



## Acerca de los autores

Este curso fue creado por integrantes del Grupo de Investigación en Lógica Programable (GILP-UNLaM)

Edgardo Gho



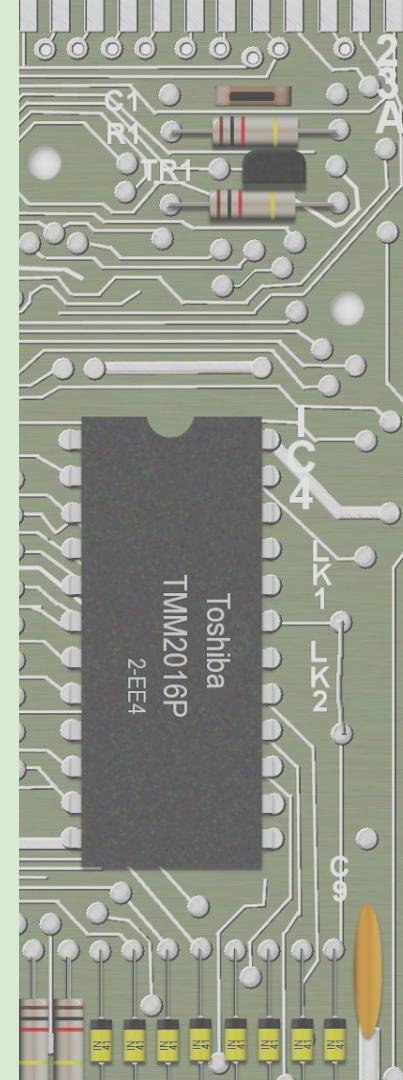
Carlos Maidana



Jair Hnatiuk



Está pensado como complemento al curso de VHDL desarrollado y dictado desde 2007. El material es un apoyo al curso presencial que se dicta en laboratorio con acceso a computadoras con el software Vivado de Xilinx y kits de desarrollo basados en la séptima familia de chips de Xilinx (Artix 7, Spartan 7, etc).



# Diseño de circuitos



Introducción 1

Módulos 2

Simulación 3

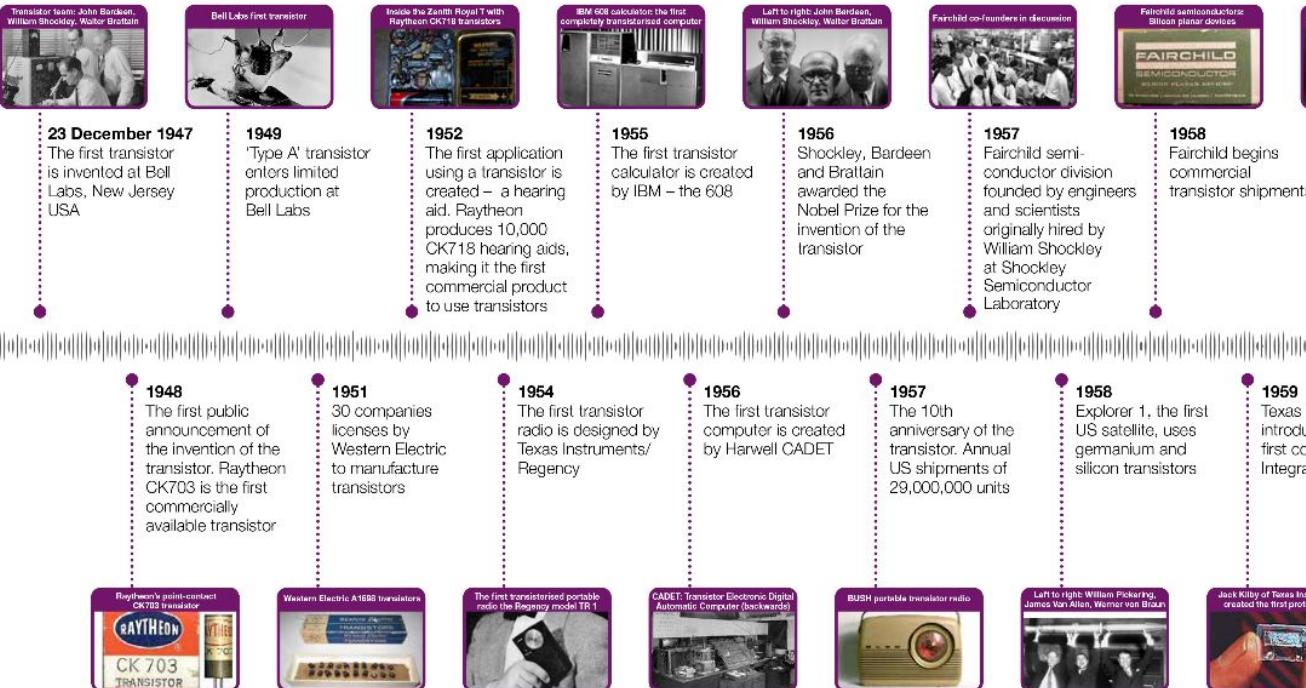
Secuenciales 4

Elementos Avanzados 5

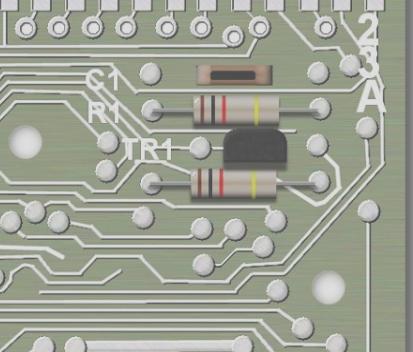
Ejemplos 6

# La lógica programable

## Celebrating 70 years of the transistor



<https://www.electronicsweekly.com/news/first-transistor-created-70-years-ago-the-device-that-changed-the-world-2017-12/>



Introducción 1

Módulos 2

Simulación 3

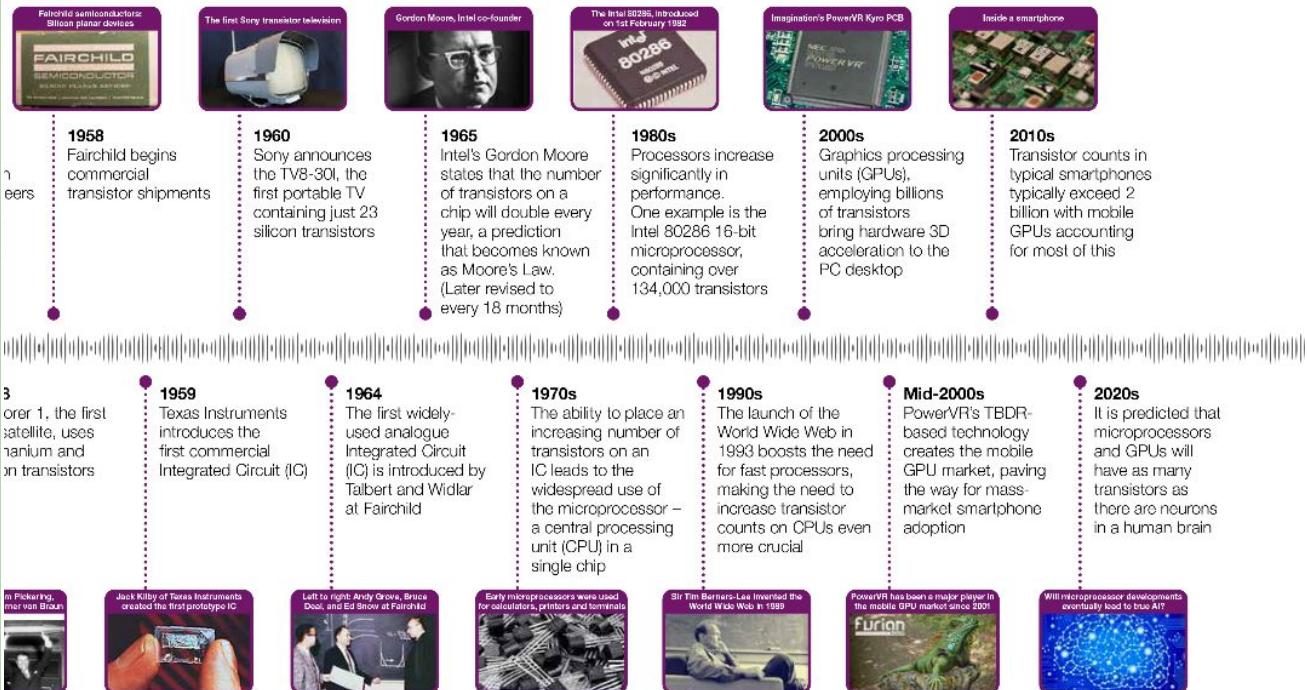
Secuenciales 4

Elementos Avanzados 5

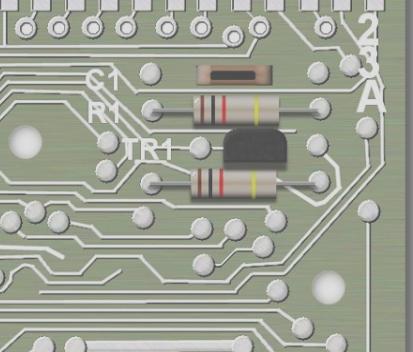
Ejemplos 6

Curso de Verilog - Básico

# La lógica programable



<https://www.electronicsweekly.com/news/first-transistor-created-70-years-ago-the-device-that-changed-the-world-2017-12/>



Introducción 1

Módulos 2

Simulación 3

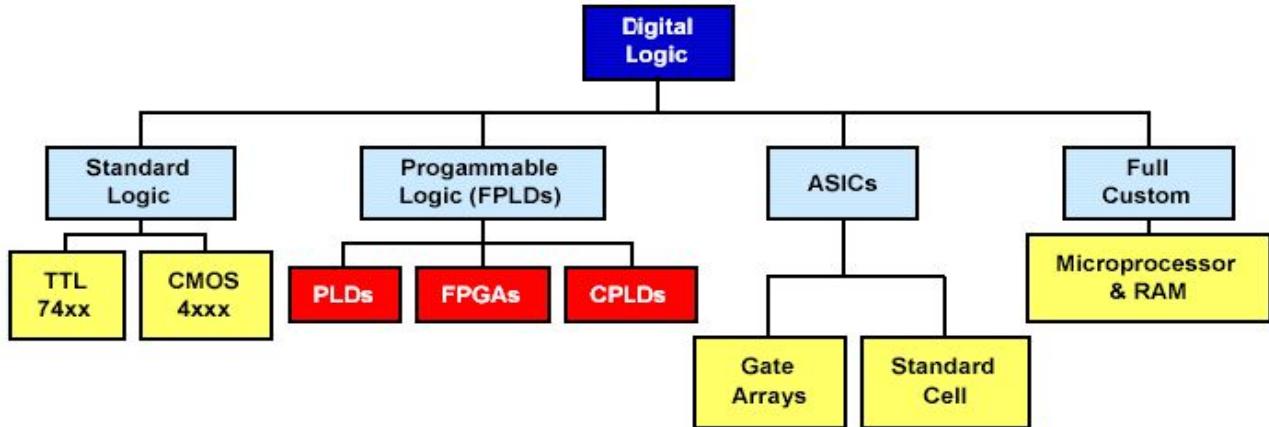
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

Curso de Verilog - Básico

# Diseño digital



Introducción 1

Módulos 2

Simulación 3

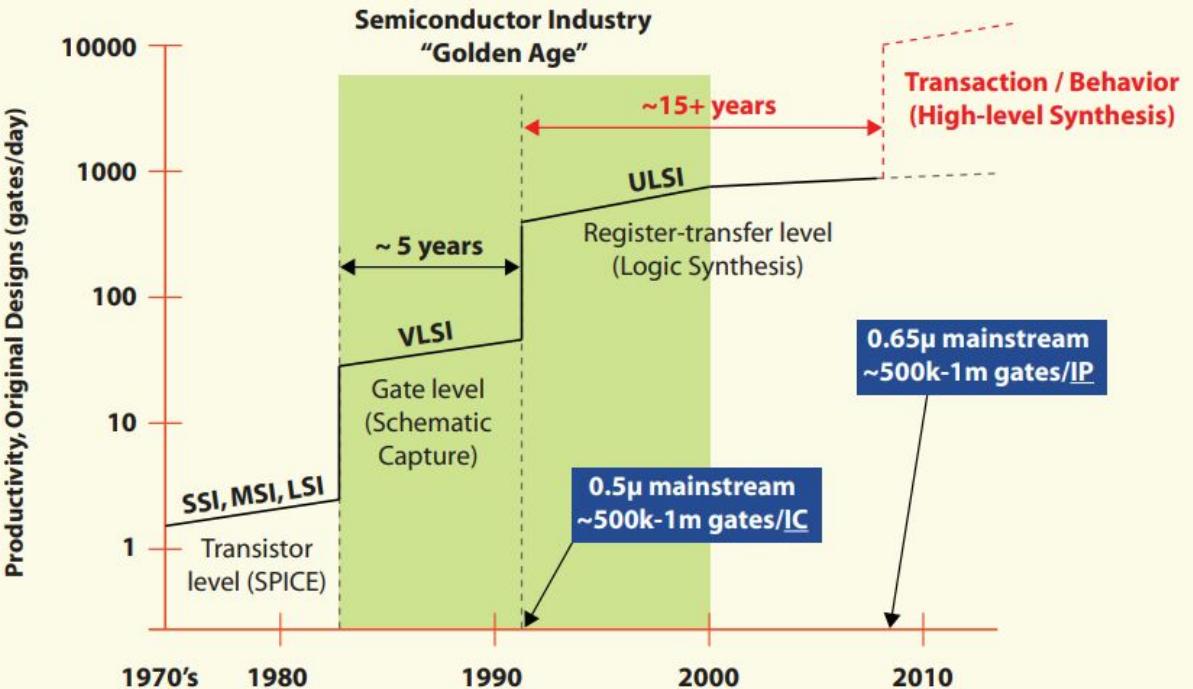
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# La lógica programable

## Evolution of IC Design



[https://www.accellera.org/images/resources/articles/icdesigntrans/  
ic\\_design\\_transition\\_feb2010.pdf](https://www.accellera.org/images/resources/articles/icdesigntrans/ic_design_transition_feb2010.pdf)



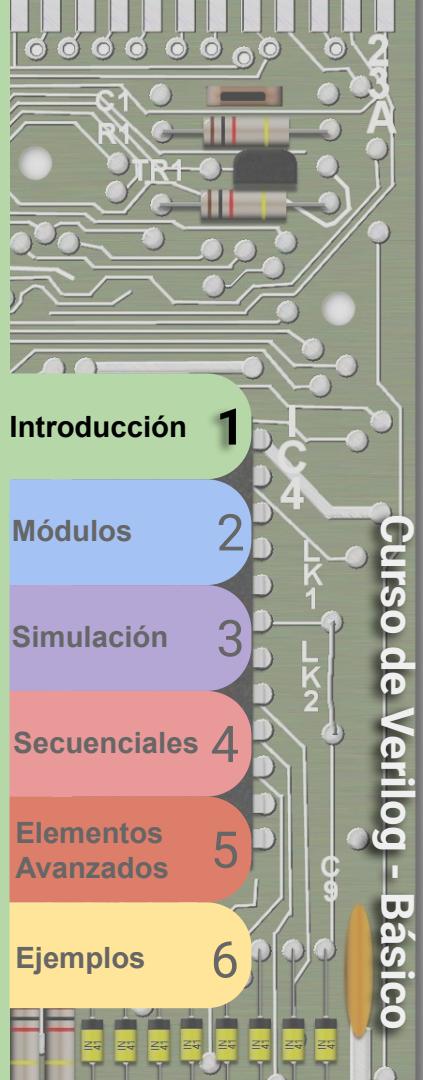
# HDL (Lenguajes descriptivos de hardware)



## VHDL

Se desarrolla en 1983 como un lenguaje para describir el funcionamiento de circuitos. El problema era que en los 80s la documentación de cómo se “comportaban” los C.I. variaba mucho según el fabricante. El depto de defensa de USA necesitaba un estándar y entonces surge VHDL. En 1987 se estandariza como IEEE 1076-1987. A partir de este punto los fabricantes de circuitos y herramientas CAD empiezan a sugerir cambios y mejoras que terminan en IEEE 1076-1993. A lo largo de los años se ha ido actualizando el estándar soportando cambios en el lenguaje.

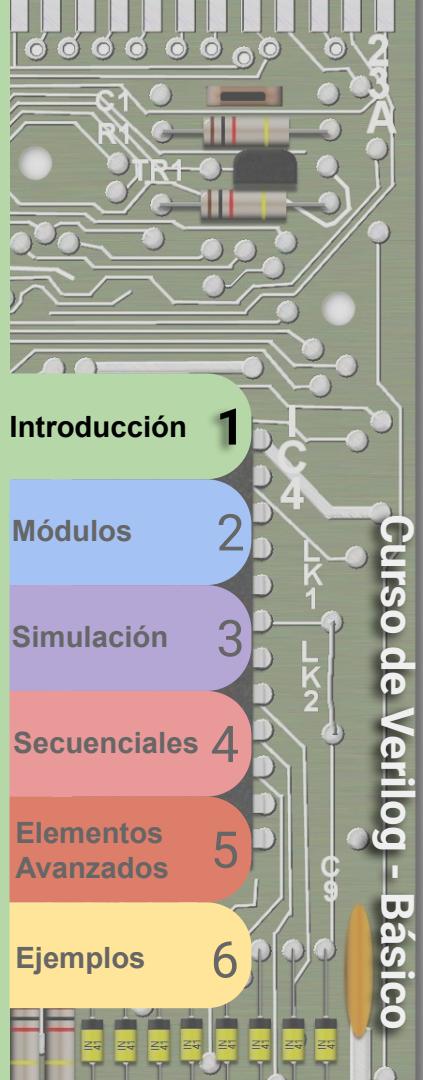
Uno de los cambios principales que incorpora la versión de 1993 es la síntesis de circuitos utilizando el mismo lenguaje. Si bien existe sintaxis específica para simular y sintaxis específica para sintetizar, el lenguaje es prácticamente el mismo para ambas tareas.



## Verilog (Verification-logic) IEEE 1364

Creado en 1984 como un producto comercial, fue estandarizado por IEEE en 1995. Comenzó como una herramienta de descripción de circuitos lógicos pero rápidamente obtuvo capacidad de simular los mismos. Cuando fue lo suficientemente popular obtuvo la capacidad de sintetizar circuitos.

Luego de ser estandarizado en 1995 se hicieron mejoras y cambios que resultaron en un nuevo estándar IEEE1364-2001. Esta es la versión más soportada por las diversas herramientas. Luego en el año 2009 se une con SystemVerilog. Desde ese entonces todo cambio se publica bajo el lenguaje SystemVerilog.



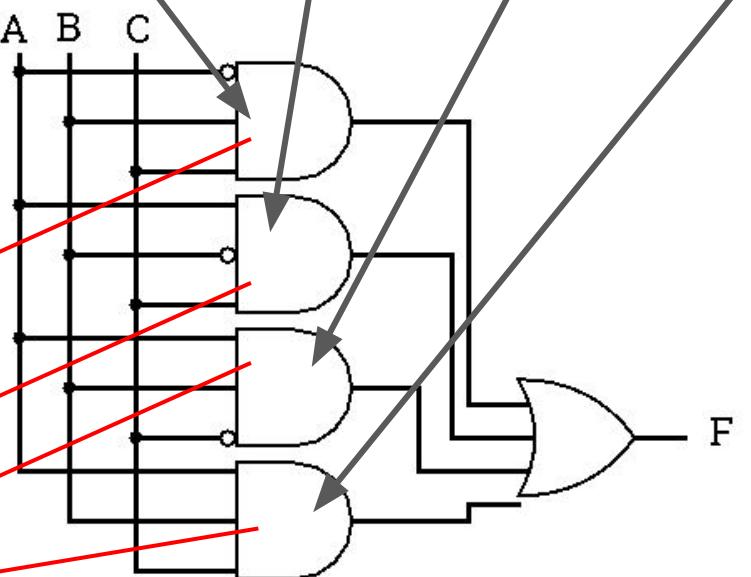
# Funciones booleanas



## Función booleana F (3 variables)

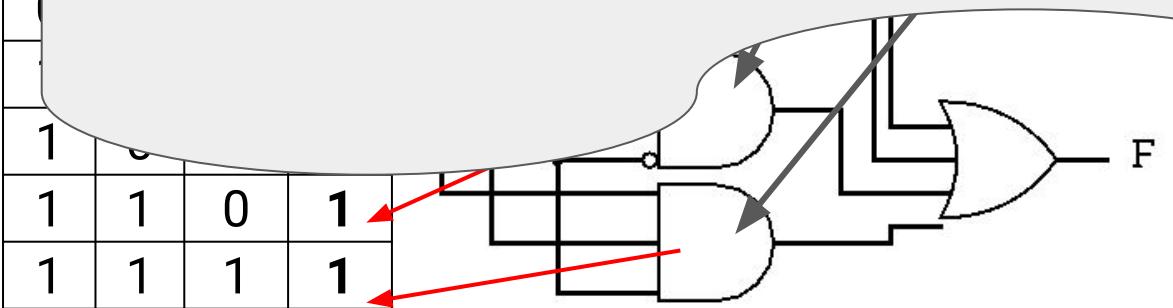
$$F(A, B, C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



## Función booleana F (3 variables)

Las tres formas de representación de la función (Tabla de verdad, circuito con compuertas o por definición de operaciones booleanas) SON EQUIVALENTES!. Es decir pasar de una forma a otra es completamente trivial. Las tres representan la misma función lógica.



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

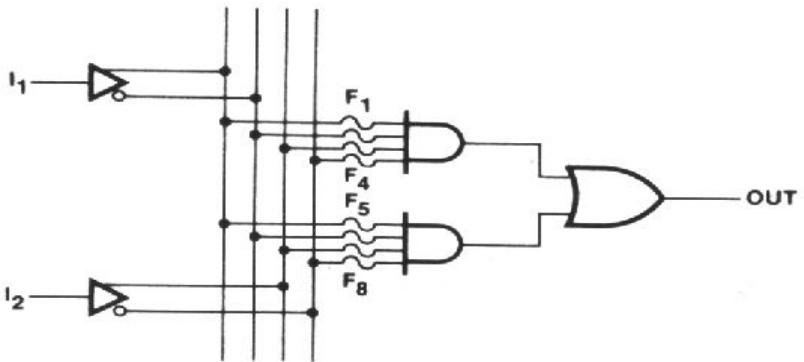
Elementos Avanzados 5

Ejemplos 6

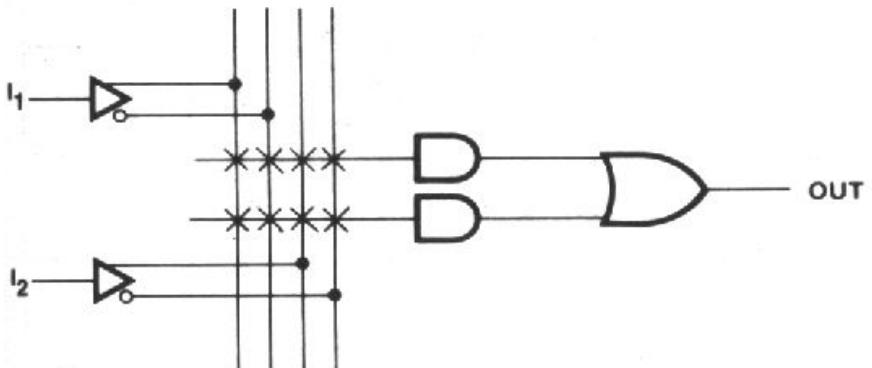
# PAL



# Circuito lógico real de una PAL



# Circuito lógico simbólico de una PAL



Introducción 1

Módulos 2

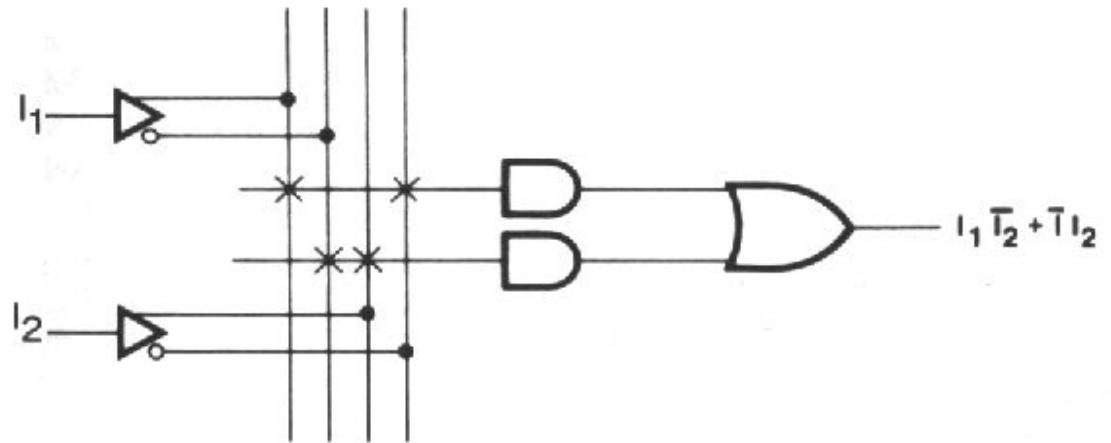
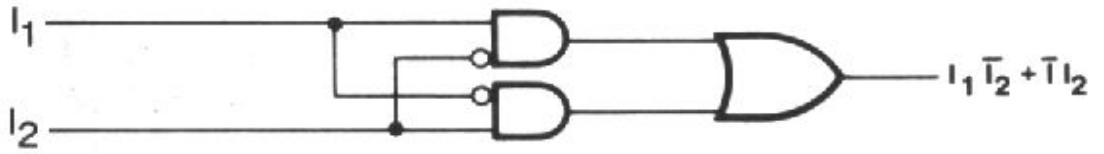
Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Implementación y esquema de quemado de fusibles



Introducción 1

Módulos 2

Simulación 3

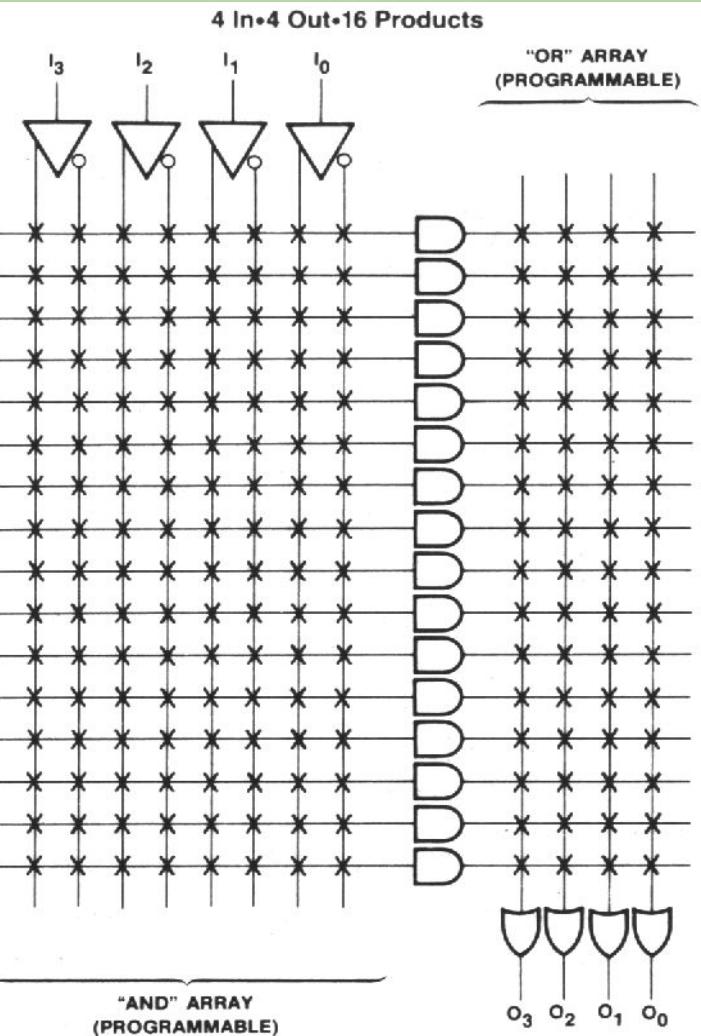
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



# Pal de 4 entradas/salidas



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Funciones con multiplexores



Introducción 1

Módulos 2

Simulación 3

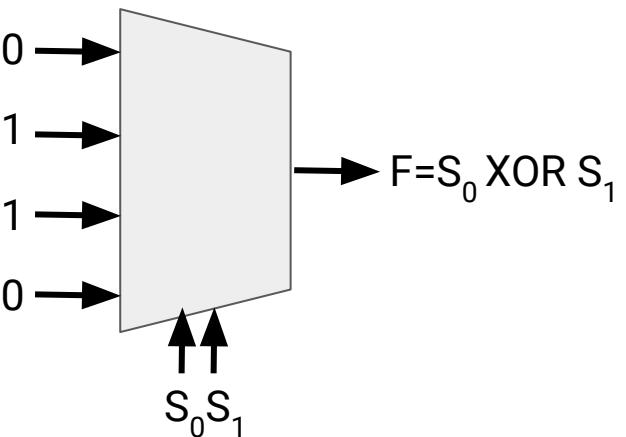
Secuenciales 4

Elementos Avanzados 5

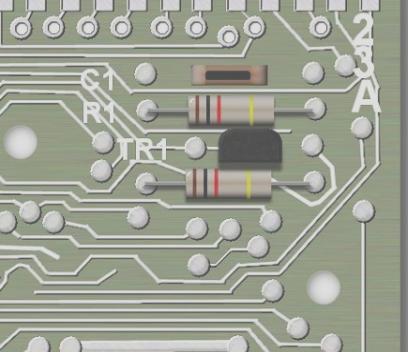
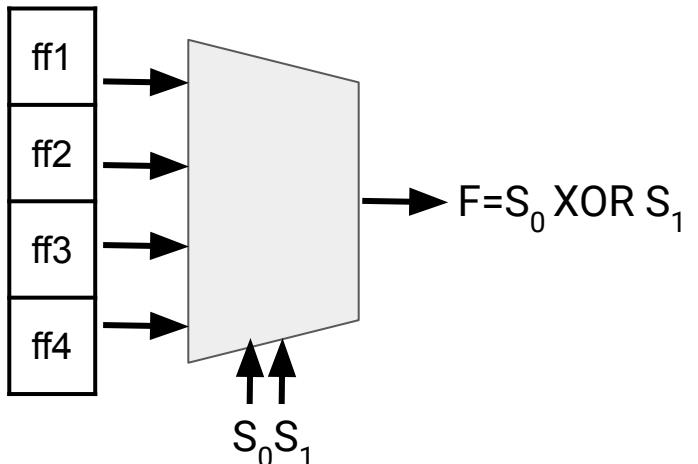
Ejemplos 6

## Un MUX con entradas fijas

$S_1$	$S_0$	F
0	0	0
0	1	1
1	0	1
1	1	0



Podemos conectar un shift register compuesto de 4 FF a las entradas del MUX. Luego cambiando los valores de FFx podemos generar cualquier "compuerta" entre  $S_0$  y  $S_1$ .



Introducción 1

Módulos 2

Simulación 3

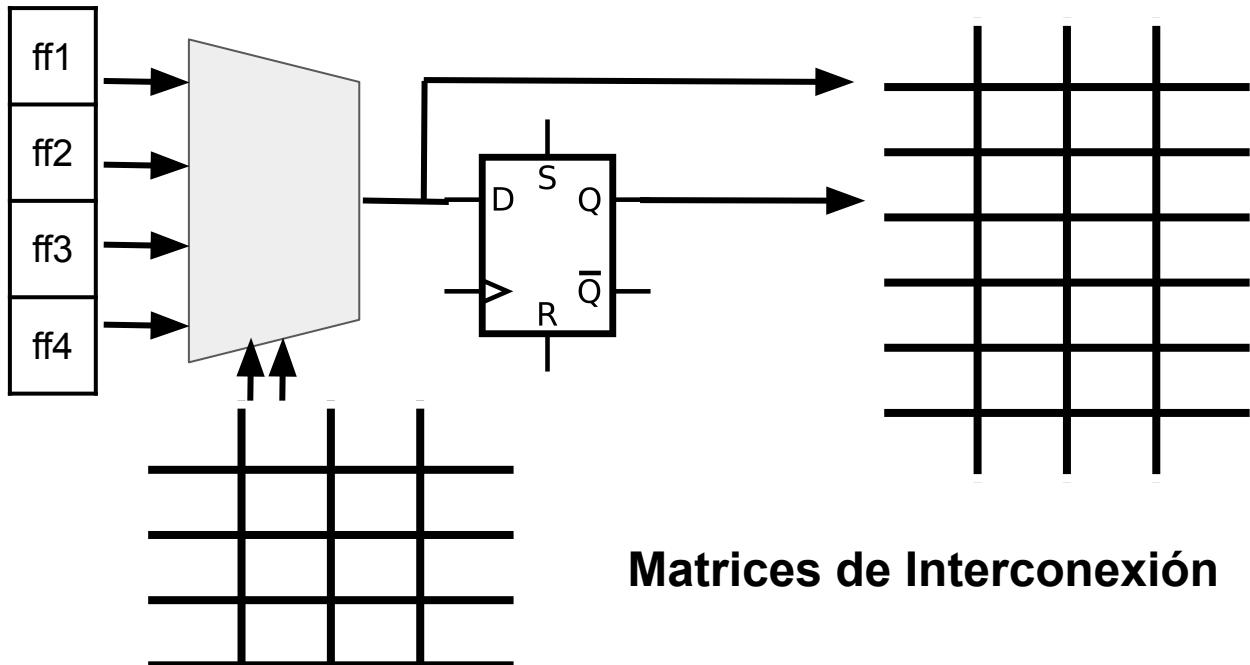
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

## Agregando un FF

Podemos incluir un FF tipo D cuya entrada esté forzada por la salida de la LUT. La selección de la LUT se conectan a la matriz, al igual que la salida del FF y la LUT.



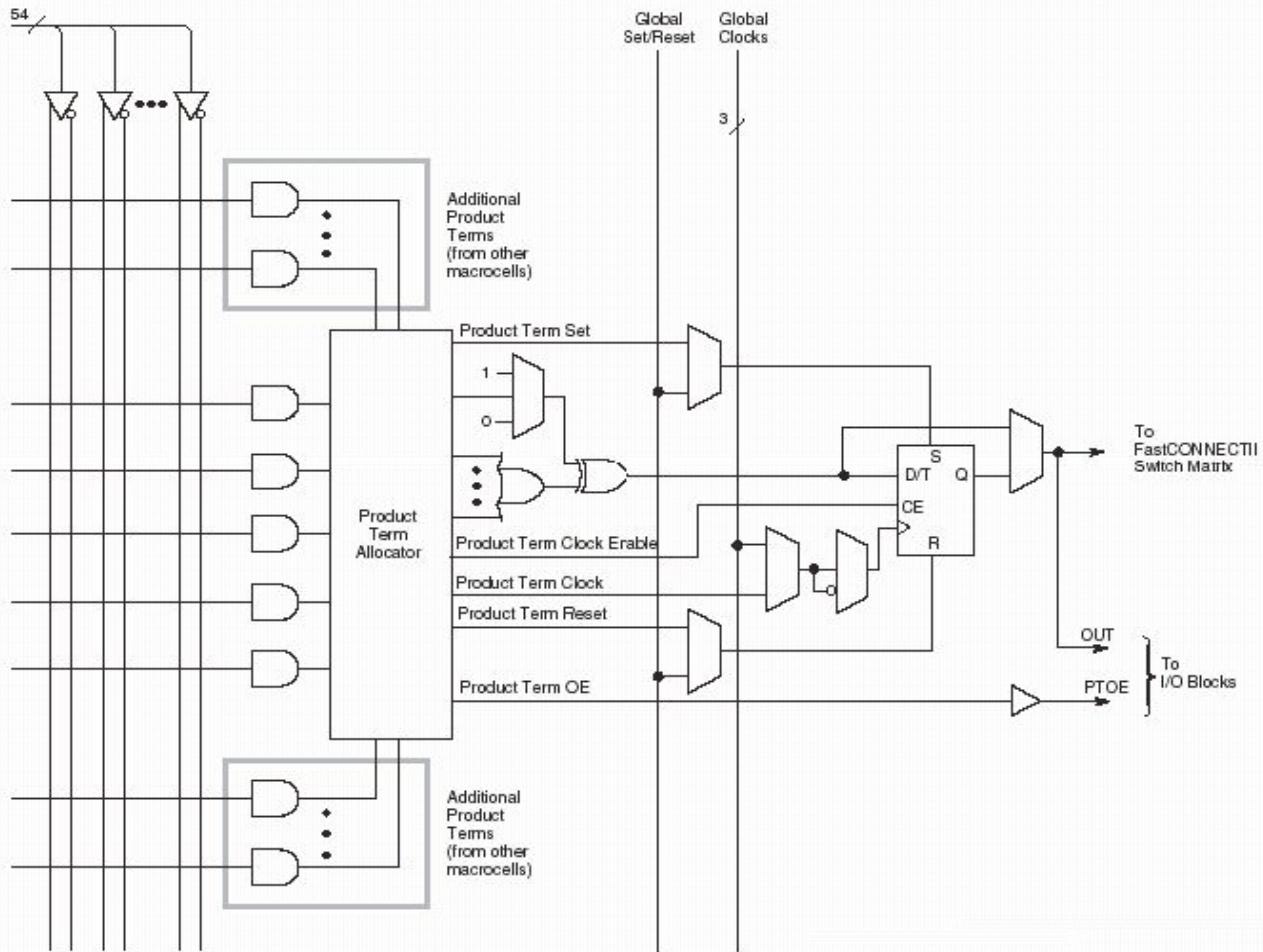
### Matrices de Interconexión



# GAL y CPLD



# CPLD



Introducción 1

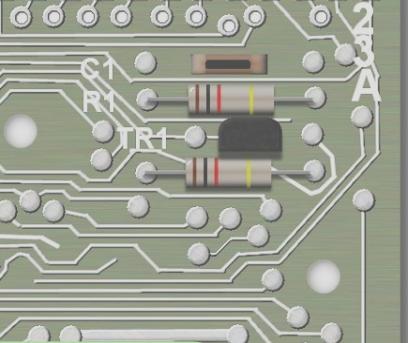
Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



# FPGA CLB



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Configurable Logic Block (CLB)

[https://www.xilinx.com/support/documentation/user\\_guides/ug474\\_7Series\\_CLB.pdf](https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf)

The 7 series configurable logic block (CLB) provides advanced, high-performance FPGA logic:

- Real 6-input look-up table (LUT) technology
- Dual LUT5 (5-input LUT) option
- Distributed Memory and Shift Register Logic capability
- Dedicated high-speed carry logic for arithmetic functions
- Wide multiplexers for efficient utilization

CLBs are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix for access to the general routing matrix (shown in [Figure 1-1](#)). A CLB element contains a pair of slices.

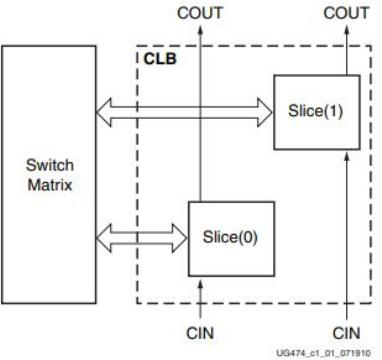
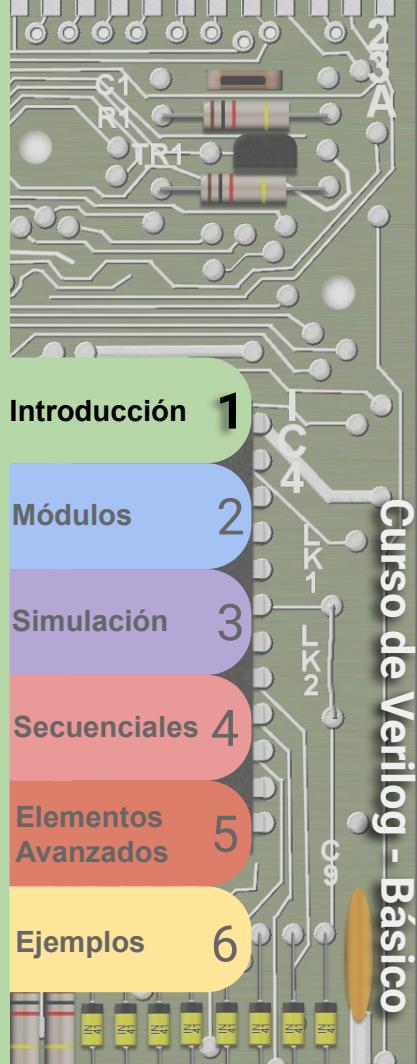
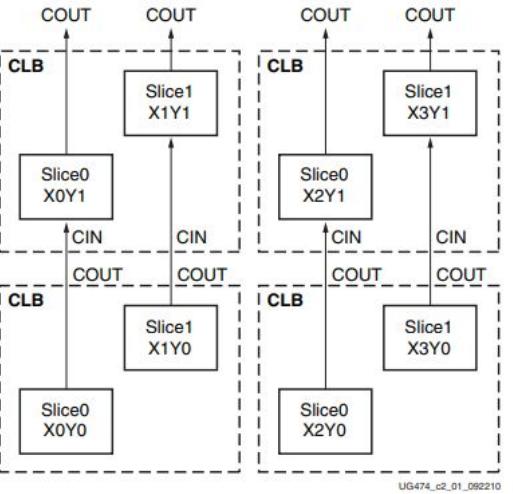
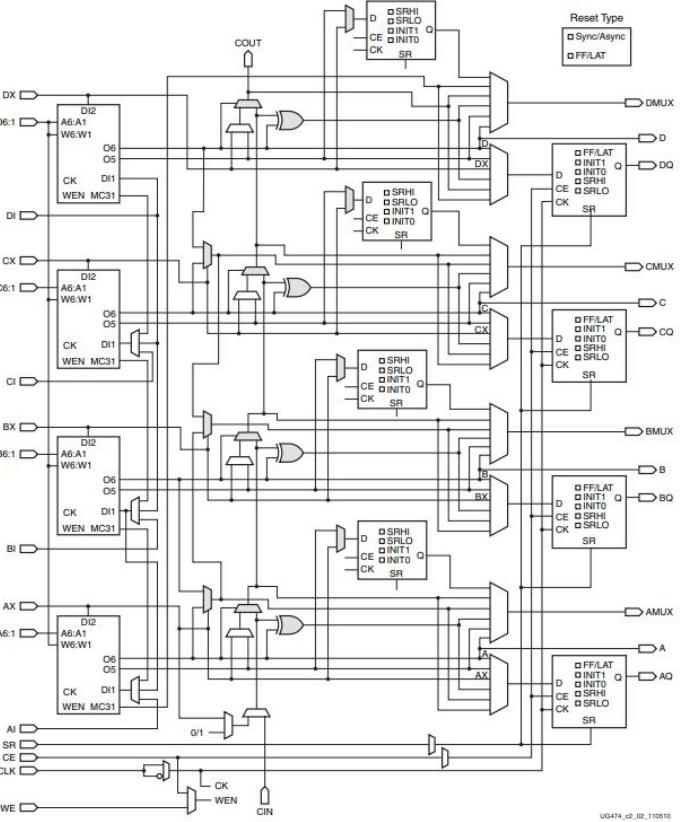
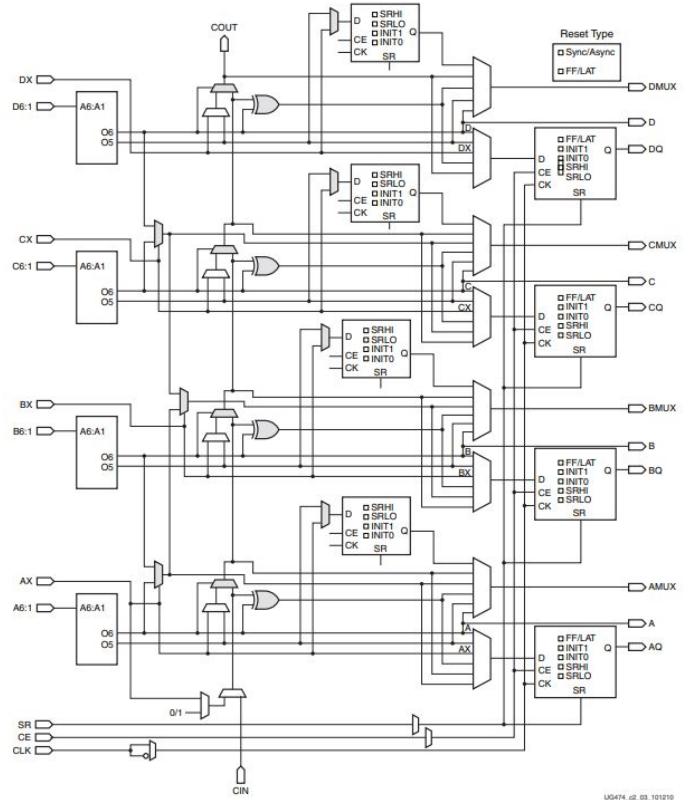


Figure 1-1: Arrangement of Slices within the CLB

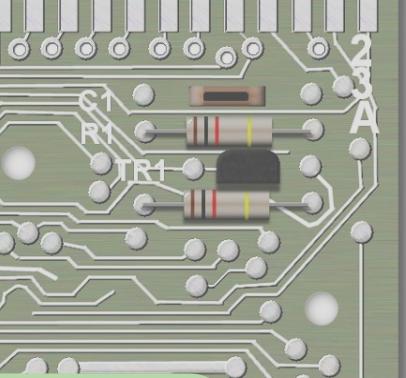
Approximately two-thirds of the slices are SLICEL logic slices and the rest are SLICEM, which can also use their LUTs as distributed 64-bit RAM or as 32-bit shift registers (SRL32) or as two SRL16s. Modern synthesis tools take advantage of these highly efficient logic, arithmetic, and memory features. Expert designers can also instantiate them.



# Configurable Logic Block (CLB)



# FPGA Simulada en Logisim-evolution



Introducción 1

Módulos 2

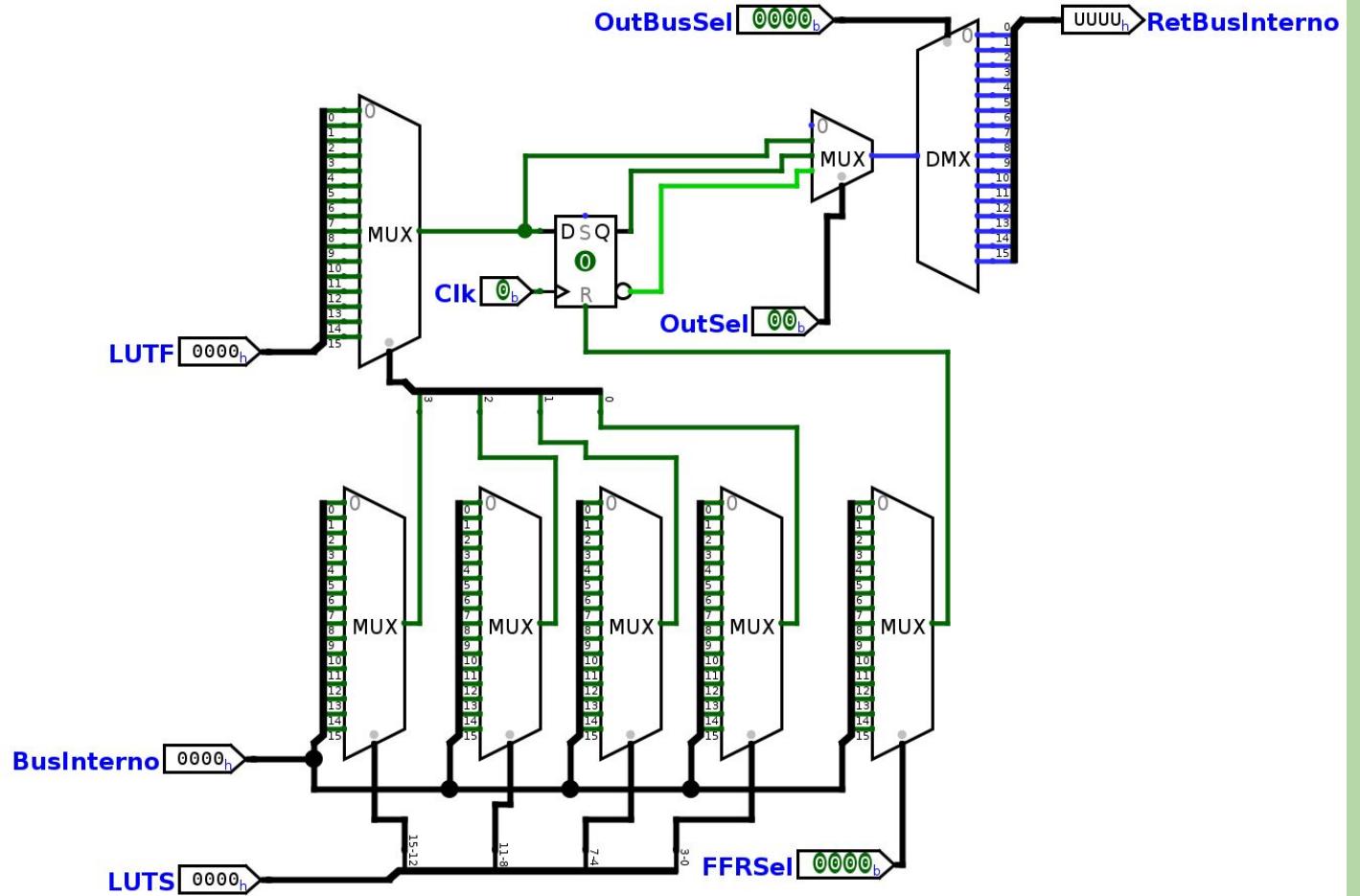
Simulación 3

Secuenciales 4

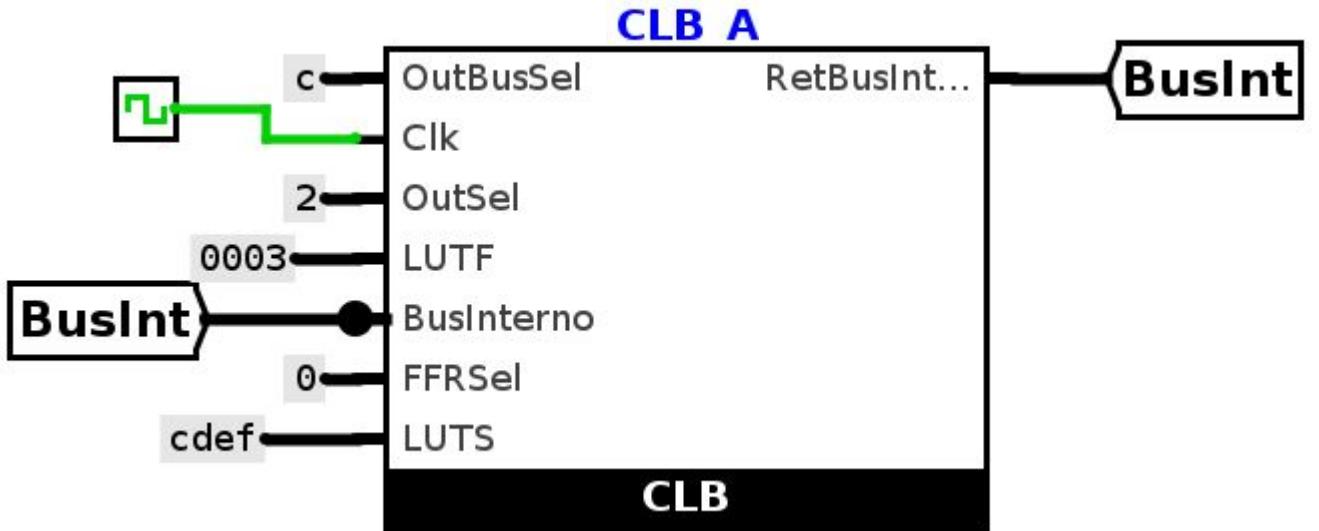
Elementos Avanzados 5

Ejemplos 6

# Bloque CLB



## Bloque CLB



## Pines de entrada

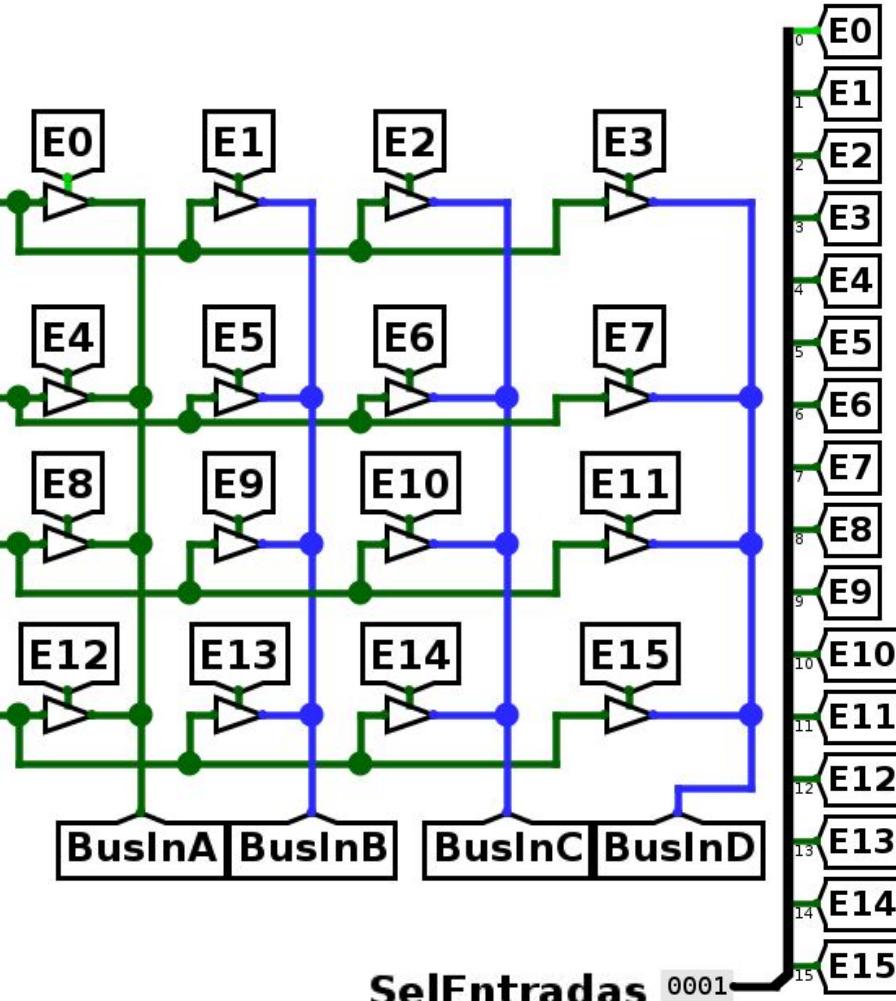
MasterClock

Entrada\_A

Entrada\_B

Entrada\_C

Entrada\_D



Introducción 1

Módulos 2

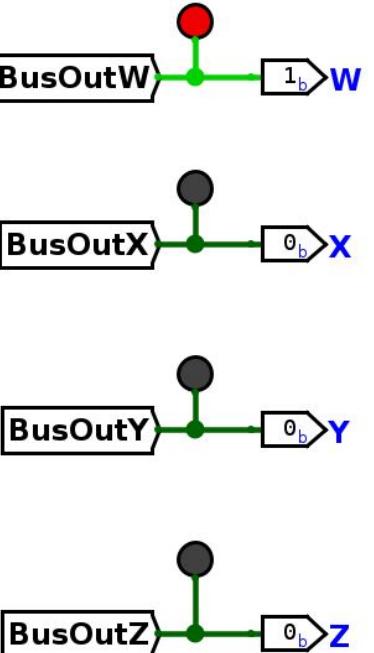
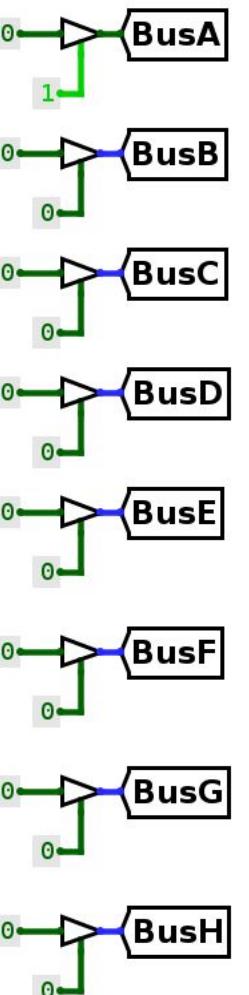
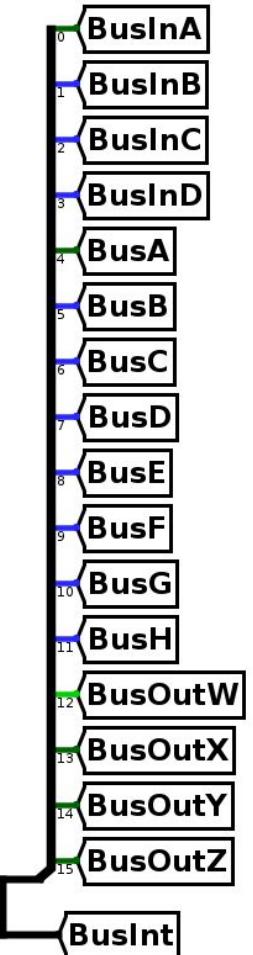
Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Bus Interno y Salidas



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

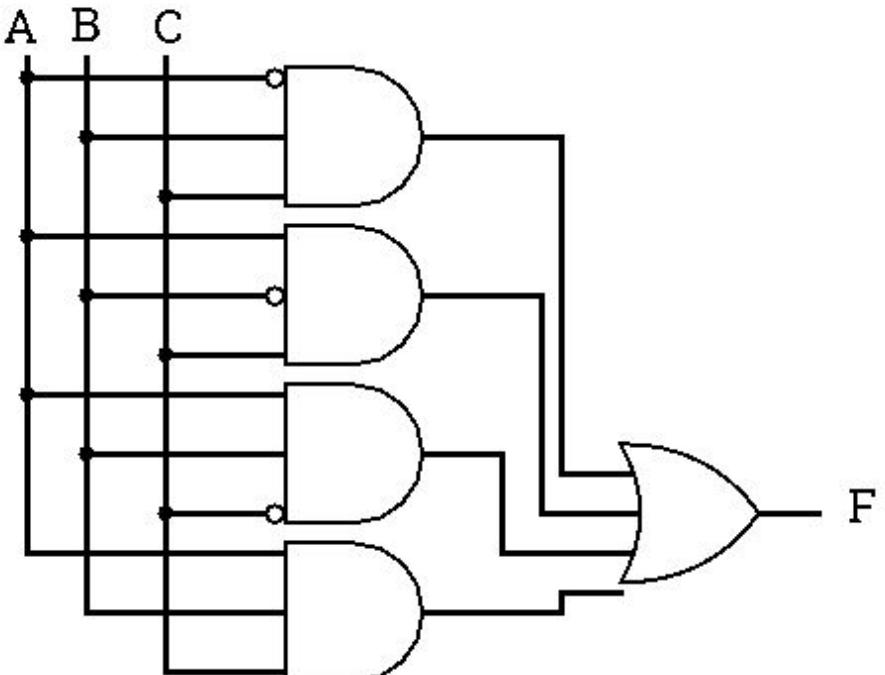
Elementos Avanzados 5

Ejemplos 6

## Ejemplo

Utilizando la FPGAGH0, genere una función mayoría (combinacional) con las entradas A, B y C.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Introducción 1

Módulos 2

Simulación 3

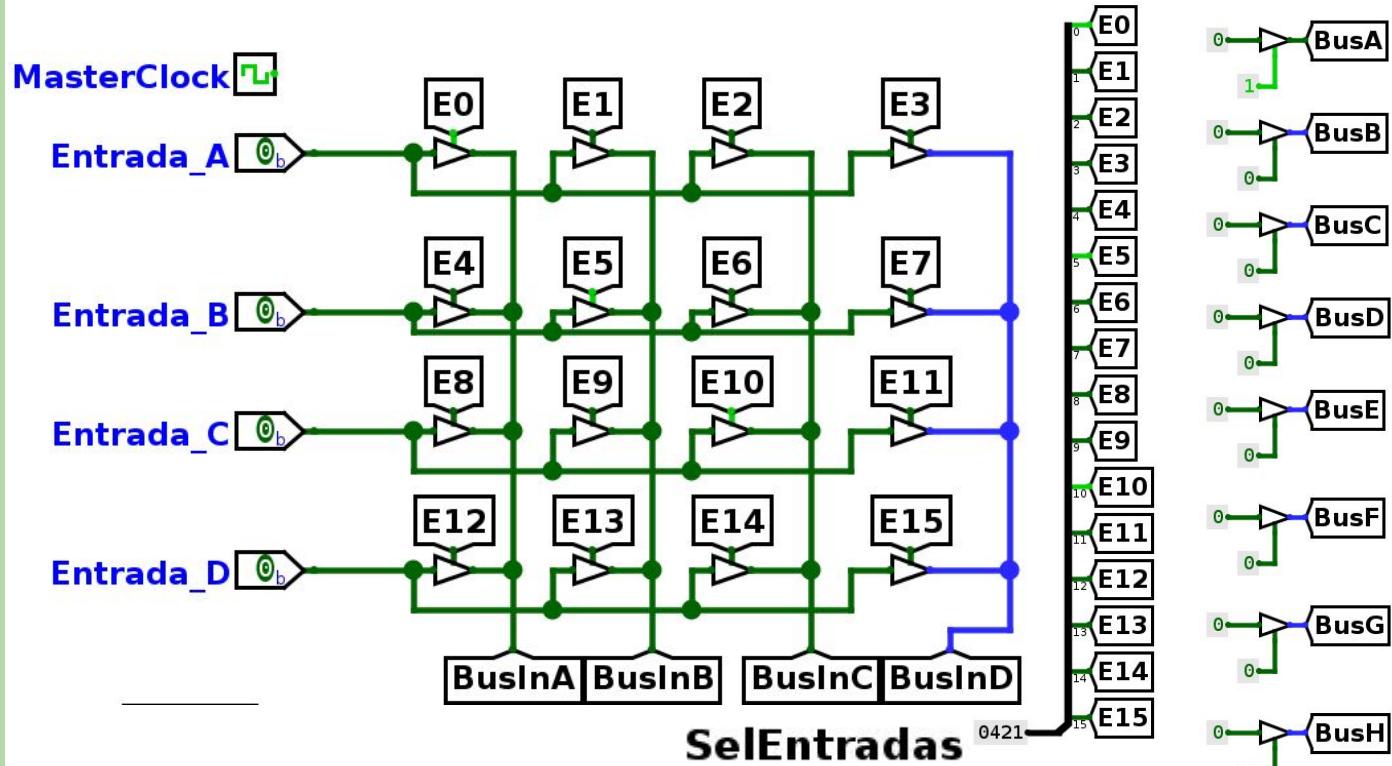
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

## Ejemplo - Conectando las entradas

Forzamos la Entrada\_A al BusInA, luego Entrada\_B al BusInB y Entrada\_C a BusInC. A tal fin seleccionamos con 0x0421. En BusA forzamos un cero a usarse en LUT4 cuando solo usamos 3.

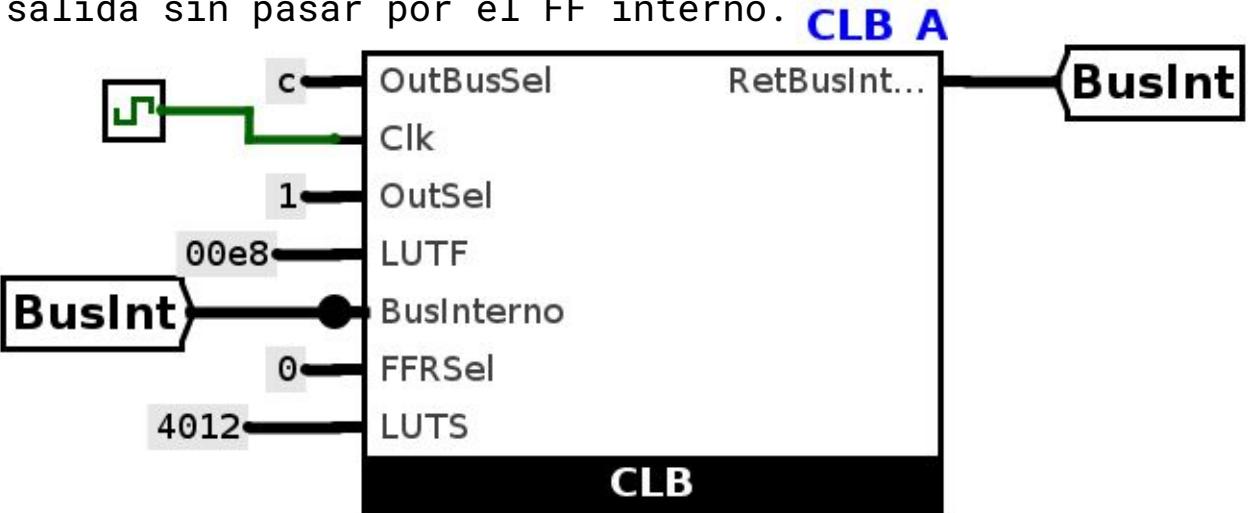


## Ejemplo - Programando la LUT

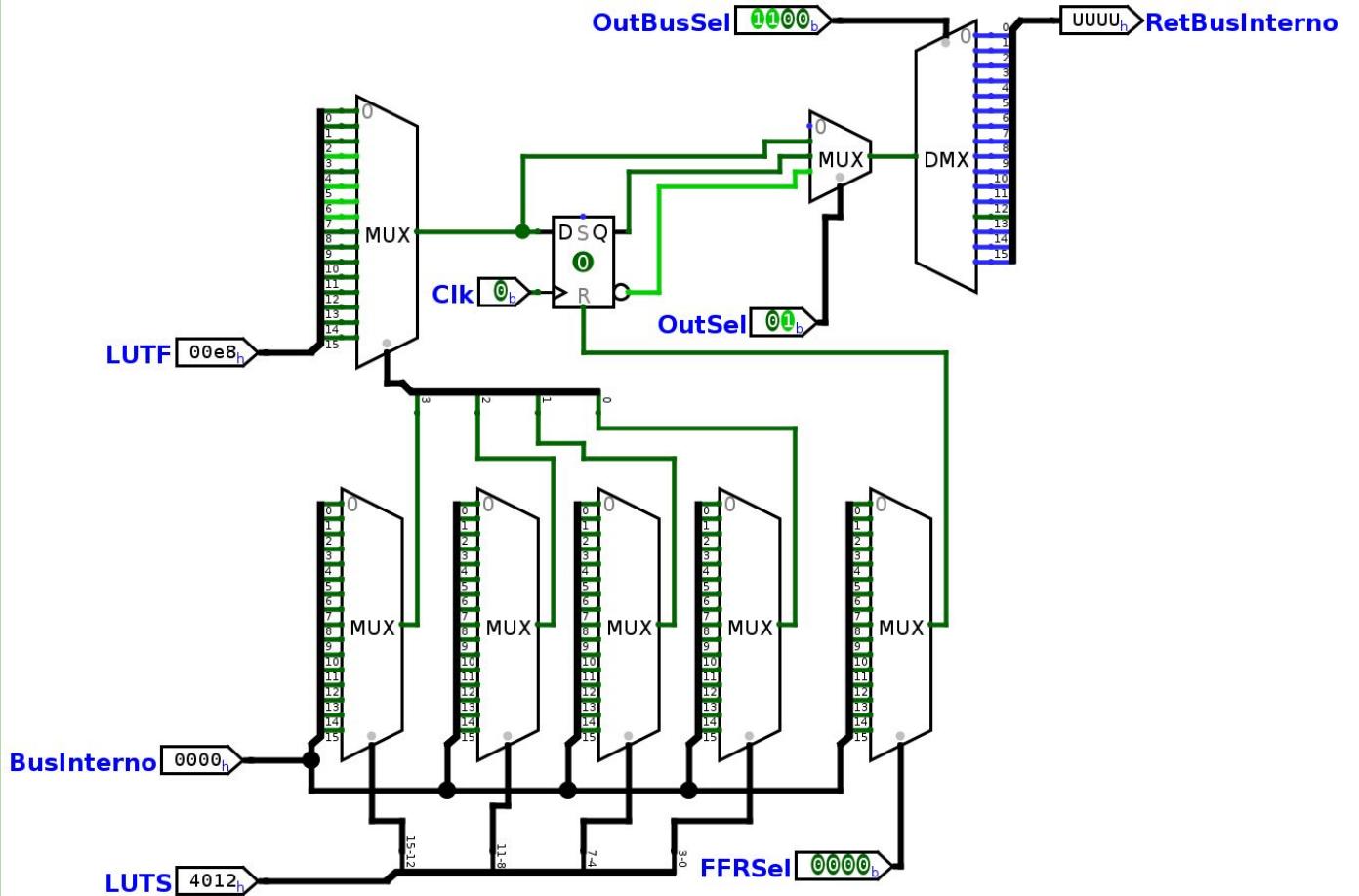
La función de la LUT es 00e8 (ver tabla de verdad expandida a 16). Las entradas de selección de la LUT serán:

- BusA (que tiene 0 forzado)
- BusInA (que tiene Entrada\_A)
- BusInB (que tiene Entrada\_B)
- BusInC (que tiene Entrada\_C)

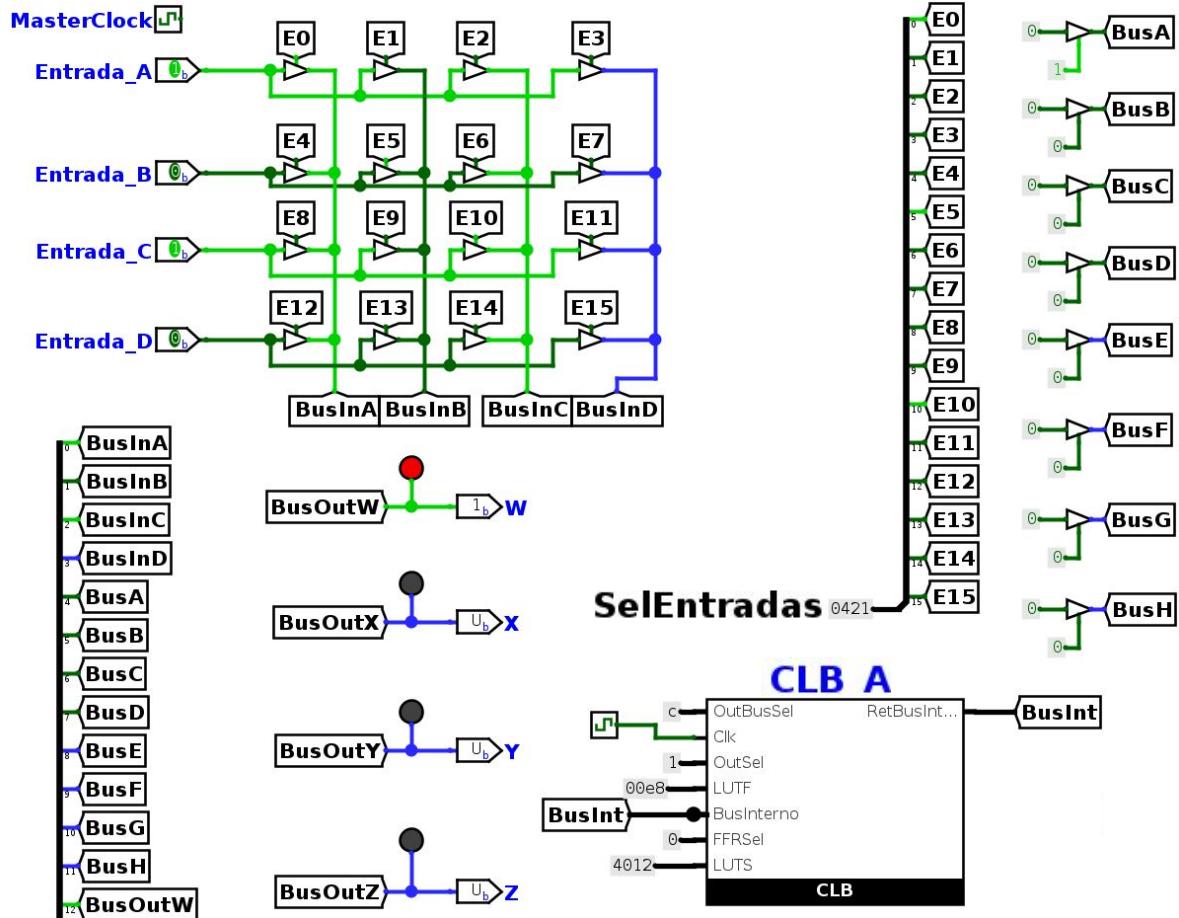
Por ende LUTS selecciona primero BusA (Elemento 4 del busint), luego BusInA (0), BusInB (1) y BusInC (2) , formando entonces 0x4012. La salida es a W (BusInt=C). Se toma la salida sin pasar por el FF interno.



# Ejemplo - Programando la LUT



# Ejemplo - Completo (FPGAGH0\_Mayoria.circ)



## Ejemplo - Bitstream (FPGAGH0\_Mayoria.circ)

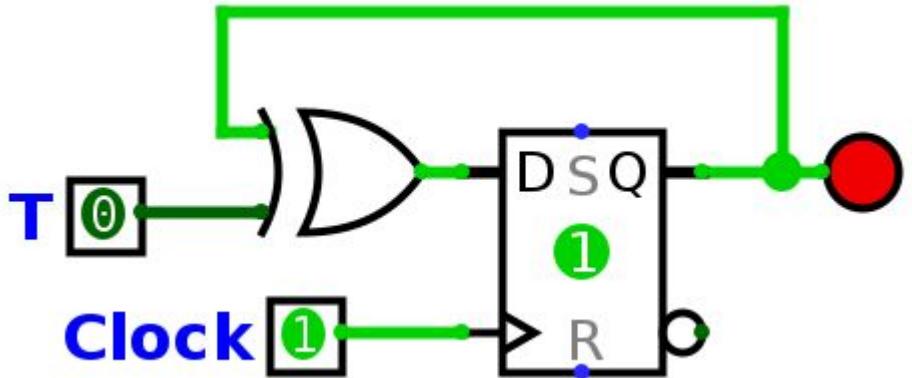
Esta tabla resume todo el funcionamiento

Bitstream Mayoría					
SelEntradas	0421	BusInA → Entrada_A	BusInB → Entrada_B	BusInC → Entrada_C	BusInD → Hi-Z
Bus Interno		BusA → 0	BusB → Z	BusC → Z	BusD → Z
		BusE → Z	BusF → Z	BusG → Z	BusH → Z
CLBs		CLB A	CLB B	CLB C	CLB D
Outsel 0→Z 1→Lut 2→FF 3→IFF	LUTF	00E8	0000	0000	0000
	LUTS	4012	0000	0000	0000
	OutSel	1	0	0	0
	OutBusSel	C	D	E	F
	FFRSel	0	0	0	0



## Ejemplo2

Utilizando la Entrada\_A como T , y la Entrada\_B como Reset, implemente un FF tipo T utilizando el FF D de una CLB. La salida es el LED W.



T	Q	D
0	0	0
0	1	1
1	0	1
1	1	0

Introducción 1

Módulos 2

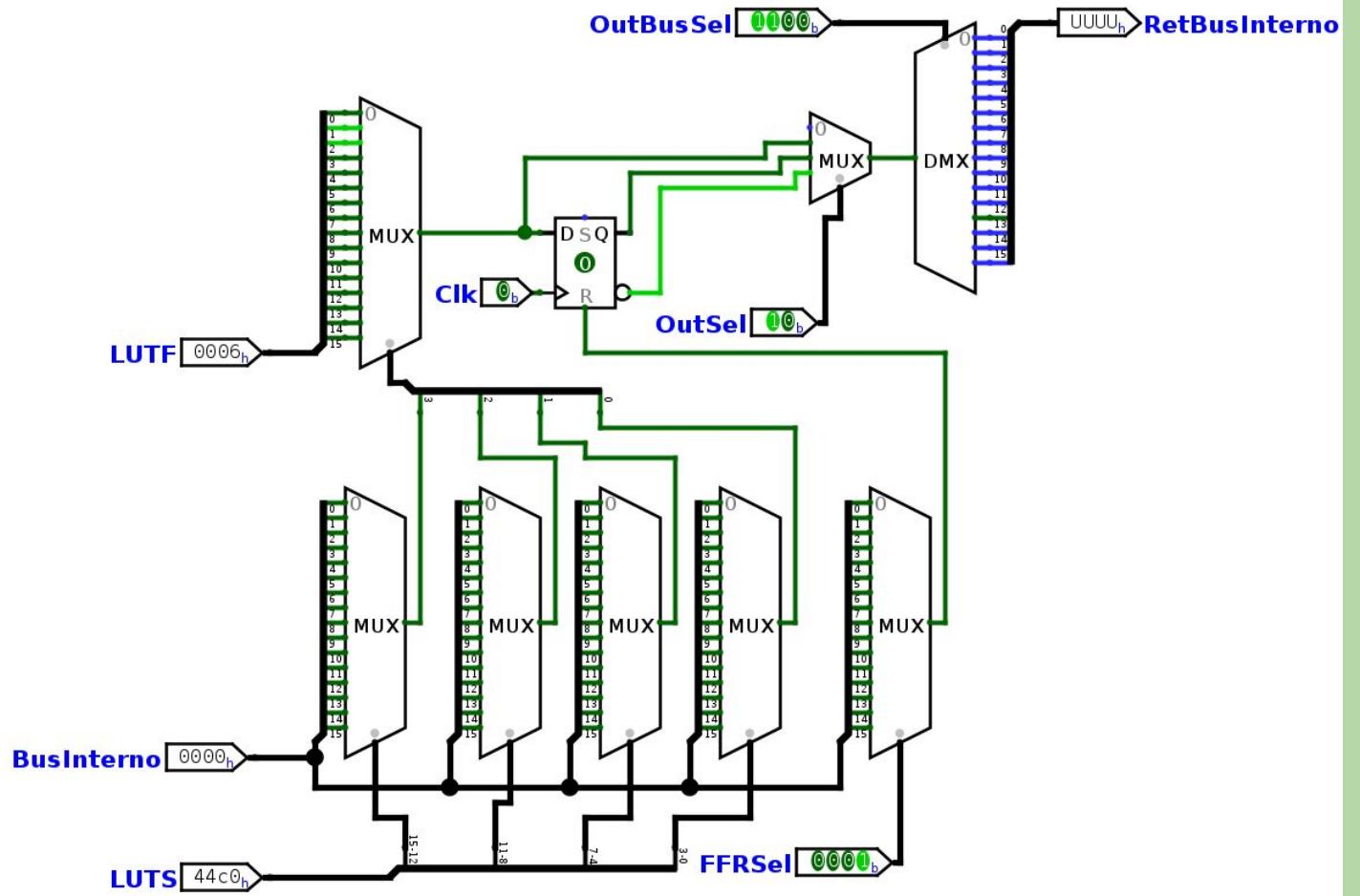
Simulación 3

Secuenciales 4

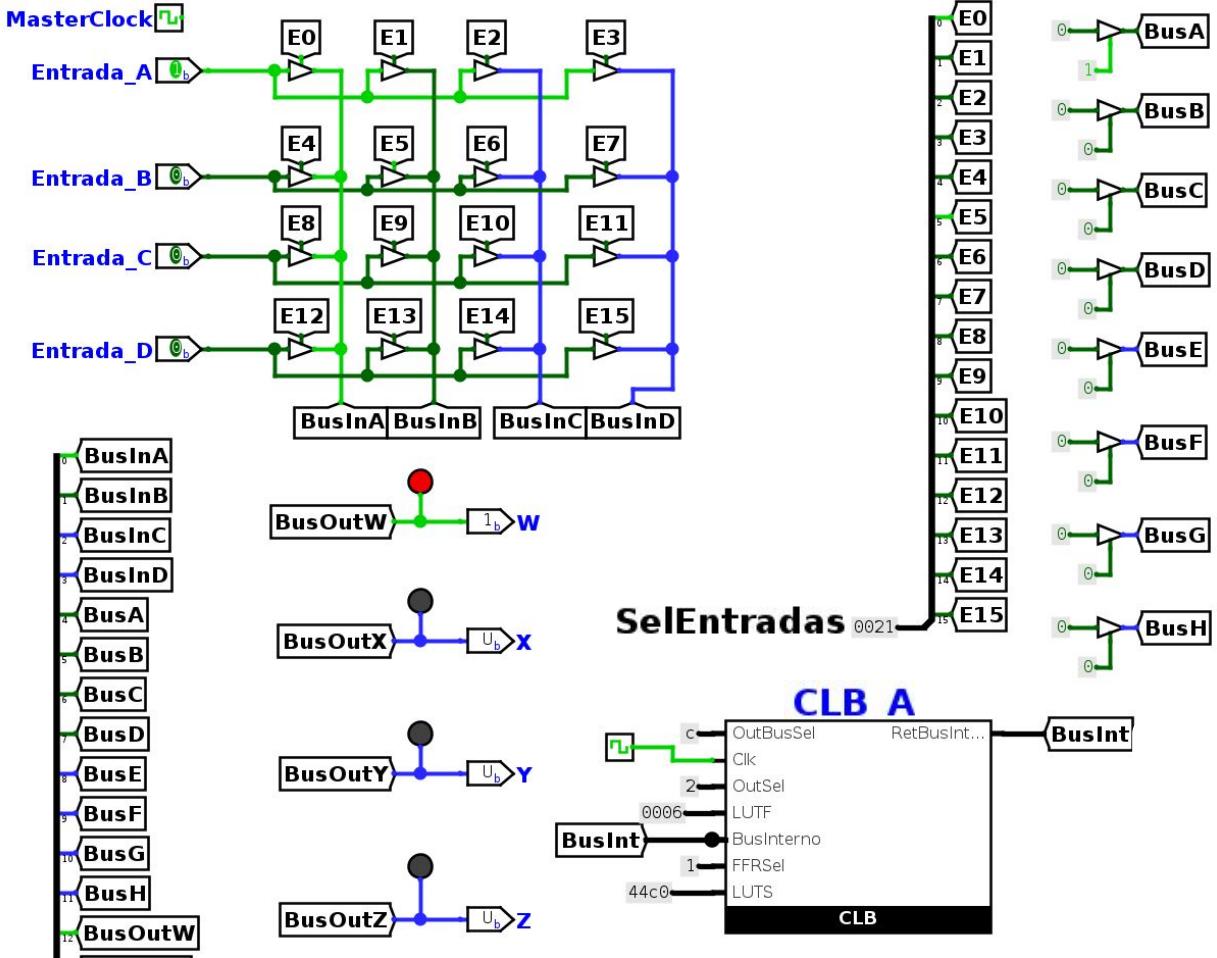
Elementos Avanzados 5

Ejemplos 6

## Ejemplo2 - LUT (0x0006)



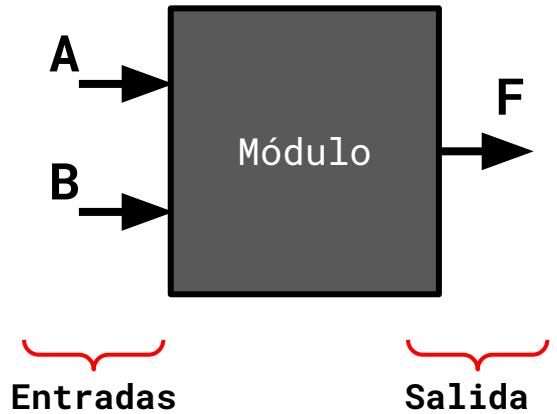
## Ejemplo2 - Completo (FPGAGHO\_TFF.circ)



# Módulos en Verilog



## Diseño modular (Caja negra)



```
module nombre ( lista de puertos );
    • Interface
        ○ Tipos de los puertos
        ○ Parámetros
    • Body
        ○ Variables
        ○ Asignaciones
        ○ Instanciación de módulos
        ○ Bloques de simulación
        ○ Bloques de comportamiento
        ○ Funciones y tareas
endmodule
```

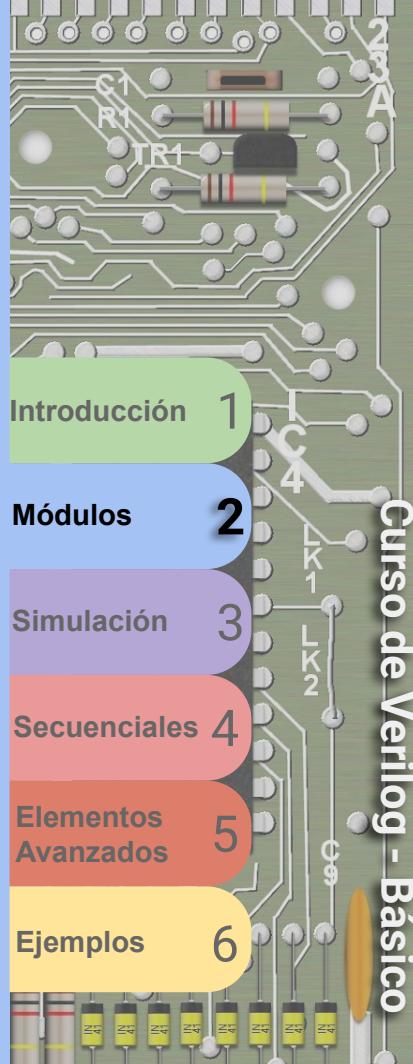
### Verilog

```
module Modulo( A,B,F);
    input A;
    input B;
    output F;
endmodule
```

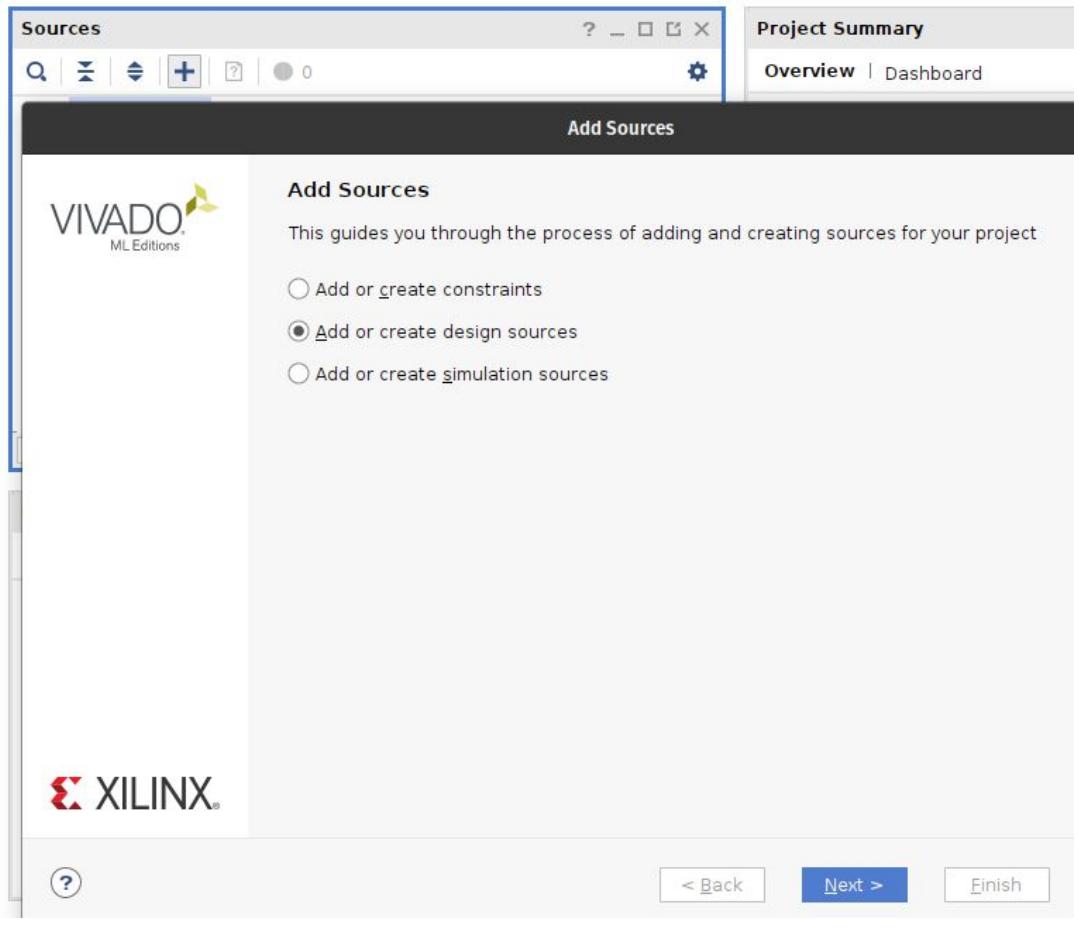
### Verilog 2001

```
module Modulo(
    input A,
    input B,
    output F
);
endmodule
```

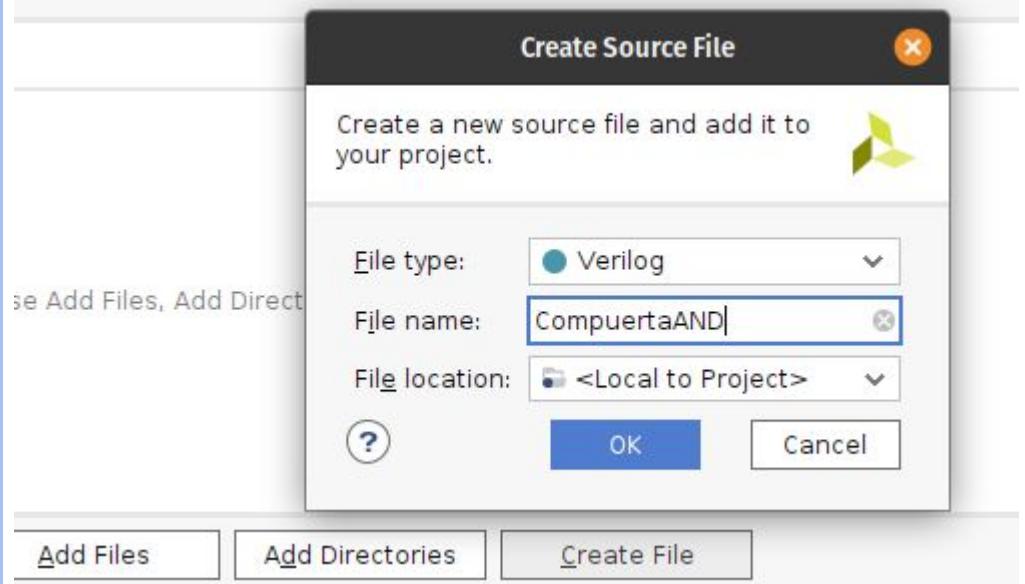
```
module Modulo( input A,B,
                output F );
endmodule
```



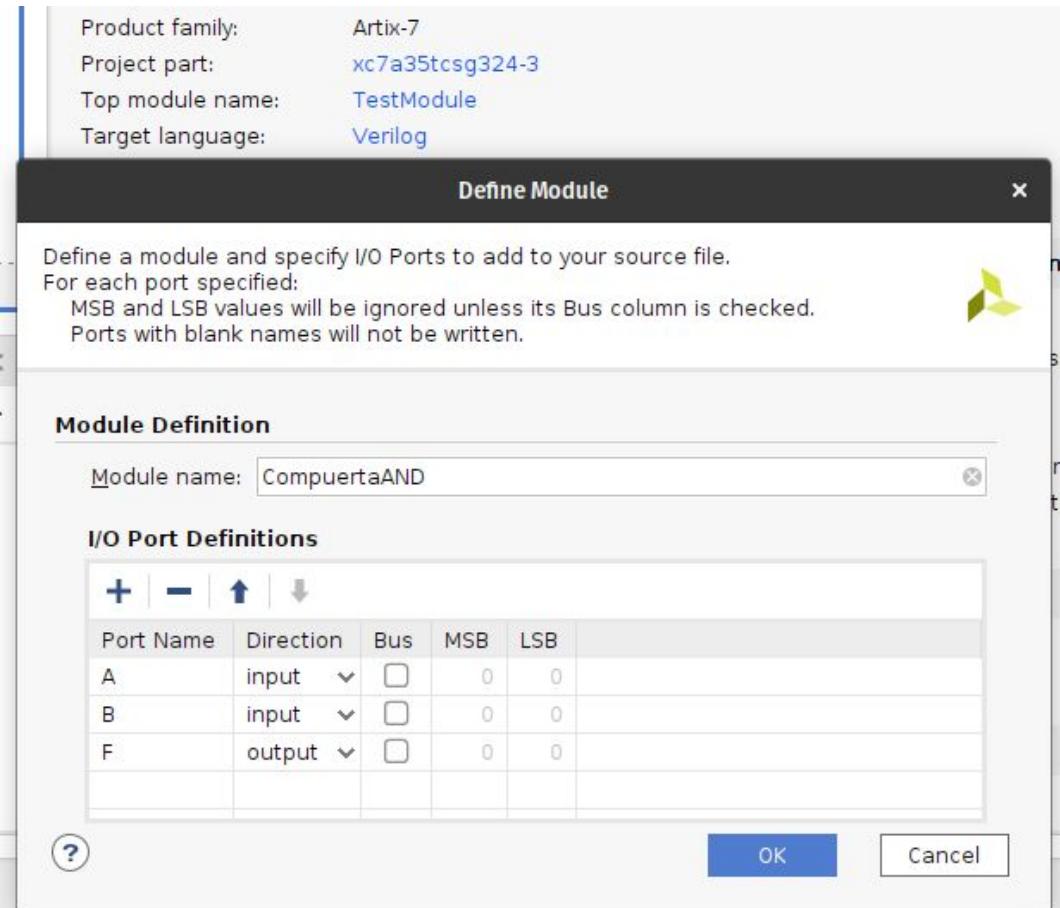
# Vivado (creando un módulo)



# Vivado (creando un módulo)



# Vivado (creando un módulo)



# Vivado (creando un módulo)

The screenshot shows the Vivado IDE interface. On the left, the 'Sources' window displays a tree structure with 'Design Sources (1)' containing 'CompuertaAND (CompuertaAND.v)', 'Constraints', 'Simulation Sources (1)' with 'sim\_1 (1)', and 'Utility Sources'. The 'CompuertaAND.v' file is selected. On the right, the code editor window titled 'CompuertaAND.v' shows the following Verilog code:

```
1 timescale 1ns / 1ps
2
3 module CompuertaAND(
4     input A,
5     input B,
6     output F
7 );
8 endmodule
9
```



# Puertos y Tipos de dato



# Puertos y tipos de dato (verilog 2001)

## Puertos

- **input**
- **output**
- **inout**

Por defecto los puertos son de tipo **wire**. Los **output** pueden definirse como '**reg**' (variable). Se puede asignar un rango **[x:y]** a un puerto para indicar un bus.

Por defecto los buses se consideran como '**unsigned**' pero puede definirse como '**signed**' de ser necesario.

## Valores posibles (wire/reg)

- 0
- 1
- X (**unknown**)
- Z (**High Impedance**)

## Tipos de datos

- **wire**
- **reg**
- *integer*
- *time*
- *real*
- *string*

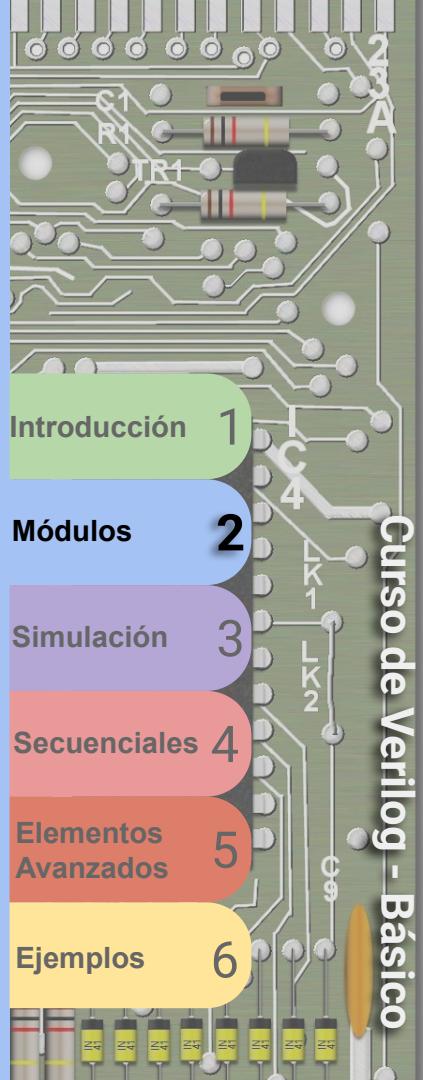
Algunos valores sólo tienen sentido en simulaciones.

## Valores literales

Se forman indicando la cantidad de **bits**, luego comilla simple y la base seguido del número.

### Ejemplos

- 4'b0110
- 4'h6 (0110<sub>2</sub>)
- 5'h1f (11111<sub>2</sub>)
- 5'd31 (11111<sub>2</sub>)



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

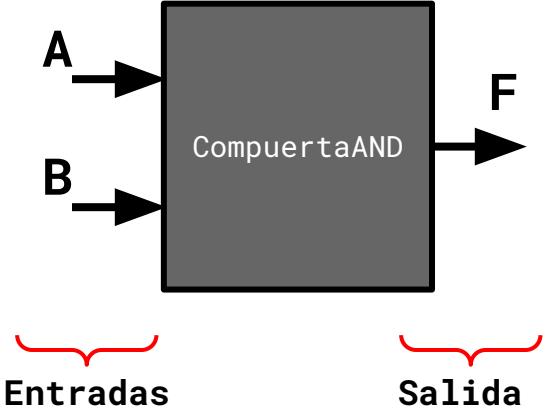
Elementos Avanzados 5

Ejemplos 6

# Lógica combinatoria



## Compuerta AND (assign)



```
module CompuertaAND (
    input A,
    input B,
    output F
);
    assign F = A & B;
endmodule
```

Operadores	
Símbolo	Operación
! o ~	Negación (Bit)
&	And (Bit)
~&	Nand (Reduc)
	Or (Bit)
~	Nor (Reduc)
^	Xor (Bit)
~^	Xnor (Bit)

### Operadores Bit

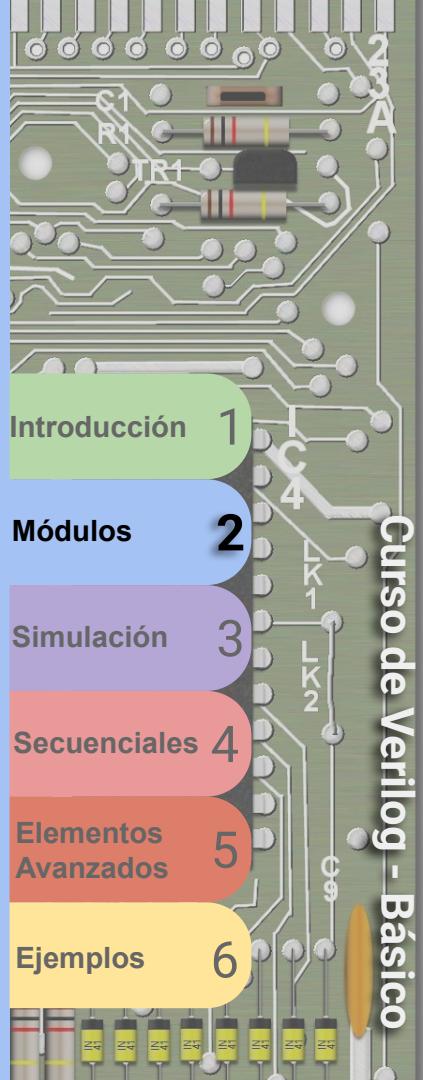
Se aplican bit a bit entre dos buses (o cables).

Ej: Si  $A[3:0]$  y  $B[3:0]$  entonces  $A \& B$  realiza  $A[0] \text{ AND } B[0]; A[1] \text{ AND } B[1] \dots$

### Operadores Reducción

Se aplican sobre todos los bits de un solo bus.

Ej: Si  $A[3:0]$  entonces  $\sim\&A$  equivale a  $A[3] \text{ NAND } A[2] \text{ NAND } A[1] \text{ NAND } A[0]$



# Vivado (Síntesis y esquemático )

## SYNTHESIS

Run Synthesis

## Open Synthesized Design

Constraints Wizard

Edit Timing Constraints

Set Up Debug

Report Timing Summary

Report Clock Networks

Report Clock Interaction

Report Methodology

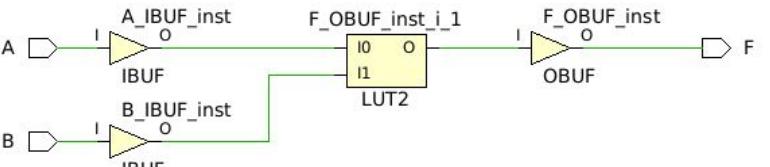
Report DRC

Report Noise

Report Utilization

Report Power

Schematic



[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_7/7series\\_scm.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_scm.pdf)

**LUT2**  
Primitive: 2-Bit Look-Up Table with General Output

**Cell Properties**

F_OBUF_inst_i_1	CLASS	cell
	FILE_NAME	/home/edgardog/Ne
INIT	INIT	4'h8

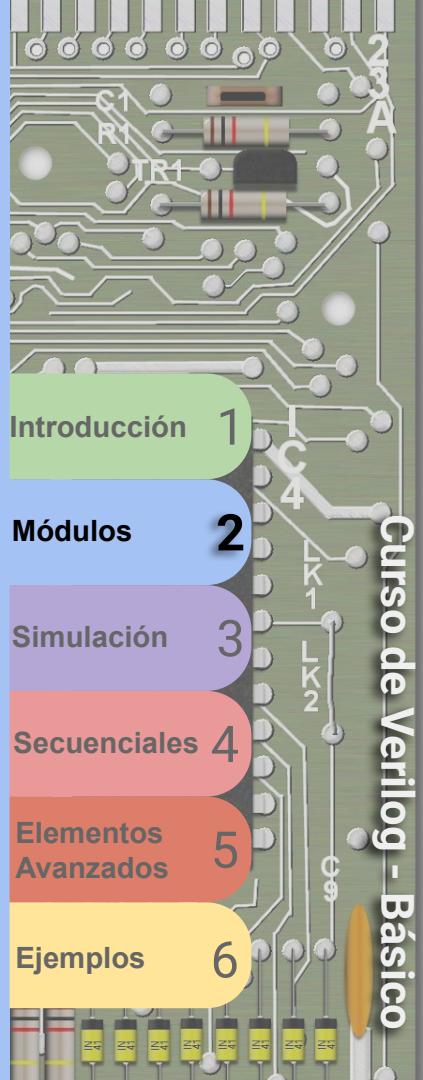
**Logic Table**

Inputs		Outputs
I1	I0	O
0	0	INIT[0]
0	1	INIT[1]
1	0	INIT[2]
1	1	INIT[3]

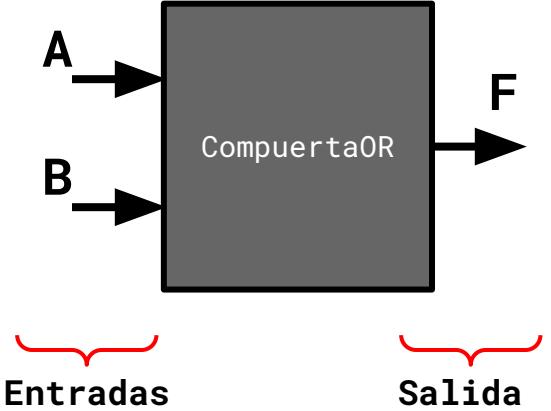
## Logic Table

Inputs		Outputs
I1	I0	O
0	0	INIT[0]
0	1	INIT[1]
1	0	INIT[2]
1	1	INIT[3]

INIT = Binary equivalent of the hexadecimal number assigned to the INIT attribute

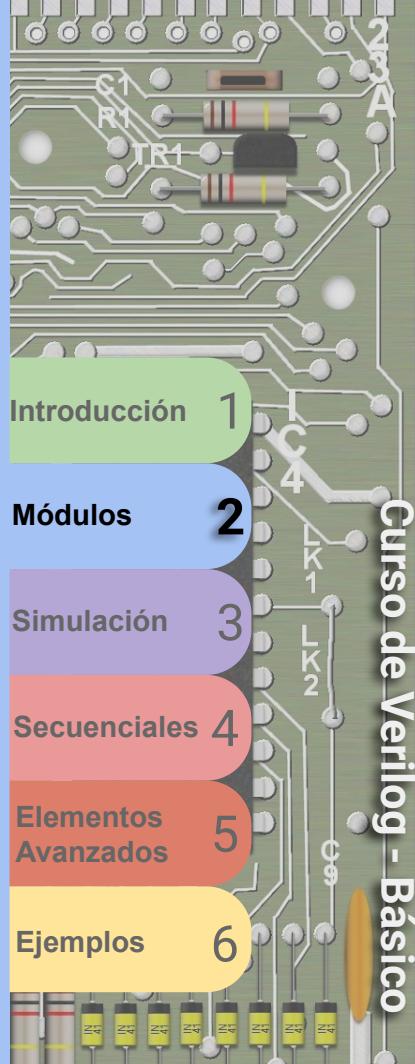
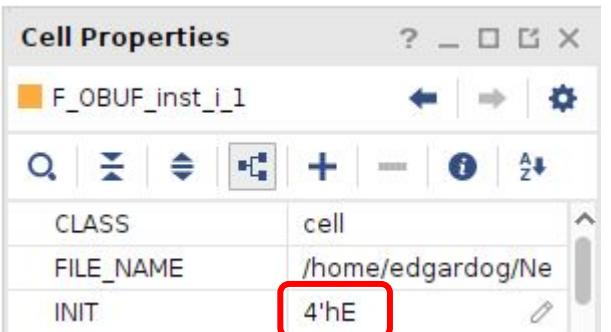
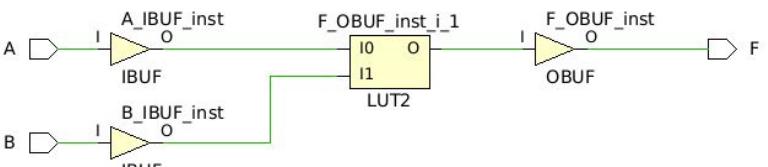


## Compuerta OR (assign)

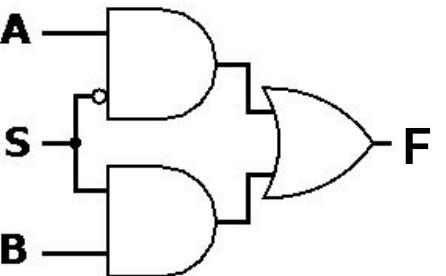
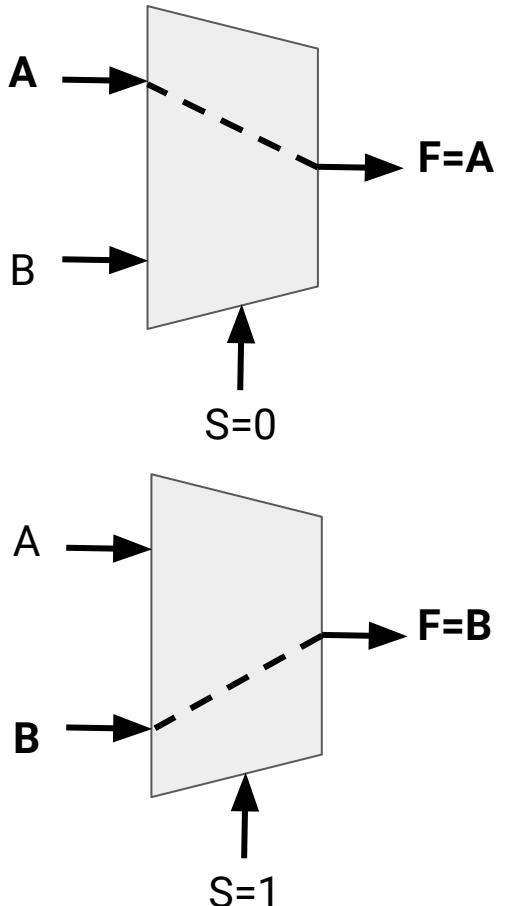


A	B	Or
0	0	0
0	1	1
1	0	1
1	1	1

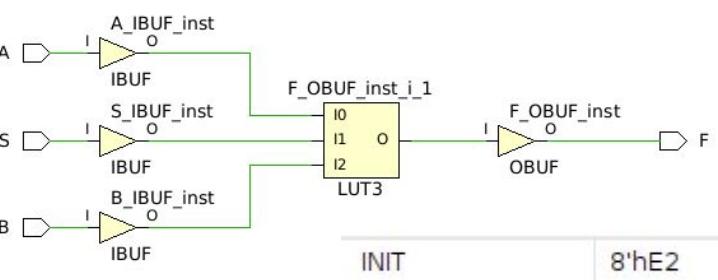
```
module CompuertaOR(
    input A,
    input B,
    output F
);
    assign F = A | B ;
endmodule
```



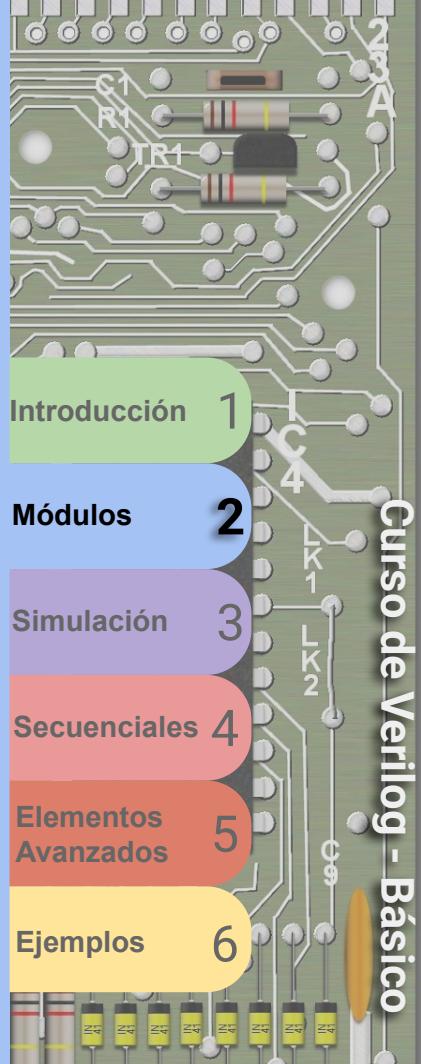
## Multiplexor (assign)



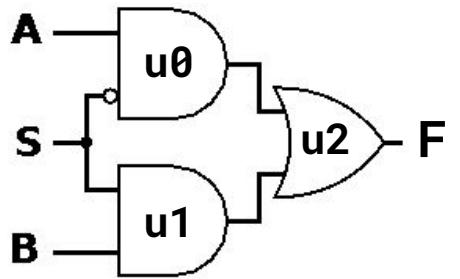
```
module Mux2(
    input A,
    input B,
    input S,
    output F
);
    assign F = ( A & !S ) | ( B & S );
endmodule
```



B	S	A	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



# Multiplexor (instancia de módulo)

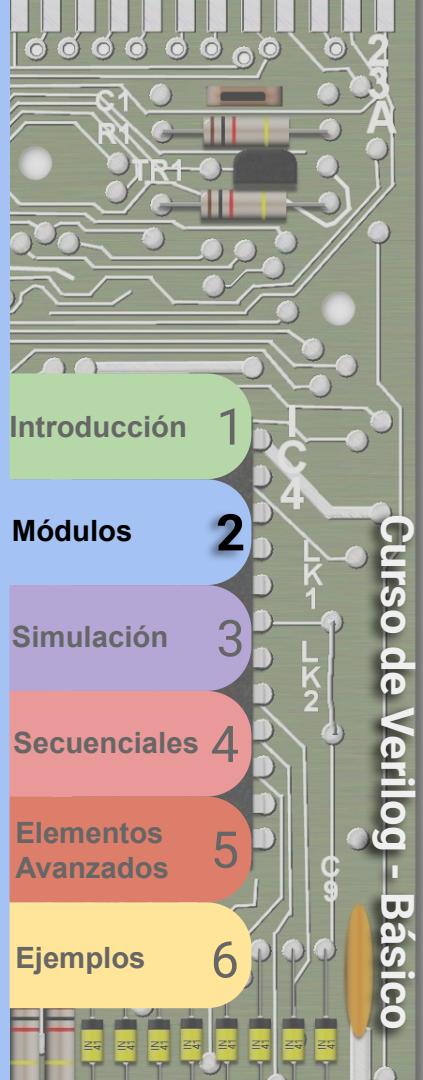
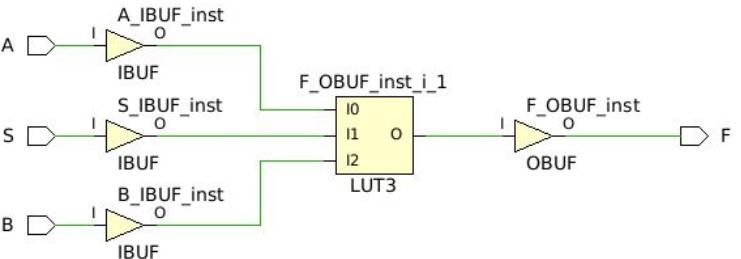
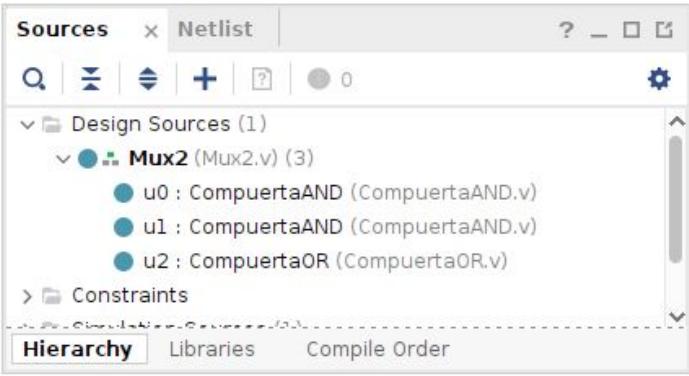


```
module Mux2(
    input A,
    input B,
    input S,
    output F
);

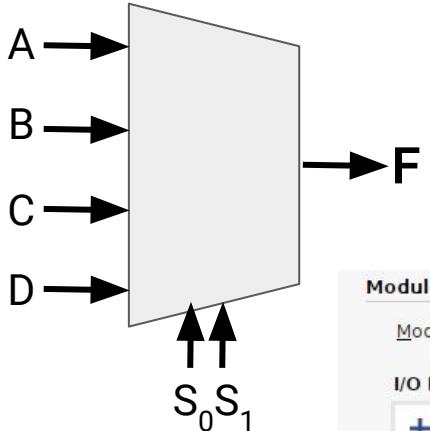
    wire outAnd0;
    wire outAnd1;
    wire notS;

    assign notS = !S;
    //instancia definiendo nombres
    CompuertaAND u0 (.A(A),
                      .B(notS),
                      .F(outAnd0));
    //instancia ordenada
    CompuertaAND u1 (B,S,outAnd1);

    CompuertaOR u2 (outAnd0,outAnd1,F);
endmodule
```



## Multiplexor (buses)



Module Definition

Module name: Mux4

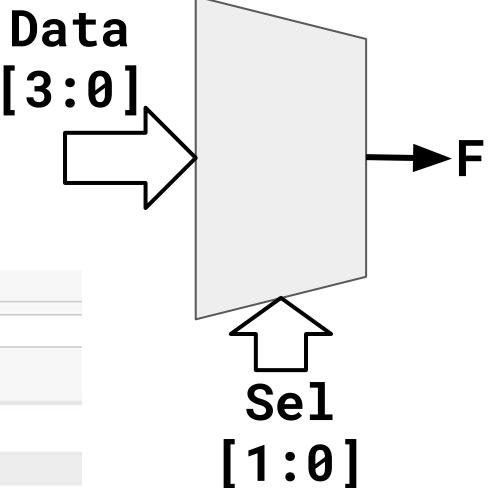
I/O Port Definitions

+ - ↑ ↓

Port Name	Direction	Bus	MSB	LSB
Data	input	<input checked="" type="checkbox"/>	3	0
Sel	input	<input checked="" type="checkbox"/>	1	0
F	output	<input type="checkbox"/>	0	0

```
module Mux4(
    input [3:0] Data,
    input [1:0] Sel,
    output F
);
endmodule
```

$S_1$	$S_0$	F
0	0	A
0	1	B
1	0	C
1	1	D



<b>Sel</b>	<b>F</b>
00	Data[0]
01	Data[1]
10	Data[2]
11	Data[3]

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

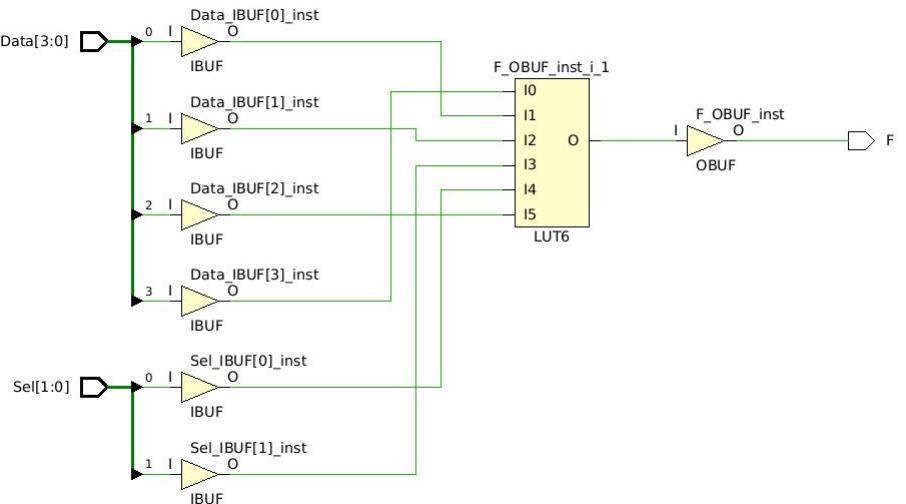
Elementos Avanzados 5

Ejemplos 6

# Multiplexor (buses)

```
module Mux4(
    input [3:0] Data,
    input [1:0] Sel,
    output F
);

    assign F = ( Data[0] & !Sel[1] & !Sel[0] ) ||
               ( Data[1] & !Sel[1] & Sel[0] ) ||
               ( Data[2] & Sel[1] & !Sel[0] ) ||
               ( Data[3] & Sel[1] & Sel[0] );
endmodule
```



Introducción 1

Módulos 2

Simulación 3

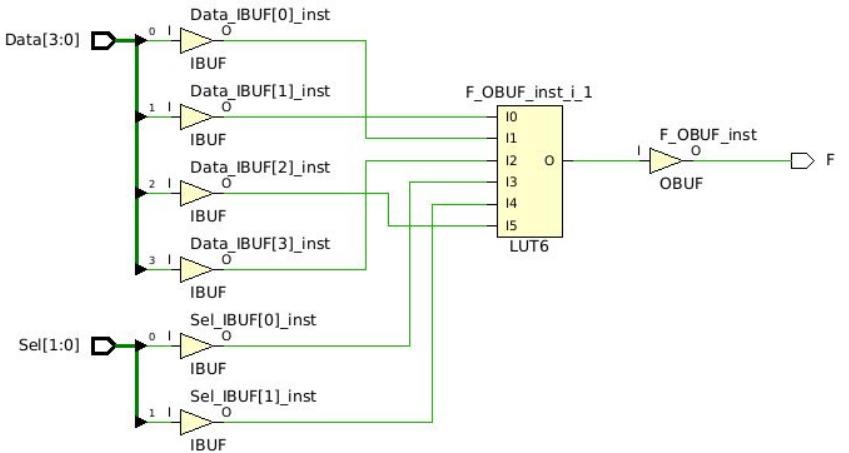
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Multiplexor (adelanto de procesos)

```
module Mux4(
    input [3:0] Data,
    input [1:0] Sel,
    output reg F
);
    always @*
    begin
        case(Sel)
            2'b00 : F = Data[0];
            2'b01 : F = Data[1];
            2'b10 : F = Data[2];
            default: F = Data[3];
        endcase
    end
endmodule
```



```
module Mux4(
    input [3:0] Data,
    input [1:0] Sel,
    output reg F
);
    always @*
    begin
        if (Sel == 2'b00)
            F = Data[0];
        else if (Sel == 2'b01)
            F = Data[1];
        else if (Sel == 2'b10)
            F = Data[2];
        else
            F = Data[3];
    end
endmodule
```



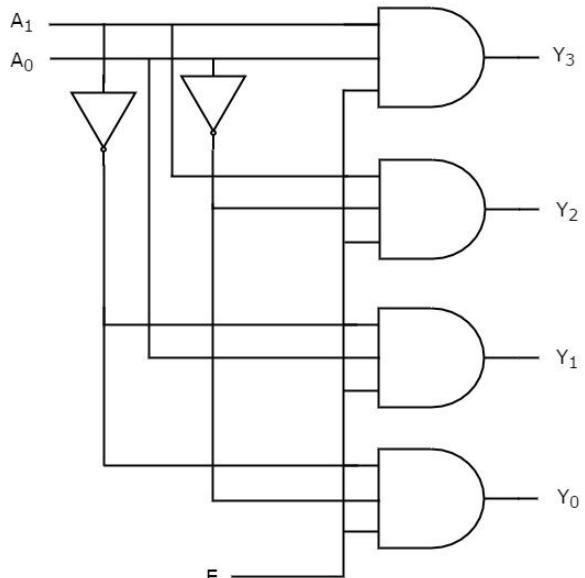
# Asignaciones condicionales



## Asignación condicional (sin always @\*)

**assign salida = (condición)? valor\_verdadero:valor\_falso**

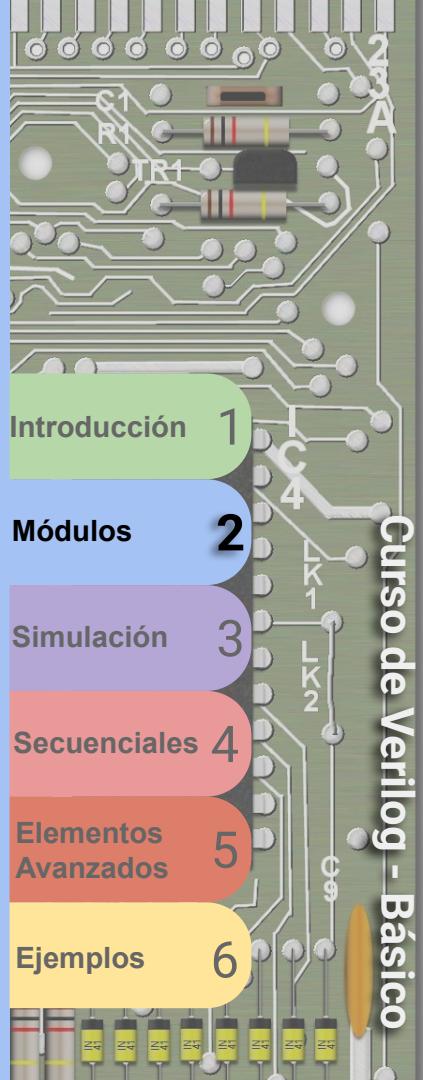
```
assign S = (Opcode[6:0] == 7'b0100011)? 1'b1:1'b0;  
assign L = (Opcode[6:0] == 7'b0000011);  
assign R = (Opcode[6:0] == 7'b0110011 & Inst[29:24]==6'd0 & Inst31[31]==1'd0)? 1'b1:1'b0;
```



**assign y3 = (a == 2'b11 & E);**

**assign y2 = (a == 2'd2 & E);**

**assign y1 = (a == 2'h1 & E);**



# Simulando un módulo



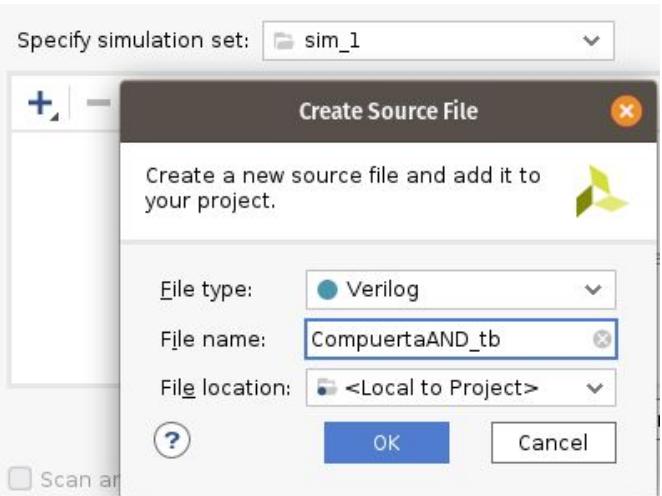
# Testbench



## Add Sources

This guides you through the process of adding and creat

- Add or create constraints
- Add or create design sources
- Add or create simulation sources



## Module Definition

Module name: CompuertaAND\_tb

## I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input		0	0

Introducción

1

Módulos

2

Simulación

3

Secuenciales

4

Elementos Avanzados

5

Ejemplos

6

# Testbench

```
timescale 1ns / 1ps

module CompuertaAND_tb();

reg a,b;
wire f;

CompuertaAND U0 ( .A(a), .B(b), .F(f) );

initial
begin

a=0;
b=0;
#4;

a=1;
#2;

b=1;
#1;

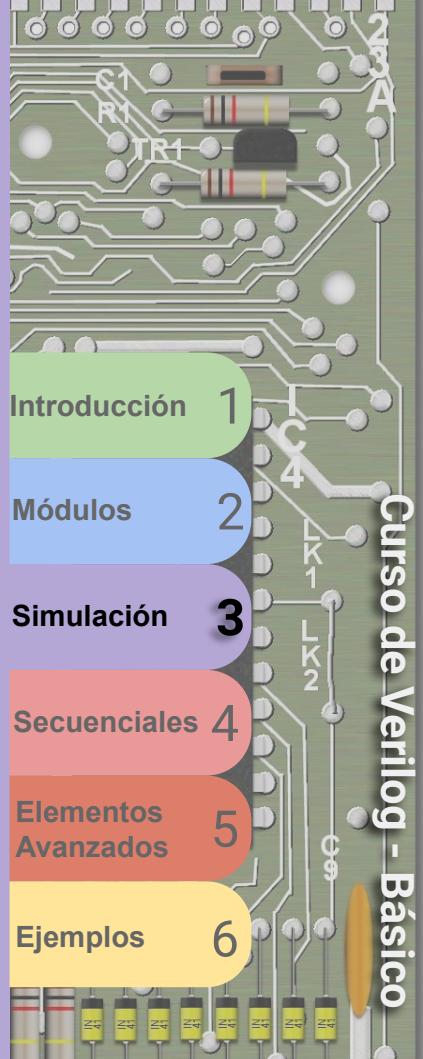
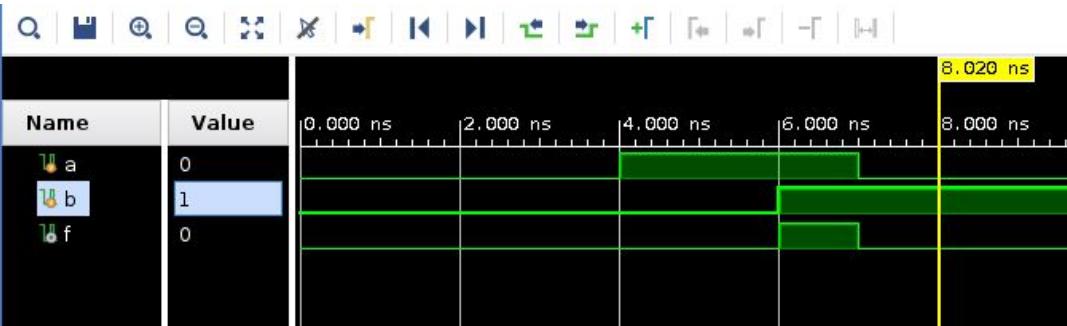
a=0;
#5;

end

endmodule
```



SIMULATION  
Run Simulation



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

Curso de Verilog - Básico

# Funciones auxiliares

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Testbench (archivo de log)

```
module CompuertaAND_tb();

reg a,b;
wire f;

CompuertaAND U0 ( .A(a), .B(b), .F(f) );

initial begin
$monitor("Time=%0t , (A[%0d] AND B[%0d]) = F[%0d]",
        $time,a,b,f);
end

initial
begin

a=0;
b=0;
#4;

a=1;
#2;

b=1;
#1;

a=0;
#5;

end
endmodule
```

# run 1000ns  
Time=0 , (A[0] AND B[0]) = F[0]  
Time=4000 , (A[1] AND B[0]) = F[0]  
Time=6000 , (A[1] AND B[1]) = F[1]  
Time=7000 , (A[0] AND B[1]) = F[0]

Introducción 1

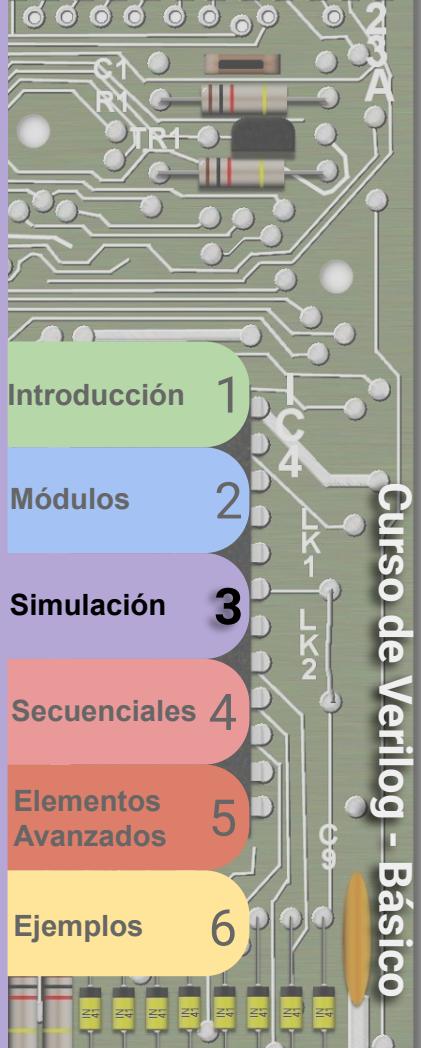
Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



# Testbench (funciones auxiliares)

```
module CompuertaAND_tb();
    reg a,b;
    wire f;

    CompuertaAND U0 ( .A(a), .B(b), .F(f) );

    initial
        begin
            a=0;
            b=0;
            #4;

            a=1;
            #2;
            $display("Time=%0t A=%d",$time,a);
            b=1;
            #1;

            a=0;
            #5;
        end
    endmodule

    # run 1000ns
    Time=6000 A=1
```

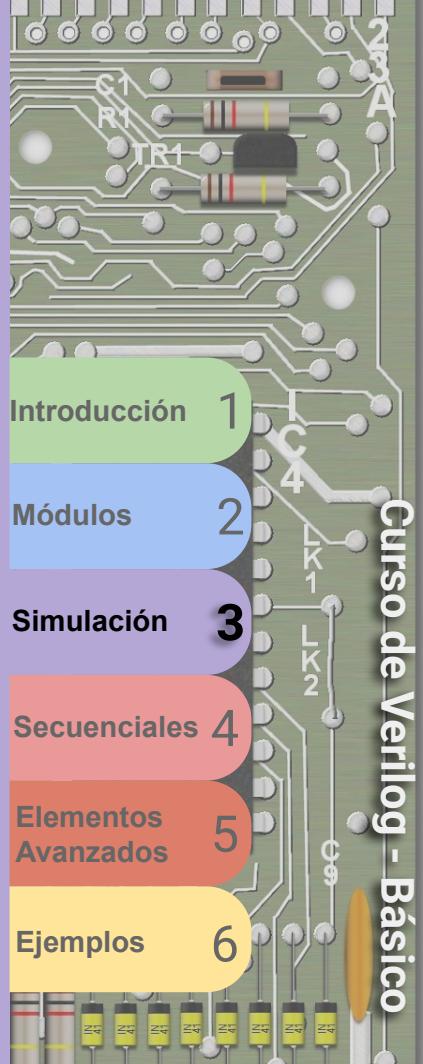
```
//Display inserta CRLF
$display("Time=%0t A=%d",$time,a);
//Write NO inserta CRLF
$write("Time=%0t A=%d",$time,a);
//Strobe muestra los valores ANTERIORES a $time
//mientras que $display muestra POSTERIORES
$strobe("Time=%0t A=%d",$time,a);
```

## Formateadores

- **%t Tiempo**
- **%h Hexadecimal**
- **%d Decimal**
- **%b Binario**
- **%s String**
- **%f Float**
- **%e Exponencial**

Existen funciones

\$f{open,close,monitor,display,  
write,strobe,gets,eof} que  
interactúan con archivos.  
\$sformat es análoga a fprintf.



# Flip Flops

Introducción 1

Módulos 2

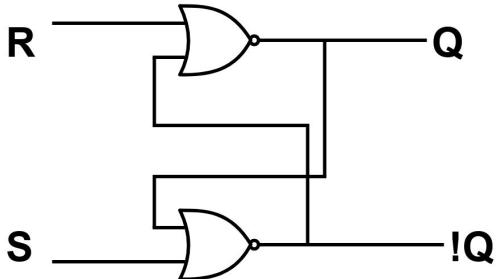
Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Latch RS (NOR)



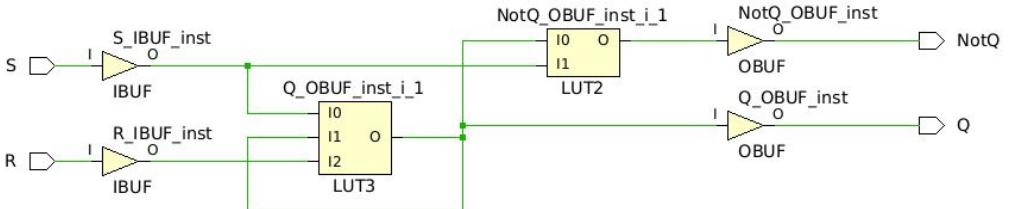
```
module LatchRS(
    input R,
    input S,
    output Q,
    output NotQ
);
    wire w1,w2;

    CompuertaNOR u0 (.A(R),.B(w2),.F(w1));
    CompuertaNOR u1 (.A(S),.B(w1),.F(w2));

    assign Q = w1;
    assign NotQ = w2;
endmodule
```

```
module CompuertaNOR(
    input A,
    input B,
    output F
);

    assign F = ~|{A,B};
    //La operacion NOR reduce un bus.
    //No es bitwise (entre dos bits)
    //Usando { } se concatenan entradas
    //y luego ~| reduce a un bit
endmodule
```



## Latch RS (NOR)

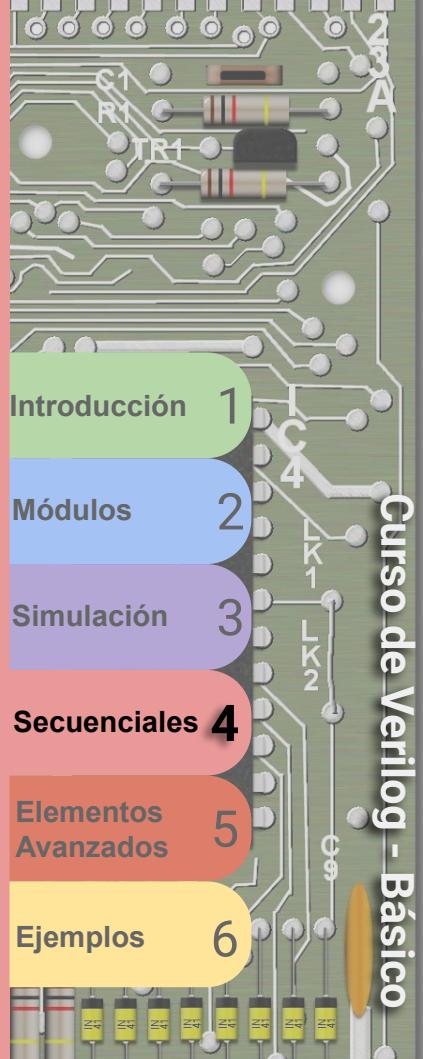
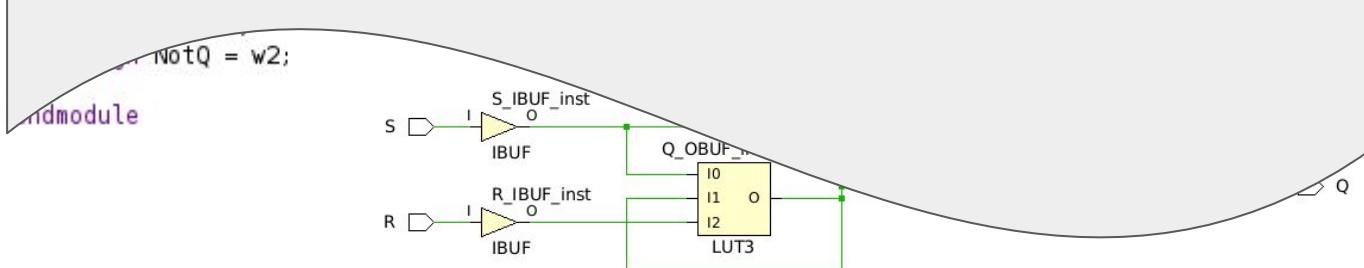


```
module CompuertaNOR(
    input A,
    input B,
    output F
);
```

$\sim\{A, B\};$   
reduce un bus.

Implementación: **Combinatorial Loop Alert:** 1 LUT cells form a combinatorial loop. This can create a race condition. Timing analysis may not be accurate. The preferred resolution is to modify the design to remove combinatorial logic loops. If the loop is known and understood, this DRC can be bypassed by..

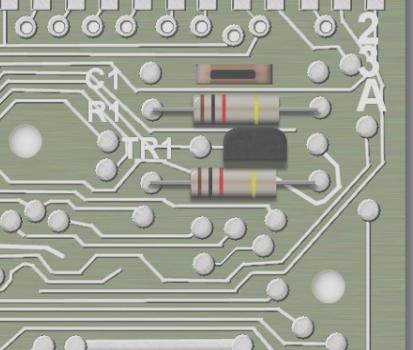
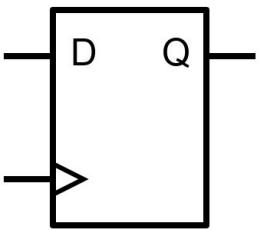
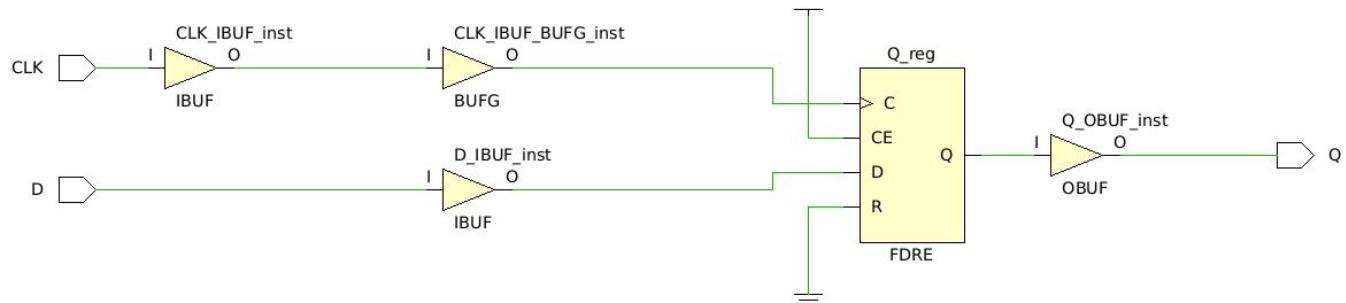
One net in the loop is Q\_OBUF. Please evaluate your design. The cells in the loop are: Q\_OBUF\_inst\_i\_1.



# Flip Flop D (flanco ascendente sin reset)

```
module FlipFlopD(
    input D,
    input CLK,
    output reg Q
);

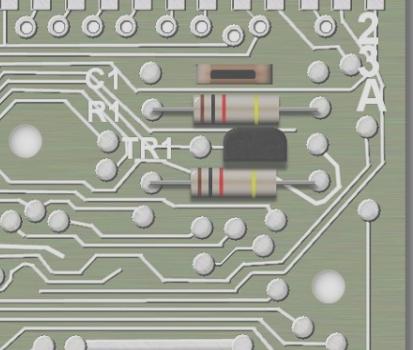
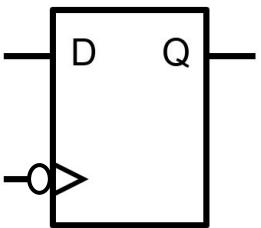
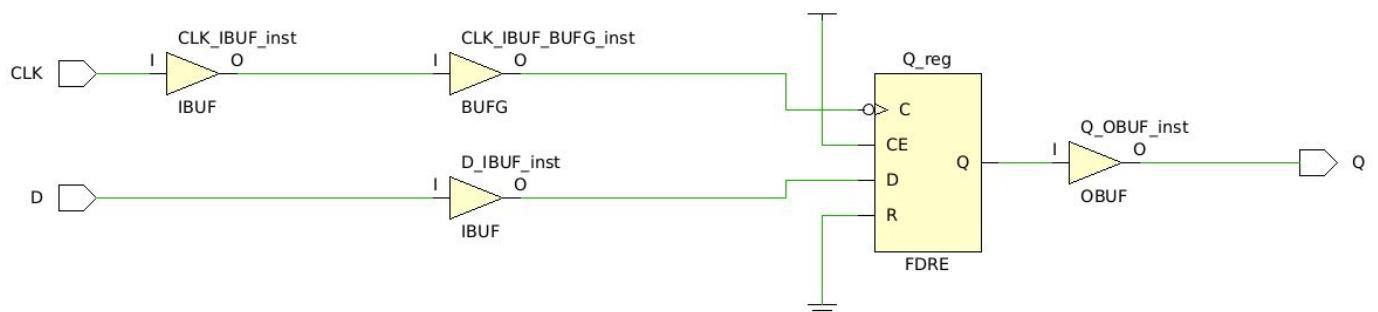
    always @ (posedge CLK)
    begin
        Q <= D;
    end
endmodule
```



# Flip Flop D (flanco descendente sin reset)

```
module FlipFlopD_desc(
    input D,
    input CLK,
    output reg Q
);

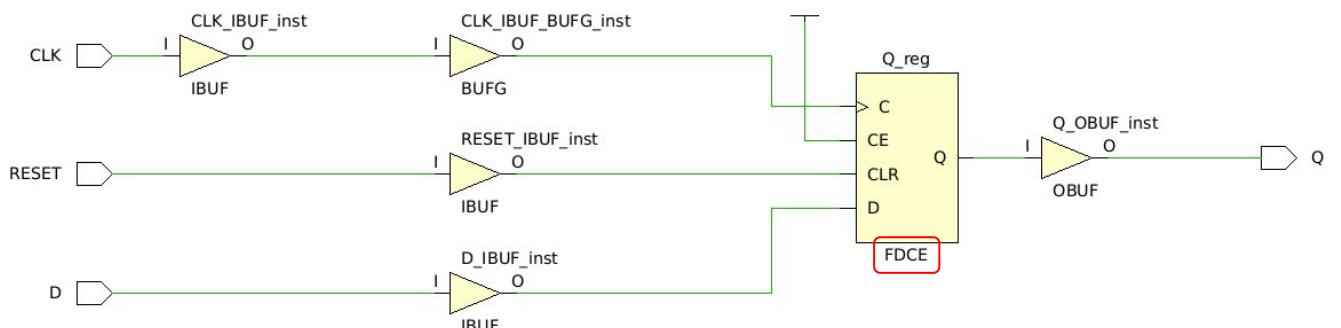
    always @ (negedge CLK)
    begin
        Q <= D;
    end
endmodule
```



# Flip Flop D (Reset Asincrónico)

```
module FlipFlopD_Rasinc(
    input D,
    input CLK,
    input RESET,
    output reg Q
);

    always @(posedge CLK or posedge RESET)
    begin
        if ( RESET==1'b1)
            Q <= 1'b0;
        else
            Q <= D;
    end
endmodule
```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

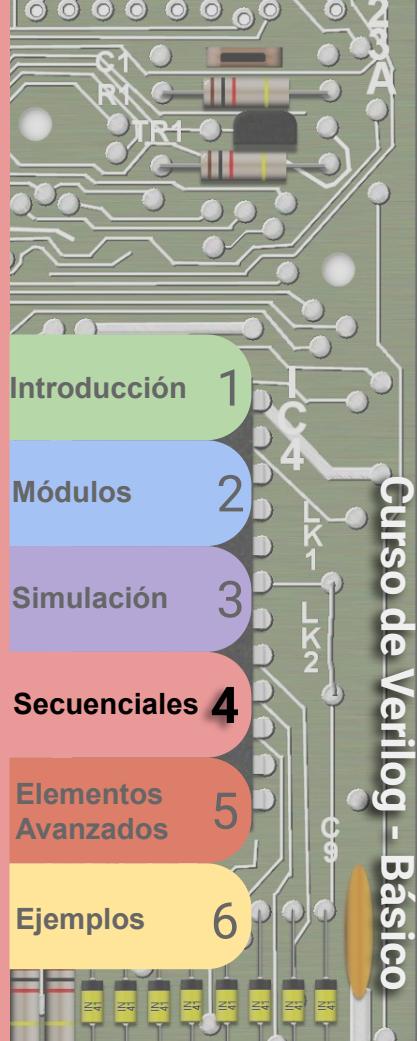
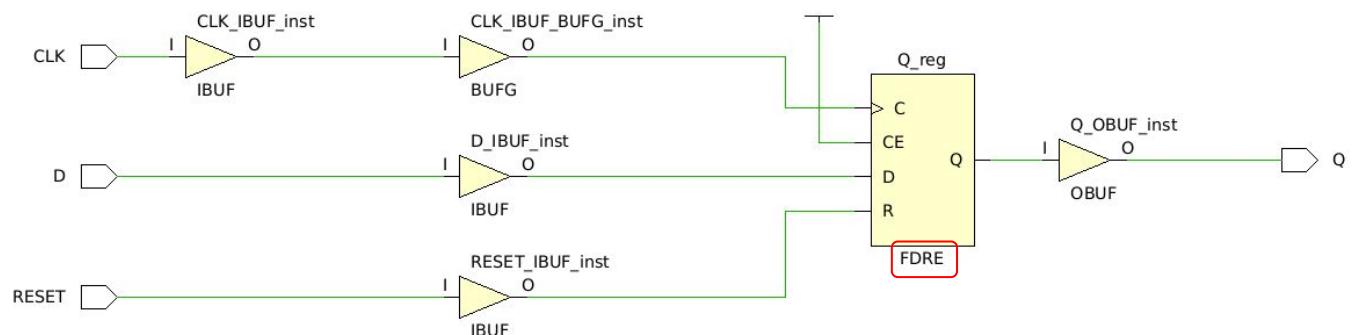
Elementos Avanzados 5

Ejemplos 6

# Flip Flop D (Reset Síncrono)

```
module FlipFlopD_Rsinc(
    input D,
    input CLK,
    input RESET,
    output reg Q
);

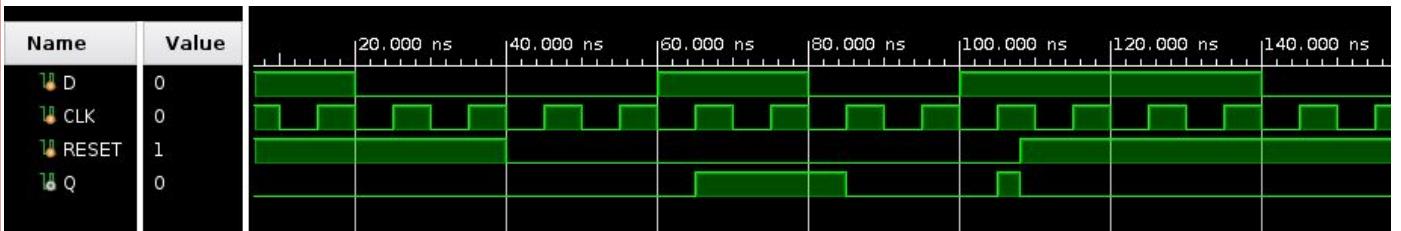
    always @ (posedge CLK)
    begin
        if (RESET==1'b1)
            Q <= 1'b0;
        else
            Q <= D;
    end
endmodule
```



# Flip Flop D (Simulando)

```
initial begin
    D = 1;
    #20;
    D = 0;
    #20;
    RESET=0;
    #20;
    D = 1;
    #20;
    D = 0;
    #20;
    D = 0;
    #20;
    D = 1;
    #8;
    RESET=1;
    #12;
    D = 1;
    #20;
    D = 0;
end
```

```
initial begin
    CLK=0;
    RESET=1;
    forever #5 CLK = !CLK;
end
endmodule
```



# Asignaciones

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

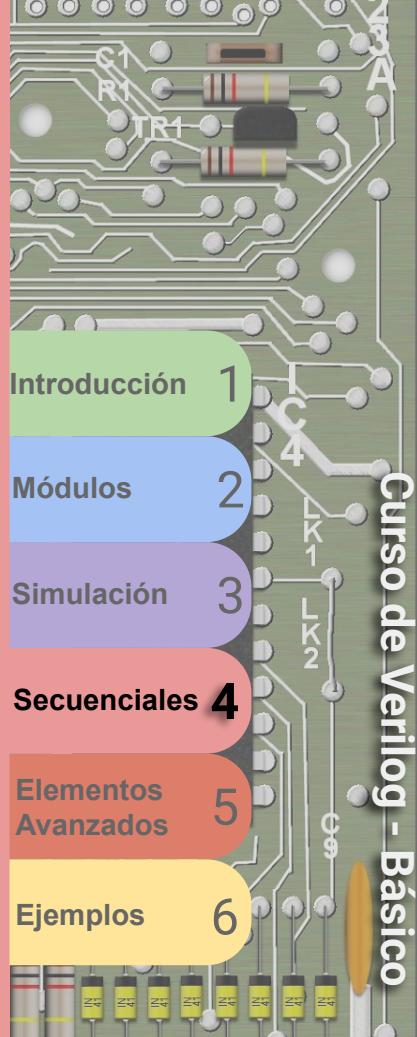
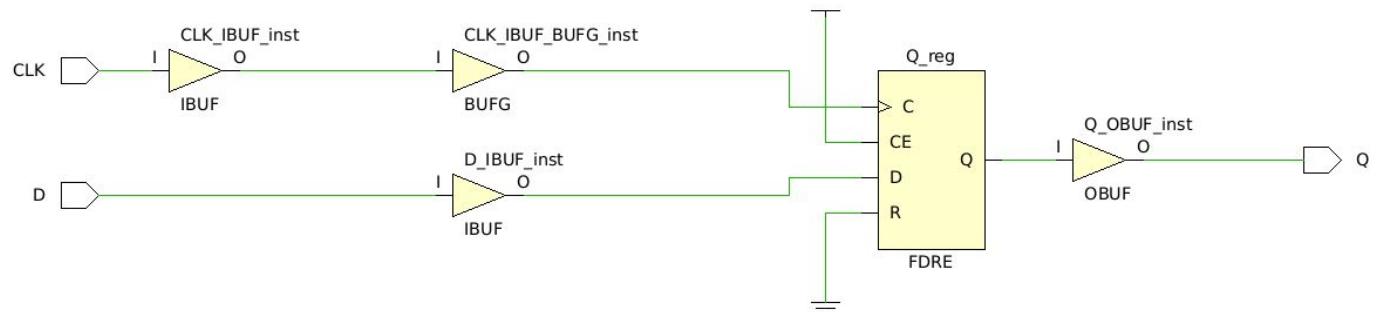
Ejemplos 6

# Asignación bloqueante

```
module AsigBlock(
    input D,
    input CLK,
    output reg Q
);

reg aux;
always @(posedge CLK)
begin
    aux = D;
    Q = aux;
end

endmodule
```

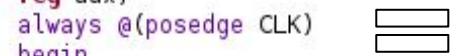


# Asignación NO bloqueante

```
module AsigNoBlock(
    input D,
    input CLK,
    output reg Q
);

reg aux;
always @ (posedge CLK)
begin
    aux <= D;
    Q <= aux;
end

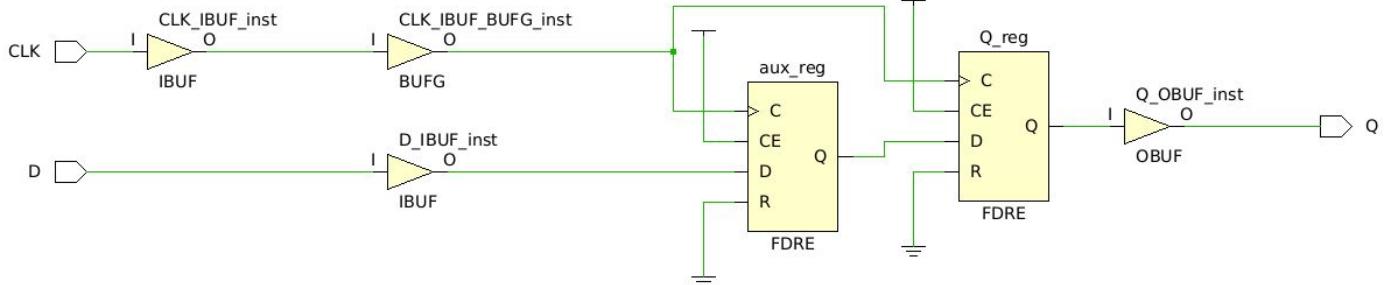
endmodule
```



```
module AsigNoBlock(
    input D,
    input CLK,
    output reg Q
);

reg aux;
always @ (posedge CLK)
begin
    Q <= aux;
    aux <= D;
end

endmodule
```



# Contadores

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

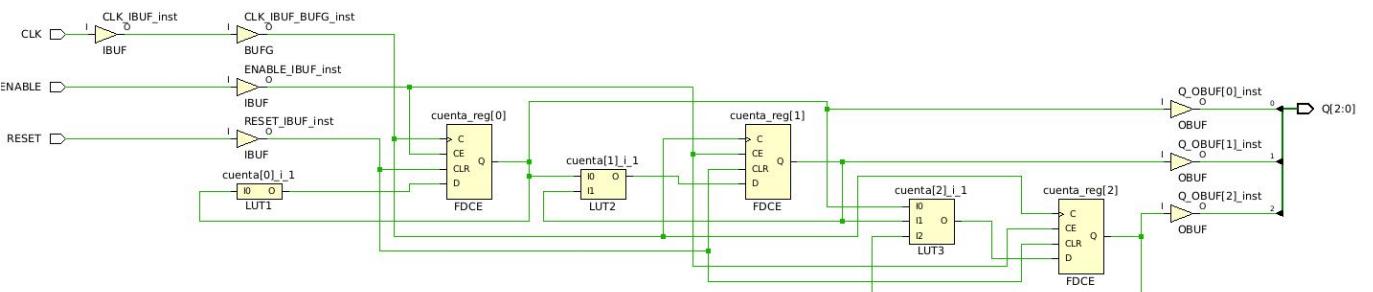
Elementos Avanzados 5

Ejemplos 6

# Contador 3 bits (Ascendente)

```
module ContadorASC(
    input CLK,
    input RESET,
    input ENABLE,
    output [2:0] Q
);
reg [2:0] cuenta;
assign Q = cuenta;

always @(posedge CLK or posedge RESET)
begin
    if (RESET)
        cuenta <= 3'd0;
    else
        begin
            if (ENABLE)
                cuenta <= cuenta + 3'd1;
        end
end
endmodule
```



Introducción

1

Módulos

2

Simulación

3

Secuenciales

4

Elementos Avanzados

5

Ejemplos

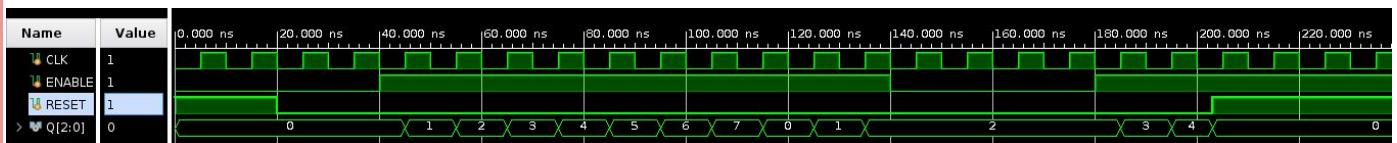
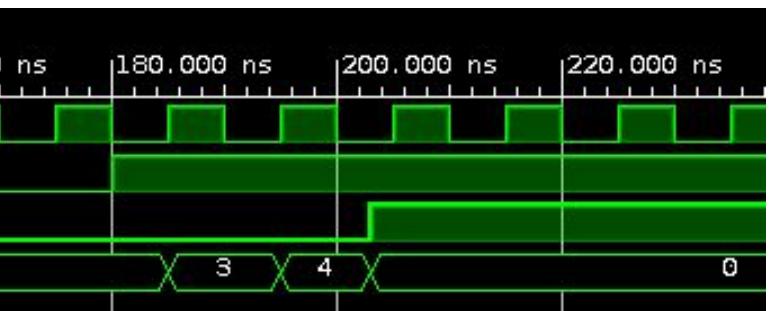
6

# Contador 3 bits (Simulación)

```
module ContadorASC_tb();
reg CLK,ENABLE,RESET;
wire [2:0] Q;

ContadorASC U0 (CLK,RESET,ENABLE,Q);
initial begin
    CLK=0;
    ENABLE=0;
    RESET=1;
    forever #5 CLK=~CLK;
end

initial begin
#20;
RESET=0;
#20;
ENABLE=1;
#100;
ENABLE=0;
#40;
ENABLE=1;
#23;
RESET=1;
end
endmodule
```



# Desplazamiento

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

## Shifter 4 bits

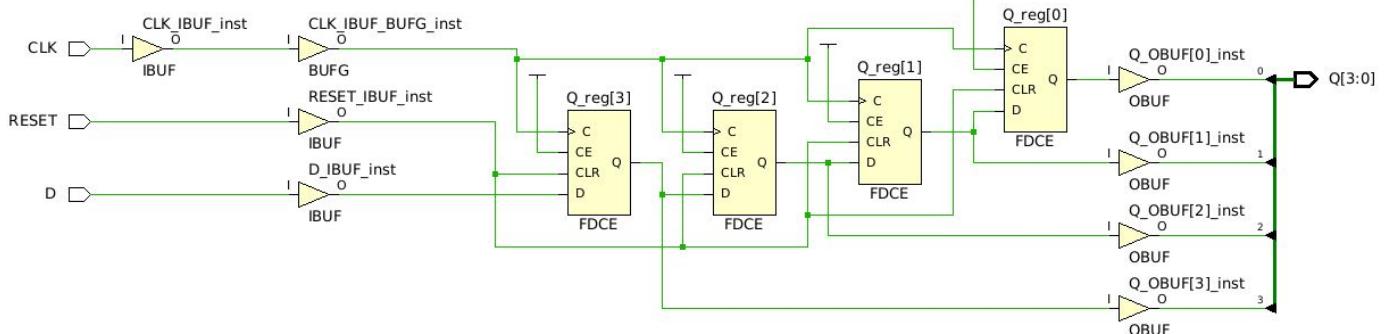
```
module ShiftRight(
    input D,
    input CLK,
    input RESET,
    output reg [3:0] Q
);

    always @(posedge CLK or posedge RESET)
    begin
        if (RESET)
            Q <= 0;
        else
            Q <= { D , Q[3:1]};
    end
endmodule
```

## Concatenación

Se agrupan cables  
utilizando { };

Ej: { a,b,c,d }; une los  
cables a,b,c y d en un bus  
donde a es el más  
significativo. Pueden  
utilizarse otros buses,  
Ej: { Carry, Suma[7:0] };  
Podemos repetir señales,  
Ej: { {8{Carry}} }, Suma[0] };



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

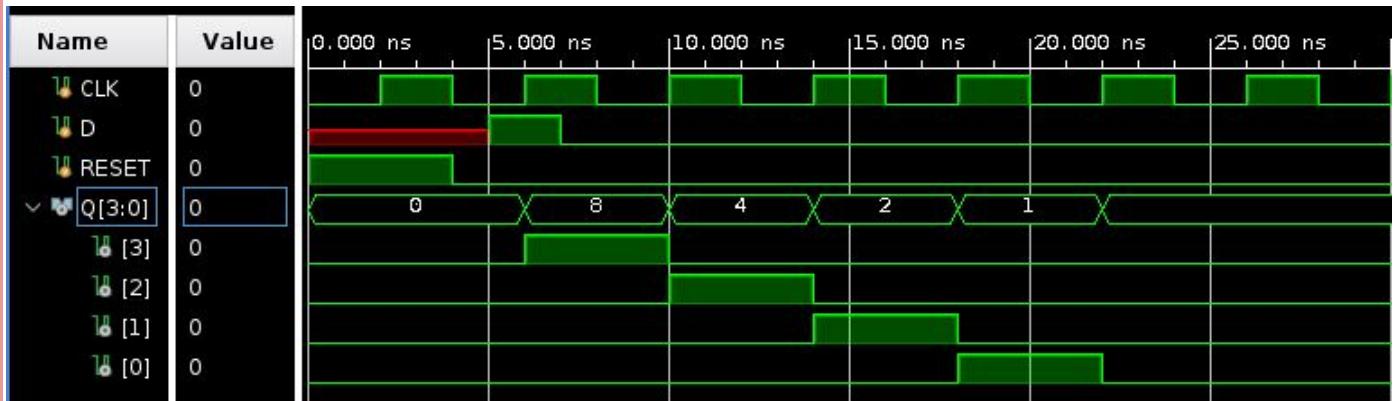
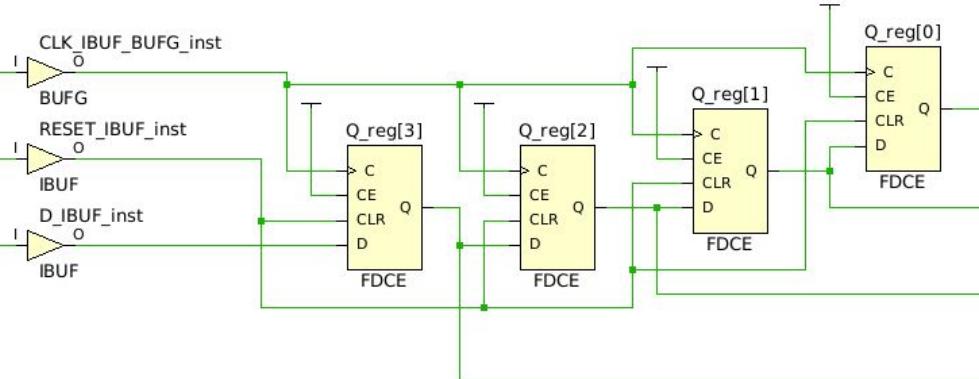
Ejemplos 6

# Shifter 4 bits (testbench)

```
module ShiftRight_tb();
reg CLK,D,RESET;
wire [3:0] Q;

ShiftRight U0 (D,CLK,RESET,Q);
initial begin
    CLK=0;
    RESET=1;
    forever #2 CLK=~CLK;
end

initial begin
    #4;
    RESET=0;
    #1;
    D=1;
    #2;
    D=0;
end
endmodule
```



# Diseño sincrónico

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

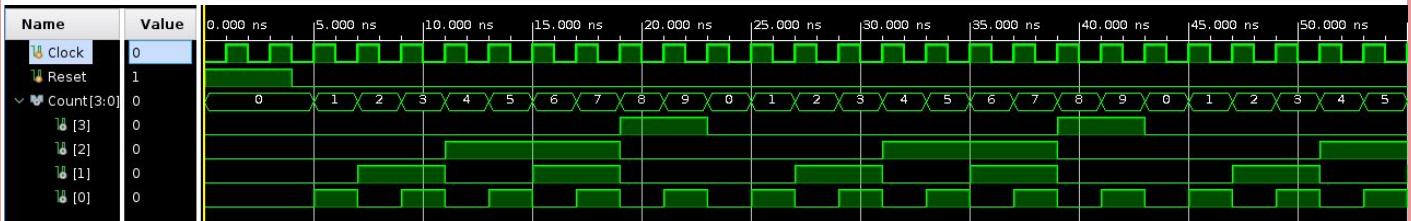
# Contador Módulo 10 (slow clock)

```
module Mod10Counter(
    input Clock,
    input Reset,
    output [3:0] Count
);
reg [3:0] counter;
assign Count=counter;
always @(posedge Clock or posedge Reset)
begin
    if (Reset | counter==4'd9)
        counter=4'd0;
    else
        counter = counter + 4'd1;
end
endmodule
```

```
module Mod10CounterTB( );
reg Clock,Reset;
wire [3:0] Count;
Mod10Counter U0(Clock,Reset,Count);

initial begin
Clock=0;
Reset=1;
forever #1 Clock=~Clock;
end

initial begin
#4;
Reset=0;
end
endmodule
```



Introducción 1

Módulos 2

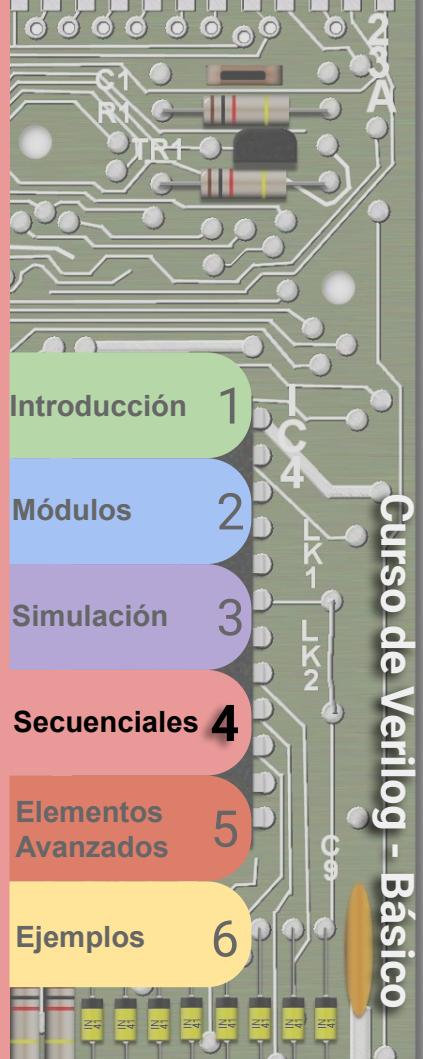
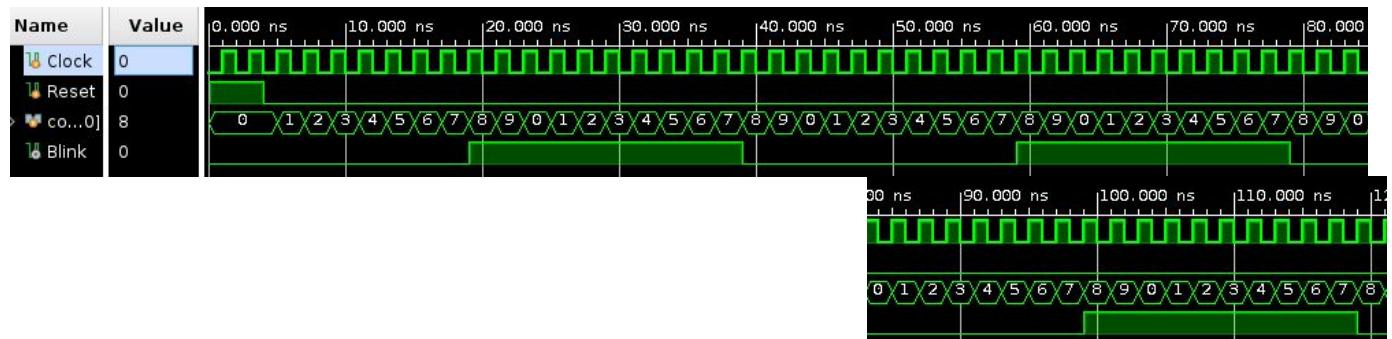
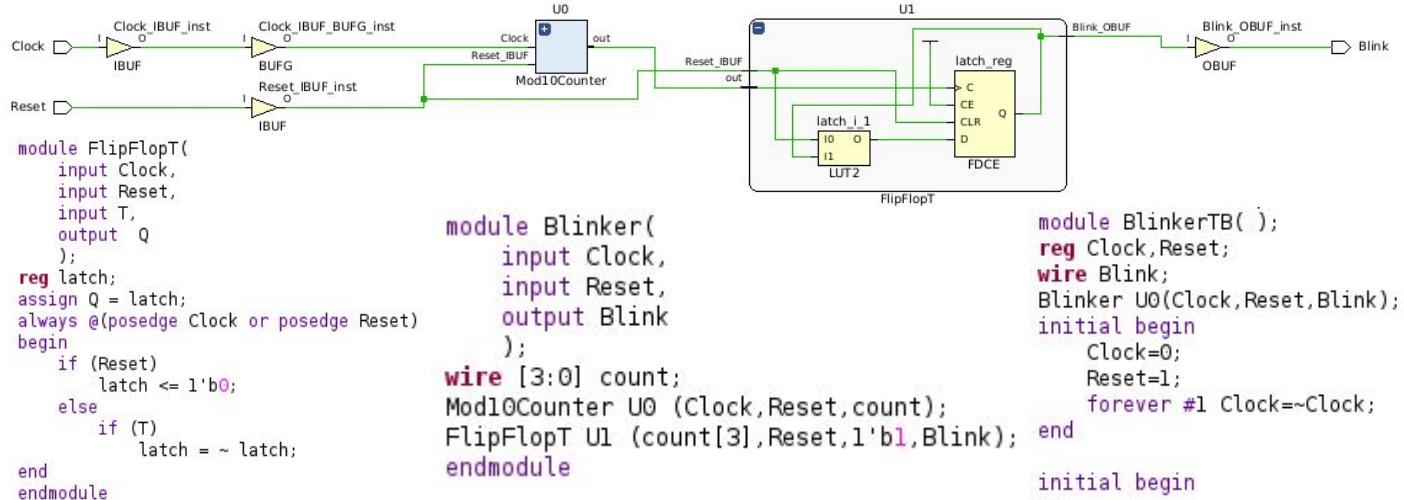
Simulación 3

Secuenciales 4

Elementos Avanzados 5

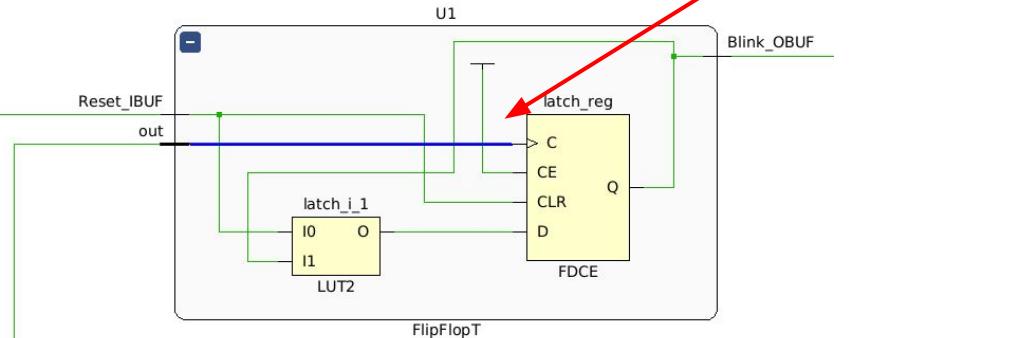
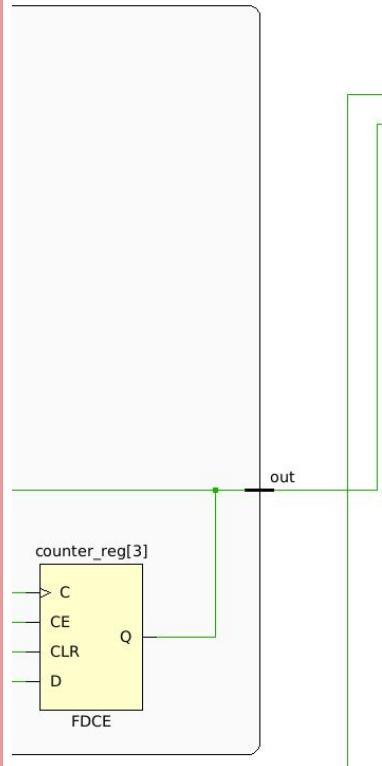
Ejemplos 6

# Contador Módulo 10 (slow clock) con FF T



# Problemas en la implementación

Clocks Lentos



Critical Warning The clock pin `latch_reg.C` is not reached by a timing clock

## Methodology Violations

WARNING! Critical violations of the methodology design rules detected. Critical violations may contribute to timing failures or cause functional issues in hardware.  
Run report\_methodology for more information.

OK

Rule Id	Severity	Description	Count
LUTAR-1	Warning	LUT drives async reset alert	1

Resets Asincrónicos

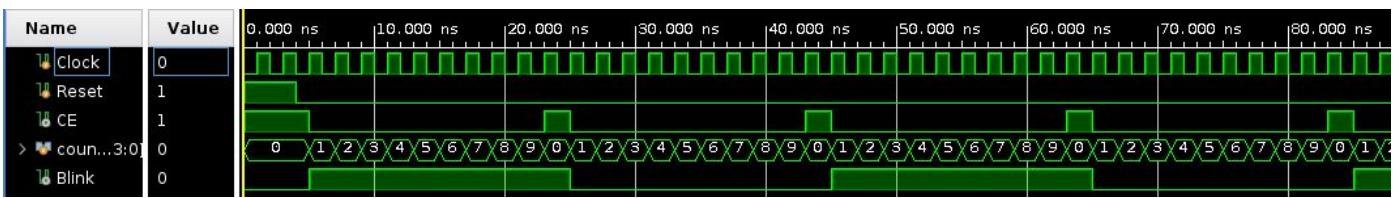
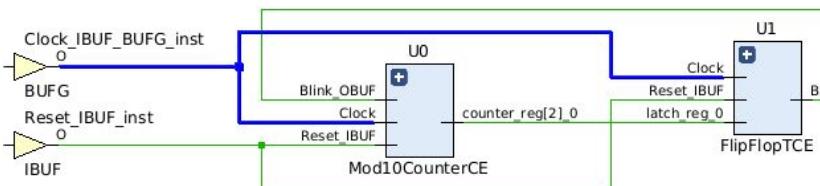


# Contador Módulo 10 y FlipFlop T con CE

```
module Mod10CounterCE(
    input Clock,
    input Reset,
    output Pulse
);
reg [3:0] counter;
assign Pulse = ~{counter};
always @(posedge Clock or posedge Reset)
begin
    if (Reset | counter == 4'd9)
        counter = 4'd0;
    else
        counter = counter + 4'd1;
end
endmodule

module Blinker(
    input Clock,
    input Reset,
    output Blink
);
wire CE;
Mod10CounterCE U0 (Clock,Reset,CE);
FlipFlopTCE U1 (Clock,Reset,CE,l'b1,Blink);
endmodule
```

```
| module FlipFlopTCE(
|     input Clock,
|     input Reset,
|     input CE,
|     input T,
|     output Q
| );
| reg latch;
| assign Q = latch;
| always @(posedge Clock or posedge Reset)
| begin
|     if (Reset)
|         latch = 1'b0;
|     else
|         if (CE & T)
|             latch = ~ latch;
| end
| endmodule
```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Indicando el clock de referencia

## Identify and Recommend Missing Timing Constraints

The Timing Constraints Wizard guides you through creating timing constraints per Xilinx design methodology. It analyzes your design for missing timing constraints and makes recommendations. You need to review and understand all of the recommendations to ensure they are appropriate for your design.

### Clocks:

- Primary Clocks
- Generated Clocks
- Forwarded Clocks
- External Feedback Delays

### Input and Output Ports:

- Input Delays
- Output Delays
- Combinational Delays

### Clock Domain Crossings:

- Physically Exclusive Clock Groups
- Logically Exclusive Clock Groups with No Interaction
- Logically Exclusive Clock Groups with Interaction
- Asynchronous Clock Domain Crossings

Clicking 'Next' on a page applies the constraints to the design in memory, so that missing constraints on subsequent pages can be identified. Each page may require considerable runtime to discover missing constraints.

The Clock Networks report is available on every page to help you review the constraints. Schematics and timing path reports are available on the Asynchronous Clock Domain Crossings page.

To leave the Wizard and automatically save the new constraints to the target XDC file, click Finish. To discard the new constraints click Cancel.

## Recommended Constraints

Object	Name	Frequency (MHz)	Period (ns)	Rise At (ns)	Fall At (ns)	Jitter (ns)
<input checked="" type="checkbox"/>	Object					
<input checked="" type="checkbox"/>	Clock	Clock	100.000	10.000	0.000	5.000

```
create_clock -period 10.000 -name Clock -waveform {0.000 5.000} [get_ports Clock]
```

## Open Synthesized Design

### Constraints Wizard

#### Implementation | Summary | Route Status

Status: Complete  
Messages: No errors or warnings  
Part: xc7a35tcsg324-1  
Strategy: Vivado Implementation Defaults  
Report Strategy: Vivado Implementation Default Reports  
Incremental implementation: None



# Parámetros

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

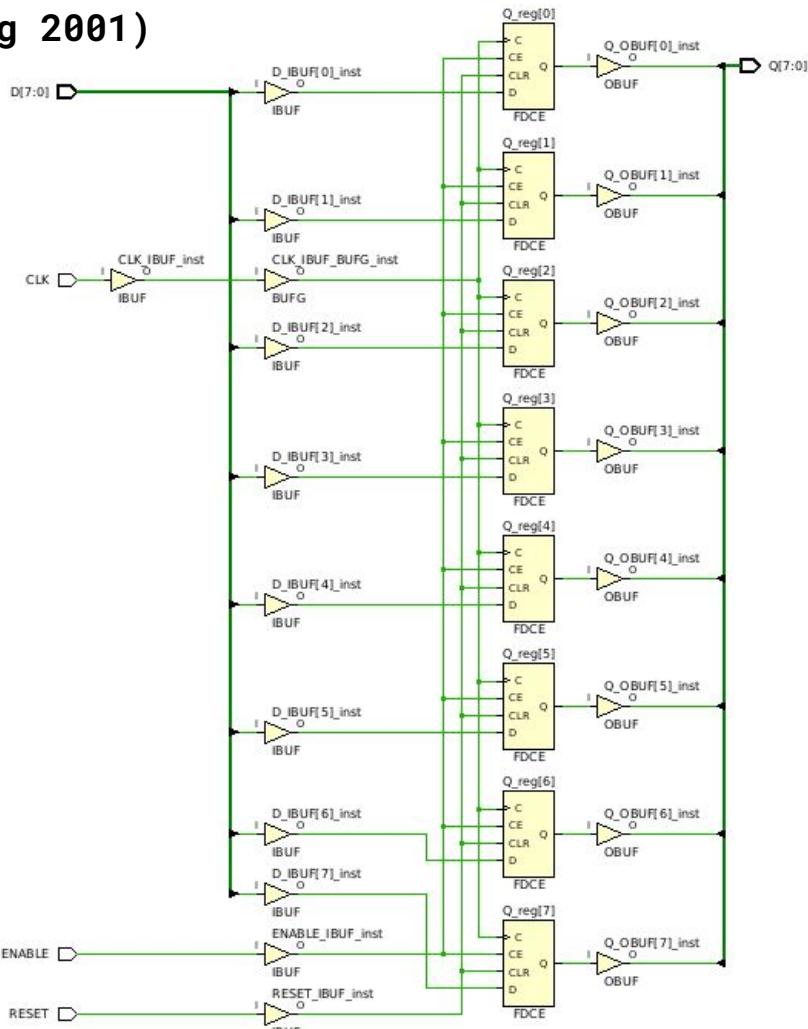
Ejemplos 6



# Definir parámetro (Verilog 2001)

```
module LatchRegister
#(parameter SIZE=8)
(
    input [SIZE-1:0] D,
    input CLK,
    input RESET,
    input ENABLE,
    output reg [SIZE-1:0] Q
);

always@(posedge CLK or posedge RESET)
begin
    if (RESET)
        Q <= 0;
    else
        if (ENABLE)
            Q <= D;
end
endmodule
```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Cambiar parámetro en instancia (Verilog 2001)

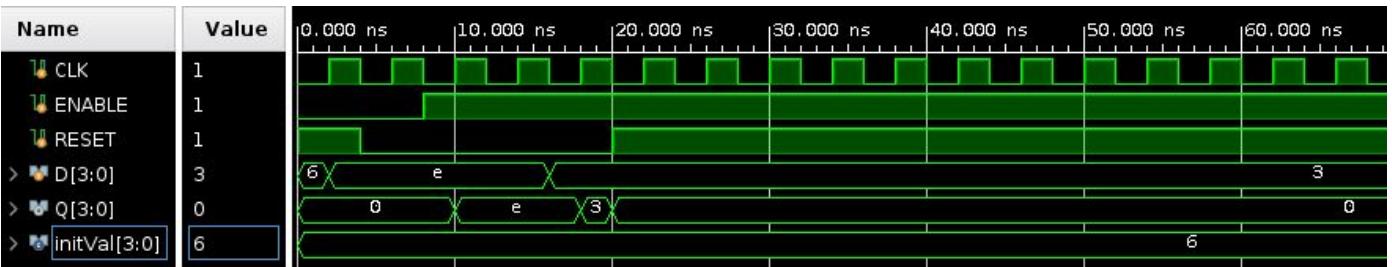
```
module LatchRegister_tb
#(parameter [3:0] initVal = 4'b0110)
();

reg CLK, ENABLE, RESET;
reg [3:0] D;
wire [3:0] Q;

LatchRegister #(SIZE(4)) U0
    (D,CLK,RESET,ENABLE,Q);

initial begin
    CLK=0;
    ENABLE=0;
    RESET=1;
    D=initVal;
    forever #2 CLK=~CLK;
end

initial begin
    #2;
    D=4'b1110;
    #2;
    RESET=0;
    #4;
    ENABLE=1;
    #8;
    D=4'b0011;
    #4;
    RESET=1;
end
endmodule
```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

Curso de Verilog - Básico

# Instanciar elementos internos



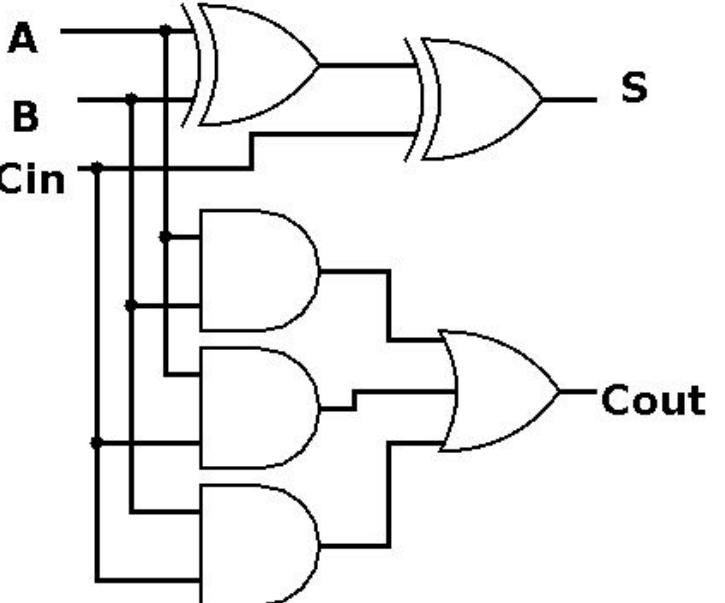
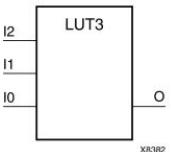
## Full Adder (LUT)

entrada	salida			
C <sub>IN</sub>	A	B	C <sub>OUT</sub>	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$C_{OUT} = E8 \quad S = 96$$

LUT3

Primitive: 3-Bit Look-Up Table with General Output



[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_7/7series\\_hdl.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf)

```
LUT3 #(
    .INIT(8'h00) // Specify LUT Contents
) LUT3_inst (
    .O(O), // LUT general output
    .I0(I0), // LUT input
    .I1(I1), // LUT input
    .I2(I2) // LUT input
);
```

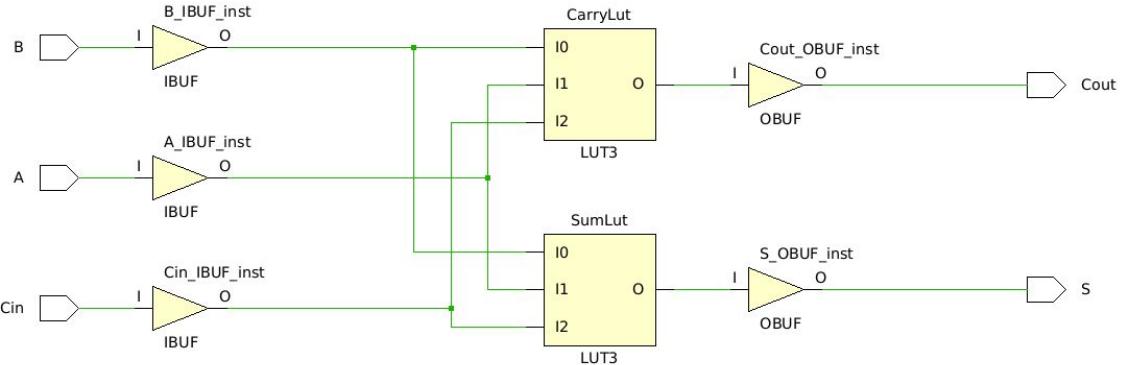


# Full Adder (LUT)

```
module FullAdderLut(
    input A,
    input B,
    input Cin,
    output S,
    output Cout
);

LUT3 #(INIT(8'hE8)) CarryLut (
    .O(Cout),
    .I0(B),
    .I1(A),
    .I2(Cin) );

LUT3 #(INIT(8'h96)) SumLut (
    .O(S),
    .I0(B),
    .I1(A),
    .I2(Cin) );
endmodule
```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# For Generate

Introducción 1

Módulos 2

Simulación 3

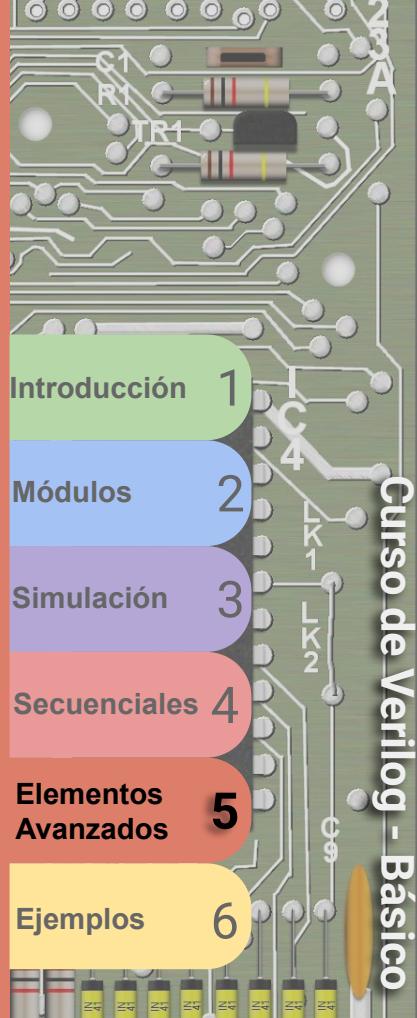
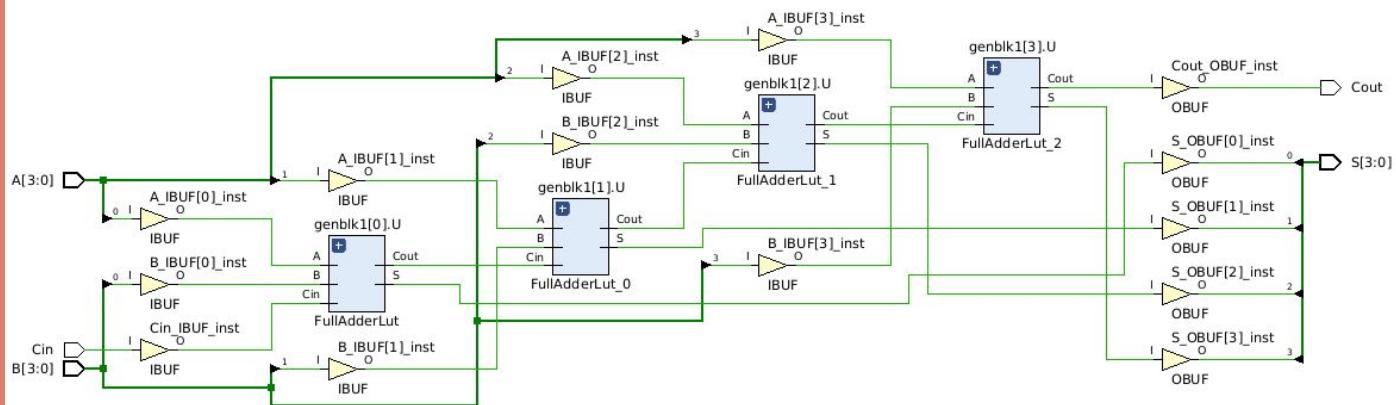
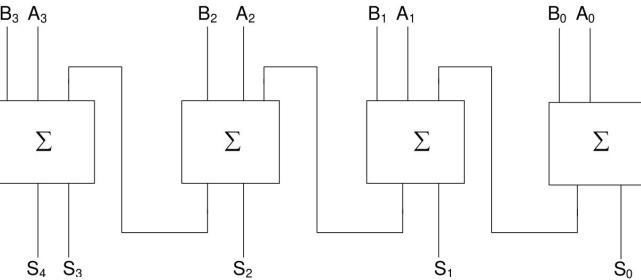
Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# Hex Adder (generate for)

```
module HexAdder(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] S,
    output Cout
);
wire [4:0] carry;
genvar i;
assign carry[0] = Cin;
assign Cout = carry[4];
generate
    for (i=0;i<4;i=i+1) begin
        FullAdderLut U (A[i],B[i],carry[i],S[i],carry[i+1]);
    end
endgenerate
endmodule
```



# IP Cores

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



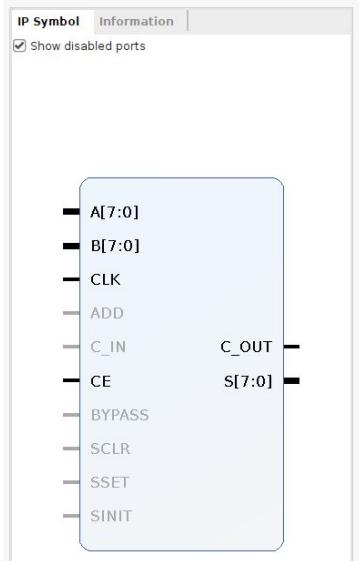
# IP Cores

## IP INTEGRATOR

Create Block Design

Open Block Design

Generate Block Design



Sumador (Sumador.bd) (1)

Sumador\_c\_addsub\_0\_0 (Sumador\_c\_addsub\_0\_0.xci)

### Create Block Design

Please specify name of block design.

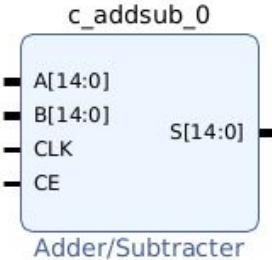
Design name: Sumador

Directory: <Local to Project>

Specify source set: Design Sources

OK Cancel

This design is empty. Press the + button to add IP.



Component Name: c\_addsub\_0

Basic Control

Implement using: DSP48

S = A +/ B

Input Type: Unsigned

Input Width: 8 [1,47]

Add Mode: Add

Output Width: 8 [8 - 9]

Latency Configuration: Automatic

Latency: 2 [0 - 2]

Constant Input

Constant Value (Bin): 00000000

IP INTEGRATOR

Create Block Design

Open Block Design

Generate Block Design

Introducción 1

Módulos 2

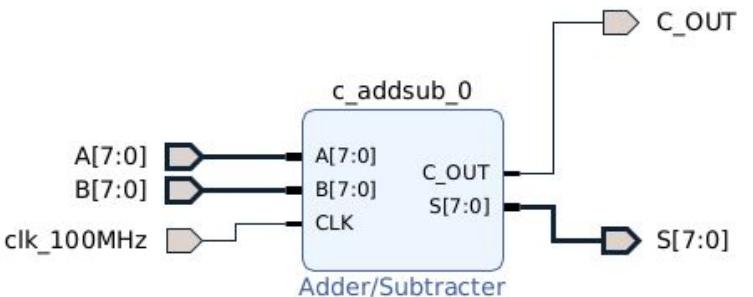
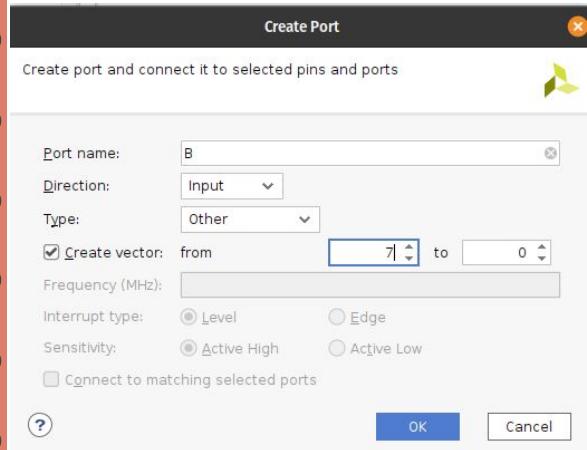
Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# IP Cores (Create port)



[https://www.xilinx.com/support/documentation/ip\\_documentation/addsub/v12\\_0/pg120-c-addsub.pdf](https://www.xilinx.com/support/documentation/ip_documentation/addsub/v12_0/pg120-c-addsub.pdf)



# IP Cores

Generate Output Products X

The following output products will be generated.

**Preview**

- Sumador.bd (OOC per IP)
  - Synthesis
  - Implementation
  - Simulation
  - Hw\_Handoff

**Synthesis Options**

- Global
- Out of context per IP
- Out of context per Block Design

**Run Settings**

- On local host: Number of jobs: 8
- On remote hosts Configure Hosts
- Launch runs on Cluster lsf

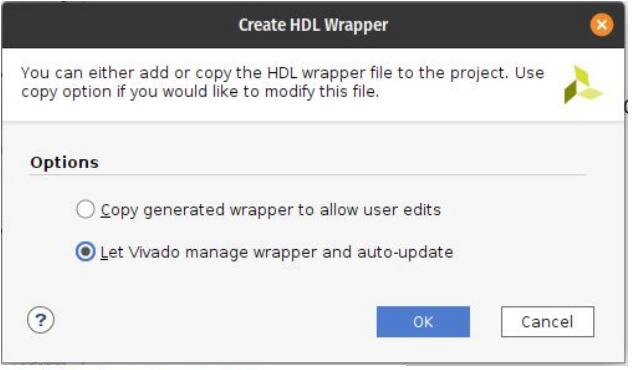
Apply Generate Cancel

● Sumador\_wrapper (Sumador\_wrapper.v) (1)

△ Sumador\_i : Sumador (Sumador.bd) (1)

● Sumador (Sumador.v) (1)

> ■ c\_addsub\_0 : Sumador\_c\_addsub\_0\_0 (Sumador\_c\_addsub\_0\_0.v) (1)

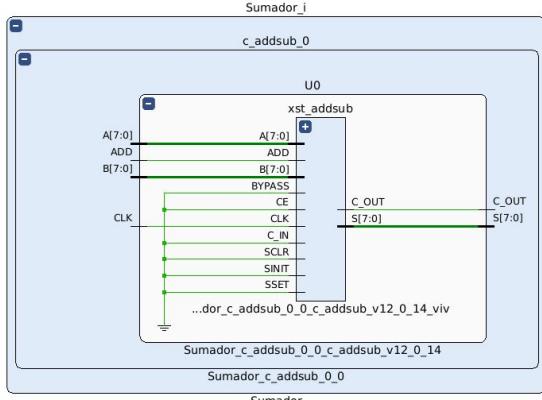


```
module Sumador_wrapper
(A,
 B,
 C_OUT,
 S,
 clk_100MHz);
input [7:0]A;
input [7:0]B;
output C_OUT;
output [7:0]S;
input clk_100MHz;

wire [7:0]A;
wire [7:0]B;
wire C_OUT;
wire [7:0]S;
wire clk_100MHz;
```

```
Sumador_i
c_addsub_0
U0
xst_addsub
A[7:0]
ADD
B[7:0]
BYPASS
CE
CLK
C_IN
SCLR
SINIT
SSET
...dor_c_addsub_0_0_c_addsub_v12_0_14_viv
Sumador_c_addsub_0_0
Sumador
```

```
Sumador Sumador_i
(A(A),
.B(B),
.C_OUT(C_OUT),
.S(S),
.clk_100MHz(clk_100MHz));
endmodule
```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

Curso de Verilog - Básico

# Sumador Simple

Introducción 1

Módulos 2

Simulación 3

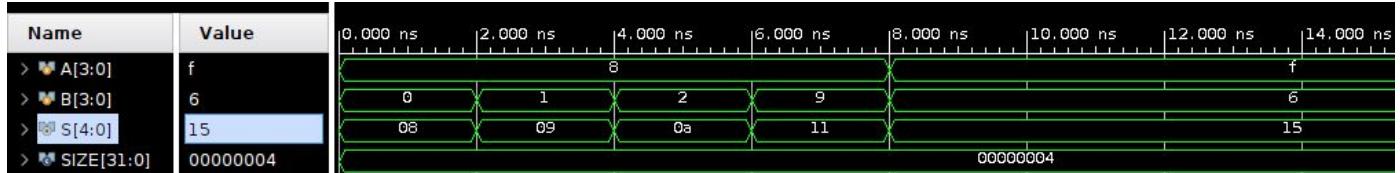
Secuenciales 4

Elementos  
Avanzados 5

Ejemplos 6

# Sumador Simple

```
module SumadorSimple
#(parameter SIZE=8)
(
    input [SIZE-1:0] A,
    input [SIZE-1:0] B,
    output[SIZE :0] S
);
wire [SIZE:0] Amas,Bmas;
assign Amas = {0,A};
assign Bmas = {0,B};
assign S = Amas + Bmas;
endmodule
```



```
module SumadorSimple_tb
#(parameter SIZE=4)
();
reg [SIZE-1:0] A,B;
wire [SIZE :0] S;

SumadorSimple #(.SIZE(SIZE)) U0 (A,B,S);

initial begin
A=4'h8;
B=4'h0;
#2;
A=4'h8;
B=4'h1;
#2;
A=4'h8;
B=4'h2;
#2;
A=4'h8;
B=4'h9;
#2;
A=4'hF;
B=4'h6;
#2;
end
endmodule
```

Introducción 1

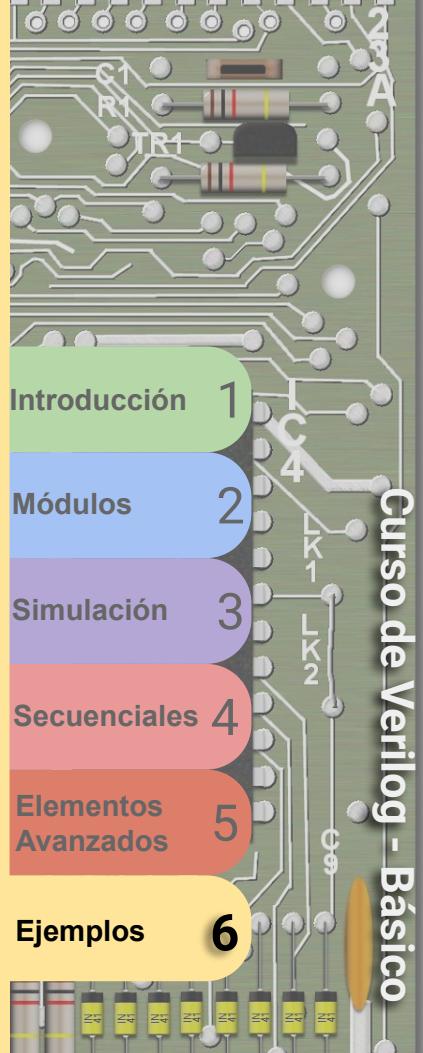
Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



# FSM Mealy

Introducción 1

Módulos 2

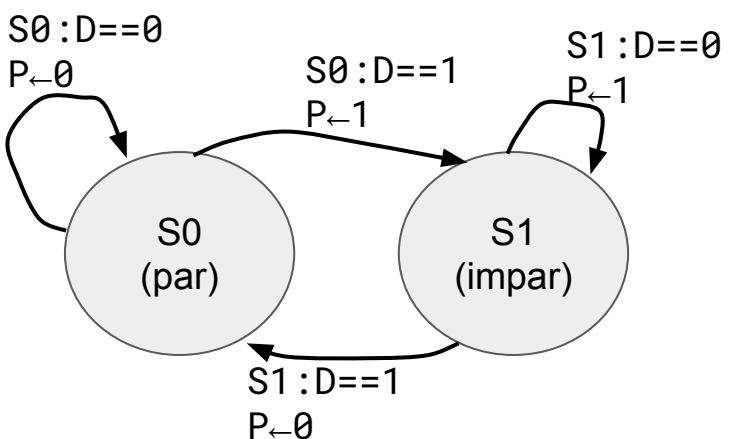
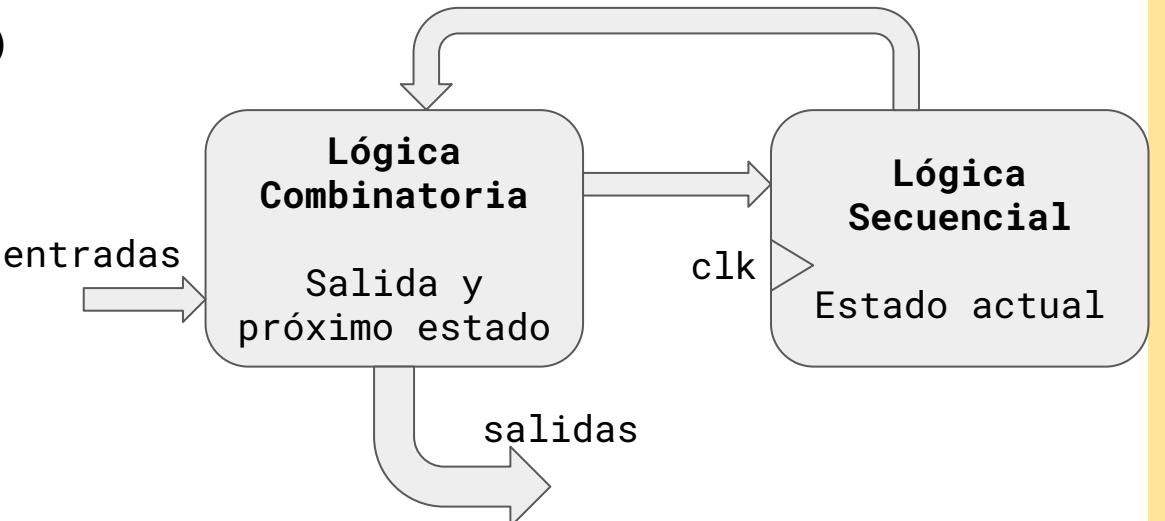
Simulación 3

Secuenciales 4

Elementos  
Avanzados 5

Ejemplos 6

# FSM (Mealy)



Estado	D	P	Nuevo estado
S0	0	0	S0
S0	1	1	S1
S1	0	1	S1
S1	1	0	S0

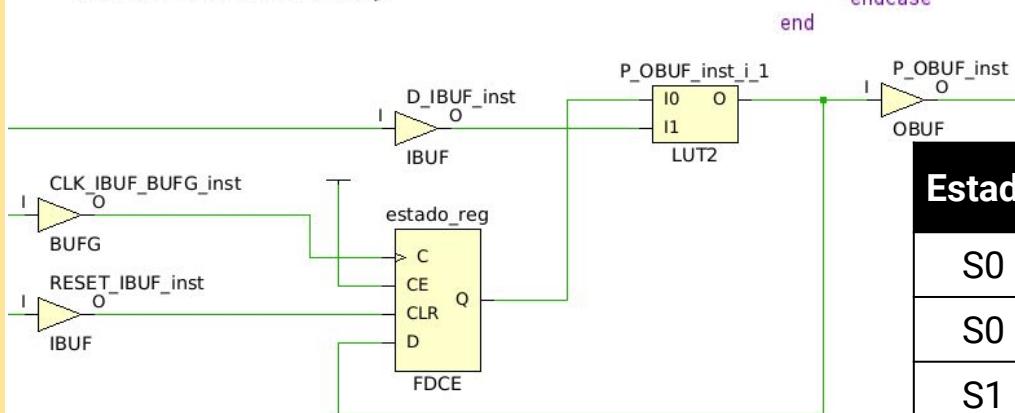


# FSM (Mealy)

```
module MealyFSM(
    input CLK,
    input RESET,
    input D,
    output reg P
);
//Definimos valor para guardar el estado
parameter S0=0, S1=1;

reg estado,nuevoEstado;

always @(posedge CLK or posedge RESET)
if (RESET)
    estado <= S0;
else
    estado <= nuevoEstado;
```



```
always @ (estado or D)
begin
    P=1'b0;
    case (estado)
        S0: if (D)
            begin
                P=1; nuevoEstado=S1;
            end
        else
            //P=0 al principio del always.
            nuevoEstado=S0;
        S1: if (D)
            //P=0 al principio del always.
            nuevoEstado=S0;
        else
            begin
                P=1; nuevoEstado=S1;
            end
        default:
            nuevoEstado=S0;
    endcase
end
```

Estado	D	P <sub>N+1</sub>	Nuevo estado
S0	0	0	S0
S0	1	1	S1
S1	0	1	S1
S1	1	0	S0

Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# FSM Moore

Introducción 1

Módulos 2

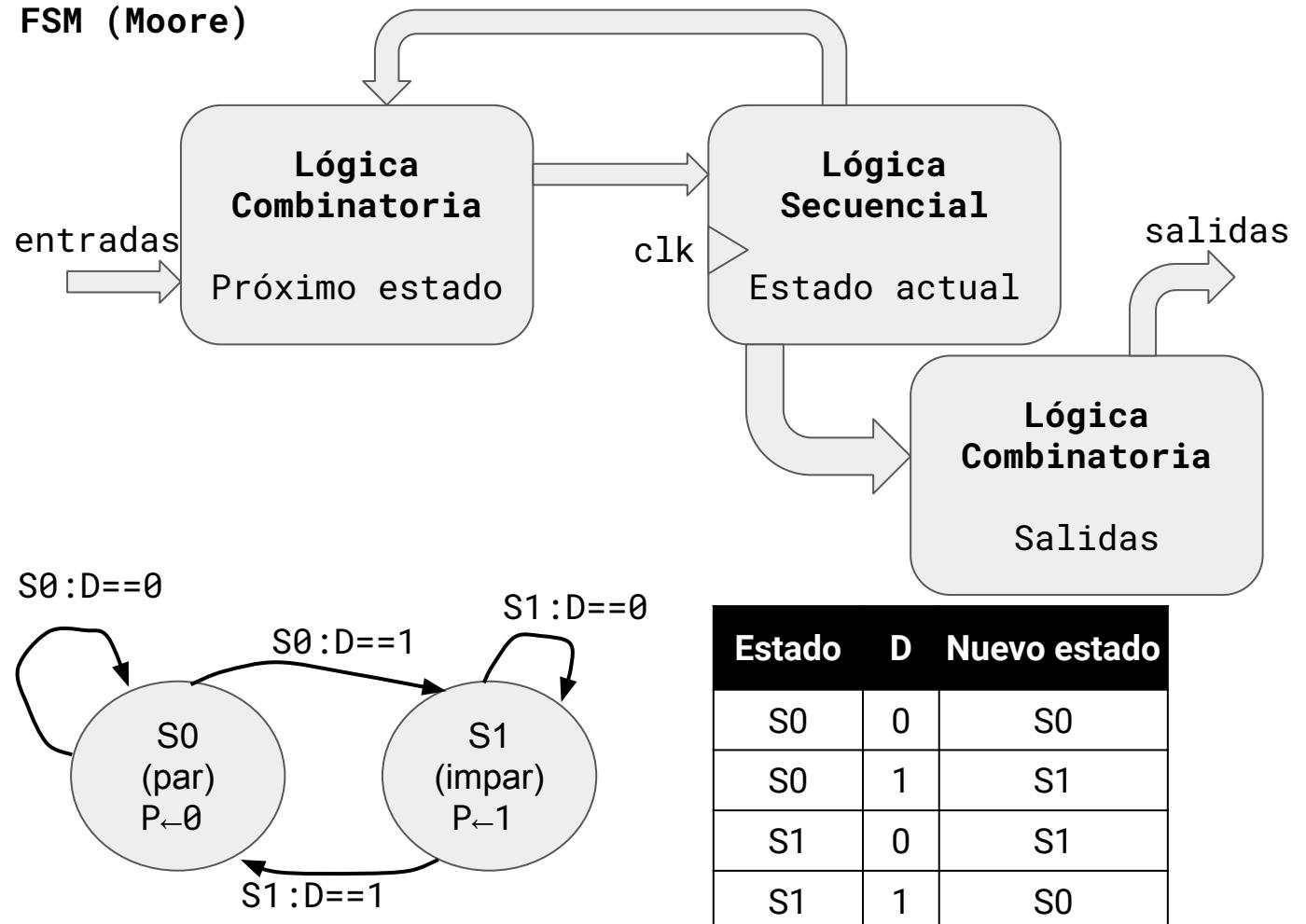
Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

## FSM (Moore)



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# FSM (Moore)

```

module MooreFSM(
    input CLK,
    input RESET,
    input D,
    output reg P
);
//Definimos valor para guardar el estado
parameter S0=0, S1=1;
reg estado,nuevoEstado;

always @(posedge CLK or posedge RESET)
if (RESET)
    estado <= S0;
else
    estado <= nuevoEstado;

```

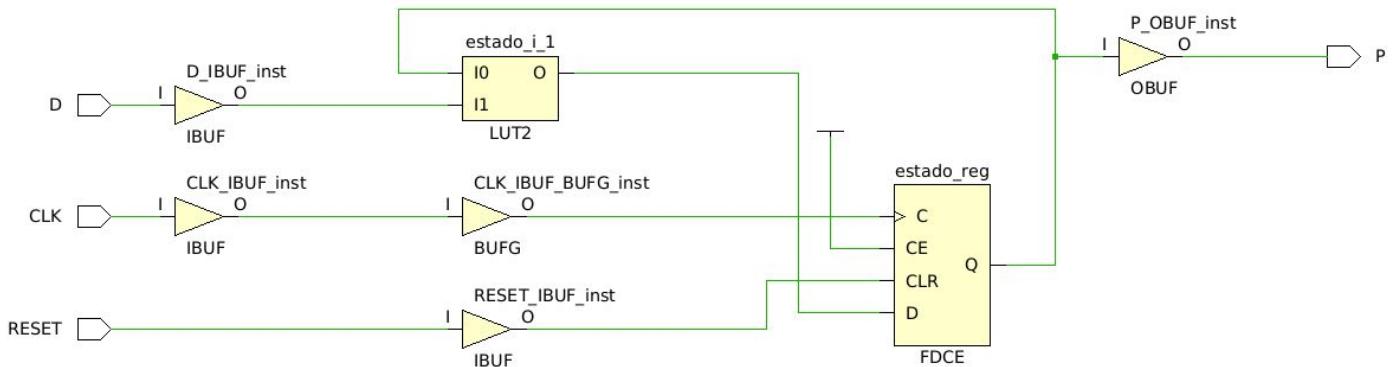
```

    always @ (estado)
    begin
        case (estado)
            S0: P=0;
            S1: P=1;
        endcase
    end

    always @ (estado or D)
    begin
        nuevoEstado=S0;
        case (estado)
            S0: if(D)
                nuevoEstado=S1;
            S1: if(!D)
                nuevoEstado=S1;
        endcase
    end
endmodule

```

Estado	D	Nuevo estado
S0	0	S0
S0	1	S1
S1	0	S1
S1	1	S0



# Interface VGA

Introducción 1

Módulos 2

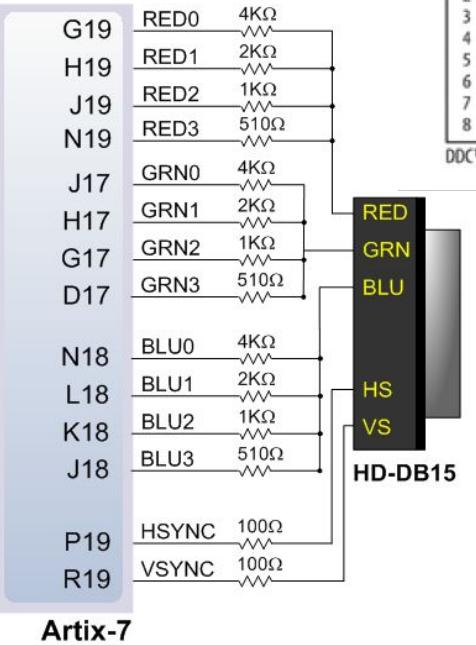
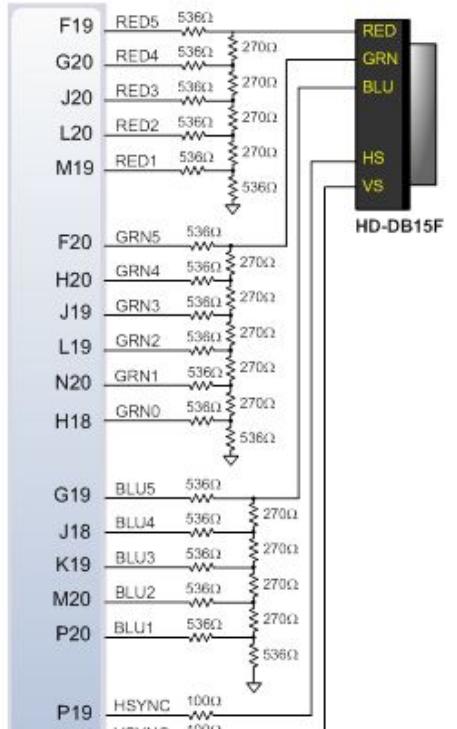
Simulación 3

Secuenciales 4

Elementos  
Avanzados 5

Ejemplos 6

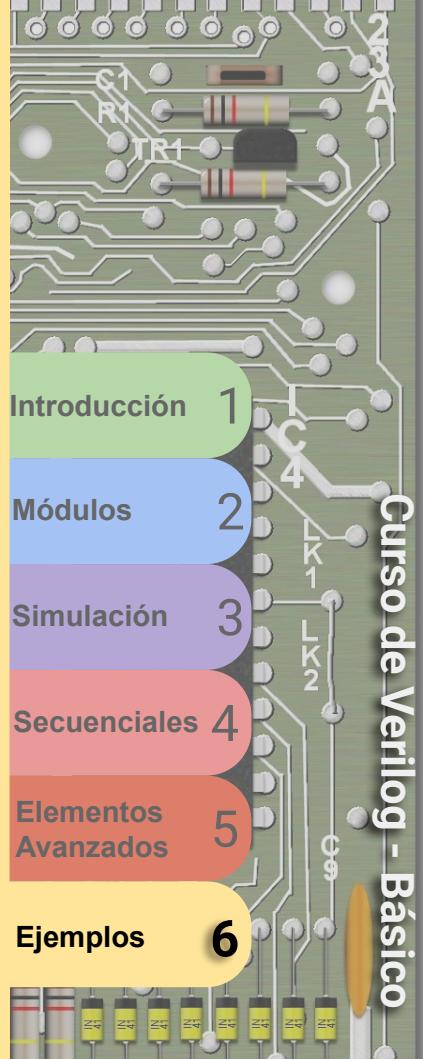
RGB : 0,7V pp  
Hs,Vs: TTL



Pin	Signal	Pin	Signal
1	Red	9	DDC +5V
2	Green	10	Sync Return
3	Blue	11	ID Bit 0
4		12	ID Bit 1
5	Ground	13	Horizontal Sync
6	Red Shield	14	Vertical Sync
7	Green Shield	15	ID Bit 3 or Data Clock
8	Blue Shield		

DDC™ of VESA (Video Electronics Standard Association)

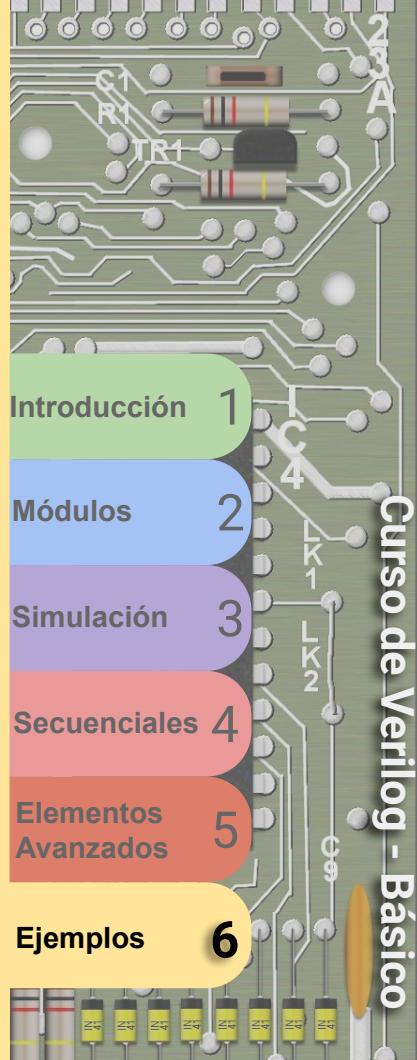
DDC es una versión simplificada de I2C.

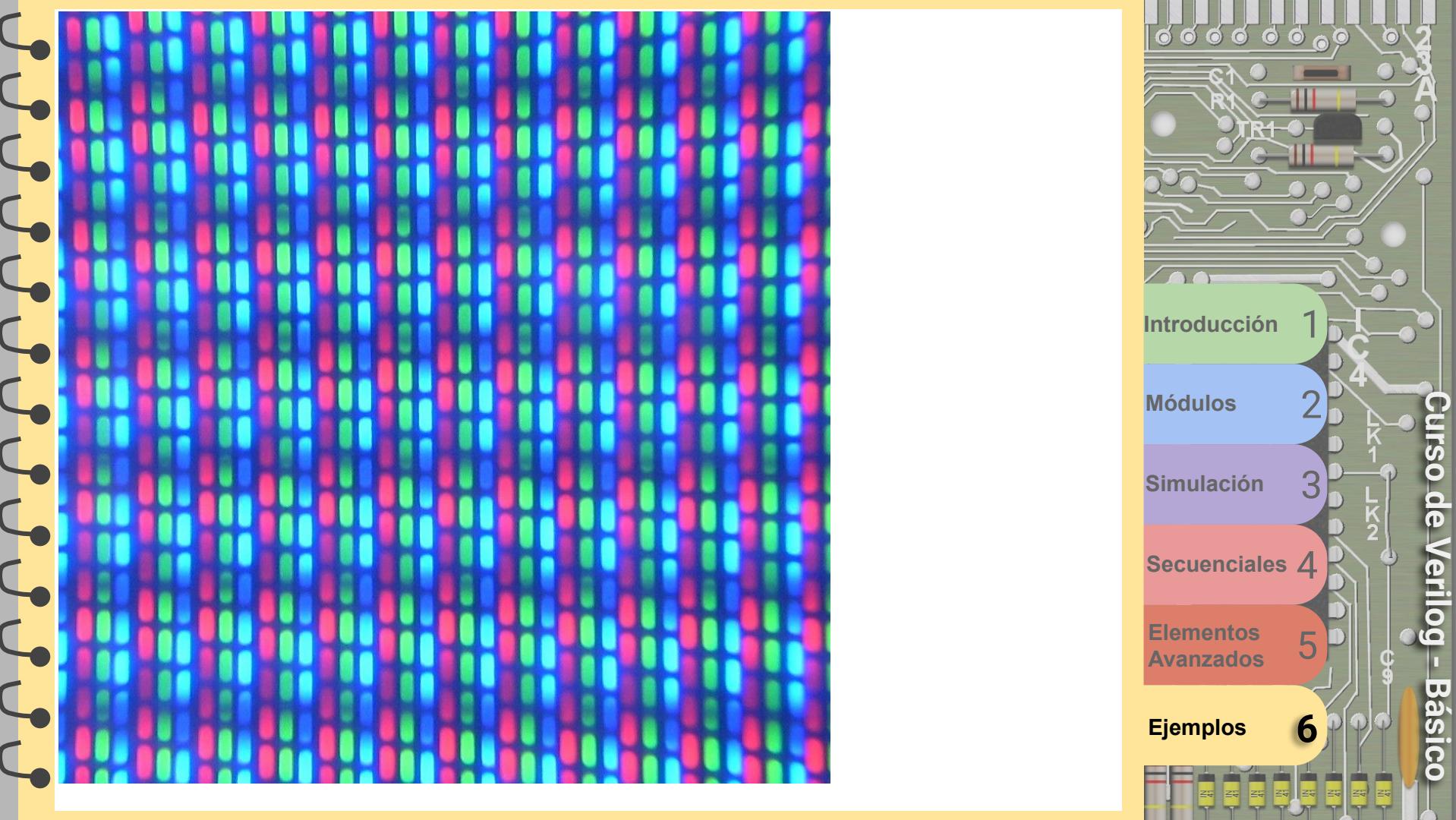


```

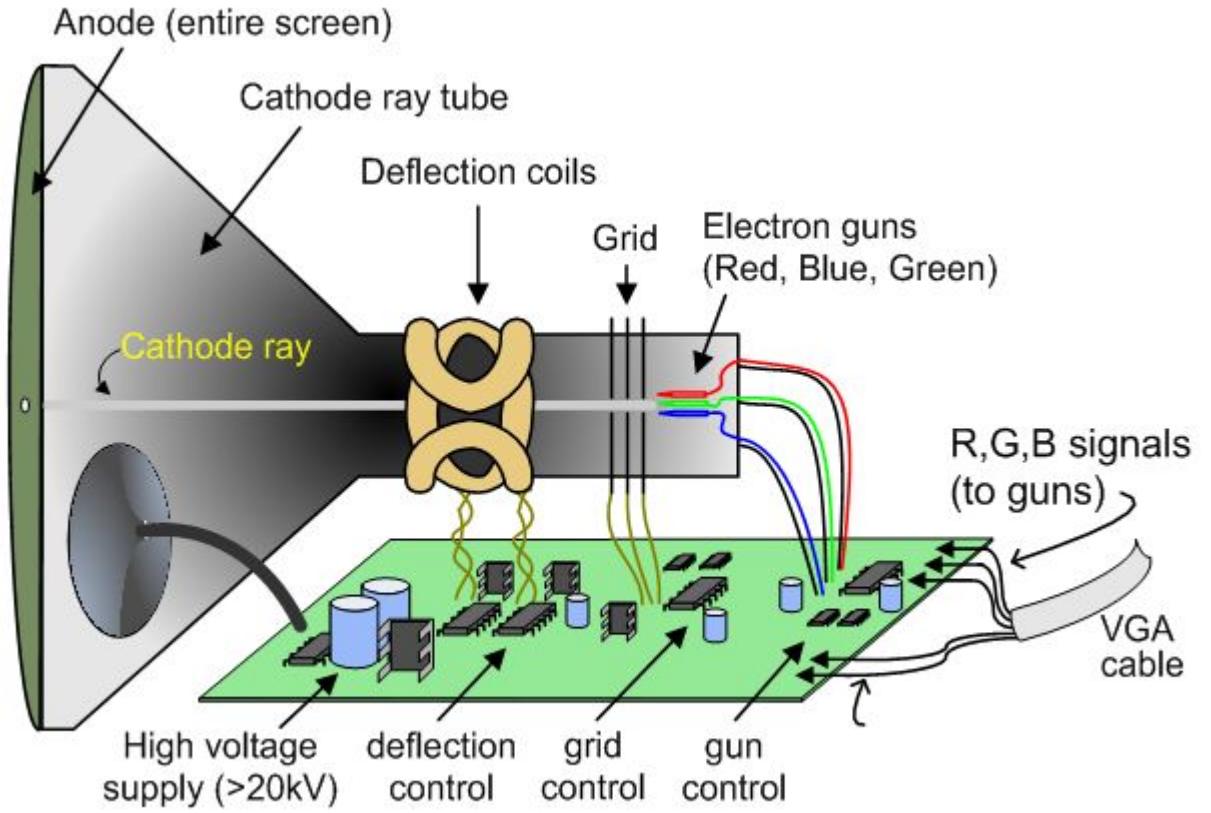
Display 1
I2C bus: /dev/i2c-5
  DRM connector: card0-HDMI-A-1
  Driver: amdgpu
  I2C address 0x50 (EDID) responsive: true
  Is eDP device: false
  Is LVDS device: false
  /sys/bus/i2c/devices/i2c-5/name AMDGPU DM i2c hw bus 3
  PCI device path: /sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0/i2c
EDID synopsis:
  Mfg id: SAM - Samsung Electric Company
  Model: U32H75x
  Product code: 3586 (0x0e02)
  Serial number: HT0J900304
  Binary serial number: 810634571 (0x30514d4b)
  Manufacture year: 2017, Week: 37
  EDID version: 1.3
  Extra descriptor:
  Video input definition: 0x80 - Digital Input
  Supported features:
    DPMS active-off
    Digital display type: RGB 4:4:4 + YCrCb 4:4:4
    Standard sRGB color space: False
  White x,y: 0.312, 0.329
  Red x,y: 0.634, 0.341
  Green x,y: 0.312, 0.636
  Blue x,y: 0.158, 0.062
  Extension blocks: 1
EDID source: I2C
EDID hex dump:
      +0      +4      +8      +c      0  4  8  c
+0000  00 ff ff ff ff ff 00 4c 2d 02 0e 4b 4d 51 30  .....L-..KMQ0
+0010  25 1b 01 03 80 46 27 78 2a 5f b1 a2 57 4f a2 28 %....F'x*_..W0.(
+0020  0f 50 54 bf ef 80 71 4f 81 00 81 c0 81 80 95 00 .PT...q0.....
+0030  a9 c0 b3 00 01 01 04 74 00 30 f2 70 5a 80 b0 58 .....t.0.pZ..X
+0040  8a 00 b9 88 21 00 00 1e 00 00 00 fd 00 18 4b 1e .....!.....K.
+0050  5a 1e 00 0a 20 20 20 20 20 00 00 00 fc 00 55 Z.....U
+0060  33 32 48 37 35 78 0a 20 20 20 20 20 00 00 00 ff 32H75x. .....
+0070  00 48 54 4f 4a 39 30 30 33 30 34 0a 20 20 01 67 .HT0J900304. .g
VCP version: 2.2
Controller mfg: Mstar
Firmware version: 0.1
Monitor returns DDC Null Response for unsupported features: false

```





- Curso de Verilog - Básico
- 1 Introducción
  - 2 Módulos
  - 3 Simulación
  - 4 Secuenciales
  - 5 Elementos Avanzados
  - 6 Ejemplos



Introducción 1

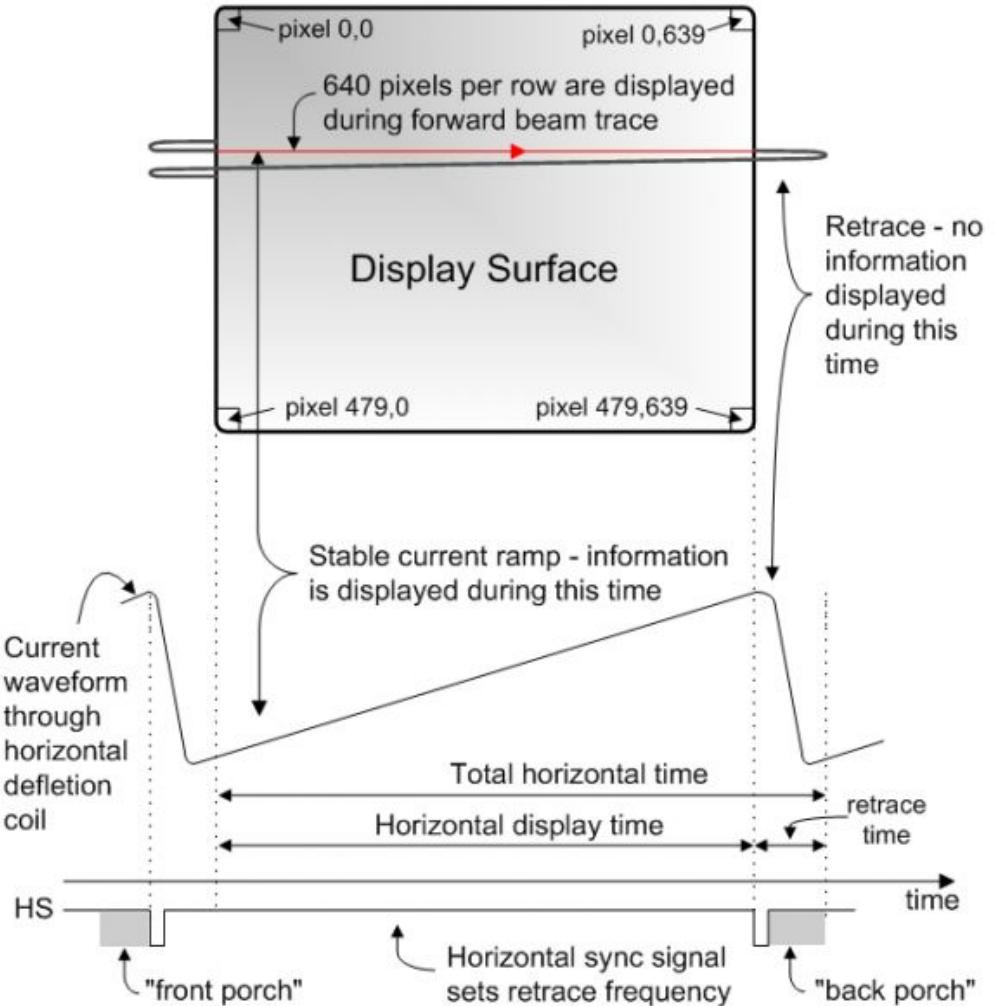
Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



Introducción 1

Módulos 2

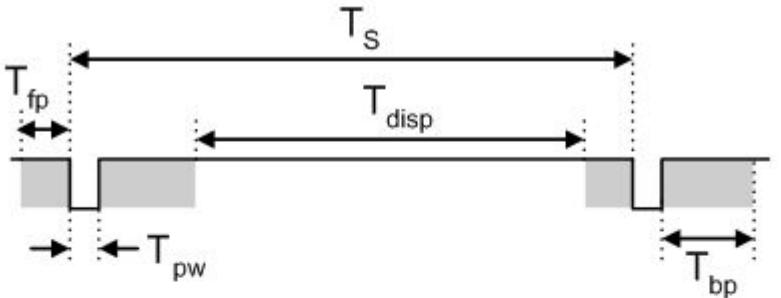
Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

# 640x480 Clock 25MHz



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
$T_S$	Sync pulse	16.7ms	416,800	521	32 us	800
$T_{disp}$	Display time	15.36ms	384,000	480	25.6 us	640
$T_{pw}$	Pulse width	64 us	1,600	2	3.84 us	96
$T_{fp}$	Front porch	320 us	8,000	10	640 ns	16
$T_{bp}$	Back porch	928 us	23,200	29	1.92 us	48

Introducción

1

Módulos

2

Simulación

3

Secuenciales

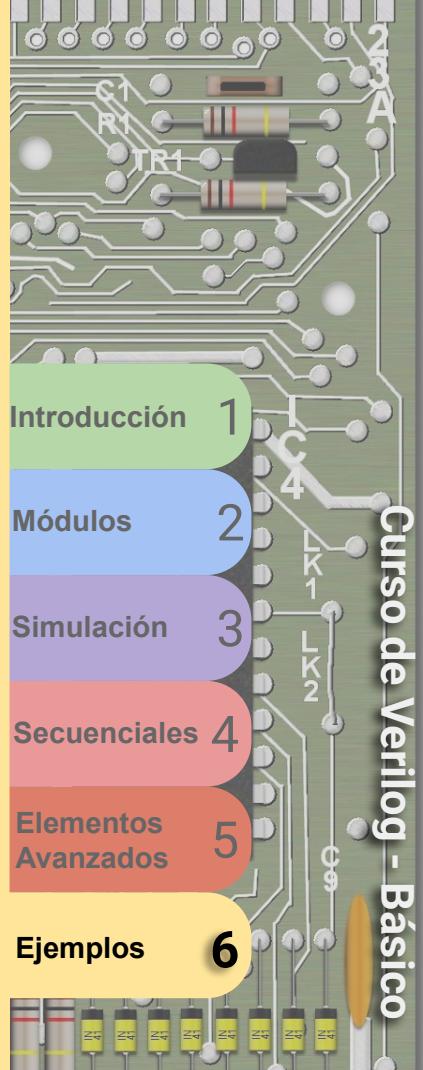
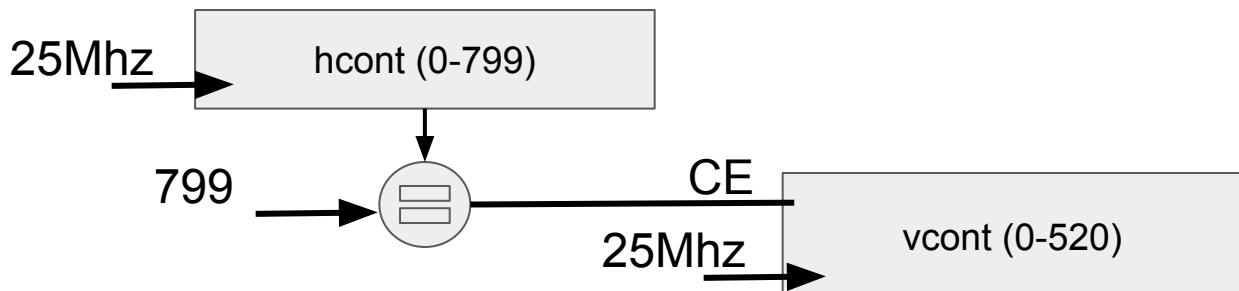
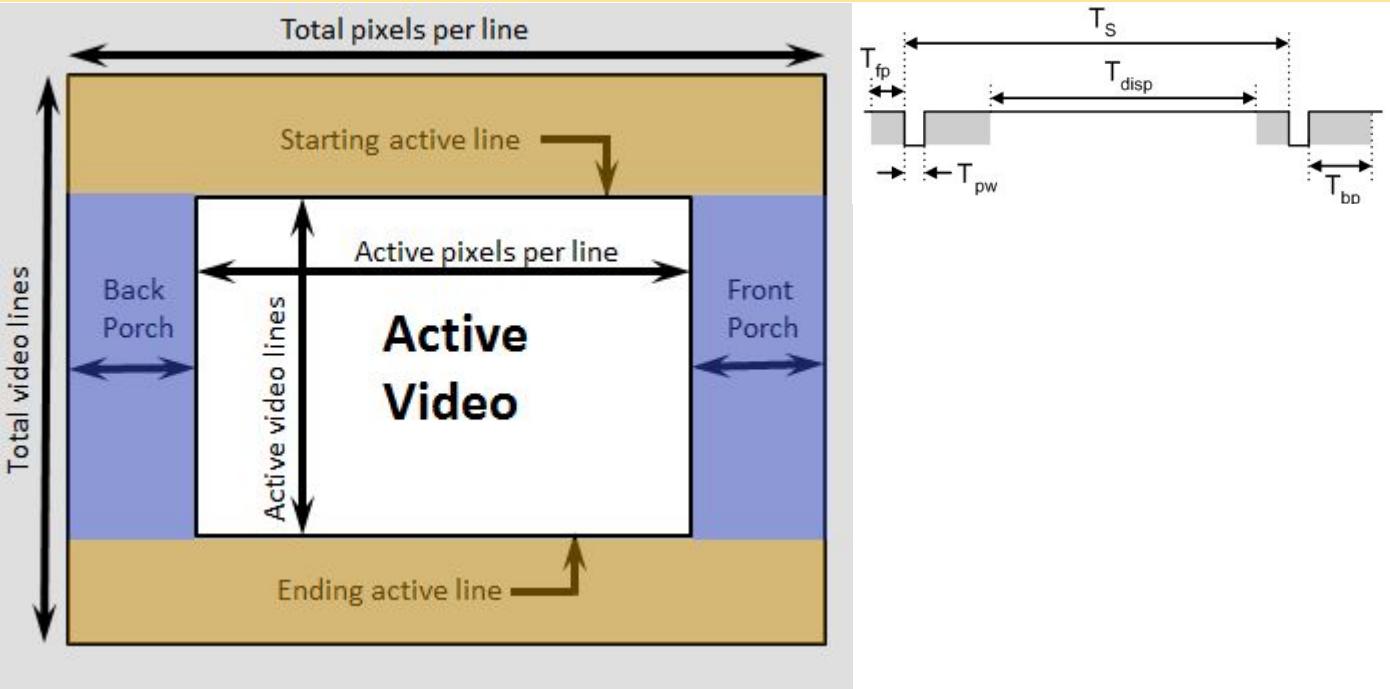
4

Elementos Avanzados

5

Ejemplos

6



```

module SalidaVGA(
    output Vsync,
    output Hsync,
    output [4:0]vga_r,
    output [5:0]vga_g,
    output [4:0]vga_b,
    input Reset
);
wire clock25;
ClockCPU U0 (.clock25(clock25));

reg [9:0] hcont,vcont;
wire HCE;
wire Red,Green,Blue;
assign Red=1'b0; assign Green=1'b0;
assign Blue=(hcont<10'd640);
assign vga_r = {5{Red}};
assign vga_g = {6{Green}};
assign vga_b = {5{Blue}};

assign HCE = (hcont==10'd799);
assign Hsync = (hcont>=10'd656 & hcont<10'd752)?1'b0:1'b1;
assign Vsync = (vcont>=10'd490 & vcont<10'd492)?1'b0:1'b1;

always @(posedge clock25 or posedge Reset)
begin
    if (Reset | hcont==10'd799)
        hcont <= 10'd0;
    else
        hcont <= hcont+1;
end

always @(posedge clock25 or posedge Reset)
begin
    if (Reset | vcont==10'd523)
        vcont <= 10'd0;
    else
        if (HCE)
            vcont <= vcont + 1;
end
endmodule

```



Introducción 1

Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6

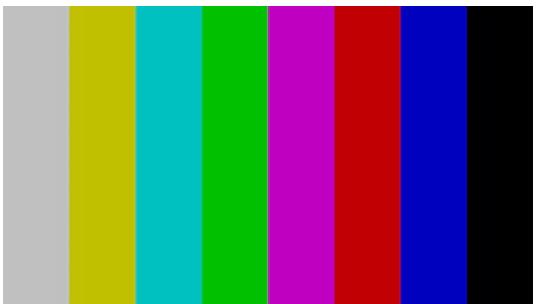
```

module SalidaVGA(
    output Vsync,
    output Hsync,
    output [4:0]vga_r,
    output [5:0]vga_g,
    output [4:0]vga_b,
    input Reset
);
wire clock25;
ClockCPU U0 (.clock25(clock25));

reg [9:0] hcont,vcont;
reg [2:0] color;
reg [6:0] contadorcolor;
wire HCE;
wire Red,Green,Blue;
assign Red=color[0];
assign Green=color[1];
assign Blue=color[2];
assign vga_r = {5{Red}};
assign vga_g = {6{Green}};
assign vga_b = {5{Blue}};

assign HCE = (hcont==10'd799);
assign Hsync = (hcont>=10'd656 && hcont<10'd752)?1'b0:1'b1;
assign Vsync = (vcont>=10'd490 && vcont<10'd492)?1'b0:1'b1;

```



```

always @(posedge clock25 or posedge Reset)
begin
    if (Reset | hcont>=10'd640 | contadorcolor==7'd79)
        contadorcolor=7'd0;
    else
        contadorcolor=contadorcolor+7'd1;
end
wire pulsoColor;
assign pulsoColor=(contadorcolor==7'd78);
always @(posedge clock25 or posedge Reset)
begin
    if (Reset)
        color <= 3'd0;
    else
        if (pulsoColor) color = color + 3'd1;
end

always @(posedge clock25 or posedge Reset)
begin
    if (Reset | hcont==10'd799)
        hcont <= 10'd0;
    else
        hcont <= hcont+1;
end

always @(posedge clock25 or posedge Reset)
begin
    if (Reset | vcont==10'd523)
        vcont <= 10'd0;
    else
        if (HCE)
            vcont <= vcont + 1;
end
endmodule

```

Introducción 1

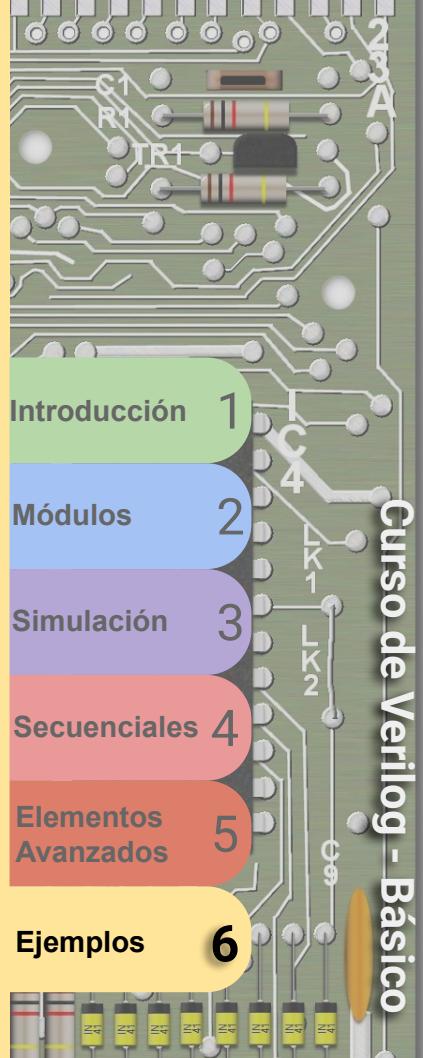
Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



```

module SalidaVGA(
    output Vsync,
    output Hsync,
    output [4:0]vga_r,
    output [5:0]vga_g,
    output [4:0]vga_b,
    input Reset
);
wire clock25;
ClockCPU U0 (.clock25(clock25));

```

```

reg [9:0] hcont,vcont;
wire HCE;
wire blanco;
assign vga_r = {5{blanco}};
assign vga_g = {6{blanco}};
assign vga_b = {5{blanco}};
wire [0:7] lin0,lin1,lin2,lin3,lin4,lin5,lin6,lin7;
assign lin0=8'b00000000;
assign lin1=8'b01111111;
assign lin2=8'b00000001;
assign lin3=8'b00000001;
assign lin4=8'b01111111;
assign lin5=8'b00000001;
assign lin6=8'b00000001;
assign lin7=8'b01111111;

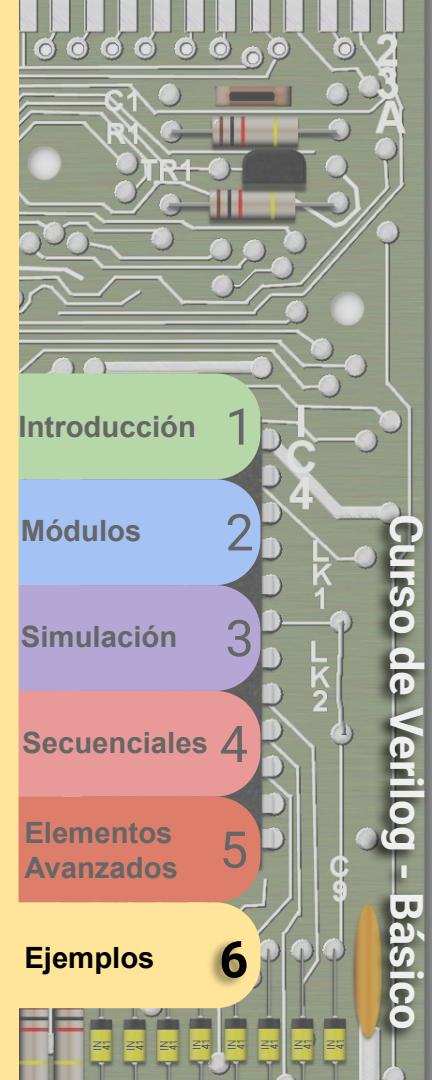
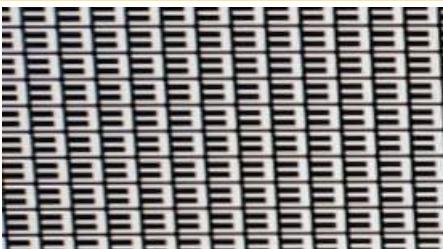
```

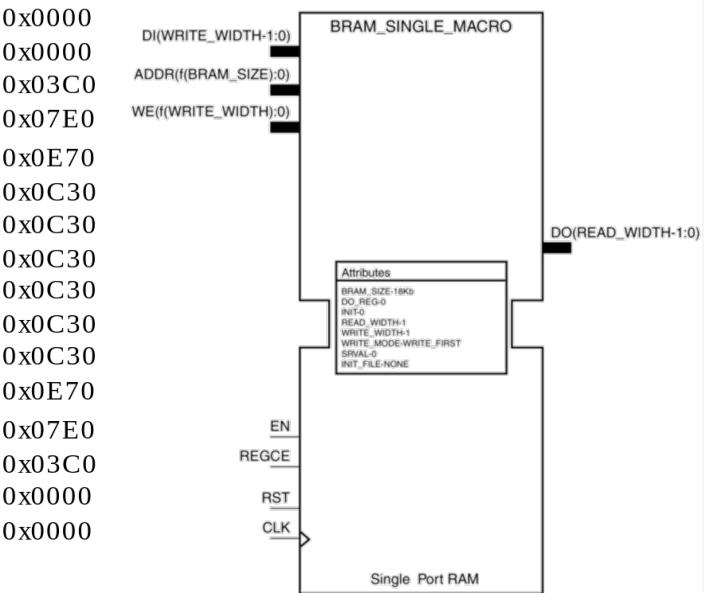
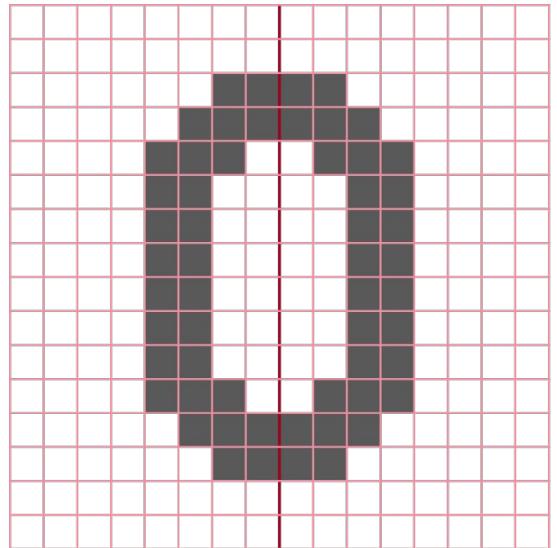
```

reg [0:7] linea;
always@*
begin
    case (vcont[2:0])
        3'd0: linea=lin0;
        3'd1: linea=lin1;
        3'd2: linea=lin2;
        3'd3: linea=lin3;
        3'd4: linea=lin4;
        3'd5: linea=lin5;
        3'd6: linea=lin6;
        default: linea=lin7;
    endcase
end

reg bit;
always@*
begin
    case (hcont[2:0])
        3'd0: bit=linea[0];
        3'd1: bit=linea[1];
        3'd2: bit=linea[2];
        3'd3: bit=linea[3];
        3'd4: bit=linea[4];
        3'd5: bit=linea[5];
        3'd6: bit=linea[6];
        default: bit=linea[7];
    endcase
end
assign blanco = (hcont<640 && vcont<480 && bit);

```





```
module char_rom(
    output [15:0] DO,
    input [10:0] ADDR,
    input CLK,
    input [15:0] DI,
    input EN,
    input REGCE,
    input RST,
    input [1:0] WE
);
```

// Output data, width defined by READ\_WIDTH parameter  
// Input address, width defined by read/write port depth  
// 1-bit input clock  
// Input data port, width defined by WRITE\_WIDTH parameter  
// 1-bit input RAM enable  
// 1-bit input output register enable  
// 1-bit input reset  
// Input write enable, width defined by write port depth

Introducción 1

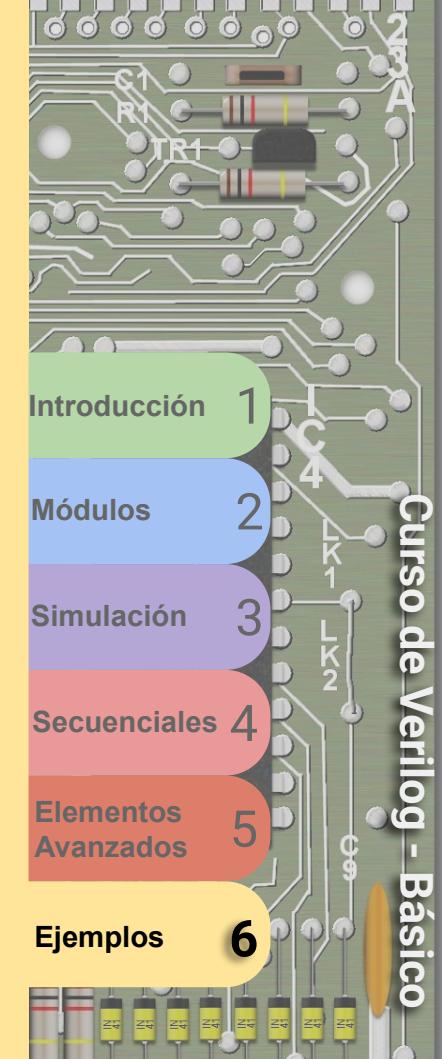
Módulos 2

Simulación 3

Secuenciales 4

Elementos Avanzados 5

Ejemplos 6



```

wire [10:0] address;
assign address={7'd67,vcont[3:0]};
wire [15:0] linea;
char_rom UI(.DO(linea),.ADDR(address),.CLK(clock25),.DI(16'd0),.EN(1'b1),.REGCE(1'b1),.RST(Rest),.WE(2'b00));

reg bit;
always@*
begin
    case (hcont[3:0])
        4'hf: bit=linea[0];
        4'he: bit=linea[1];
        4'hd: bit=linea[2];
        4'hc: bit=linea[3];
        4'hb: bit=linea[4];
        4'ha: bit=linea[5];
        4'h9: bit=linea[6];
        4'h8: bit=linea[7];
        4'h7: bit=linea[8];
        4'h6: bit=linea[9];
        4'h5: bit=linea[10];
        4'h4: bit=linea[11];
        4'h3: bit=linea[12];
        4'h2: bit=linea[13];
        4'h1: bit=linea[14];
        default: bit=linea[15];
    endcase
end
assign blanco = (hcont<640 && vcont<480 && bit);

```



NUNCA! elegir letras simétricas en algún eje para las pruebas!!!

