OPERATORS

BITWISE

The operations are performed on each bit of the bus. All buses (output and inputs) must have same size.

```
~  // NOT: Invert a single-bit signal or each bit in a bus
&  // AND two single bits or each bit between two buses
|  // OR two single bits or each bit between two buses
^  // XOR two single bits or each bit between two buses
```

ARITHMETIC

```
+  // Addition
-  // Subtraction
*  // Multiplication
```

COMPARATORS

```
==  // Equal
!=  // Not Equal
>   // Greater-than
>=  // Greater-than or Equal
<   // Less-than
<=  // Less-than or Equal
```

LOGICAL

The following logical operators are used in **conditional TRUE/FALSE statements** in order to specify the condition for the operation.

```
!   // Not True
&&  // Both Inputs True
||  // Either Input True
==  // Inputs Equal
!=  // Inputs Not Equal
<   // Less-than
<=  // Less-than or Equal
>   // Greater-than
>=  // Greater-than or Equal
```

BIT SWIZZLING

To select and to rearrange the elements of a vector.

```
{A, B, C}      // Concatenate "A", "B" and "C" into a bus
{3{A}}         // Replicate "A" 3 times
{{5{A}}, B}    // Replicate "A" 5 times and concatenate to "B"
A[4]           // Select bit 4 of bus A
A[3:1]         // Select bits 3, 2 and 1 (3 to 1) of bus A
```

LIST OF KEYWORDS

always	end	ifnone	not	rnmos	tri
and	endcase	incdir	notif0	rpmos	tri0
assign	endconfig	include	notif1	rtran	tri1
automatic	endfunction	initial	or	rtranif0	triand
begin	endgenerate	inout	output	rtranif1	trior
buf	endmodule	input	parameter	scalared	trireg
bufif0	endprimitive	instance	pmos	showcancelled	unsigned
bufif1	endspecify	integer	posedge	signed	use
case	endtable	join	primitive	small	vectored
casex	endtask	large	pull0	specify	wait
casez	event	liblist	pull1	specparam	wand
cell	for	library	pulldown	strong0	weak0
cmos	force	localparam	pullup	strong1	weak1
config	forever	macromodule	pulsestyle_onevent	supply0	while
deassign	fork	medium	pulsestyle_ondetect	supply1	wire
default	function	module	rcmos	table	wor
defparam	generate	nand	real	task	xnor
design	genvar	negedge	realtime	time	xor
disable	highz0	nmos	reg	tran	
edge	highz1	nor	release	tranif0	
else	if	noshowcancelled	repeat	tranif1	

SYNTHESIS CONSTRUCTS

- ## CONTINUOUS ASSIGNMENT

```
assign wire_output = statement;
```

- ## CONDITIONAL ASSIGNMENT

```
assign wire_output = 1-bit_condition? value_for_true : value_for_false;
```

- ## CASE

```
case (2-bit_select)
  2'd0: reg_output = statement_for_0;
  2'd1: reg_output = statement_for_1;
  2'd2: reg_output = statement_for_2;
  2'd3: reg_output = statement_for_3;
endcase
```

```
case (2-bit_select)
  2'd1:   reg_output = statement_for_1;
  2'd3:   reg_output = statement_for_3;
  default:reg_output = default_statement;
endcase
```

```
case (3-bit_select)
  3'd0: reg_output = statement_for_0;
  3'd1: reg_output = statement_for_1;
  3'd2: reg_output = statement_for_2;
  3'd3: reg_output = statement_for_3;
  3'd4: reg_output = statement_for_4;
  3'd5: reg_output = statement_for_5;
  3'd6: reg_output = statement_for_6;
  3'd7: reg_output = statement_for_7;
endcase
```

```
case (3-bit_select)
  3'd0:   reg_output = statement_for_0;
  3'd2:   reg_output = statement_for_2;
  3'd3:   reg_output = statement_for_3;
  3'd5:   reg_output = statement_for_5;
  3'd6:   reg_output = statement_for_6;
  default:reg_output = default_statement;
endcase
```

- ## IF

```
if (condition)
  statement;
else if (condition)
  statement;
else
  statement;
```

- ## INSTANTATION WITHOUT PARAMETER

```
module_name  label(
    .module_port (variable_top_design),
    .module_port (variable_top_design)
);
```

- ## INSTANTATION WITH PARAMETER

```
module_name #(parameter_value) label(
    .module_port (variable_top_design),
    .module_port (variable_top_design)
);
```

- ## GENERATE

```
genvar var;

generate
  for (var initialization; condition; step)
  begin: label
        statement or instantiation
  end
endgenerate
```

EJEMPLOS

```
assign F = A | (B & C);              // operaciones bit a bit
assign busA = {bitX, bitY, bitZ};    // concatenación
assign busA[1] = bitY;
assign bitX = busA[2];
assign busB = busA[2:1];
assign busB = {bitX, bitY};
```

- **COMPARADORES**

```
assign Mayor = (A > B);
assign Menor = (A < B);
```

- **SUMADOR**

```
assign SUMA = A + B;
```

- **RESTADOR**

```
assign RESTA = A - B;
```

- **MULTIPLEXOR 2:1**

```
assign Y = sel? I1 : I0;
```

- **MULTIPLEXOR 8:1**

```
always @(*)
  case ({sel2,sel1,sel0})
    3'd0: Y = I0;
    3'd1: Y = I1;
    3'd2: Y = I2;
    3'd3: Y = I3;
    3'd4: Y = I4;
    3'd5: Y = I5;
    3'd6: Y = I6;
    3'd7: Y = I7;
  endcase
```

- **DECODIFICADOR 3 A 8 (1 DE 8)**

```
always @(*)
  case ({I2,I1,I0})
    3'b000: Y = 4'b00000001;
    3'b001: Y = 4'b00000010;
    3'b010: Y = 4'b00000100;
    3'b011: Y = 4'b00001000;
    3'b100: Y = 4'b00010000;
    3'b101: Y = 4'b00100000;
    3'b110: Y = 4'b01000000;
    3'b111: Y = 4'b10000000;
  endcase
```

- **REGISTRO**

```
always@(posedge CLOCK)
    Q <= D;
```

- **REGISTRO CON "CLOCK ENABLE"**

```
always@(posedge CLOCK)
    if (CE)
        Q <= D;
```

- **REGISTRO CON "RESET" SÍNCRONO**

```
always@(posedge CLOCK)
    if (RESET)
        Q <= {n{1'b0}};
    else
        Q <= D;
```

- **REGISTRO CON "CLEAR" ASÍNCRONO**

```
always@(posedge CLOCK, posedge CLEAR)
    if (CLEAR)
        Q <= {n{1'b0}};
    else
        Q <= D;
```

- **REGISTRO CON "RESET" SÍNCRONO Y "CLOCK ENABLE"**

```
always@(posedge CLOCK)
    if (RESET)
        Q <= {n{1'b0}};
    else if (CE)
        Q <= D;
```

SIMULATION CONSTRUCTS

`$finish` is a system task that ends simulation.
`$monitor` is a system task that outputs the value of the variables if they change.

▪ *$urandom*

Returns a 32-bit pseudo-random number.

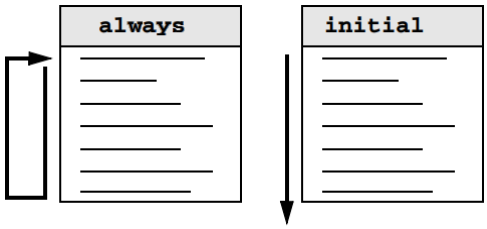
▪ *$urandom_range(*maxval*,*minval*)*

Returns an unsigned integer between **minval** and **maxval**. If **minval** is omitted, the function shall return a value between 0 and **maxval**.

▪ *LOOP STATEMENTS*

There are several ways to create a looping statement within a Verilog testbench. Each of these constructs must appear within an "`initial`" or "`always`" block.

- **initial** procedure executes once at start of simulation. The synthesis standard does not support the initial construct.
- **always** blocks act like a continuous loop, but **initial** blocks operate once and stop.



▪ *FOREVER LOOP*

The **forever** loop is used to create an infinite loop.

```
forever begin
  statements;
end
```

▪ *REPEAT LOOP*

A **repeat** loop is used to perform an action a finite number of times and the loop variable is not needed for the function.

```
repeat (value) begin
  statements;
end
```

▪ *WHILE LOOP*

The **while** loop is a good way to create a conditional loop that will execute as long as a condition is met.

```
while (condition) begin
  statements;
end
```

▪ *FOR LOOP*

The **for** loop is used when a finite loop is desired, and it is necessary to key off the loop variable.

```
integer var;

for (var initialization; condition; step)
begin
  statements;
end
```

El archivo de estímulos de simulación debe tener las siguientes cuatro líneas de comandos para poder visualizar la simulación con **gtkwave**.

```
initial begin
    $dumpfile("ARCHIVO.vcd");  // Archivo en el que se guardarán los resultados de la simulación
    $dumpvars;
end
```