T1: HARDWARE DESCRIPTION LANGUAGES (MODULES)

SystemVerilog

```
module sillyfunction(input  logic a, b, c,
                     output logic y);

  assign y = ~a & ~b & ~c |
             a & ~b & ~c |
             a & ~b &  c;
endmodule
```

T1: HARDEWARE DESCRIPTION LANGUAGES

Un módulo comienza con un listado de las entradas y salidas. La sentencia assign describe la lógica combinacional. ~ indica NOT, & indica AND, y | indica OR.
Las señales lógicas como las entradas y salidas son variables booleanas (0 o 1). También pueden tener valores flotantes e indefinidos que se discutirán en la sección A.2.8.
El tipo lógico se introdujo en Systemverilog. Sustituye al tipo reg, que era una fuente perenne de confusión en Verilog. La lógica debe utilizarse en todas partes excep
to en las redes con múltiples controladores, como se explicará en la sección A.7.

T1: HARDEWARE DESCRIPTION LANGUAGES(MODULES)

SystemVerilog

```
module adder(input  logic [31:0] a,
             input  logic [31:0] b,
             output logic [31:0] y);

  assign y = a + b;
endmodule
```

Note that the inputs and outputs are 32-bit busses.

T1: HARDWARE DESCRIPTION LANGUAGES SIMULATION)

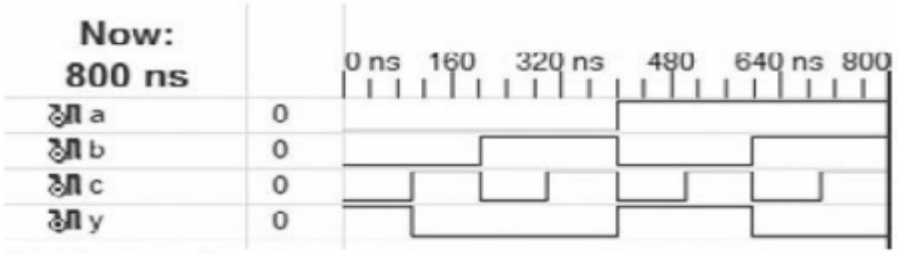

FIGURE A.1 Simulation waveforms

T1: HARDWARE DESCRIPTION LANGUAGES (SYNTHESIS)

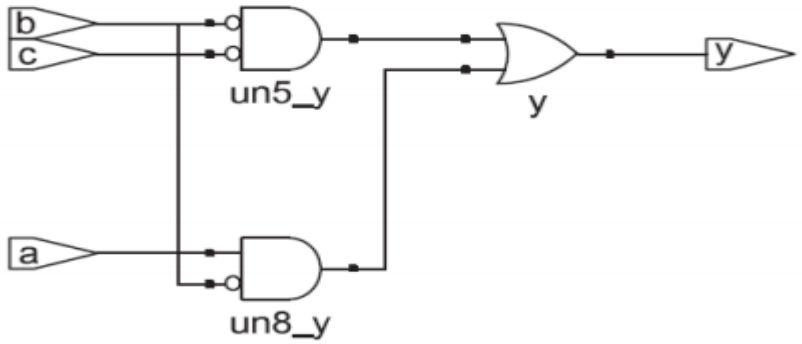

FIGURE A.2 Synthesized silly_function circuit

T1: HARDWARE DESCRIPTION LANGUAGES (SYNTHESIS)

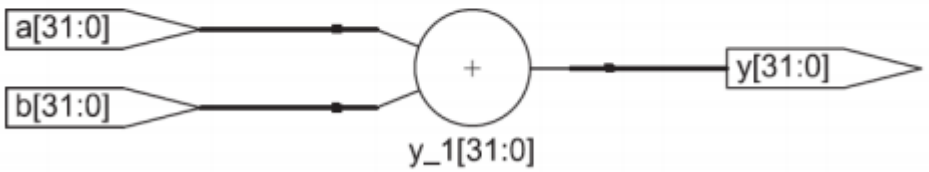

FIGURE A.3 Synthesized adder

T1: HARDWARE DESCRIPTION LANGUAGES(operadores combinados por bits)

SystemVerilog

```
module inv(input  logic [3:0] a,
           output logic [3:0] y);

  assign y = ~a;
endmodule
```

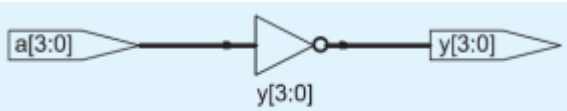

FIGURE A.4 inv

T1: HARDWARE DESCRIPTION LANGUAGES (combinational bit wise operators, comments)

SystemVerilog

```
module gates(input  logic [3:0] a, b,
             output logic [3:0] y1, y2,
                                y3, y4, y5);

   /* Five different two-input logic
      gates acting on 4 bit busses */
   assign y1 = a & b;    // AND
   assign y2 = a | b;    // OR
   assign y3 = a ^ b;    // XOR
   assign y4 = ~(a & b); // NAND
   assign y5 = ~(a | b); // NOR
endmodule
```

T1: HARDWARE DESCRIPTION LANGUAGES (combinational bit wise operators)

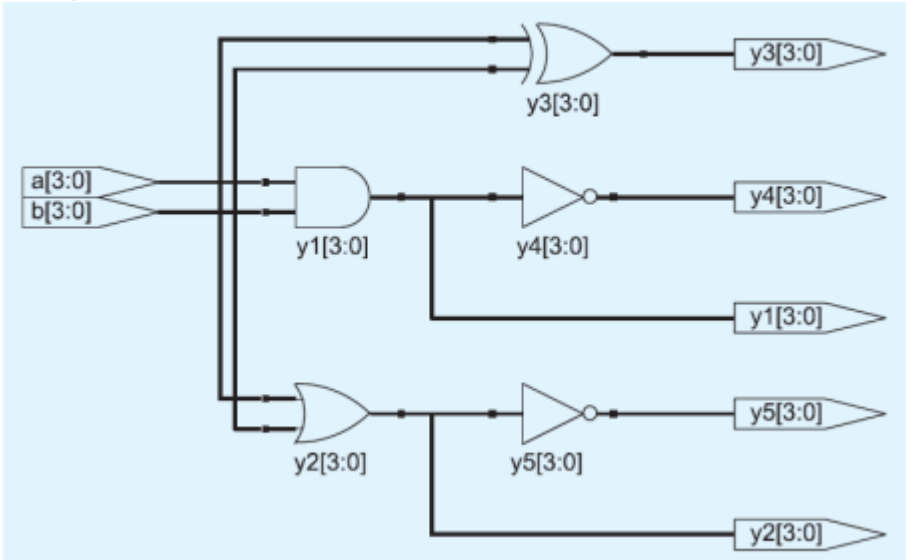

FIGURE A.5 Gates

T1: hardaware desciption languages(operadores de reducción / generar)

SystemVerilog

```
module and8(input  logic [7:0] a,
            output logic       y);

   assign y = &a;

   // &a is much easier to write than
   // assign y = a[7] & a[6] & a[5] & a[4] &
   //            a[3] & a[2] & a[1] & a[0];
endmodule
```

Como es de esperar, los operadores de reducción |, ^,~& y ~| también están disponibles para OR, XOR, NAND y NOR. Recordemos que un XOR multientrada realiza la paridad, devolviendo TRUE si un número impar de entradas son TRUE.

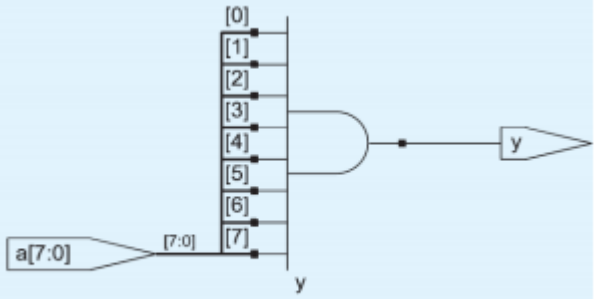

FIGURE A.6 and8

T1: HARDWARE DESCRIPTION LANGUAGES (Conditional Assignments)

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
  port(d0, d1:in  STD_LOGIC_VECTOR(3 downto 0);
       s:     in  STD_LOGIC;
       y:     out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2 is
begin
  y <= d0 when s = '0' else d1;
end;
```

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);

   assign y = s ? d1 : d0;
endmodule
```

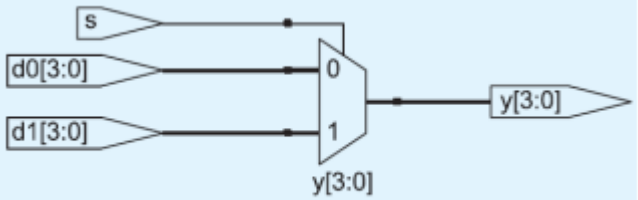

FIGURE A.7 mux2

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mux4 is
  port(d0, d1,
    d2, d3: in  STD_LOGIC_VECTOR(3 downto 0);
    s:      in  STD_LOGIC_VECTOR(1 downto 0);
    y:      out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth1 of mux4 is
begin
  y <= d0 when s = "00" else
       d1 when s = "01" else
       d2 when s = "10" else
       d3;
end;
```

```
architecture synth2 of mux4 is
begin
  with s select y <=
    d0 when "00",
    d1 when "01",
    d2 when "10",
    d3 when others;
end;
```

```
module mux4(input  logic [3:0] d0, d1, d2, d3,
            input  logic [1:0] s,
            output logic [3:0] y);

   assign y = s[1] ? (s[0] ? d3 : d2)
                   : (s[0] ? d1 : d0);
endmodule
```

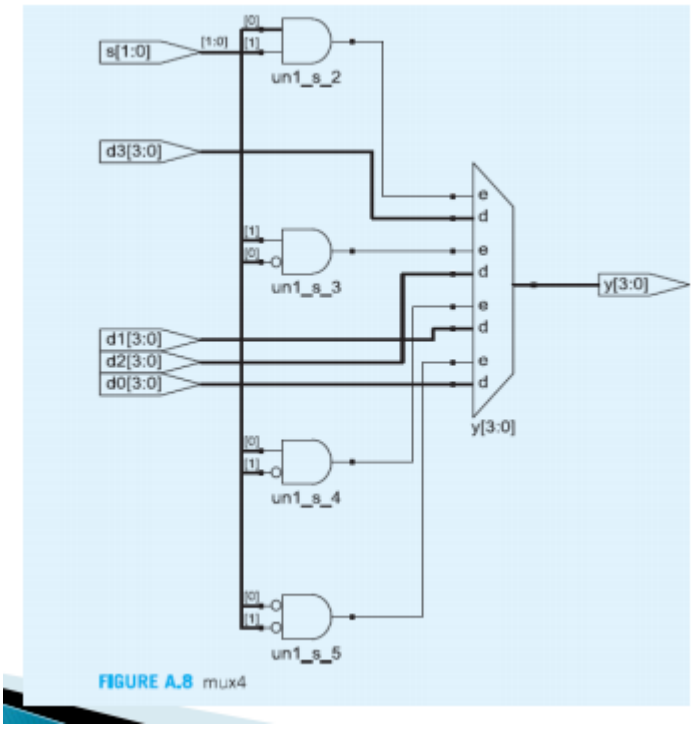

FIGURE A.8 mux4

T1: HARDWARE DESCRIPTION LANGUAGES(Variable internas/Seniales/Concurrencia)

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Si definimos las señales intermedias P y G

$$P = A \oplus B$$
$$G = AB$$

podemos reescribir el sumador completo como

$$S = P \oplus C_{in}$$
$$C_{out} = G + PC_{in}$$

P y G se denominan variables internas porque no son ni entradas ni salidas, sino que sólo se utilizan internamente en el módulo. Son similares a las variables locales en los lenguajes de programación. El ejemplo A.8 muestra cómo se utilizan en los HDLs

En system verilog, las señales internas suelen declararse como lógicas

```
module fulladder(input  logic a, b, cin,
                 output logic s, cout);

  logic p, g;

  assign p = a ^ b;
  assign g = a & b;

  assign s = p ^ cin;
  assign cout = g | (p & cin);
endmodule
```

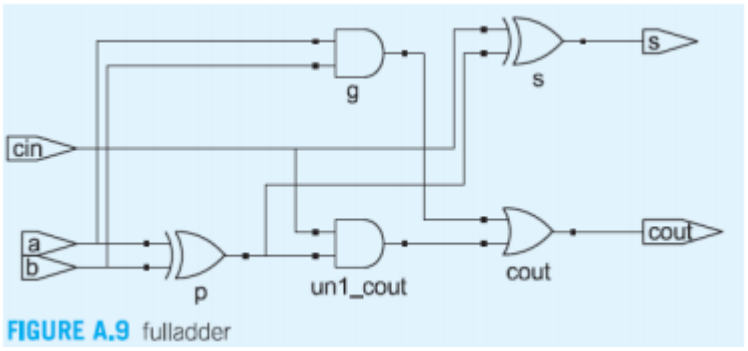

FIGURE A.9 fulladder

T1: HARDWARE DESCRIPTION LANGUAGES(Precdencia y otros operadores)

SystemVerilog

TABLE A.1 SystemVerilog operator precedence

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, -	PLUS, MINUS
	<<, >>	Logical Left / Right Shift
	<<<, >>>	Arithmetic Left / Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	Conditional

T1: HARDWARE DESCRIPTION LANGUAGES(Numeros)

SystemVerilog

Como se muestra en la Tabla A.3, los números de SystemVerilog pueden especificar su base y su tamaño (el número de bits utilizados para representarlos). El formato para declarar constantes es N' Bvalue, donde N es el tamaño en bits, Bis la base, y value da el valor. Por ejemplo 9'h25 indica un número de 9 bits con un valor de 2516 = 3739 = 0001001015.
SystemVerilog soporta 'b para binario (base 2), 'o para octal (base 8), 'd para decimal (base 10), y 'h para hexadecimal (base 16). Si se omite la base, la base por defecto es la decimal.

Si no se indica el tamaño, se supone que el número tiene tantos bits como la expresión en la que se utiliza. Los ceros se rellenan automáticamente en la parte delantera del número para que tenga el tamaño completo. Por ejemplo, si w es un bus de 6 bits, asignar w = 'bll da a w
el valor 000011. Es mejor práctica dar explícitamente el tamaño. Una excepción es que '0 y '1 son abreviaturas de SystemVerilog para llenar un bus con todos los Os y todos los 1s.

TABLE A.3 SystemVerilog numbers

Numbers	Bits	Base	Val	Stored
3'b101	3	2	5	101
'b11	?	2	3	000...0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00...0101010
'1	?	n/a		11...111

T1: HARDWARE DESCRIPTION LANGUAGES(Zs and Xs)

SystemVerilog

```
module tristate(input  logic [3:0] a,
                input  logic       en,
                output tri   [3:0] y);

  assign y = en ? a : 4'bz;
endmodule
```

Observe que y se declara como tri en lugar de lógica. Las señales lógicas sólo pueden tener un único controlador, los buses Tristate pueden tener múltiples controladores, por lo que deben ser declarados como una red. Dos tipos de redes en SystemVerilog se llaman tri y trireg. Típicamente, exactamente un conductor en una red está activo a la vez, y la red toma ese valor. Si ningún controlador está activo, un tri flota (z), mientras que un trireg retiene el valor anterior. Si no se especifica ningún tipo para una entrada o salida, se asume tri.

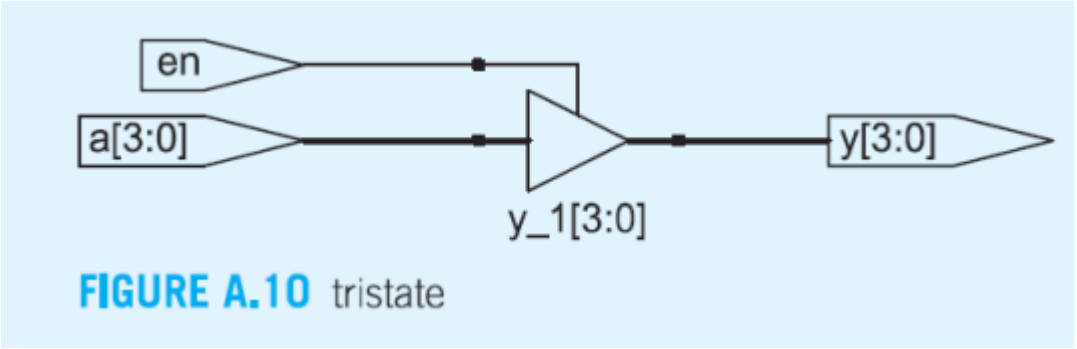

FIGURE A.10 tristate

SystemVerilog

Los valores de las señales de SystemVerilog son O, 1, z y x. Las constantes que empiezan con z o x se rellenan con zs o xs iniciales (en lugar de Os) para alcanzar su longitud completa cuando sea necesario.

La Tabla A.5 muestra una tabla de verdad para una puerta AND utilizando los cuatro posibles valores de la señal. Obsérvese que la puerta puede a veces determinar la salida a pesar de que algunas entradas son desconocidas. Por ejemplo, 0 a z devuelve O porque la salida de una puerta AND ¡siempre es O si alguna de las entradas es O! Si no es así, las entradas flotantes o no válidas provocan salidas no válidas, que se muestran como x.

TABLE A.5 SystemVerilog AND gate truth table with z and x

&		A			
		0	1	z	x
B	0	0	0	0	0
	1	0	1	x	x
	z	0	x	x	x
	x	0	x	x	x

T1: HARDWERE DESCRIPTION LANGUAGES(Bit Swizzling)

```
assign y = {c[2:1], {3{d[0]}}, c[0], 3'b101};
```

El operador () se utiliza para concatenar buses.

13(d[0])) indica tres copias de d[0].

No confundas la constante binaria de 3 bits 3'b101 con el bus b.

Observe que fue crítico especificar la longitud de 3 bits en la constante; de lo contrario, habría tenido un número desconocido de ceros a la izquierda que podrían aparecer en medio de y.

Si y fuera más ancha que 9 bits, los ceros se colocarían en los bits más más significativos.

SystemVerilog

```
module mul(input  logic [7:0] a, b,
           output logic [7:0] upper, lower);

  assign {upper, lower} = a*b;
endmodule
```

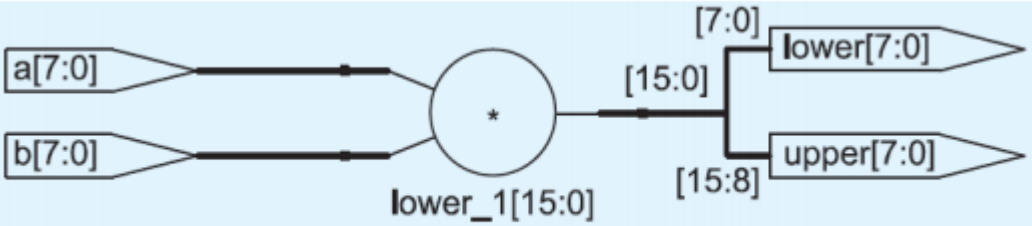

FIGURE A.11 Multipliers

SystemVerilog

```
module signextend(input  logic [15:0] a,
                  output logic [31:0] y);

   assign y = {{16{a[15]}}, a[15:0]};
endmodule
```

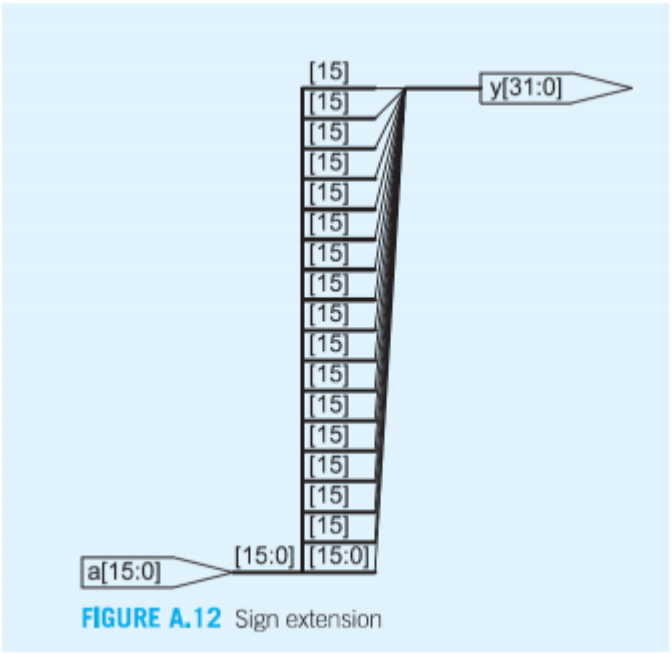

FIGURE A.12 Sign extension

T1: HARDWARE DESCRIPTION LANGUAGES(Delays)

SystemVerilog

```
`timescale 1ns/1ps

module example(input  logic a, b, c,
               output logic y);

  logic ab, bb, cb, n1, n2, n3;

  assign #1 {ab, bb, cb} = ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```

Los archivos SystemVerilog pueden incluir una directiva de escala de tiempo que indica el valor de cada unidad de tiempo. La directiva es de la forma 'unidad de escala de tiempo/paso'. En este archivo, cada unidad es 1ns, y la simulación tiene una resolución de 1 ps. Si no se da ninguna directiva de escala de tiempo en el archivo,
se utiliza una unidad y un paso por defecto (normalmente 1 ns para ambos). En SystemVerilog, el símbolo + se utiliza para indicar el número de unidades de retardo. Puede ser colocado en declaraciones de asignación, así como en asignaciones no bloqueantes (<=) y bloqueantes (=) que serán discutidas en la Sección ASA.

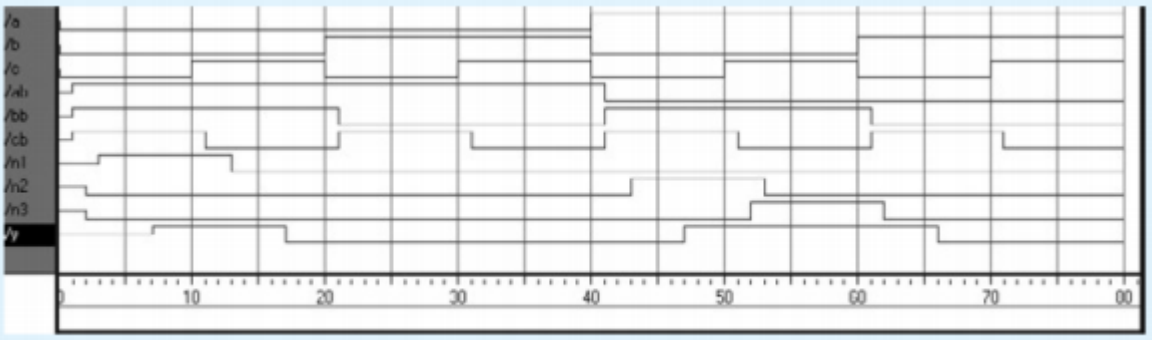

FIGURE A.13 Example simulation waveforms with delays