

SISTEMAS DIGITALES PROGRAMABLES

31/01/2023

Verilog para Diseño y Simulación

Verilog para Diseño y Simulación

2

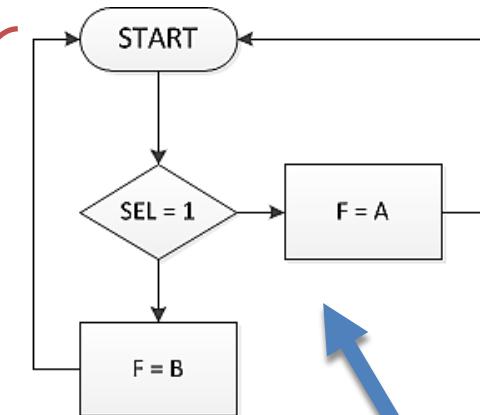
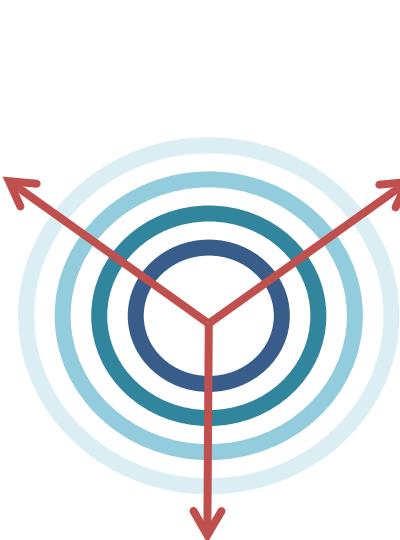
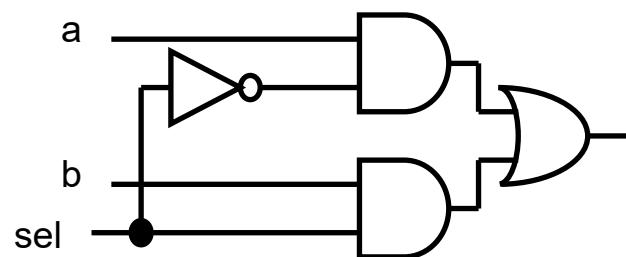
Sistemas Digitales Programables

- Dominios y Niveles de Modelización
- Flujos de Diseño y Verificación
- El lenguaje de Descripción de Hardware VERILOG
 - Jerarquía y Estructuras fundamentales
 - Bloques Procedurales y Asignaciones Continuas
 - Concurrencia
 - VERILOG para Síntesis (VERILOG - RTL)

Dominios y Niveles de Modelización

3

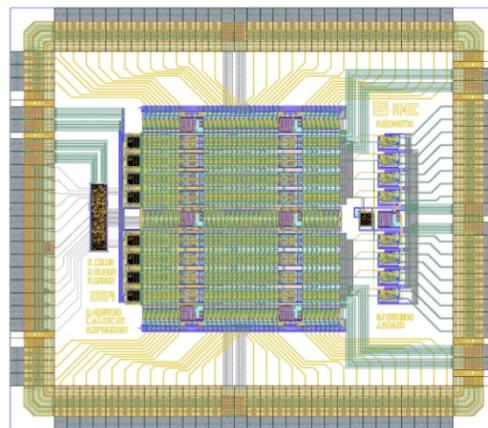
Sistemas Digitales Programables



```
module mux2_1(f, a, b, sel);
output f;
input a, b, sel;
reg f;

always @(a or b or sel)
  if (sel) f = a;
  else f = b;

endmodule
```



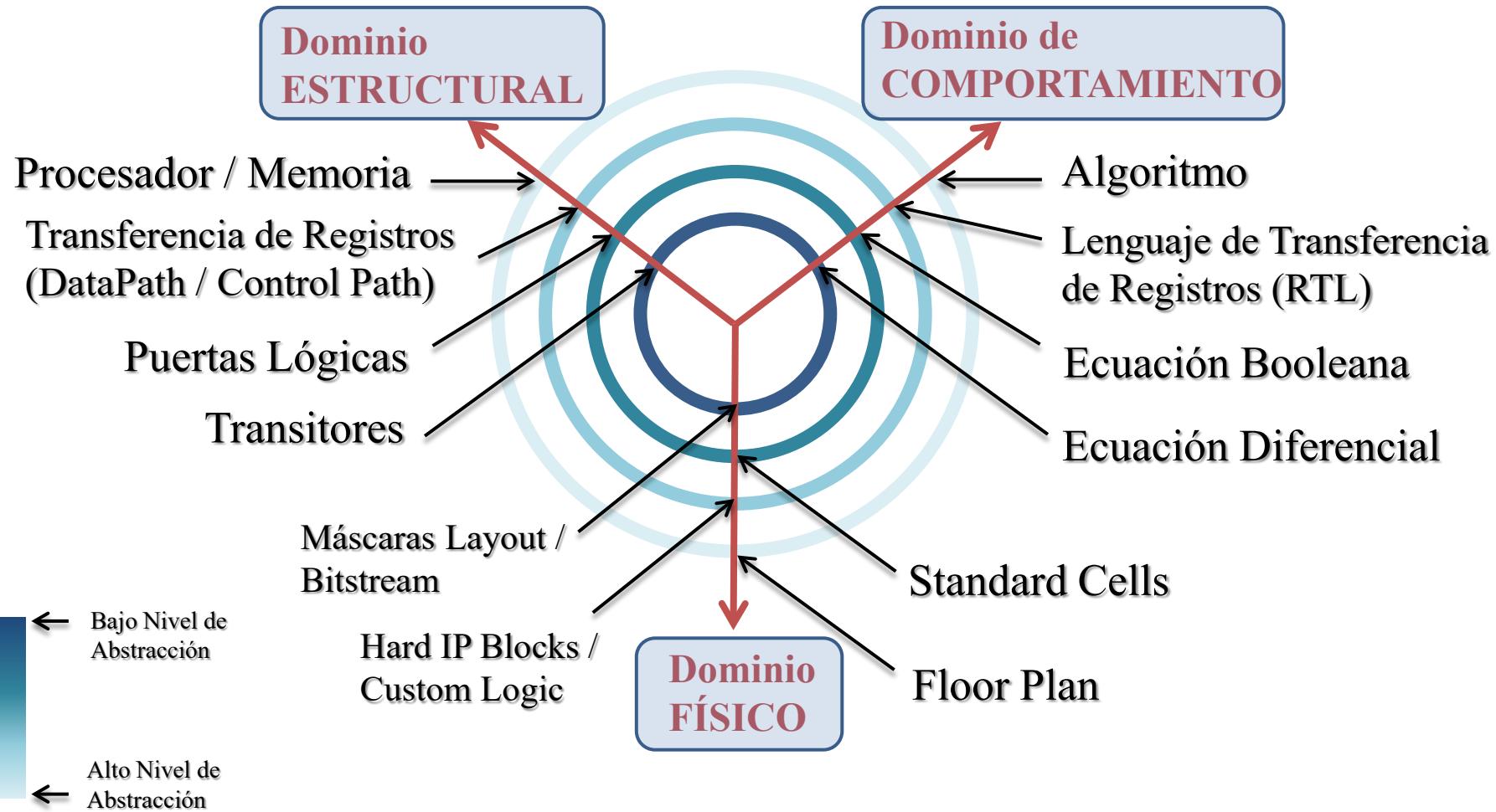
← Bajo Nivel de Abstracción

← Alto Nivel de Abstracción

Dominios y Niveles de Modelización

4

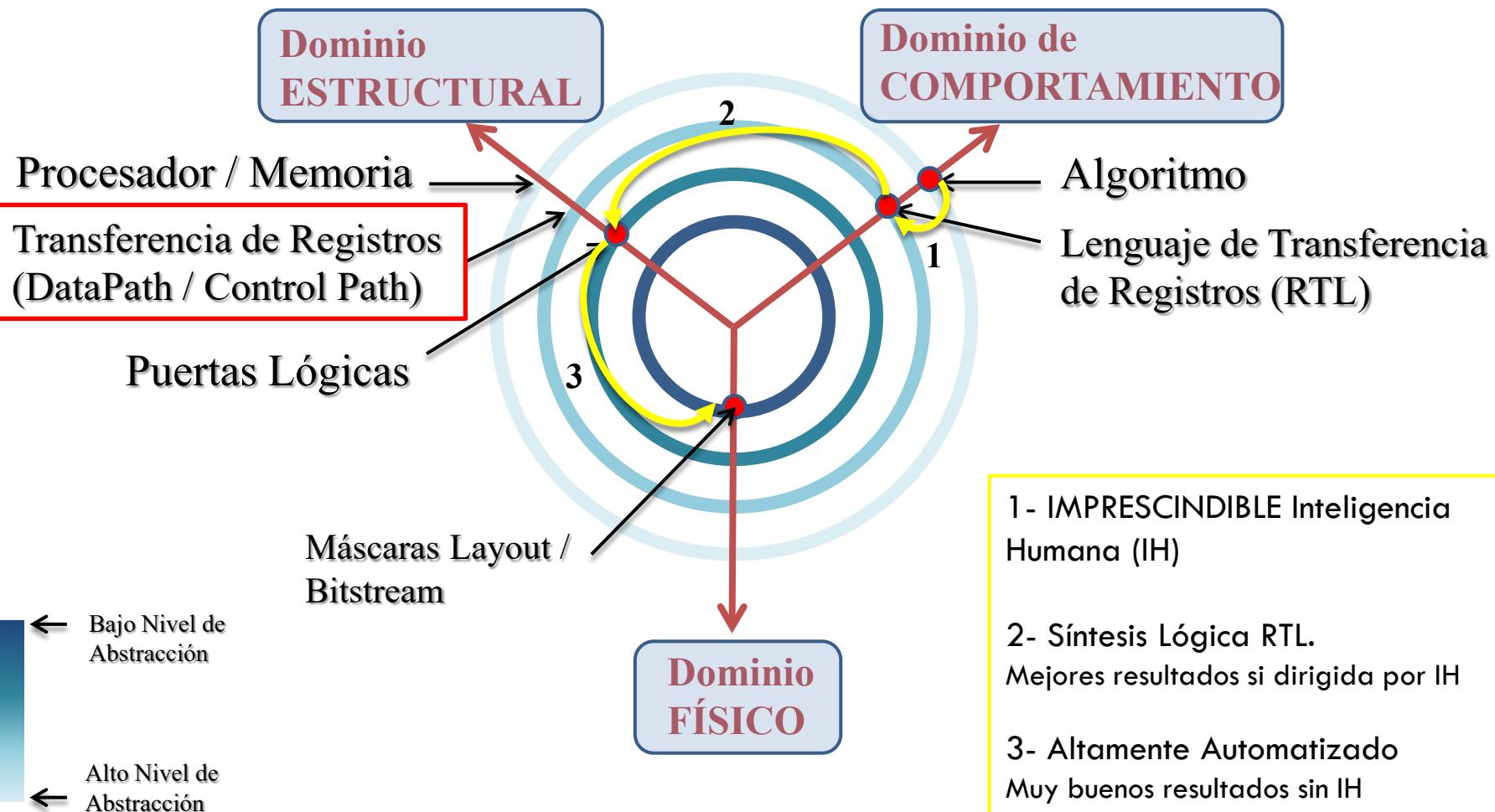
Sistemas Digitales Programables



Dominios y Niveles de Modelización

5

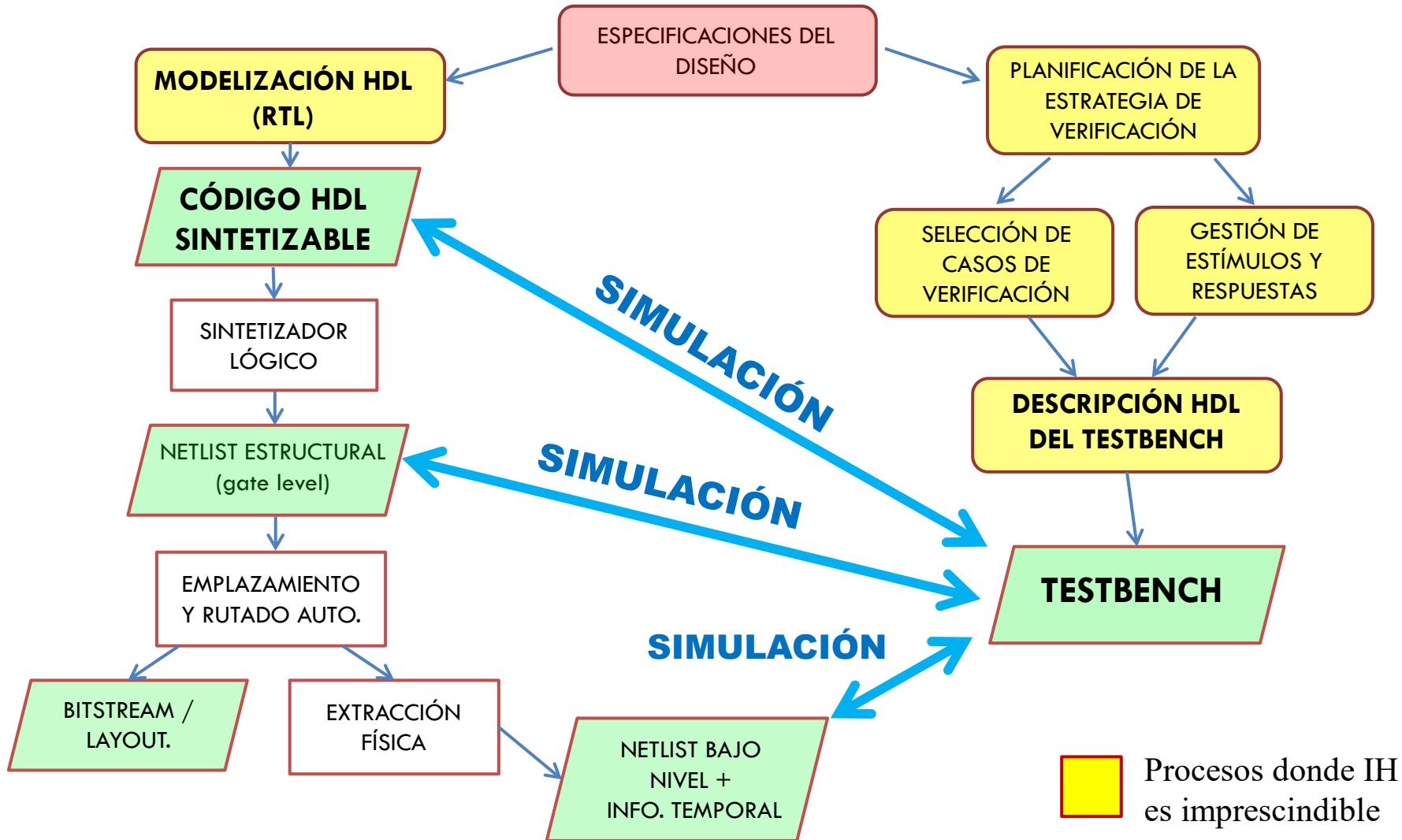
Sistemas Digitales Programables



Flujos de Diseño y Verificación

6

Sistemas Digitales Programables

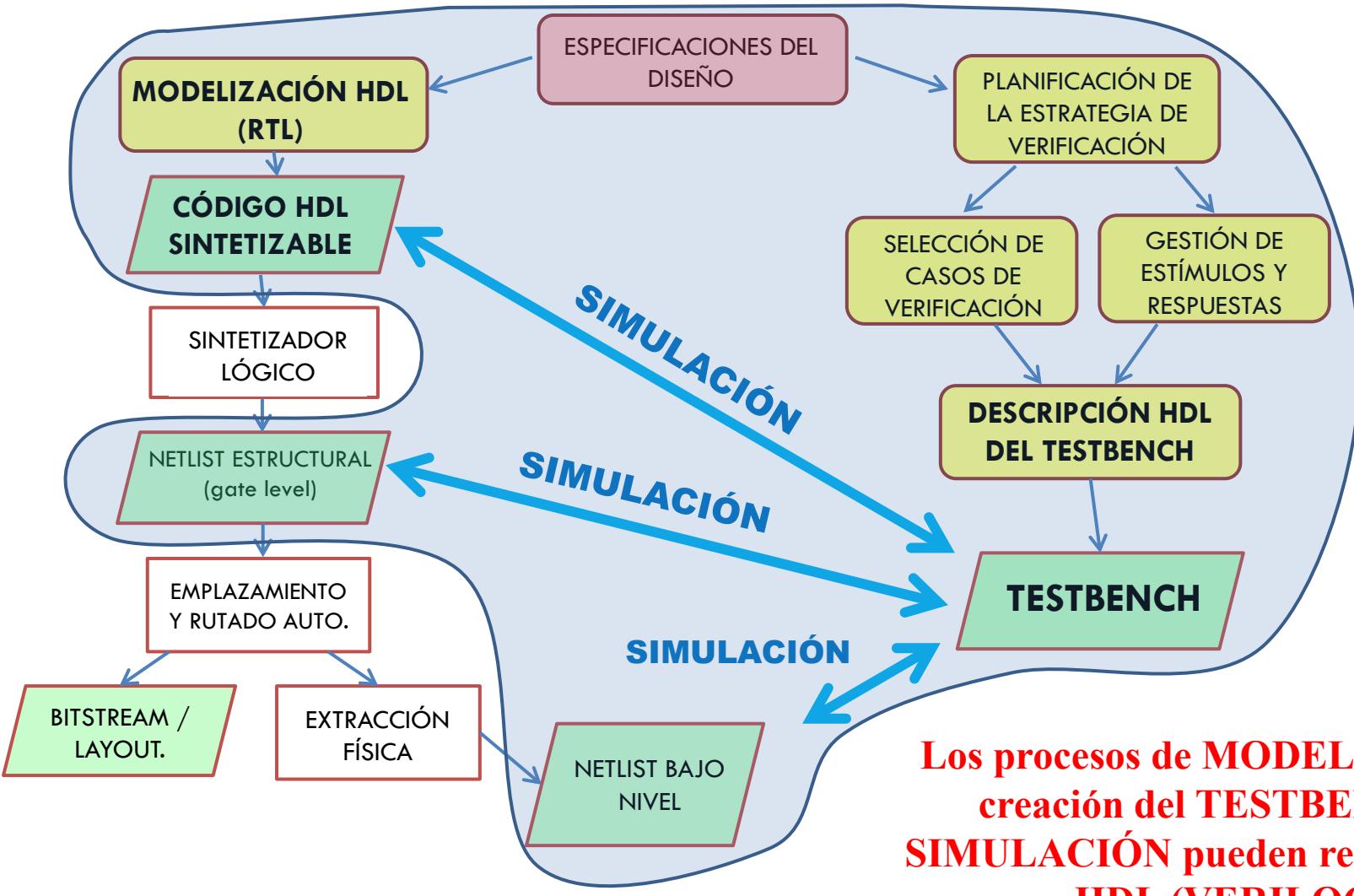


Procesos donde IH
es imprescindible

Flujos de Diseño y Verificación

7

Sistemas Digitales Programables



Introducción a Verilog

- Verilog HDL: Lenguaje de descripción de Hardware.
- Describir sistemas digitales:
 - ▣ Desde un sistema basado en microprocesador hasta un simple flip-flop.
 - ▣ Descripción de hardware a diferentes niveles: Niveles de Abstracción.
- El Verilog HDL, se describe en la norma IEEE1364-1995 y su posterior actualización de 2001.
 - ▣ Las herramientas utilizan un subconjunto de la norma.
 - ▣ No toda la sintaxis descrita en la norma es **sintetizable**.
 - Síntesis: Proceso por el cual el diseño se adapta a un hardware concreto, ya sea una FPGA o un ASIC.

Niveles de Abstracción en Verilog

9

Sistemas Digitales Programables

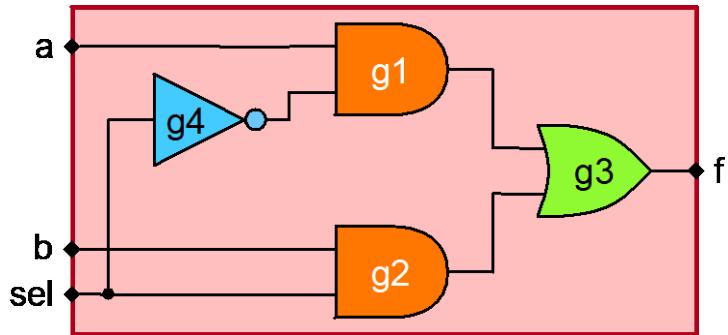
- Abstracción: Acción de separar conceptualmente algo de algo.
- Nivel de abstracción: cantidad de detalles que se utilizan para describir un sistema electrónico.
 - Alto nivel: descripción funcional de un sistema.
 - Bajo nivel: detallar todas las características eléctricas, niveles de tensión, tiempos, etc.
- Verilog puede describir un circuito con diferentes niveles de abstracción:
 - Nivel de puerta o nivel estructural.
 - Nivel de transferencia de registro o nivel RTL.
 - Nivel de comportamiento (Behavioral Level).

Nivel de puerta

10

Sistemas Digitales Programables

- Descripción a bajo nivel.
- Descripción mediante primitivas lógicas (AND, OR, NOT, etc.), conexiones lógicas.
 - ▣ Se pueden añadir propiedades temporales.
- Las señales toman los valores '0', '1', 'X' y 'Z'.



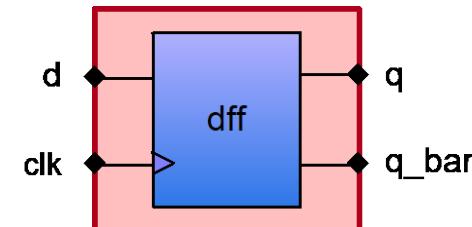
```
module mux(f, a, b, sel);
  input a, b, sel;
  output f;
  and #5 g1(f1, a, nsel),
        g2(f2, b, sel);
  or  #5 g3(f, f1, f2);
  not g4(nsel, sel);
endmodule
```

Nivel de Transferencia de registro o RTL

11

Sistemas Digitales Programables

- Especificar las características de un circuito mediante
 - ▣ operaciones
 - ▣ transferencia de datos entre registros.
- La especificación a nivel RTL confiere la propiedad de diseño sintetizable.



```
module dff(d, clk, q, q_bar);
  input d, clk;
  output q, q_bar;
  reg q, q_bar;
  always @ (posedge clk)
    begin
      q <= #1 d; // Retardo 1 unidad
      q_bar <= #1 ~d; // Retardo 1 unidad
    end
endmodule
```

Nivel de comportamiento

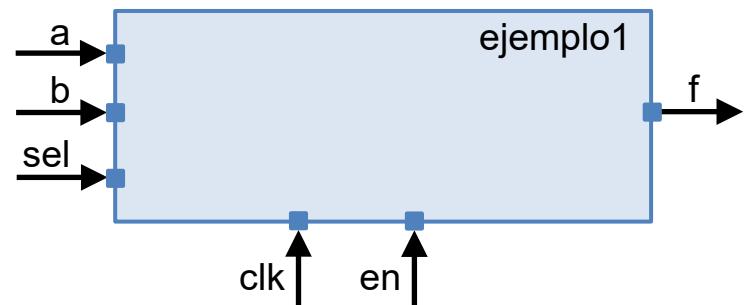
- Independiente de la estructura del diseño.
- El diseñador define el comportamiento.
- El diseño se define mediante algoritmos en paralelo.
 - Son instrucciones que se ejecutan de forma secuencial.
- La descripción puede hacer uso de sentencias o estructuras no sintetizables.
- Realización de *testbenches*:
 - Verificar el correcto funcionamiento de un modulo o diseño.
 - Escritura tan compleja o más como la realización en RTL del propio diseño.
 - La escritura de un testbench no tiene porque ser sintetizable.

Comenzamos con el diseño

13

Sistemas Digitales Programables

- Contenedor del diseño:
 - *module*
- Declaración entradas y salidas:
 - *input, output, inout*
 - Declarar tamaño.
 - Parámetros
- Descripción del diseño:
 - Asignaciones
 - Bloques procedurales
- Finalización de la descripción:
 - *endmodule*



```
module     ejemplo1(a, b, sel, clk, en, f);
// Declaración Inputs y Outputs
input      a, b, sel, clk, en;
output     f;

// Descripción del diseño

endmodule
```

Consideraciones sobre Verilog HDL

14

Sistemas Digitales Programables

□ Entradas y salidas:

- Se pueden agrupar formando buses:
 - Comparten nombre.
- Hay que establecer el tamaño:
 - El número de elementos.
- Se pueden referenciar, por el índice.

```
// Entradas y salidas de un solo bits
input      a, b, c, d;
output     f;
// Entradas y salidas de varios bits
input      [3:0] bus_a;
output     [7:0] bus_f;

assign    f = bus_a[2];
assign    bus_f[5] = a;
```

□ Comentarios:

- De una sola línea, se precede de: //
- De varias líneas: /* comentario */

□ Uso de mayúsculas:

- Sensible al uso de mayúsculas.
- Recomendable usar solo minúsculas.

□ Identificadores:

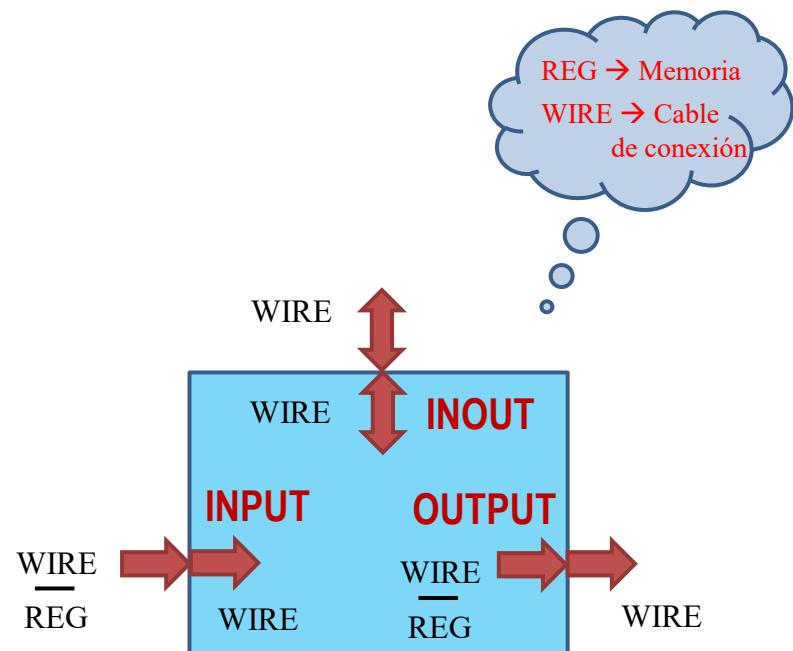
- Deben comenzar por un carácter.
- Cualquier letra, excepto la ñ y la ç,
- cualquier número
- y los símbolos “_” y “\$”.

Tipos de datos

15

Sistemas Digitales Programables

- **Nets:** cables de conexión.
 - No almacenan información.
 - Utilizar tipo *wire*.
- **Registers:** representan variables con capacidad de almacenar información.
 - Utilizar tipo *reg* y de manera excepcional *integer* para testbenches.



Tipos de datos

16

Sistemas Digitales Programables

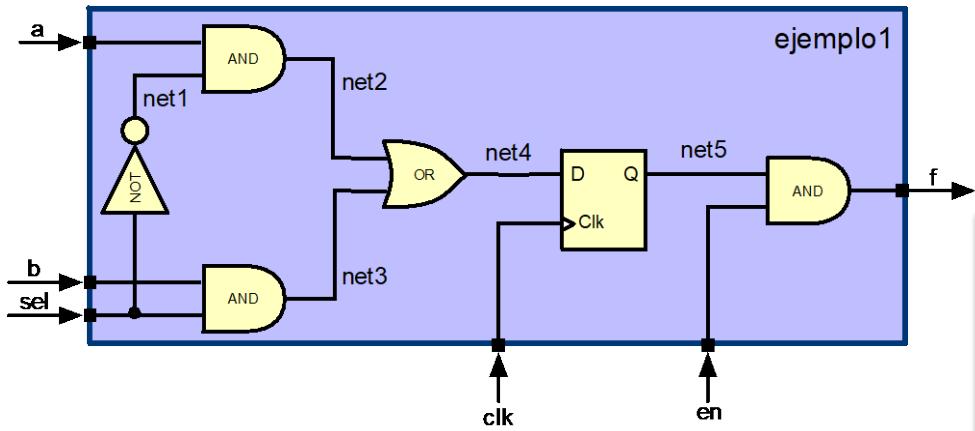
Un dato deber ser cuando le asignemos su valor ...
net (wire)	Con una asignación continua (assign) Conectándolo a la salida de un submódulo.
registro (reg)	Dentro de un proceso (always / initial).

Un reg puede ser secuencial o combinacional.

Ejemplo: definición nodos

17

Sistemas Digitales Programables



```
module     ejemplo1(a, b, sel, clk, en, f);
// Declaración Inputs y Outputs
input      a, b, sel, clk, en;
output     f;
wire      f;
// Descripción de los nodos internos
reg       net5;
wire      net1, net2, net3, net4;

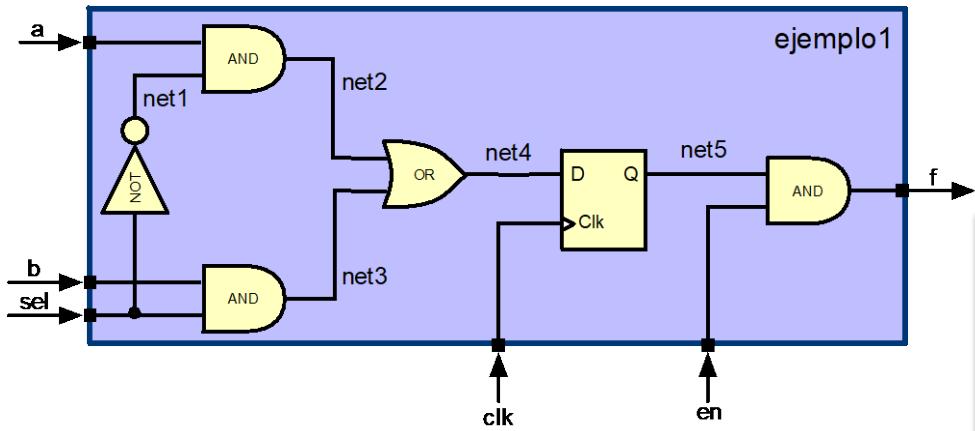
// Descripción del diseño

endmodule
```

Ejemplo: definición nodos

18

Sistemas Digitales Programables



```
module     ejemplo1(a, b, sel, clk, en, f);
// Declaración Inputs y Outputs
input      a, b, sel, clk, en;
output     f;
wire      f;
// Descripción de los nodos internos
reg       net4, net5;
wire      net1, net2, net3;

// Descripción del diseño

endmodule
```

Declaración señales y nodos

- **Inputs:** El tipo de las señales de entrada no se definen, se considera *wire*.
- **Outputs:** Las salidas pueden ser *wire* o *reg*, dependiendo de la capacidad de almacenamiento.
 - ▣ OJO: en Verilog un nodo del tipo *wire* puede atacar a una salida.
- **Nodos Internos:** siguen la misma filosofía que las salidas.

Asignaciones Continuas

20

Sistemas Digitales Programables

- La asignación continua se utiliza exclusivamente para modelar lógica combinacional.
- Se ejecuta de forma continua.
- Sintaxis:
 - `assign #<delay> variable = f(wire, reg, constante)`
- La variable solo puede ser del tipo `wire`.
- La asignación continua debe estar declarada fuera de cualquier proceso, NUNCA dentro de bloques `always` o `initial`.

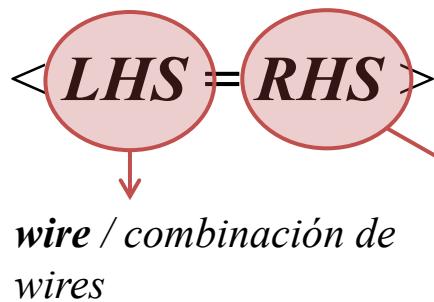
Asignaciones Continuas

21

Sistemas Digitales Programables

□ Elementos del Módulo. Asignaciones Continuas

assign <retardo> <<*LHS* = *RHS*>>



Asignación sobre combinación de señales

```
assign #102 {cout,sum}=a+b+cin;
```

Implícita en la declaración del wire

```
module m7485 (G, L, a, b);
output G,L;
input [3:0] a,b;
wire #(10,5) G = (a >b),
          L = (a<b);
endmodule
```

Ejemplos

22

Sistemas Digitales Programables

```
// Ejemplo de buffer triestado
module    buffer(data, enable, out);
input     [7:0]      data;
input          enable;
output    [7:0]      out;

wire      [7:0]      out;

assign   #1 out = (enable) ? data : 8'bz;
endmodule
```

```
/*Asignación de una suma de operandos de
4 bits a una concatenación de suma y
carry de salida*/
module    sumador(a, b, cin, sum, cout);
input     [3:0]      a, b;
input          cin;
output    [3:0]      sum;
output          cout;

reg      [3:0]      a,b;
wire      [3:0]      cout;
wire      [3:0]      sum;

assign   #1 {cout, sum} = a + b + cin;
endmodule
```

Procesos

23

Sistemas Digitales Programables

- Principal característica del Verilog:
 - ▣ Concurrencia: procesos que se ejecutan en paralelo.
- Toda descripción del comportamiento debe hacerse dentro de un proceso.
 - ▣ Excepto las asignaciones continuas.
- En Verilog hay dos tipos de procesos, también denominados bloques procedurales o concurrentes:
 - ▣ El denominado: *Initial*
 - ▣ El denominado: *Always*

Procesos: Initial y Always

24

Sistemas Digitales Programables

□ Proceso *Initial*:

- Se ejecuta una sola vez.
- No es Sintetizable.
- No se puede utilizar en descripción RTL.
- Uso solo para testbenches.

□ Proceso *Always*:

- Se ejecuta continuamente en modo bucle.
- Es Sintetizable.
- Se ejecuta por eventos o por temporización.
- Al conjunto de eventos, se denomina lista de sensibilidad.

initial	c
c	sentencia



CONTROLES TEMPORALES

#<delay> → Retardo de “delay” unidades de tiempo (ver ‘timescale’)

@(<signals list>) → Retardo hasta que cambia una “signal” (**posedge** o **negedge** si se desea cambio por flanco)

wait(<expresión>) → Retardo hasta que “expresión” sea cierta



always	c
c	sentencia

Procesos: Initial y Always

25

Sistemas Digitales Programables

```
// Ejemplo de proceso initial

initial
begin
    clk = 0;
    reset = 0;
    enable = 0;
    data = 0;
end
```

No
Sintetizable

```
// Ejemplo de proceso always

always @(a or b or sel)
begin //Sobra en este caso
    if (sel == 1)
        y = a;
    else
        y = b;
end //Sobra en este caso
```

Sintetizable

Procesos: Initial y Always

- *Begin, end:*
 - Necesarios si el proceso engloba más de una asignación procedural (=) o más de una estructura (if-else, case, etc.).
- *Initial:*
 - Se ejecuta a partir del instante cero.
 - Las asignaciones se ejecutan secuencialmente.
 - Si hay varios procesos *initial* se ejecutan en paralelo.
- *Always:*
 - Se ejecuta cada vez que se produzcan eventos en su lista de sensibilidad:
 - Variación de la señal *a*, de la señal *b* o de la señal *sel*.
 - En este caso solo contiene una estructura de control, se puede prescindir el *begin* y *end*.

Procesos: Initial y Always

- Las asignaciones dentro de un proceso se deben realizar sobre variables del tipo *reg* NUNCA sobre nodos *wire*.
 - Blocking: *variable* = *f(wire, reg, constante)*
 - Non-Blocking: *variable* <= *f(wire, reg, constante)*
- El origen de la asignación puede ser *reg*, *wire*, *constante* o una función.

```
// Ejemplo de proceso initial
wire      clk, reset;
reg       enable, data;
initial
  begin
    clk = 0;    //Error
    reset = 0;  //Error
    enable = 0;
    data = 0;
  end
```

Incorrecto

```
// Ejemplo de proceso initial
reg      clk, reset;
reg      enable, data;
initial
  begin
    clk = 0;
    reset = 0;
    enable = 0;
    data = 0;
  end
```

Correcto

Eventos

- Se considera como el cambio de una variable o señal.
- Sirve para controlar el instante en que se produce una asignación procedural o la ejecución de un proceso *always*.
- Se emplea el carácter @ seguido del evento.
- Hay dos tipos de eventos:
 - ▣ Eventos de nivel: Cambio de valor de una variable o de una lista de sensibilidad.
 - ▣ Eventos de flanco: Se produce por flanco de subida o bajada de una variable o de una lista de sensibilidad.

Eventos de nivel

29

Sistemas Digitales Programables

- Cambio de valor de una variable o de una lista de sensibilidad.

Evento	Descripción
<code>always @(a) b = b + c;</code>	Cada vez que varia a se evalúa la expresión.
<code>always @(a or b or c) d = a + b;</code>	Cada vez que varia a ó b ó c se evalúa la expresión. En este caso a, b y c son la lista de sensibilidad.

Eventos de flanco

30

Sistemas Digitales Programables

- Se produce por flanco de subida o bajada de una variable o de una lista de sensibilidad.

Evento	Descripción
<code>always @(posedge clk or posedge rt) b <= b + c;</code>	Cada vez que se produce un flanco de subida de clk o de rt se evalúa la expresión.
<code>always @(posedge clk or negedge rt) b <= b + c;</code>	Cada vez que se produce un flanco de subida de clk o de bajada de rt se evalúa la expresión.

Sentencia condicional if - else

31

Sistemas Digitales Programables

```
if (condición)
    sentencias; //condición verdadera
else
    sentencias; //condición falsa
```

```
if (condición1)
    sentencias; //condición1 verdadera
else if (condición2)
    sentencias; //condición2 verdadera
else if (condición3)
    sentencias; //condición3 verdadera
else
    sentencias; //todas las condiciones son falsas
```

```
if (reset == 1'b0)
    count <= 8'b0;
else if (enable == 1'b1 && up_down == 1'b1)
    count <= count + 1'b1;
else if (enable == 1'b1 && up_down == 1'b0)
    count <= count - 1'b1;
else
    count <= count;
```

Usar solo dentro de
un proceso

Sentencia case

32

Sistemas Digitales Programables

```
case (expresión)
  <caso1> : sentencia;
  <caso2> : begin
    sentencias;
    sentencias;
  end
  <caso3> : sentencia;
  default : sentencia;
end case
```

```
case (sel)
  2'b00 : y = ina;
  2'b01 : y = inb;
  2'b10 : y = inc;
  2'b11 : y = ind;
  default : y = ind;
end case
```

- ✓ Hay que considerar todos los posibles casos.
- ✓ Las variables pueden tomar valores como 'z' o 'x', no solo el '1' y el '0'.
- ✓ Usar el *default* para completar los casos.

Usar solo dentro de
un proceso

Bucle for

33

Sistemas Digitales Programables

```
for (<valor inicial>; <expresión>; <incremento>)  
    sentencias;
```

```
for (k = 0; k < n-1; k = k+1)  
    Q[k] <= Q[k+1];
```

- ✓ Hay declarar la variable *k*.
- ✓ Lo normal es declarar *k* como un *integer*.
- ✓ Ejemplo: *integer k*;

Usar solo dentro de
un proceso

Jerarquía

34

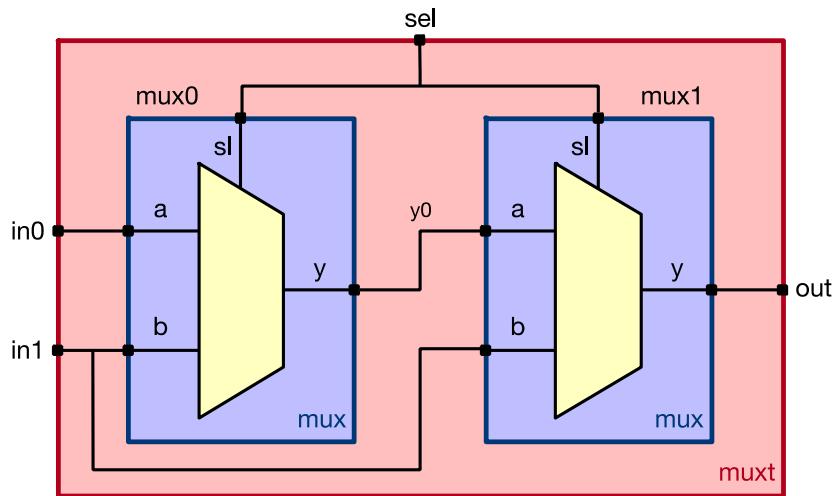
Sistemas Digitales Programables

- Diseño con jerarquía:
 - Construir diseños de mayor complejidad, a partir de módulos o diseños más sencillos.
 - La manera de incluir un módulo en un diseño es:
 - *master_name instance_name(port_list);*
 - Conexionado de un módulo dentro del diseño:
 - Por orden: las señales se asignan a los puertos del módulo según el orden en que han sido declarados los puertos.
 - Por nombre: las señales se asignan a los puertos del módulo especificando el nombre del puerto, sin importar el orden.
- Ejemplo:
 - Módulo o diseño denominado *muxt* formado por dos módulos denominados *mux*, cuya declaración de interface es:
 - *module mux(a, b, sl, y); // a, b e y son señales de 8 bits.*

Ejemplo: jerarquía

35

Sistemas Digitales Programables



```
module muxt(in0, in1, sel, out);
// Declaración Inputs y Outputs
input [7:0] in0, in1;
input sel;
output [7:0] out;
// Descripción de los nodos internos
wire [7:0] y0;
// Conexionado por orden
mux mux0(in0, in1, sel, y0);
// Conexionado por nombre
mux mux1(.y(out), .b(in1), .a(y0), .sl(sel));
endmodule
```

Parámetros

36

Sistemas Digitales Programables

- Diseño para el reúso.
 - Reutilización de diseños.
 - Optimización del proceso de diseño.
 - Uso de diseños parametrizados.
- La parametrización de un módulo se consigue mediante el uso del término **parameter**:
 - **parameter <nombre> = <valor por defecto>**
 - El número de parámetros no está limitado.
- Se especifican durante la llamada al módulo.
 - La sintaxis para incluir un módulo en un diseño, ahora, es:
 - **master_name #<valor_parameters> instance_name(port_list);**
 - Si no se especifican se toma el valor por defecto.
 - Se deben especificar en el orden en que fueron declarados o usar asignación por nombre.

Ejemplo: Parámetros

37

Sistemas Digitales Programables

```
/*Asignación de una suma de operandos de  
4 bits a una concatenación de suma y  
carry de salida*/  
  
module      sumador(a, b, cin, sum, cout);  
    input      [3:0]      a, b;  
    input          cin;  
    output     [3:0]      sum;  
    output          cout;  
  
    assign      #1 {cout, sum} = a + b + cin;  
  
endmodule
```

```
/*Asignación de una suma de operandos de  
n bits a una concatenación de suma y  
carry de salida*/  
  
module      sumador(a, b, cin, sum, cout);  
    parameter n = 8;  
    input      [n-1 :0]  a, b;  
    input          cin;  
    output     [n-1 :0]  sum;  
    output          cout;  
  
    assign      #1 {cout, sum} = a + b + cin;  
endmodule
```

```
/*Modo de incluir un módulo en un diseño*/  
sumador sum8bits(.a(), .b(), .cin(), .sum(), .cout());  
sumador #(16) sum16bits(.a(), .b(), .cin(), .sum(), .cout());  
sumador #(.n(4)) sum4bits(.a(), .b(), .cin(), .sum(), .cout());
```

Números en Verilog

- Pueden especificarse en:
 - decimal, hexadecimal, octal y binario.
- El carácter “_” se usa para una representación más clara, pero no se interpreta.
- Sintaxis de una constante numérica:
 - `<tamaño>'<base><valor>`
- Verilog, expande el valor hasta llenar el tamaño indicado.
 - El número se expande de derecha a izquierda.
 - Cuando el tamaño es menor que el valor, se truncan los bits mas significativos.
 - Cuando el tamaño es mayor que el valor, se rellena con el bit más significativo cuando este es “0”, “x” o “z” y se rellena con ceros si el bit más significativo es “1”, ver ejemplos.
- Los números negativos se representan en complemento a 2.
 - Se especifican con el signo “-” delante del tamaño.

Ejemplos de números en Verilog

39

Sistemas Digitales Programables

Entero	Almacenado como	Descripción
1	00000000000000000000000000000001	Sin tamaño, 32 bits
8'hAA	10101010	Hexadecimal con tamaño
6'b10_0011	100011	Binario con tamaño
'hF	00000000000000000000000000000001111	Hexadecimal sin tamaño, 32 bits
6'hCA	001010	Valor truncado
6'hA	001010	Relleno de ceros a la izquierda
16'bz	zzzzzzzzzzzzzz	Relleno de zetas a la izquierda
8'bx	xxxxxxx	Relleno de equis a la izquierda
8'b1	00000001	Relleno de ceros a la izquierda
-8'd2	11111110	Negativo relleno de unos a la izquierda

Operaciones

40

Sistemas Digitales Programables

TIPO	SÍMBOLO	OPERACION	EJEMPLO
ARITMÉTICO	+	Suma	$A=4'b0011; B=4'b0100; // A y B vectores$ $D=6; E=4; // D y E enteros$ $A*B // 4'b1100$ $D/E // 1$ $A+B // 4'b0111$
	-	Resta	
	*	Multiplicación	
	/	División, devuelve la parte entera	$in1=4'b101x; in2=4'b1010;$ $sum=in1 + in2; // 4'bx$ $-10 \% 3 = -1$ $14 \% -3 = 2 // Toma el signo del primer operador$
	%	Resto entero de la división	

Operaciones

41

Sistemas Digitales Programables

TIPO	SÍMBOLO	OPERACION	EJEMPLO
BIT A BIT	~	NOT	
	&	AND	$\sim(101011) = 010100$
		OR	$(010101) \& (001100) = 000100$ $(010101) (001100) = 011101$
	\wedge	OR EXCLUSIVA	$(010101) ^ (001100) = 011001$
	$\sim\wedge$	NOR EXCLUSIVA	

Operaciones

42

Sistemas Digitales Programables

TIPO	SÍMBOLO	OPERACION	EJEMPLO
LÓGICOS DE REDUCCIÓN (1 BIT DE SALIDA)	&	AND	&(10101010) =1'b0 (10101010) =1'b1 &(10x0x0x) =1'b0 (10x01010) =1'b1
	~&	NAND	
		OR	
	~	NOR	
	\wedge	OR EXCLUSIVA	
	~ \wedge	NOR EXCLUSIVA	

Operaciones

43

Sistemas Digitales Programables

TIPO	SÍMBOLO	OPERACION	EJEMPLO
LÓGICOS	!	NOT	$A = 4, B = 3, I = 4'b1010, J = 4'b1101, K = 4'b1xxz, L = 4'b1xxz, M = 4'b1xxx$
	&&	AND	
		OR	$A == B$ // resultado lógico 0, por ser falsa. $I != J$ // resultado lógico 1, por ser verdadero. $I == K$ // resultado lógico x.
	==	IGUAL	
	!=	DISTINTO	$K === L$ // resultado lógico 1 (todos los bits iguales, incluyendo x y z)
	==:, !=:	IGUAL, DISTINTO (CASE)	$K === M$ // resultado lógico 0 (el bit menos significativo no es igual)

Operaciones

44

Sistemas Digitales Programables

TIPO	SÍMBOLO	OPERADOR	EJEMPLO
RELACIONAL	>	Mayor que	
	<	Menor que	
	\geq	Mayor o Igual	
	\leq	Menor o Igual	

Operaciones

45

Sistemas Digitales Programables

TIPO	SÍMBOLO	OPERADOR	EJEMPLO
DESPLAZA-MIENTO	<<	Lógico a izquierda	
	>>	Lógico a derecha	
	>>>	Aritmético a derecha	
CONCATENA-CIÓN	{ }		byte = {4'h5, 4'h5} // byte = 01010101 {co, sum} = a + b + cin
REPLICACIÓN	{ N{DATO} }		byte = {4{2'b01}} // byte=8'b01010101 word = {{8{byte[7]}}, byte} // Extensión Signo
CONDICIO-NAL	?	(condición) ? exp_verd : exp_fal	out = (enable) ? a : b //out toma el valor de a si enable vale 1, sino toma el valor de b.

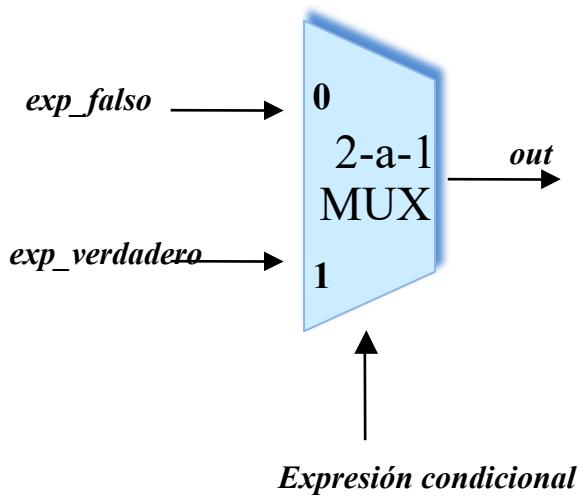
Operaciones

46

Sistemas Digitales Programables

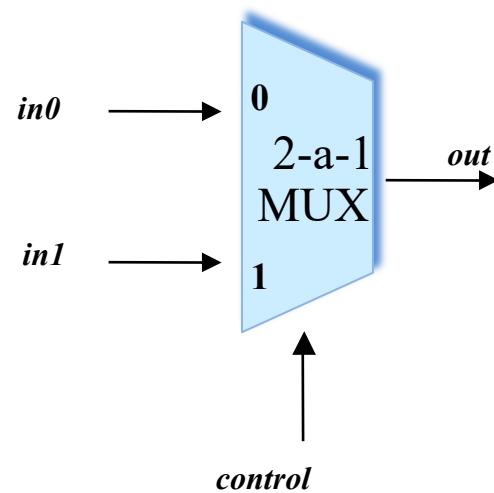
■ El Operador Condicional

<Expresión CONDICIONAL> ? <exp_VERDADERO> : <exp_FALSO>;



Ejemplo

```
out = control ? in1 : in0;
```

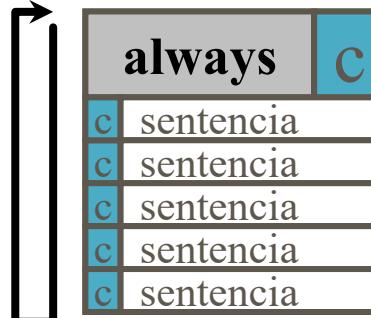
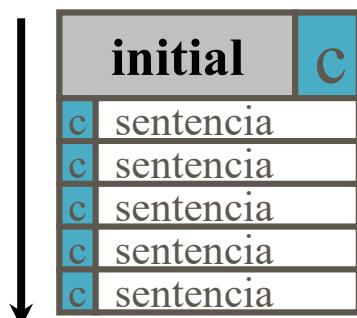


Verilog no sintetizable

47

Sistemas Digitales Programables

□ Elementos del Módulo. Bloques Procedurales



- Los Bloques Procedurales se ejecutan en **paralelo** entre si.
- Las sentencias internas de los BP se ejecutan secuencialmente.
- Las sentencias internas describen un comportamiento. Para ello se usan ...

ASIGNACIONES PROCEDURALES

LHS siempre **reg** ← **salida** = (a | b) & c

CONTROLES TEMPORALES

#<delay> → Retardo de "*delay*" unidades de tiempo (ver '*timescale*')

@(<signal>) → Retardo hasta que cambia "*signal*" (*posedge* o *negedge* si se desea cambio por flanco)

Wait(<expresión>) → Retardo hasta que "*expresión*" sea cierto

CONSTRUCCIONES DE ALTO NIVEL

If - else

While

Repeat

Case

For

Function / Task

Verilog no sintetizable

48

Sistemas Digitales Programables

□ Elementos del Módulo. Bloques Procedurales. Ejemplos (I)

```
initial  
begin  
    @(negedge reset) in23=0;  
    #5 wen =0;  
    address=16'h154a;  
    @(sel1 or sel2) donde=1;r1= 0;  
end
```

Activación por flanco de bajada de la señal de reset

Activación por nivel con cualquiera de las dos señales

```
initial  
begin  
    read=0;  
    wait (en1|en2) read =1;  
    #5 read=0;  
end
```

Detiene la ejecución del bloque hasta que se cumple la condición (en1== '1' o bien en2=='1')

```
always  
#(cycle/2) clk=~clk;
```

Retardo parametrizable

```
always @ (posedge clk)  
begin  
    #5 q=d;  
end
```

Flanco activo de reloj

```
always @ (reset , posedge clk, set)  
begin  
    if(!reset) q=0;  
    else if (!set) q=1;  
    else q=d;  
end  
assign _q=~q;
```

OR de todos los eventos

Verilog no sintetizable

49

Sistemas Digitales Programables

□ Elementos del Módulo. Bloques Procedurales. Ejemplos (II)

Cuando hay más de una
sentencia
(begin / end → bloque
secuencial)

```
i=0;  
while (i<10)  
begin  
    $display("i=%0d", i);  
    i=i+1;  
end
```

```
for (i=0; i<10; i=i+1)  
    $display("i=%0d", i);
```

```
i=0  
repeat (5)  
begin  
    $display("i=%0d", i);  
    i=i+1;  
end
```

CON PRIORIDAD

```
if (a>b)  
    $display("A is greater \n");  
else  
    $display("B is greater or equal to A\n");
```

SIN PRIORIDAD

```
case (number)  
3'b001: $display ("number is 001\n");  
3'b010: $display("number is 010\n");  
3'bxx1: $display ("number is xx1\n");  
default: $display("number is another value\n")  
endcase
```

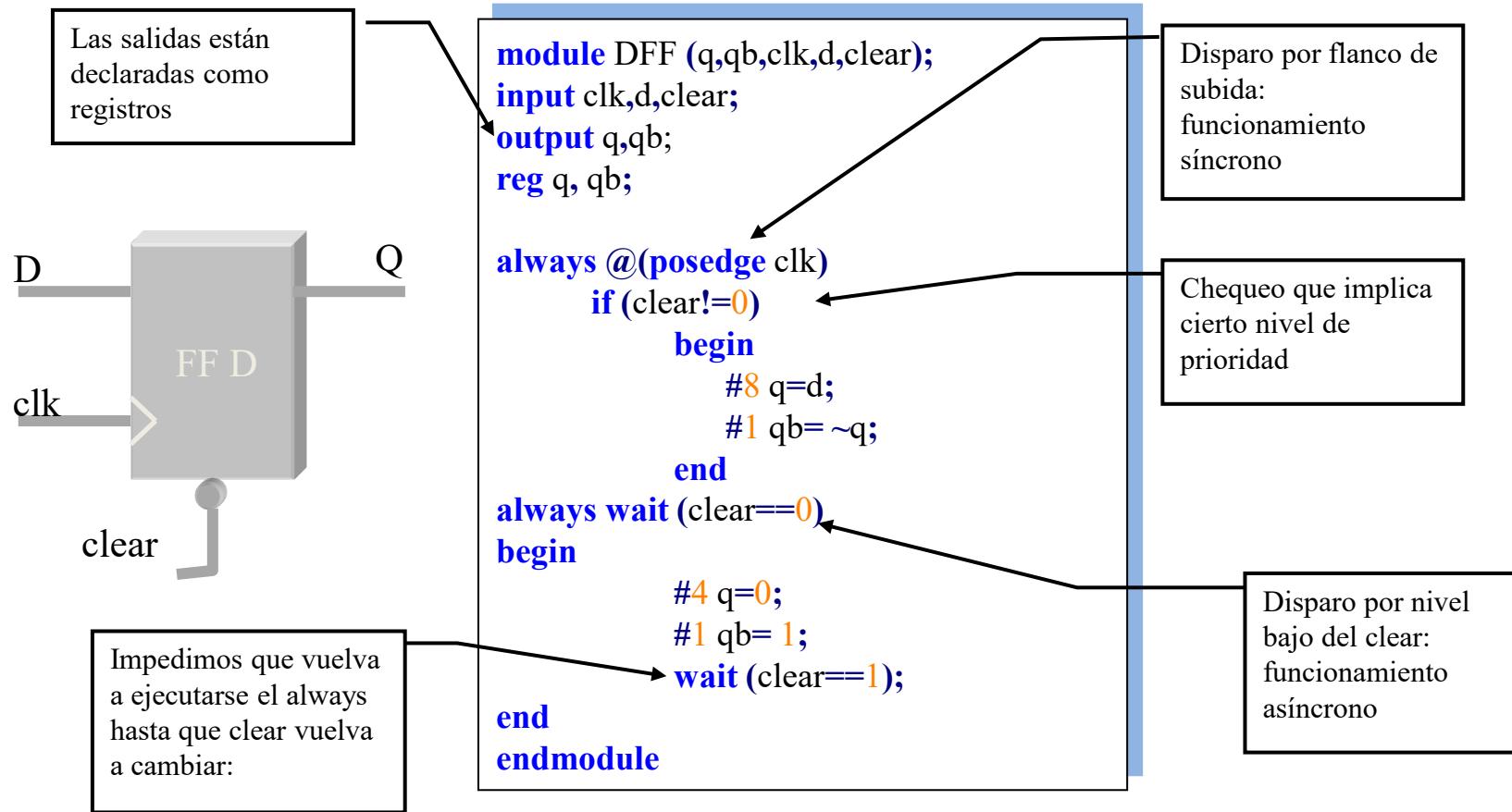
Las *x* se consideran *don't care*
sólo si se emplea **casex** o **casez**

VERILOG HDL (Lenguaje de Descripción de Hardware)

50

Sistemas Digitales Programables

□ Ejemplo: Descripción Behavioral de un DFF (NO SINTETIZABLE)



Verilog no sintetizable

51

Sistemas Digitales Programables

□ Elementos del Módulo. Funciones y Tareas

```
function <rango> <nombre>;  
    <declaraciones entradas>  
    <declaraciones variables>  
    begin  
        <sentencias>  
    end  
endfunction
```

```
task <task_name>;  
    <declaraciones entradas/salidas>  
    <declaraciones variables>  
    begin  
        <sentencias>  
    end  
endtask
```

- Se ejecuta en 0 de tiempo de simulación
- No admite controles temporales
- Debe tener al menos una entrada
- Solo puede devolver un valor

- Puede ejecutarse durante un tiempo de simulación diferente de 0
- Puede contener controles temporales
- Puede tener argumentos de entrada y salida arbitrarios (o no tener argumentos)

VERILOG no sintetizable

52

Sistemas Digitales Programables

□ Elementos del Módulo. Funciones y Tareas. Ejemplos

```
module alu(result, funcion,opa,opb);
output [31:0]result;
input [15:0] opa,opb;
input [4:0] funcion;
reg [15:0 ] result;

always @ (funcion or opa or opb)
    if (funcion == 5'h0)
        result = add(opa,opb);

function [15:0] add;
    input [15:0 ] a,b;
    add = a+b;
endfunction

endmodule
```

```
module operation;
parameter delay = 10;
reg [15:0] A, B;
reg [15:0] AB_AND, AB_OR, AB_XOR;

initial $monitor( ...);

always @(A or B)
begin
bitwise_oper(AB_AND, AB_OR, AB_XOR, A, B);
end

task bitwise_oper;
output [15:0] ab_and, ab_or, ab_xor;
input [15:0] a, b;
begin
#delay ab_and = a & b;
ab_or = a | b;
ab_xor = a ^ b;
end
endtask
endmodule
```

Concurrencia

53

Sistemas Digitales Programables

CONCURRENCIA

Controles Temporales aplicados a Concurrencia

Zero Delay

```
module event_control;
reg [4:0] N;
initial begin
    $display ("AAA");
    $display("BBB");
end
initial
for (N=0; N>=3; N= N+1)
    $display (N);
endmodule
```

Posibles
Diferencias de
Ejecución

AAA	0
BBB	1
0	2
1	3
2	AAA
3	BBB

```
module event_control;
reg [4:0] N;
initial begin
    $display ("AAA");
    $display("BBB");
end
initial
#0 for (N=0; N>=3; N= N+1)
    $display (N);
endmodule
```

AAA
BBB
0
1
2
3

El ‘zero delay’ fuerza el orden
de ejecución

Concurrencia

54

Sistemas Digitales Programables

□ CONCURRENCIA

□ Controles Temporales aplicados a Concurrencia

Retardo Intra-Asignación

```
x = #1 y;           // Retardo Intra-asignación  
  
// Código Equivalente al Retardo Intra-asignación.  
begin  
    hold = y;          // Sample & hold y  
    #1;                // Retardo.  
    x = hold;          // Asignación a x.  
end
```

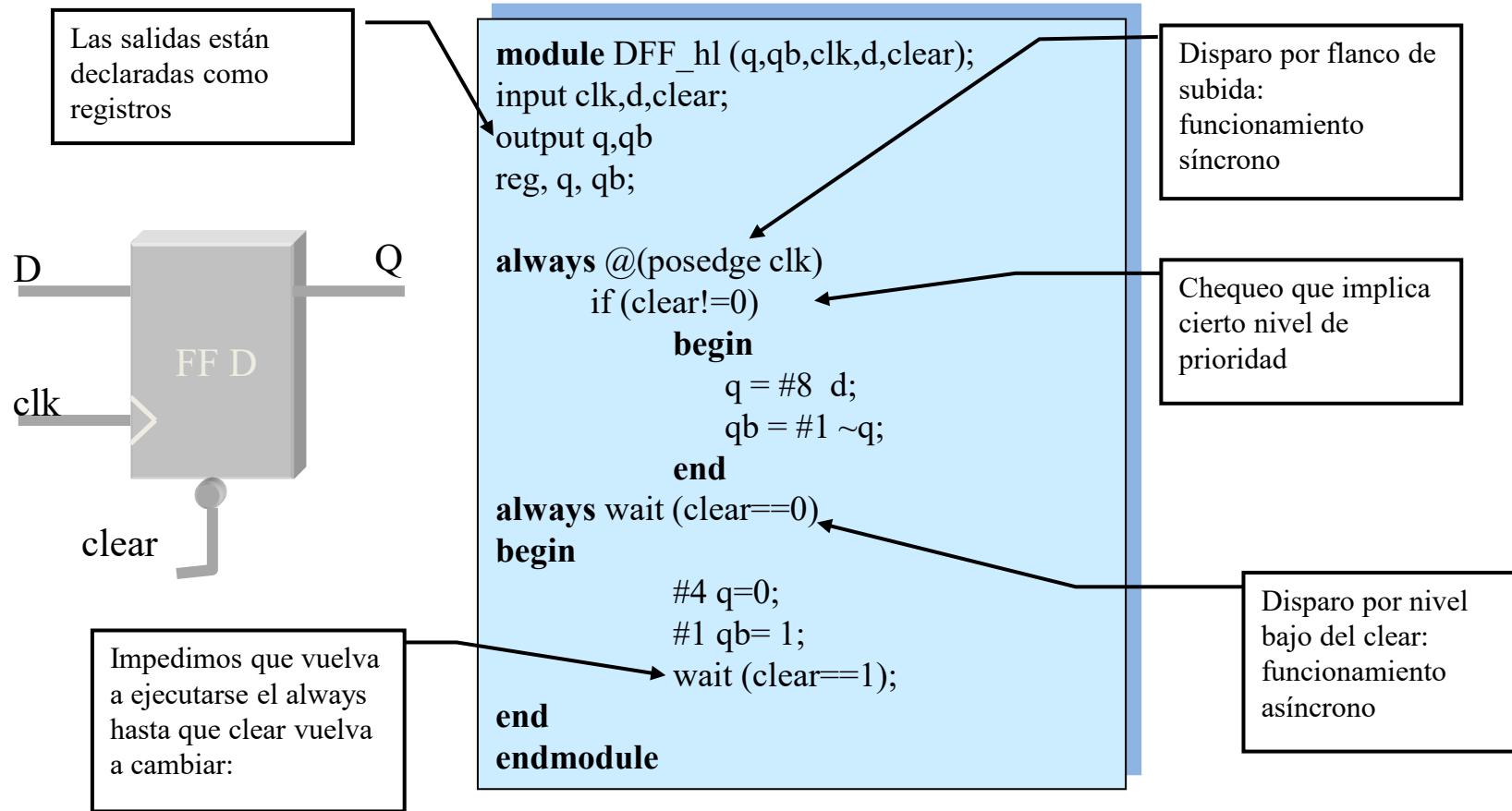
Se almacena temporalmente el valor de **y** →
1 *timestep* después de realiza la asignación a **x**

Verilog no sintetizable

55

Sistemas Digitales Programables

□ Ejemplo: Descripción Behavioral de un DFF (NO SINTETIZABLE)



Concurrencia

56

Sistemas Digitales Programables

□ CONCURRENCIA

□ Controles Temporales aplicados a Concurrencia

Control por Eventos Nombrados

```
event received_data;  
  
always @(posedge clk)  
    if (last_dat_packet)  
        ->received_data;  
  
always @(*(received_data))  
    data_buf={data_pkt[0],datapkt[1]};
```

Se declara un tipo *event* que puede ser lanzado voluntariamente

Un bloque procedural puede ser sensible al evento

Concurrencia

57

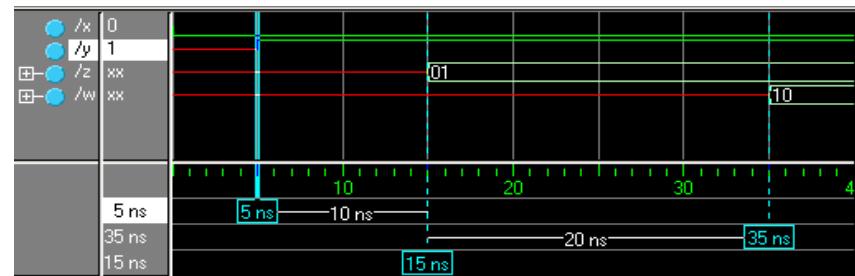
Sistemas Digitales Programables

CONCURRENCIA

Bloques de Ejecución Paralela (FORK / JOIN)

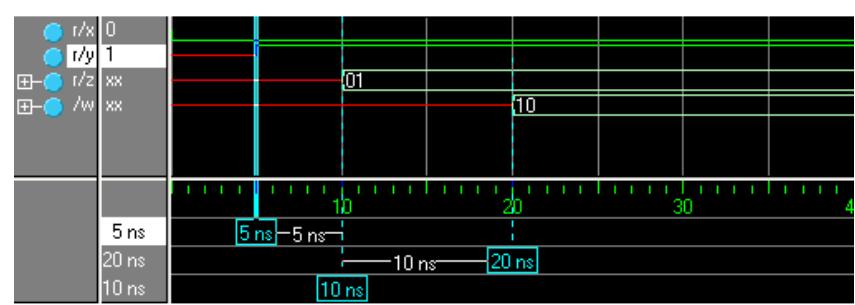
- Dentro de los bloques *BEGIN / END* la ejecución es **SECUENCIAL**

```
initial
begin
    x=1'b0;
    #5      y=1'b1;
    #10     z={x,y};
    #20     w={y,x};
end
```



- Los bloques FORK / JOIN permiten una ejecución paralela de sus sentencias con control temporal y retardos relativos al tiempo de inicio del bloque *

```
initial
fork
    x=1'b0;
    #5      y=1'b1;
    #10     z={x,y};
    #20     w={y,x};
join
```



* Será necesario vigilar la aparición de condiciones de carrera

Concurrencia

58

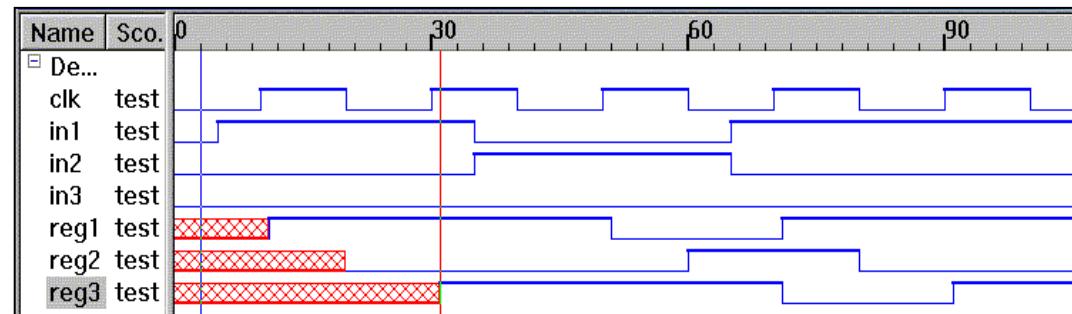
Sistemas Digitales Programables

CONCURRENCIA

Asignaciones Blocking vs. Non - Blocking

- Una asignación **Blocking** (=) se realiza de forma instantánea
- Una asignación **Non-Blocking** (<=) no bloquea la ejecución y se realiza en dos fases:
 - 1º Se evalua RHS de **todas** las asignaciones / 2º Se realizan la asignaciones

```
always @(posedge clk)
begin
    reg1 <= #1 in1;
    reg2 <= @(negedge clk) in2^in3;
    reg3 <= #1 reg1;
end
```



Almacena el valor anterior de reg1

Concurrencia

59

Sistemas Digitales Programables

CONCURRENCIA

Blocking vs. Non – Blocking. Eliminación de Condiciones de Carrera

```
reg d1, d2, d3, d4;  
  
always @(posedge clk) d2 = d1;  
always @(posedge clk) d3 = d2;  
always @(posedge clk) d4 = d3;
```

Resultado Impredecible

```
reg d1, d2, d3, d4;
```

```
always @(posedge clk) d2 <= d1;  
always @(posedge clk) d3 <= d2;  
always @(posedge clk) d4 <= d3;
```

Segundo se
actualizan d2,
d3 y d4

Primero se
almacenan los
valores
anteriores de
d1,d2,d3

Concurrencia

60

Sistemas Digitales Programables

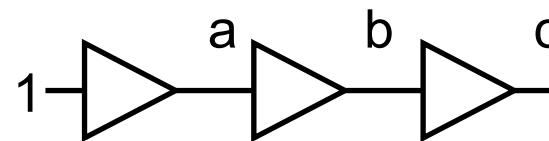
□ CONCURRENCIA

□ Blocking vs. Non – Blocking. Ejemplos

$a = 1;$

$b = a;$

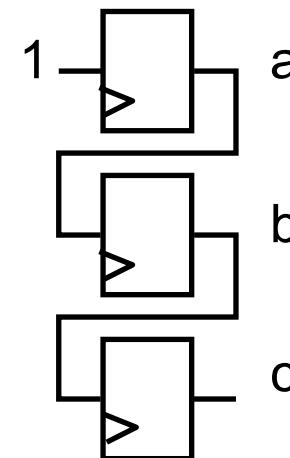
$c = b;$



$a \leq 1;$

$b \leq a;$

$c \leq b;$



Concurrencia

61

Sistemas Digitales Programables

```
reg x,y,z;
reg [7:0] rega,regb;
integer count;
initial
begin
    x=0;y=1;z=1;
    count=0;
    rega=8'b0;regb=rega;
    #15 rega[2]=1'b1;
    #10 regb[7:5]={x,y,z};
    count=count+1;
end
```

```
reg x,y,z;
reg [7:0] rega,regb;
integer count;
initial
begin
    x=0;y=1;z=1;
    count=0;
    rega=8'b0; regb=rega;
    rega[2] <= #15 1'b1;
    regb[7:5] <= #10 {x,y,z};
    count <= count+1;
end
```

	T = 0	T = 10	T = 15	T = 20	T = 25
x	0	1			
y	1				
z	1				
rega	0		2		
regb	0		1'b1		
count					count+1;
regb[7:5]					= {x,y,z};

Concurrencia

62

Sistemas Digitales Programables

□ CONCURRENCIA

□ Blocking vs. Non – Blocking. Ejemplos

```
a=0; b=1; c=2  
  
a = #5 b + c;  
d = a;
```

T = 0

```
a=0;  
b=1;  
c=2;
```

T = 5

```
a = 3  
d = 3
```

```
a=0; b=1; c=2  
  
a <= #5 b + c;  
d = a;
```

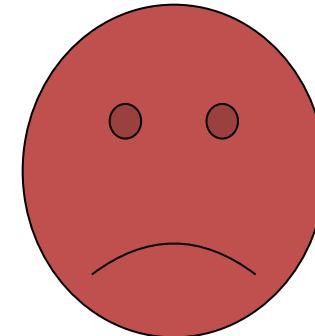
```
a=0;  
b=1;  
c=2;  
d=0;
```

```
a = 3
```

Modelo de un registro de desplazamiento

Sistemas Digitales Programables

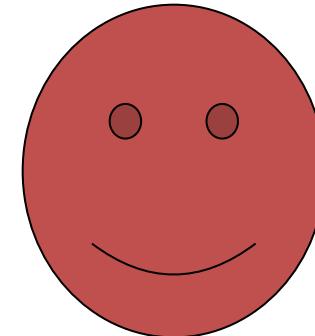
```
module shift_register
    input clk;
    input reset;
    input datain;
    output dataout;
    reg [3:0] shift;
    always @(posedge clk)
        if (!reset)
            shift=0;
        else
            begin
                shift[3]= datain;
                shift[2]=shift[3];
                shift[1]=shift[2];
                shift[0]=shift[1];
            end
        assign dataout=shift[0];
endmodule
```



Modelo de un registro de desplazamiento

Sistemas Digitales Programables

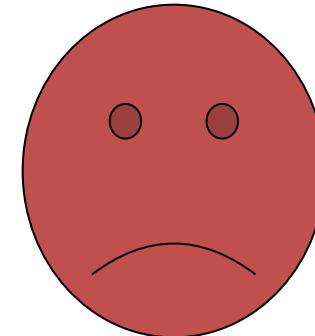
```
module shift_register
    input clk;
    input reset;
    input datain;
    output dataout;
    reg [3:0] shift;
    always @(posedge clk)
        if (!reset)
            shift=0;
        else
            begin
                shift[0]=shift[1];
                shift[1]=shift[2];
                shift[2]=shift[3];
                shift[3]= datain;
            end
        assign dataout=shift[0];
endmodule
```



Modelo de un registro de desplazamiento

Sistemas Digitales Programables

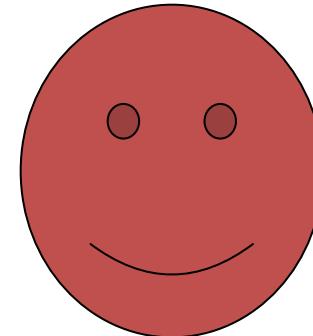
```
module shift_register
    input clk;
    input reset;
    input datain;
    output dataout;
    reg [3:0] shift;
    always @(posedge clk)
        if (!reset)
            shift=0;
        else
            fork
                shift[0]=shift[1];
                shift[1]=shift[2];
                shift[2]=shift[3];
                shift[3]= datain;
            join
    assign dataout=shift[0];
endmodule
```



Modelo de un registro de desplazamiento

Sistemas Digitales Programables

```
module shift_register
    input clk;
    input reset;
    input datain;
    output dataout;
    reg [3:0] shift;
    always @(posedge clk)
        if (!reset)
            shift=0;
        else
            fork
                shift[0]<=shift[1];
                shift[1]<=shift[2];
                shift[2]<=shift[3];
                shift[3]<=datain;
            join
    assign dataout=shift[0];
endmodule
```



Concurrencia

67

Sistemas Digitales Programables

□ CONCURRENCIA

□ Bloques Nombrados

begin:<nombre>

...

end

fork:<nombre>

...

join

- Los bloques nombrados forman parte de la jerarquía del diseño
- Pueden ser **DESHABILITADOS**

Concurrencia

68

Sistemas Digitales Programables

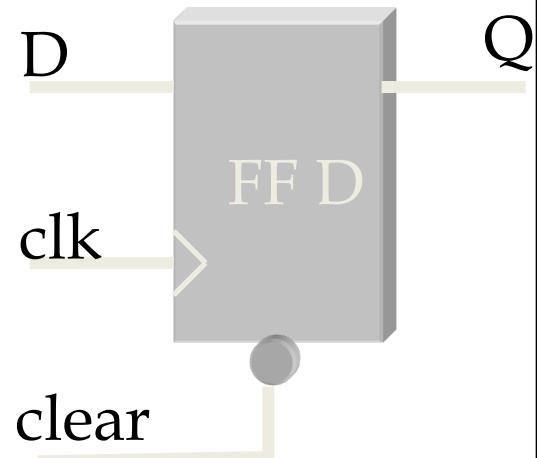
□ Bloques Nombrados. Ejemplos

```
module monoestable(trigger,out);
    output out; input trigger; reg out;
initial out=0;
always (@posedge trigger)
begin
    disable time_out;
    out=1;
    end;
always
begin:time_out
#50 out=0;
end
endmodule
```

```
module find_true_bit;
reg [15:0] flag;
integer i;
initial
begin
flag = 16'b 0010_0000_0000_0000;
i = 0;
begin: block1
while(i < 16)
begin
if (flag[i])
begin
$display("Encontrado un 1 en posicion %d", i);
 disable block1;
end // if
i = i + 1;
end // while
end // block1
end //initial
endmodule
```

Modelización behavioral: Un ejemplo simple.

Las salidas están declaradas como registros



Impedimos que vuelva a ejecutarse el always hasta que clear vuelva a cambiar:

```
module DFF
(q,qb,clk,d,clear);
input clk,d,clear;
output q,qb;
reg , q, qb;

always @(posedge clk)
if (clear!=0)
begin : sincrono
q= #8 d;
qb= #1 ~q;
end
always wait (clear==0)
Begin
 disable sincrono;
#4 q=0;
#1 qb= 1;
wait (clear==1);
end
endmodule
```

Disparo por flanco de subida: funcionamiento síncrono

Chequeo que implica cierto nivel de prioridad

Disparo por nivel bajo del clear: funcionamiento asíncrono

Verilog para síntesis

70

Sistemas Digitales Programables

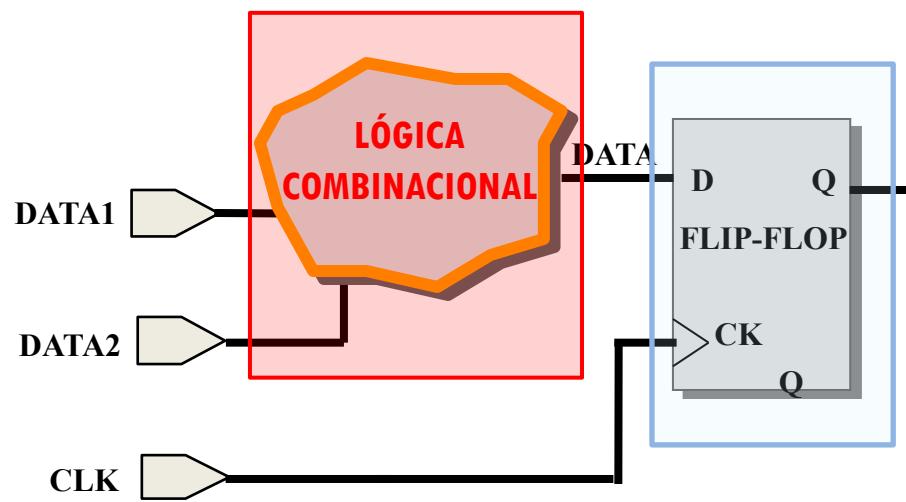
□ Verilog para Síntesis (Verilog – RTL)

□ Recomendaciones para mejorar el proceso de síntesis.

- SEPARAR LÓGICA COMBINACIONAL Y
SECUENCIAL

```
always @ (data1 or data2)
begin: combinacional
  data=COMBI(data1,data2);
end
```

```
always @ (posedge clk)
begin: secuencial
  q<=data;
end
```



Modelización Circuitos Combinacionales

71

Sistemas Digitales Programables

- Característica fundamental: Las salidas cambian cuando lo hacen las entradas.
- Ejemplos: Multiplexores, decodificadores, codificadores, demultiplexores, comparadores, sumadores, ...
- Recomendaciones:
 - Utilizar bloques procedurales *always*.
 - Lista de sensibilidad completa con todas las señales de entrada.
 - Utilizar asignaciones bloqueantes (*blocking =*).
 - Asignar valor a todas las señales de salida en todos los caminos condicionales.

Verilog para síntesis

72

Sistemas Digitales Programables

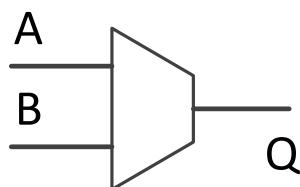
□ Verilog para Síntesis (Verilog – RTL)

□ Modelado de Circuitos Combinacionales

ASIGNACIONES CONCURRENTES

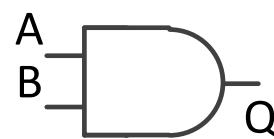
```
module MUX_mod(Q,A,B);
output Q;
input a,b;

assign Q=sel?B:A;
endmodule
```



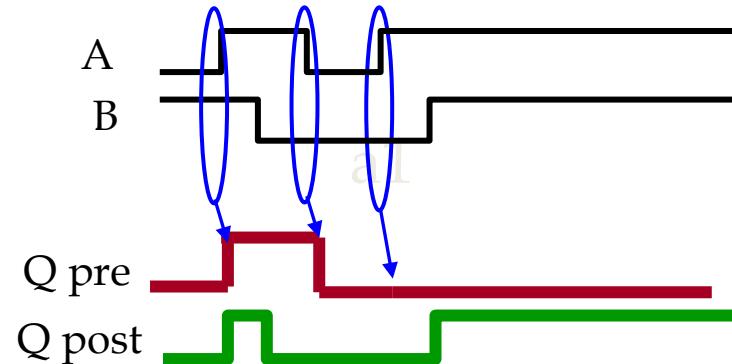
BLOQUES PROC. COMBINACIONALES

```
module and2(Q,A,B);
output Q;
input a,b;
reg Q;
always @(A or B)
  Q=A&B;
endmodule
```



Incluir todas las señales de entrada en la sensibilidad

```
module and2(Q,A,B);
output Q; input a,b;
reg Q;
always @(A)
  Q=A&B;
endmodule
```



Verilog para síntesis

73

Sistemas Digitales Programables

□ Verilog para Síntesis (Verilog – RTL)

□ Modelado de Circuitos Combinacionales

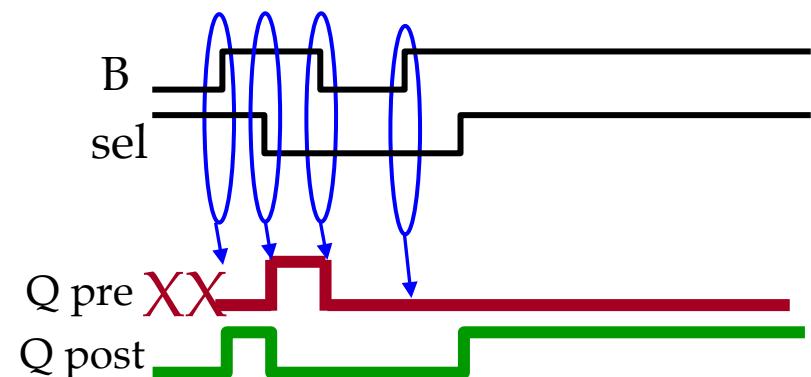
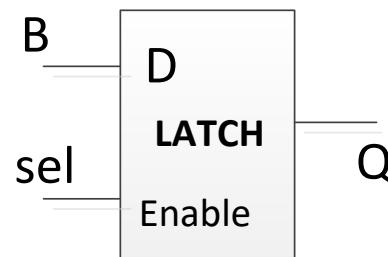
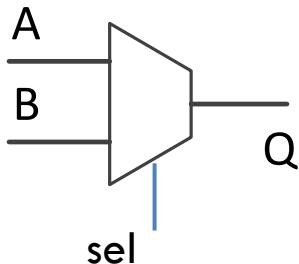
BLOQUES PROCEDURALES CON IF

```
reg Q;  
always @(A or B or sel)  
if (sel) Q=B;  
else Q=A;
```

```
reg Q;  
always @(A or B or sel)  
if (sel) Q=B;
```

```
reg Q,temp;  
always @(B or sel)  
begin  
    Q=temp;  
    temp=1'b0;  
    if (sel) temp=B;  
end
```

Evita la
aparición de
latches pero
cuidado con la
simulación

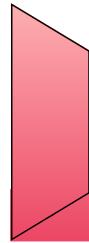


Ejemplos: Combinacionales

74

Sistemas Digitales Programables

```
/*multiplexor 2 a 1*/
module mux(A, B, sel, Q);
input A, B, sel;
output Q;
reg Q;
always @ (A or B or sel)
    if (sel) Q = B;
    else Q = A;
endmodule
```



Incluir todas las señales de entrada en la sensibilidad

```
/*decoder 2 a 4*/
module decoder (bin_in, dec_out, enable);
input [1:0] bin_in;
input enable;
output [3:0] dec_out;
reg [3:0] dec_out;
always @ (enable or bin_in)
    if (enable)
        case (bin_in)
            2'h0 : dec_out = 4'b0001;
            2'h1 : dec_out = 4'b0010;
            2'h2 : dec_out = 4'b0100;
            2'h3 : dec_out = 4'b1000;
            default : dec_out = 0;
        endcase
    else dec_out = 0;
endmodule
```

Verilog para síntesis

75

Sistemas Digitales Programables

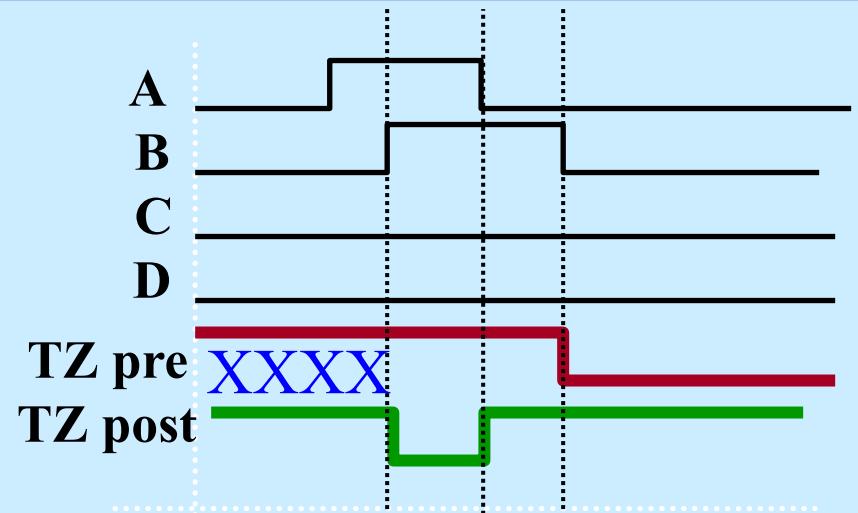
□ Verilog para Síntesis (Verilog – RTL)

□ Recomendaciones para mejorar el proceso de síntesis.

- USO DE ASIGNACIONES BLOCKING EN BLOQUES COMBINACIONALES

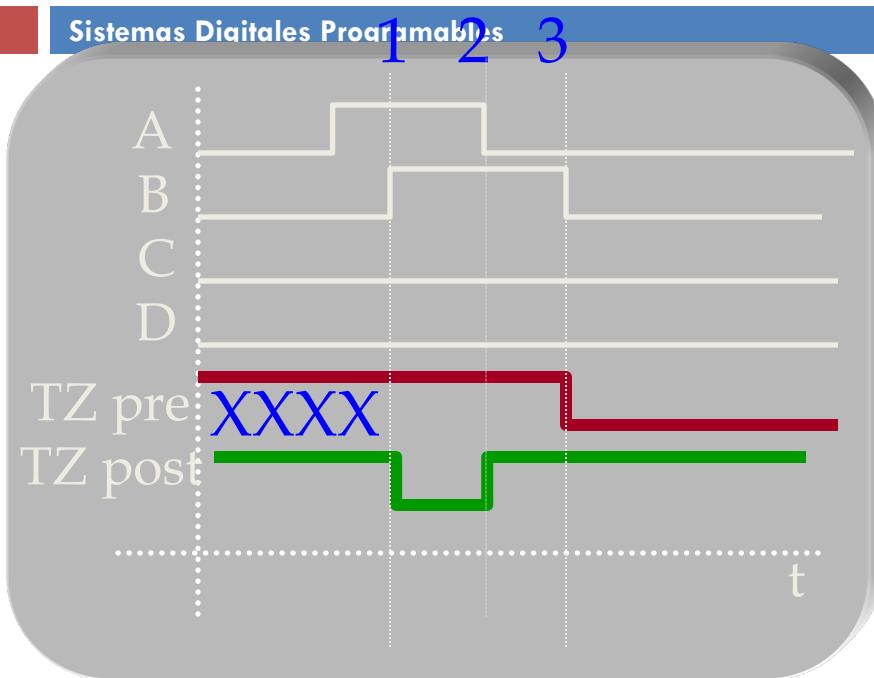
```
reg TM, TN, TO, TZ;  
always @((A or B or  
          C or D ))  
begin: combinacional  
    TM= A&B;  
    TN= C&D;  
    TO=TM | TN;  
    TZ=!TO;  
end
```

```
reg TM, TN, TO, TZ;  
always @((A or B or  
          C or D ))  
begin: combinacional  
    TM<= A&B;  
    TN<= C&D;  
    TO<=TM | TN;  
    TZ<=!TO;  
end
```



El resultado de la síntesis es el mismo pero la simulación no

Verilog: Uso de Blocking assignments



```
reg TM, TN, TO, TZ;  
always @(A or B or C or D)  
begin: combinacional  
    TM<= A&B;  
    TN<= C&D;  
    TO<=TM | TN;  
    TZ<=!TO;  
end
```

- Ambas descripciones sintetizan el mismo circuito
- Sin embargo en la solución B existirá una discrepancia entre la simulación antes y después de la síntesis

Modelización Circuitos Secuenciales

77

Sistemas Digitales Programables

- Característica fundamental: Las salidas cambian con el flanco activo de la señal de reloj.
- Pueden tener una señal asíncrona, mayoritariamente un reset.
- Ejemplos: Registros, Contadores, Registros de Desplazamiento, Maquinas de Estados Finitos (FSM), ...
- Recomendaciones:
 - Utilizar bloques procedurales `always`.
 - Lista de sensibilidad: la señal de reloj indicando el flanco activo y si es necesaria una señal asíncrona.
 - Utilizar asignaciones no bloqueantes (Non-blocking `<=`).
 - Se infiere un registro por cada señal.

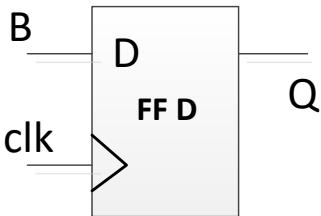
Verilog para síntesis

78

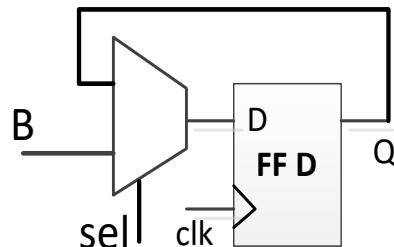
Sistemas Digitales Programables

- Verilog para Síntesis (Verilog – RTL)
- Modelado de Circuitos Secuenciales

```
reg Q;  
always @(posedge clk)  
  Q <= B;
```

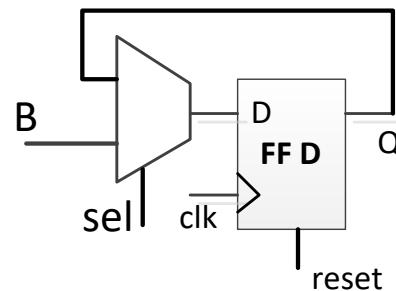


```
reg Q;  
always @(posedge clk)  
  if (sel)  
    Q <= B;
```



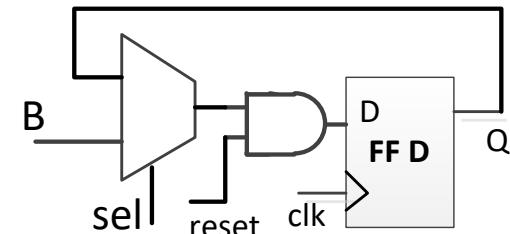
RESET ASÍNCRONO

```
reg SAL;  
always  
  @(posedge clk or  
    negedge reset)  
  if(!reset)  
    SAL<=1'b0;  
  else if (sel)  
    SAL<=B;
```



RESET SÍNCRONO

```
reg SAL;  
always  
  @(posedge clk)  
  if(!reset)  
    SAL<=1'b0;  
  else if (sel)  
    SAL<=B;
```



Verilog para síntesis

79

Sistemas Digitales Programables

□ Verilog para Síntesis (Verilog – RTL)

□ Recomendaciones para mejorar el proceso de síntesis.

- El uso de NON-BLOCKING dentro de un bloque procedural sensible a reloj infiere registros.
- Emplear asignaciones continuas si se desea evitar la inferencia de registros
- **NORMA GENERAL:** Usar NON-BLOCKING dentro de bloques secuenciales y extraer la información al exterior mediante asignaciones continuas o bloques combinacionales

```
reg [3:0] curr_state;
always @(posedge clk)
begin: secuencial
    curr_state<=next_state;
end
assign salida=curr_state;
endmodule
```

4
REGISTROS

```
reg [3:0] curr_state;
always @(posedge clk)
begin: secuencial
    curr_state<=next_state;
    salida<=curr_state;
end
endmodule
```

8
REGISTROS

Ejemplo: Registro 4 bits

80

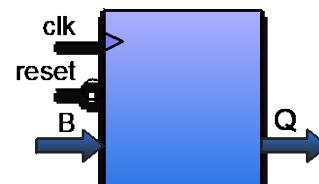
Sistemas Digitales Programables

```
/*Registro 4 bits, reset asincrono*/
module    reg4bits(clk, reset, B, Q);

input      clk, reset;
input      [3:0] B;
output     [3:0] Q;

reg       [3:0] Q;

always @(posedge clk or negedge reset)
  if (!reset)
    Q <= 4'b0000;
  else
    Q <= B;
endmodule
```



¿Cuántos flip-flops se infieren?

- a) 4
- b) 8
- c) 1

Ejemplo: Registro n bits

81

Sistemas Digitales Programables

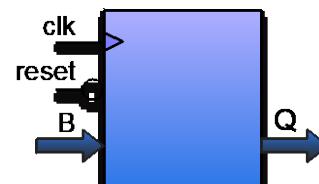
```
/* Registro n bits, reset asincrono*/
module    regnbits(clk, reset, B, Q);

parameter          n = 8;

input              clk, reset;
input [n-1:0] B;
output [n-1:0] Q;

reg [n-1:0] Q;

always @(posedge clk or negedge reset)
  if (!reset)
    Q <= {n{1'b0}};
  else
    Q <= B;
endmodule
```



¿Cuántos flip-flops se infieren?

- a) n
- b) 4
- c) 1

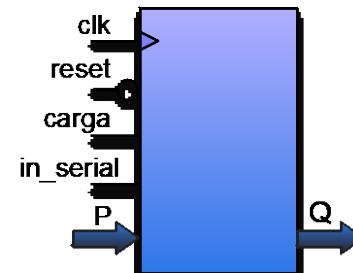
- Reúso de código Verilog HDL.
- Realización de diseños parametrizables.
- Se realiza todo en función de un parámetro, que fija el tamaño del registro.

Ejemplo: Registro desplazamiento 4 bits

82

Sistemas Digitales Programables

```
/*Registro 4 bits, reset asincrono*/
module    reg4shift(clk, reset, carga,
in_serial, P, Q);
input        clk, reset, carga,in_serial;
input      [3:0] P;
output reg [3:0] Q;
always @(posedge clk or negedge reset)
  if (!reset)
    Q <= 4'b0000;
  else
    if (carga == 1'b1)
      Q <= P;
    else
      begin
        Q[0] <= Q[1]; Q[1] <= Q[2];
        Q[2] <= Q[3]; Q[3] <= in_serial;
      end
endmodule
```



¿Cuántos flip-flops se infieren?

- a) 4
- b) 8
- c) 1

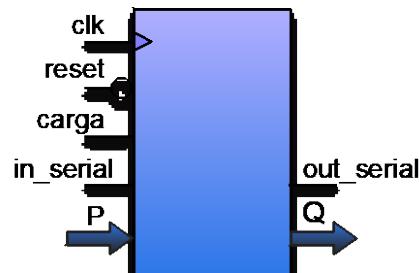
- Una sola estructura “if-else”, con anidaciones.
- Las señales de entrada se evalúan de forma anidada.
- El desplazamiento se realiza a cada ciclo de reloj.
- Habría que añadir un “enable”.

Ejemplo: Registro desplazamiento 4 bits

83

Sistemas Digitales Programables

```
/*Registro 4 bits, salida serie*/  
  
assign out_serial = Q[0];  
  
always @ (posedge clk or negedge reset)  
    if (!reset)  
        Q <= 4'b0000;  
    else  
        if (carga == 1'b1)  
            Q <= P;  
        else  
            begin  
                Q[0] <= Q[1];  
                Q[1] <= Q[2];  
                Q[2] <= Q[3];  
                Q[3] <= in_serial;  
            end  
endmodule
```



¿Cuántos flip-flops se infieren?

- a) 4
- b) 8
- c) 5

- La salida serie, es la misma señal que el bit de menor peso.
- A la señal de salida serie no se le debe asignar valor dentro del proceso always.
- Se le asigna valor a la salida serie, mediante una asignación continua. Concurrencia.

Ejemplos: Registro desplazamiento 4 bits

84

Sistemas Digitales Programables

- Formas fácilmente parametrizables.

```
/* Registro 4 bits, concatenación */

assign out_serial = Q[0];

always @(posedge clk or negedge reset)
  if (!reset)
    Q <= 0;
  else
    if (carga == 1'b1)
      Q <= P;
    else
      Q[3:0] <= {in_serial, Q[3:1]};
endmodule
```

```
/* Registro 4 bits, bucle for */

assign out_serial = Q[0];

always @(posedge clk or negedge reset)
  if (!reset)
    Q <= 0;
  else
    if (carga == 1'b1)
      Q <= P;
    else
      begin
        for (k = 0; k < 4; k = k + 1)
          Q[k] <= Q[k+1];
        Q[3] <= in_serial;
      end
endmodule
```

Ejemplos: Registro desplazamiento n bits

- Formas fácilmente parametrizables.

```
/* Registro n bits, concatenación */

parameter n = 4;
assign out_serial = Q[0];

always @(posedge clk or negedge reset)
  if (!reset)
    Q <= 0;
  else
    if (carga == 1'b1)
      Q <= P;
    else
      Q[n-1:0] <= {in_serial, Q[n-1:1]};
endmodule
```

```
/* Registro n bits, bucle for */

parameter n = 4;
assign out_serial = Q[0];

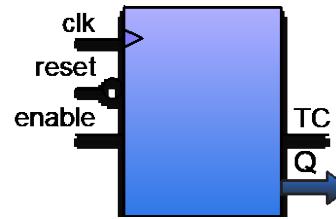
always @(posedge clk or negedge reset)
  if (!reset)
    Q <= 0;
  else
    if (carga == 1'b1)
      Q <= P;
    else
      begin
        for (k = 0; k < n; k = k + 1)
          Q[k] <= Q[k+1];
        Q[n-1] <= in_serial;
      end
endmodule
```

Ejemplo: Contador 4 bits

86

Sistemas Digitales Programables

```
/*Contador 4 bits, reset asincrono*/
module cont4bits(clk, reset, enable, Q, TC);
input clk, reset, enable;
output reg [3:0] Q;
output TC;
always @(posedge clk or negedge reset)
if (!reset)
    Q <= 4'b0000;
else
    if (enable == 1'b1)
        if (Q == 4'b1111)
            Q <= 4'b0000;
        else
            Q <= Q + 1;
    else
        Q <= Q;
assign TC = (Q == 4'b1111) ? 1'b1 : 1'b0;
endmodule
```



¿Cuántos flip-flops se infieren?

- a) 4
- b) 8
- c) 5

- Una sola estructura “if-else”, con anidaciones.
- Las señales de entrada se evalúan de forma anidada.
- Se evalúa cuando el contador llega a la última cuenta.
- La señal TC se genera fuera del always.

Ejemplo Incorrecto: Contador 4 bits

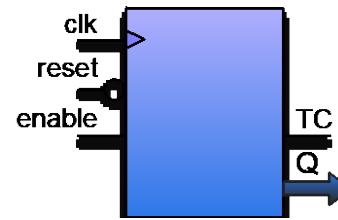
87

Sistemas Digitales Programables

```
/*Contador 4 bits, reset asincrono*/
module cont4bits(clk, reset, enable, Q, TC);
    input clk, reset, enable;
    output reg [3:0] Q;
    output reg TC;
    always @(posedge clk or negedge reset)
        if (!reset)
            Q <= 4'b0000;
        else
            if (enable == 1'b1)
                if (Q == 4'b1111)
                    begin
                        Q <= 4'b0000;
                        TC <= 1'b1;
                    end
                else Q <= Q + 1;
            else Q <= Q;
endmodule
```

4

Incorrecto



¿Cuántos flip-flops se infieren?

- a) 4
- b) 8
- c) 5

- TC se genera un ciclo de reloj tarde.
- TC no tiene valor en todas las condiciones.
- TC nunca se pone a cero lógico.
- Se infiere un flip-flop más.

Ejemplo: Contador Parametrizable

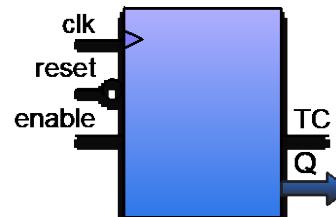
88

Sistemas Digitales Programables

```
/*Contador 4 bits, reset asincrono*/
module cont4bits(clk, reset, enable, Q, TC);
parameter M = 16; //Modulo
parameter N = $clog2(M-1); //Número de bits
input clk, reset, enable;
output reg [N-1:0] Q;
output TC;

always @(posedge clk or negedge reset)
if (!reset)
    Q <= {N{1'b0}};
else
    if (enable == 1'b1)
        if (Q == M-1)
            Q <= {N{1'b0}};
        else
            Q <= Q + 1;
assign TC = (Q == M-1) ? 1'b1 : 1'b0;

endmodule
```



¿Cuántos flip-flops se infieren?

- a) N
- b) M
- c) M-1

- Se estable el número de cuentas a partir del parámetro M .
- La función $\$clog2$ permite obtener el número de bits necesarios para poder alcanzar la cuenta de $M-1$.

Verilog para verificación

89

Sistemas Digitales Programables

□ Ejemplo: TestBench DFF

```
// Simple Test Bench for a Flip Flop  
`timescale 1 ns/ 100 ps
```

```
module tb_Flip_Flop();  
  // General Parameters  
  localparam T=5;  
  // Wire & Regs declaration  
  reg CLK;    wire Q, Qb;  reg D, CLR;  
  
  DFF_hl DUV(.clk(CLK),.d(D),  
             .clear(CLR), .q(Q), .qb(Qb));
```

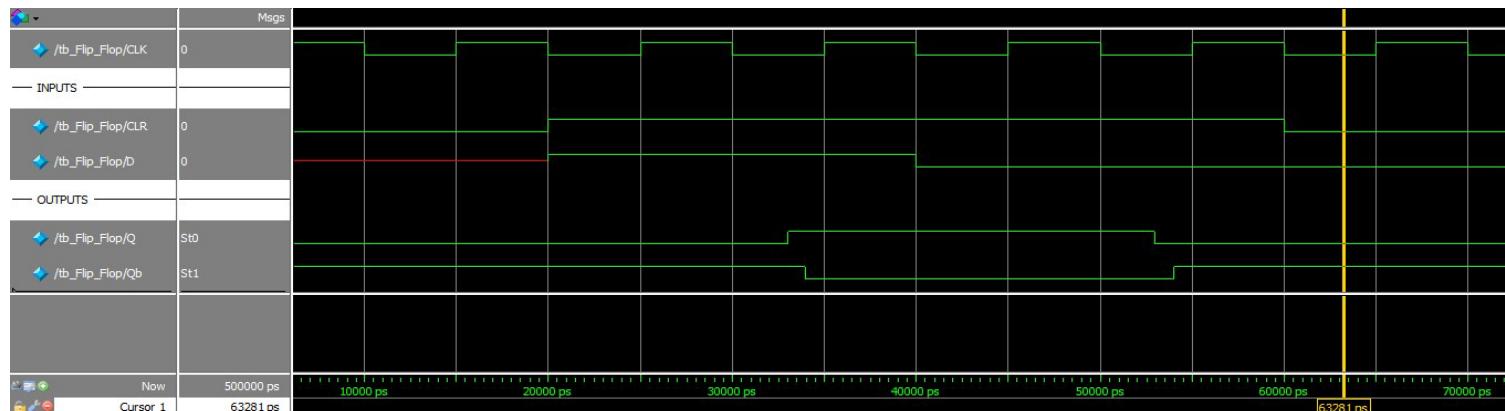
Declaración de parámetros y wires/reg para conexiones

Instanciación DUV

```
initial  
begin  
#0 CLK = 0;  
// Power On Reset operation  
CLR = 1'b0; #(T*4);  
// Release CLR signal and input D=1  
CLR = 1'b1; D = 1'b1; #(T*4);  
// Input D=0  
D = 1'b0; #(T*4);  
// Activate CLR  
CLR = 1'b0; #(T*4);  
end  
  
always #T CLK = ~CLK;  
endmodule
```

Casos de Verificación

Generación de Reloj



Verilog para verificación

90

Sistemas Digitales Programables

```
module tb_FF_2();
// General Parameters
localparam T=5;
// Wire & Regs declaration
reg CLK; wire Q, Qb; reg D, RST; reg EN;
DFFAR_rtl DUV(.clk(CLK),.d(D), .sel(EN), .reset(RST), .q(Q), .qb(Qb),);
```

```
initial
begin
#0 CLK = 1;
// Power On Reset operation
D = 1'b0; EN = 1'b0;
RESET();

// Release RST signal and input D=1
DATA(1'b1);
EN = 1'b1;

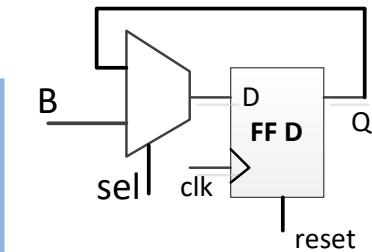
// Input D=0
WAIT_n(4);
DATA(1'b0);

// Turn off ENABLE
EN = 1'b0;
WAIT_n(2);
@(negedge CLK) EN = 1'b1;
end
```

```
task DATA;
input d_bit;
begin
@(negedge CLK); D = d_bit;
end
endtask

task WAIT_n;
input integer n;
begin
repeat(n) @(posedge CLK);
end
endtask

task RESET;
begin
RST=1'b0;
repeat(2) @(negedge CLK);
RST=1'b1;
end
endtask
```



```
// Clock Generator
always
begin
#T CLK = ~CLK;
end
endmodule
```

□ Test Bench con Tasks

Verilog para verificación

91

Sistemas Digitales Programables

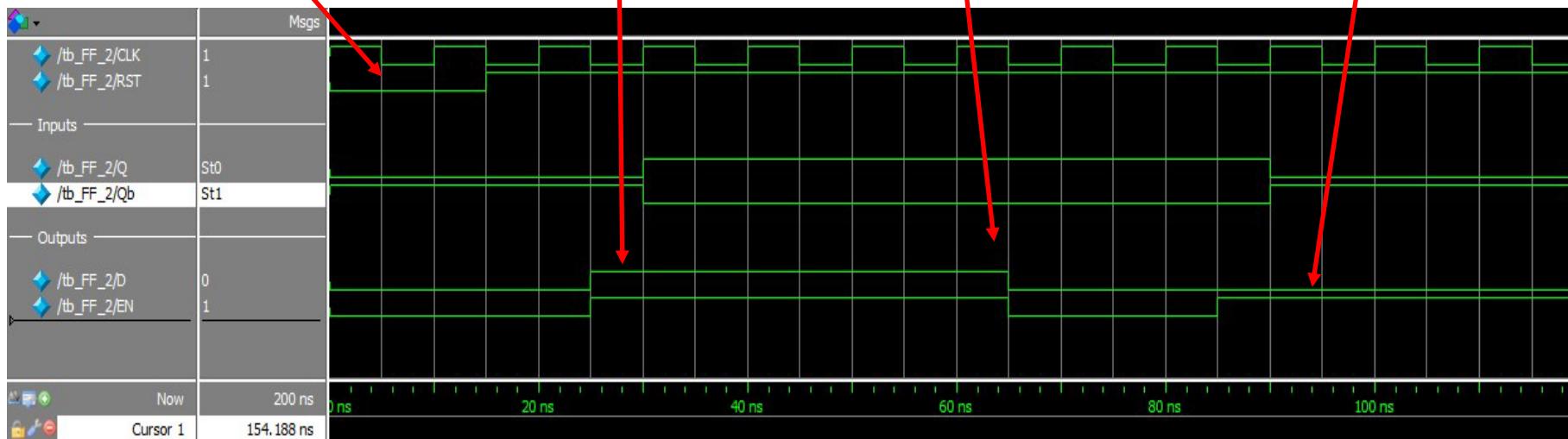
□ Test Bench con Tasks

```
initial
begin
#0 CLK = 1;
// Power On Reset operation
D = 1'b0; EN = 1'b0;
RESET();
```

```
// Release RST signal and
// input D=1
DATA(1'b1);
EN = 1'b1;
```

```
// Input D=0
WAIT_n(4);
DATA(1'b0);
```

```
// Turn off ENABLE
EN = 1'b0;
WAIT_n(2);
@(negedge CLK) EN = 1'b1;
end
```



Verilog HDL: Resumen

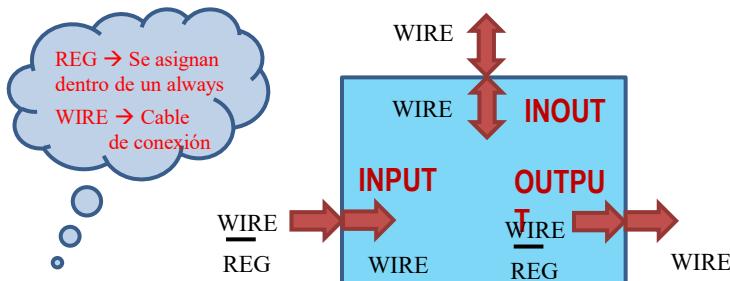
92

Sistemas Digitales Programables

□ Jerarquía

```
module <nombre_modulo> (<lista puertos>);  
<declaraciones>  
<elementos del módulo>  
endmodule
```

PUERTOS: Tipos y Reglas de conexión



```
module MODELO(CYC_O, GNT_n, GNT_n_next);
```

```
parameter masters = 4;  
  
input [(masters-1):0] CYC_O;  
input [(masters-1):0] GNT_n;  
output [(masters-1):0] GNT_n_next;  
// Definición de nodos  
reg [(masters-1):0] GNT_n_next;  
wire [3:0] conex;  
integer posicion_master=0;
```

// Bloques Procedurales

```
always @(CYC_O, GNT_n)  
begin  
...  
end
```

// Instanciaciones de Módulos

```
HalfAdder #(4,128) H1 ( a1, b1, c1, s1 );  
posicional  
HalfAdder #(.(delay(4)..bits(128))  
H2 ( .a(a2), .b(b2), .c(c2), .s(s2) );  
nombrada
```

Valores de los parámetros

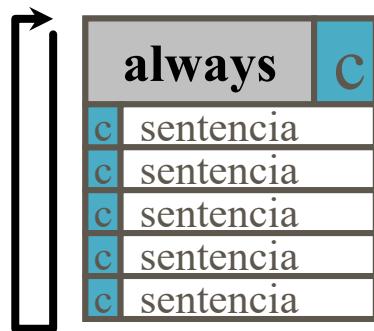
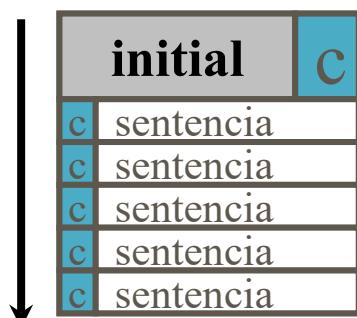
```
assign GNT_NEXT=GNT_n && CYC_O;  
endmodule
```

Verilog HDL: Resumen

93

Sistemas Digitales Programables

□ Elementos del Módulo. Bloques Procedurales



- Los Bloques Procedurales se ejecutan en **paralelo** entre si.
- Las sentencias internas de los BP se ejecutan secuencialmente.
- Las sentencias internas describen un comportamiento. Para ello se usan ...

