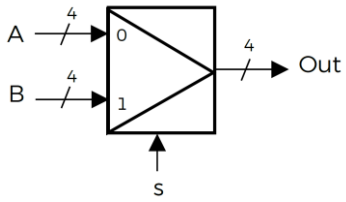
2.3. Realiza una implementación alternativa que no utilice los "Full-Adder" de 1 bit, sino asignación continua y operador de concatenación. Tu profesor te pedirá en la corrección que uses uno u otro.

```
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 3. MULTIPLEXOR 2-1 DE 4 BITS

Basándonos en los ejemplos vistos en las sesiones tutorizadas, implementar 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 4

La siguiente tabla resume las operaciones que debe ser capaz de realizar nuestra ALU:

Op	Descripción	R
000	Suma de A y B	A + B + c_in
001	Incremento de A	A + 1 + c_in
010	Negativo de A	-A + c_in
011	Negativo de B	-B + c_in
100	Y bit a bit entre A y B	A and B
101	O bit a bit entre A y B	A or B
110	O exclusiva bit a bit entre A y B	A xor B
111	Negación bit a bit de A	inv(A)

Con los módulos desarrollados hasta el momento deberíamos poder construir una ALU capaz de realizar la operación aritmética de suma y todas las operaciones lógicas. Sin embargo, no seríamos capaces de realizar el resto de operaciones aritméticas.

La solución es crear un pequeño módulo de preprocesamiento de los operandos que sea capaz de realizar transformaciones en **A** y **B** antes de usarlos en el sumador.

4.1. Crear un módulo capaz de calcular el negativo de un número en complemento a 1. El módulo debe incluir una señal de control **cpl**, de manera que podamos elegir entre dejar pasar el dato sin modificar o calculando el negativo. Elabora el módulo con el prototipo siguiente:

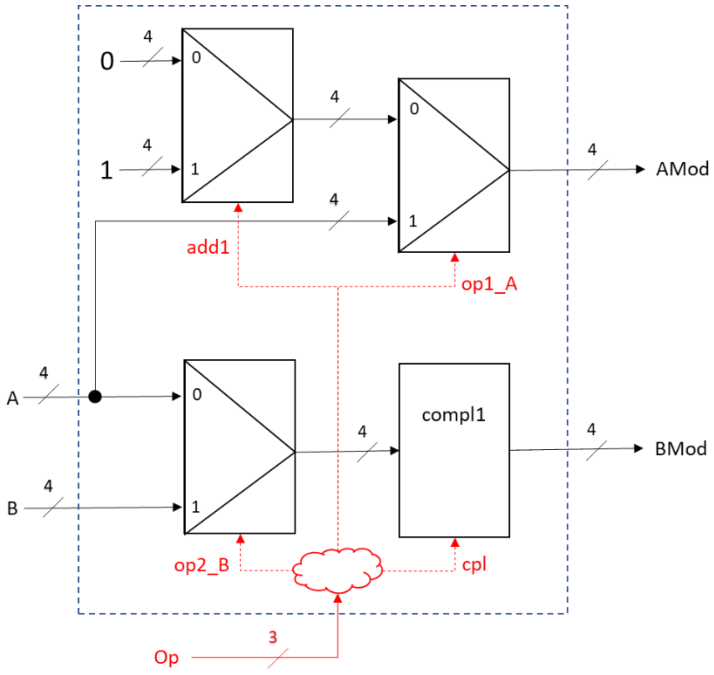
```
compl1(output wire [3:0] Out, input wire [3:0] Inp, input wire cpl);
```

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

4.2. Crear el módulo de preprocesamiento de acuerdo al siguiente prototipo

```
preprocess(output wire [3:0] AMod, output wire [3:0] BMod, input wire [3:0] A, input wire [3:0] B, input wire [2:0] Op);
```

En este módulo las salidas **AMod** y **BMod** son modificaciones de **A** y **B** cuyo valor final depende de las entradas de control a cada multiplexor y al "complementador a 1". Estas entradas se calculan como una función de la señal general de control de la ALU: **Op**. Por ejemplo, para calcular el negativo de A, lo podemos hacer en complemento a 2, negando en complemento a 1 (**cpl** = 1) y sumando 1 (**add1** = 1 y **op1_A** = 0). Para realizar las operaciones lógicas o la operación de suma debemos configurar las señales de control (**add1**, **op1_A**, **op2_B**, **cpl**) para que **AMod = A** y **BMod = B**. Y así con todas las operaciones.



Para crear las funciones lógicas que determinan el valor de las señales de control (**add1**, **op1_A**, **op2_B**, **cpl**) a partir del valor de **Op**, lo más recomendable es escribir las tablas de verdad de cada una y construir mediante puertas o expresiones lógicas estas funciones.

OBJETIVO 5

5.1. Unir todos los componentes como aparecía en la figura de la primera página del enunciado de acuerdo al siguiente prototipo:

```
alu(output wire [3:0] R, output wire zero, carry, sign, input wire [3:0] A, B, input wire c_in, input wire [2:0] Op);
```

5.2. Por último, también es necesario diseñar las funciones lógicas de los *flags* de acarreo (**carry**), cero (**zero**) y signo (**sign**):

- **zero** tiene valor de 1 si el resultado fue 0 (R == 0)
- **sign** tomará el valor del bit más significativo de R
- **carry** será 1 si hubo acarreo en la operación aritmética realizada

Los bits de acarreo y signo no importan en las operaciones lógicas, así que se pueden dejar conectados igual que en las operaciones aritméticas.

OBJETIVO 6

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

OBJETIVO 7

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

ENTREGA Y EVALUACIÓN

Una vez que el/la profesor/a 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.

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). - Presentando únicamente la parte básica de la práctica, sin ser capaces de realizar la modificación. - Si no responden satisfactoriamente a las preguntas del profesor (aunque la práctica funcione).
5-6	Si la modificación no funciona del todo, pero: - Han sabido responder adecuadamente a las preguntas del profesor
7-10	- Si la parte básica de la práctica funciona. - Si implementan la modificación con éxito. - Si responden correctamente a las preguntas del profesor La calificación final dependerá de la calidad de la implementación y las respuestas