

Especificación de funciones combinatoriales con Verilog (Práctica 4)

Introducción

Verilog es un lenguaje de descripción de hardware (HDL) que, como su nombre indica, permite describir formalmente el funcionamiento de un circuito

Con ayuda de herramientas como **ISE Design Suite (Xilinx)**, especificar circuitos en verilog, simular su funcionamiento e incluso implementar el circuito hardware

Es similar a un lenguaje de programación imperativo: formado por un conjunto de sentencias que indican como realizar una tarea.

Algunas diferencias:

- La mayoría de las sentencias se ejecutan concurrentemente
- Cada sentencia corresponde a un bloque de circuito

Estructura general de una descripción Verilog

Un diseño Verilog consta de uno o varios módulos interconectados

La descripción del módulo Verilog (module) consiste en:

```
module mi_circuito (  
    input x, y,  
    input z,  
    output f1, f2  
);  
  
    wire cable_interno;  
    reg variable_a;  
  
    ...  
    ...  
    ...  
  
endmodule
```

Declaración del módulo con
sus entradas y salidas

Declaración de señales y variables que se
utilizarán internamente en la descripción

Descripción del comportamiento módulo.
Hay **varias alternativas** para realizarla

Tipos de descripciones

Descripción **funcional**:

- a partir de las **expresiones algebraicas de las funciones** (operadores AND, OR, XOR...)
- Previamente tenemos que usar kmapa y deducir expresiones algebraicas

Descripción **estructural**:

- Interconectando de los **componentes de un circuito previamente diseñado**.
- Por ejemplo, a partir del diseño a nivel de puertas lógicas que previamente hemos diseñado "a mano"

Descripción **procedimental**:

- Descripción **algorítmica** utilizando estructuras de control (if... else, case...)
- Es la opción más potente

Descripción Funcional (**assign**)

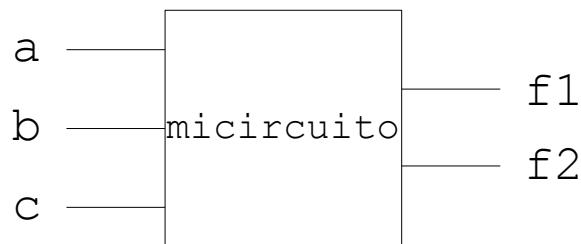
Modela circuitos combinaciones.

Asignamos (**assign**) funciones lógicas a las salidas de cada módulo

Las salidas dependen de los valores de las entradas en cada instante (combinacional).

Necesitamos una sentencia **assign** para cada función de salida

Todas las sentencias **assign** se ejecutan al mismo tiempo, por lo que da igual el orden en el que aparezcan en la especificación.



$$f1 = ab + ac$$
$$f2 = a'b + bc$$



```
module micircuito(
    input a,b,c,
    output f1,f2);

    assign f1 = (a&b) | (a&c);
    assign f2 = (~a&b) | (b&c);

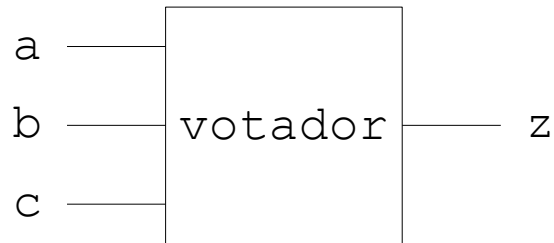
endmodule
```

Algunos operadores a nivel de bits (bitwise) (**assign**)

Operador	Ejemplo de código Verilog
&	<code>c = a&b; // Operación AND de todos los bits</code>
	<code>c = a b; // Operación OR de todos los bits</code>
^	<code>c = a^b; // Operación XOR de todos los bits</code>
~	<code>b = ~a; // NOT Inversión de todos los bits</code>
~&	<code>d = a ~& b; // Operador NAND a nivel de bits</code>
~	<code>d = a ~ b; // Operador NOR a nivel de bits</code>
~^	<code>d = a ~^ b; // Operador EXNOR a nivel de bits</code>

- *Estos operadores trabajan con todos los bits de a y todos los bits de b*
- *Si las variables son de un bit operan como los operadores del álgebra de conmutación.*

Ejemplo: circuito votador



► Expresión lógica:

► $z = ab + ac + bc$

```
module votador (input a,b,c,output z);  
    assign z= (a & b) | (a & c) | (b & c);  
endmodule
```

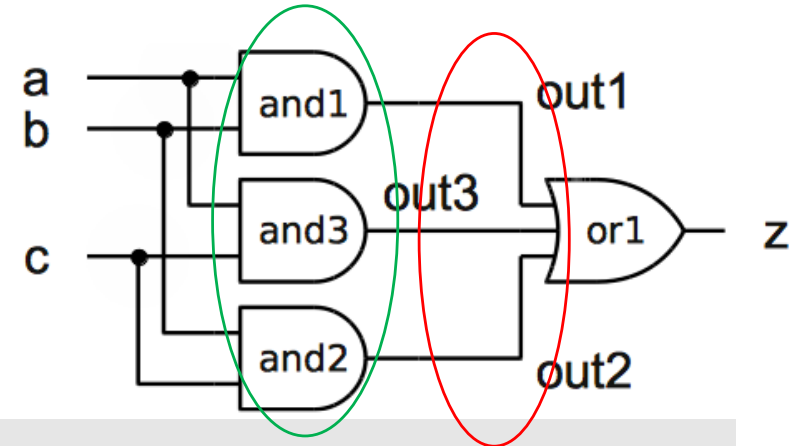
Descripción estructural

- Permite conectar módulos que ya están definidos previamente
- Las puertas lógicas básicas ya están predefinidas en Verilog

and, or, nand, nor, xor, xnor, not

- Las conexiones internas hay que declararlas tipo **“wire”** (cable)

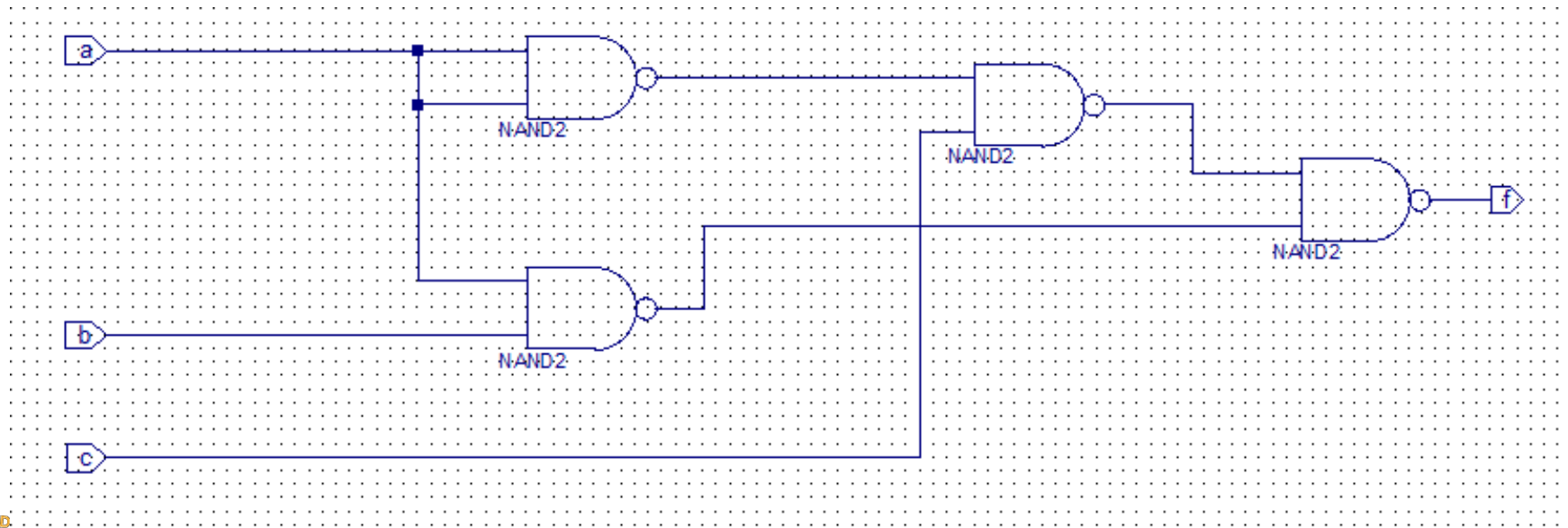
Es muy útil para la interconexión de los módulos más complejos ya creados



```
module votador(  
    input a,b,c,  
    output z);  
  
    wire out1,out2,out3;  
  
    and and1(out1,a,b);  
    and and2(out2,b,c);  
    and and3(out3,a,c);  
    or or1(z,out1,out2,out3);  
  
endmodule
```


Descripción estructural mediante captura de esquemáticos

- ISE Design Suite incluye **Schematics**, una herramienta para la captura de esquemáticos.
- La propia herramienta genera la especificación verilog del módulo
- Sólo para circuitos previamente diseñados "a mano" y relativamente sencillos



Simulación de circuitos combinatoriales

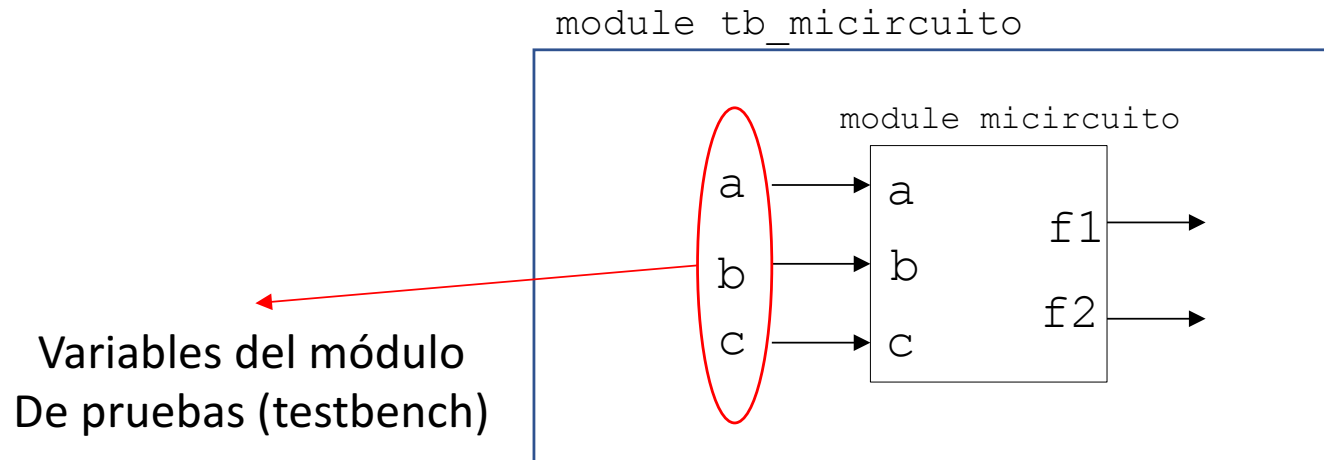
(Prácticas 4 y 6)

Simulación de módulos combinacionales

ISE Design Suite incluye la herramienta **ISim**, para simular el funcionamiento de nuestros diseños antes de implementarlo físicamente

Proceso: **New Source -> Verilog Test Module**, que debe ser asociado al módulo que queremos simular.

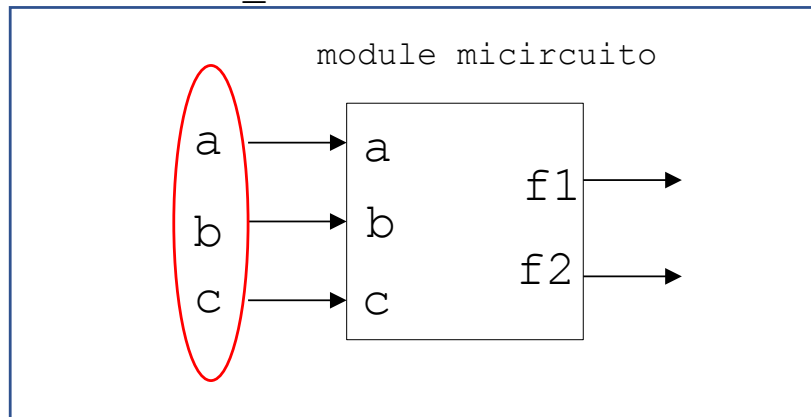
Se creará un fichero **testbench** al que sólo tendremos que **añadir las órdenes para la simulación**



Ejemplo: fichero testbench generado

```
module micircuito(  
    input a,b,c,  
    output f1,f2);  
  
    assign f1 = (a&b) | (a&c);  
    assign f2 = (~a&b) | (b&c);  
  
endmodule
```

```
module tb_micircuito
```



```
module tb_micircuito;  
    //inputs  
    reg a,b,c;  
    //outputs  
    wire f1,f2;  
    micircuito uut(  
        .a(a),.b(b),.c(c),  
        .f1(f1),.f2(f2)  
    );  
    initial begin  
        //Initialize Inputs  
        a=0;  
        b=0;  
        c=0;  
        #100;  
  
        //Add stimulus here  
        //***** COMPLETAR *****  
  
    end  
endmodule
```

Simulación de módulos combinacionales (ii)

Un fichero de pruebas (**testbench**) contiene siempre un bloque **initial**, que define los valores iniciales de las variables, y cómo queremos que cambien con el tiempo. Por ejemplo:

```
//Initial Inputs
initial begin
    a=0; //puedo definir, uno a uno, los valores iniciales
    b=0; //no importa el orden, son valores iniciales
    c=0;
    #10; //espero 10 ns
    c=1; //a y b no cambian
    #10; //espero 10 ns
    {a,b,c}=0'b010; //puedo asignar valores a los 3 con {}
    b=0; //no importa el orden, son valores iniciales
    c=0;
    #50; //espero 10 ns
    $finish; fin de la simulación
end
```

Simulación de módulos combinacionales (iii)

Si quiero probar las 2^n combinaciones, es muy útil el bloque **always** que se ejecuta en paralelo al bloque **initial**

```
always #10 {a,b,c} = {a,b,c} + 1; //se generan 000, 001, 010.... 111
                                //cambiando de valor cada 10ns
//Initial Inputs
initial begin
    a=0; //puedo definir, uno a uno, los valores iniciales
    b=0; //no importa el orden, son valores iniciales
    c=0;
    #100; //espero 100 ns
$finish; //fin de la simulación
end
```

