

10: Σχεδίαση και Προσομοίωση Ολοκληρωμένων Συστημάτων με Verilog (II)

Vasileios Tenentes

University of Ioannina

Outline

Παραμετροποιήσιμα Modules

Δηλώσεις Συνθήκης

Δηλώσεις Επανάληψης

Δηλώσεις συνεχούς ανάθεσης

Tri-state buffer

Tasks/Διεργασίες και functions/συναρτήσεις

Παράδειγμα - CPU Write / Read Task

Βοηθητικά tasks/functions

Single-bit Arithmetic logic unit

Παραμετροποιήσιμα Modules

Parameters - Παράμετροι

Χρησιμότητα Παραμέτρων

Η Verilog παρέχει τις παραμέτρους, με την δήλωση **parameter**, που συμπεριφέρονται σαν σταθερές τοπικές μεταβλητές για κάθε ενότητα κώδικα.

Για σχέδια που μπορούν να υλοποιηθούν με την ίδια λογική, αλλά με χρήση διαφορετικού μεγέθους modules.

Παράδειγμα: Ένας counter των **N bits**.

```
module counterNBits(clk, vout);
```

```
    parameter N=8;
```

```
    input clk;
```

```
    output [N-1:0] vout;
```

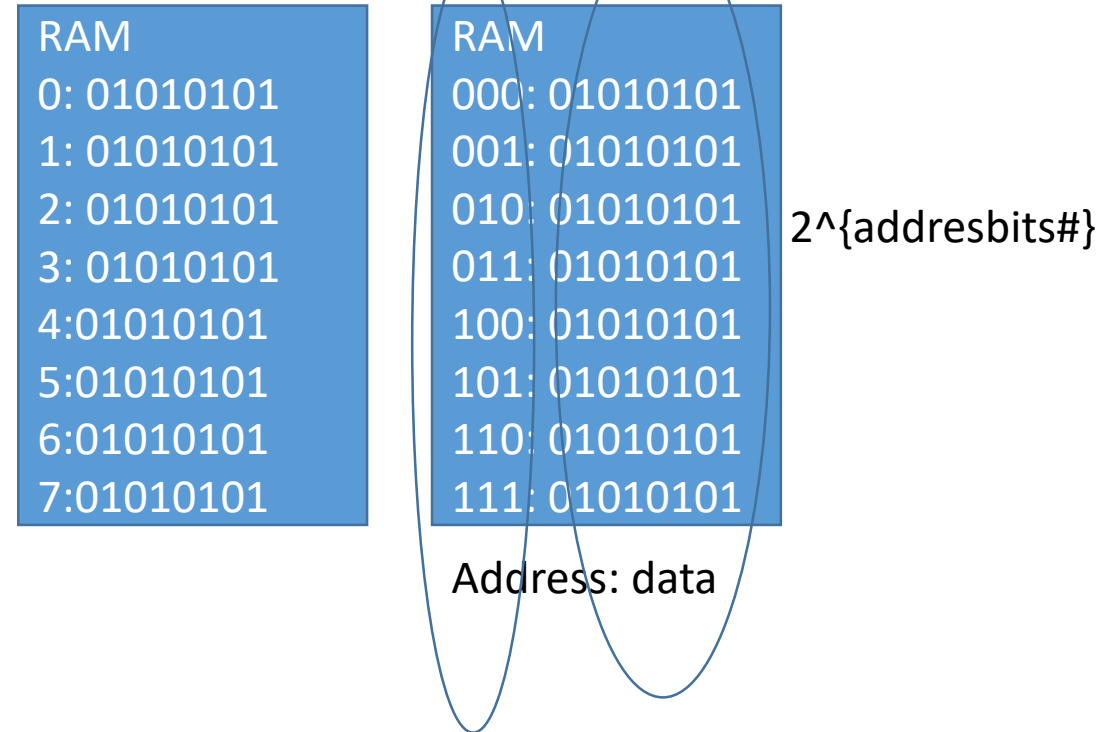
```
    wire clk;
```

```
    reg [N-1:0] vout;
```

```
    always @(posedge clk)
```

```
        vout<=vout+1;
```

```
endmodule
```



Παράκαμψη Παραμέτρων

Μπορούμε να **παρακάμψουμε** τις προεπιλεγμένες τιμές με την δήλωση **defparam** περνώντας ένα νέο σύνολο παραμέτρων στο instance που έχουμε σκοπό να φτιάξουμε.

```
1 module secret_number;
2   parameter my_secret = 0;
3
4   initial begin
5     $display("My secret number is %d", my_secret);
6   end
7
8 endmodule
9
10 module defparam_example();
11
12   defparam U0.my_secret = 11;
13   defparam U1.my_secret = 22;
14
15   secret_number U0();
16   secret_number U1();
17
18 endmodule
```

Πολλαπλές Παράμετροι (Παράδειγμα RAM)

```
RAM_DEPTH = 1 << ADDR_WIDTH
```

```
0000000000001
```

```
0001000000000
```

Παράδειγμα: Μια RAM διαφορετικού πλήθους γραμμών και μεγέθους κάθε γραμμής.

Addresswidth=3

Datawidth=8

```
1 module ram_sp_sr_sw (  
2   clk      , // Clock Input  
3   address  , // Address Input  
4   data     , // Data bi-directional  
5   cs       , // Chip Select  
6   we       , // Write Enable/Read Enable  
7   oe       , // Output Enable  
8 );  
9  
10 parameter DATA_WIDTH = 8 ;  
11 parameter ADDR_WIDTH = 8 ;  
12 parameter RAM_DEPTH = 1 << ADDR_WIDTH;  
13 // Actual code of RAM here  
14  
15 endmodule
```

RAM	
000:	01010101
001:	01010101
010:	01010101
011:	01010101
100:	01010101
101:	01010101
110:	01010101
111:	01010101

RAM_DEPTH

Address: data

Παράκαμψη παραμετροποίησης πολλών παραμέτρων

Παραμετροποίηση instance με σωστή σειρά

```
1 module ram_controller (); //Some ports
2
3 // Controller Code
4
5 ram_sp_sr_sw #(16, 8, 256) ram(clk, address, data, cs, we, oe);
6
7 endmodule
```

Όταν δημιουργείτε περισσότερες από μία παραμέτρους, οι τιμές των παραμέτρων πρέπει να διαβιβάζονται με τη σειρά που δηλώνονται στο module. Το οποίο είναι λίγο κουραστικό και απαιτεί να θυμάσαι τη σειρά. Για τον λόγο αυτό γίνεται και με ανάθεση των παραμέτρων με το όνομά τους:

Παραμετροποίηση instance με τα ονόματα των παραμέτρων

```
1 module ram_controller (); //Some ports
2
3 ram_sp_sr_sw #(
4     .DATA_WIDTH(16),
5     .ADDR_WIDTH(8),
6     .RAM_DEPTH(256)) ram(clk, address, data, cs, we, oe);
7
8 endmodule
```

Δηλώσεις Συνθήκης

Συνθήκη if-then-else

Syntax: if

if (condition)
statements;

Syntax : if-else

if (condition)
statements;
else
statements;

Syntax : nested if-else-if

if (condition)
statements;
else if (condition)
statements;
...
else
statements;

Παράδειγμα if

```
1 module simple_if();  
2  
3 reg latch;  
4 wire enable,din;  
5  
6 always @ (enable or din)  
7 if (enable) begin  
8     latch <= din;  
9 end  
10  
11 endmodule
```

Παράδειγμα if-else

```
1 module if_else();  
2  
3 reg dff;  
4 wire clk,din,reset;  
5  
6 always @ (posedge clk)  
7 if (reset) begin  
8     dff <= 0;  
9 end else begin  
10     dff <= din;  
11 end  
12  
13 endmodule
```

If-else: Σύνθεση με προτεραιότητα

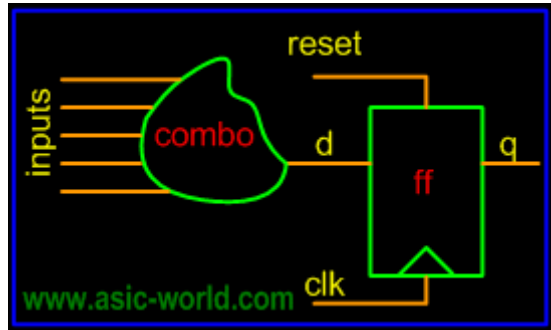
Παράδειγμα με προτεραιότητα

```
1 module nested_if();  
2  
3 reg [3:0] counter;  
4 reg clk, reset, enable, up_en, down_en;  
5  
6 always @ (posedge clk)  
7 // If reset is asserted  
8 if (reset == 1'b0) begin  
9     counter <= 4'b0000;  
10 // If counter is enable and up count is asserted  
11 end else if (enable == 1'b1 && up_en == 1'b1) begin  
12     counter <= counter + 1'b1;  
13 // If counter is enable and down count is asserted  
14 end else if (enable == 1'b1 && down_en == 1'b1) begin  
15     counter <= counter - 1'b1;  
16 // If counting is disabled  
17 end else begin  
18     counter <= counter; // Redundant code  
19 end
```

To enable==1 && up_en==1
υπερισχύει του
enable==1 && down_en==1

If-else: Σύνθεση με παραλληλία

Η παραλληλία είναι επιθυμητή γιατί απαιτεί λιγότερο υλικό και πρέπει να γίνεται όταν ξέρουμε πως κάποιες είσοδοι έχουν αλληλεξάρτηση



Στο παράδειγμα δεξιά, προϋποθέτει ότι τα up_en και το down_en δεν θα έρθουν ποτέ μαζί σε 1. Αλλιώς δύο blocks, αυτό που κάνει πρόσθεση και αυτό που κάνει αφαίρεση, θα οδηγούν ταυτόχρονα τον καταχωρητή counter

Παράδειγμα με παραλληλία

```
1 module parallel_if();
2
3 reg [3:0] counter;
4 wire clk, reset, enable, up_en, down_en;
5
6 always @ (posedge clk)
7 // If reset is asserted
8 if (reset == 1'b0) begin
9     counter <= 4'b0000;
10 end else begin
11     // If counter is enable and up count is mode
12     if (enable == 1'b1 && up_en == 1'b1) begin
13         counter <= counter + 1'b1;
14     end
15     // If counter is enable and down count is mode
16     if (enable == 1'b1 && down_en == 1'b1) begin
17         counter <= counter - 1'b1;
18     end
19 end
20
21 endmodule
```

Συνθήκη με περιπτώσεις - Case

Η **δήλωση περιπτώσεων (case)** συγκρίνει μια έκφραση με μια σειρά περιπτώσεων και εκτελεί κώδικα που αντιστοιχεί στην πρώτη περίπτωση που ταιριάζει:

Υποστηρίζει μεμονωμένες ή πολλαπλές περιπτώσεις.

Ομαδοποιεί πολλές δηλώσεις χρησιμοποιώντας λέξεις-κλειδιά έναρξης και λήξης.

Παράδειγμα

Η σύνταξη του **case** φαίνεται παρακάτω

s

< case1 > : < statement >

< case2 > : < statement >

.....

default : < statement >

endcase

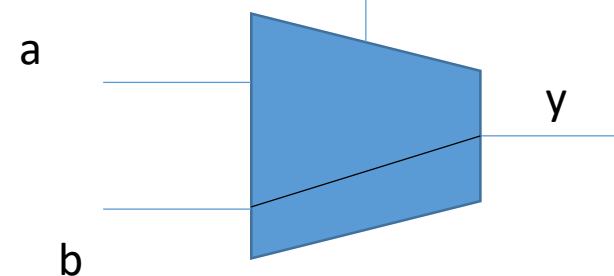
```
1 module mux (a,b,c,d,sel,y);  
2 input a, b, c, d;  
3 input [1:0] sel;  
4 output y;  
5  
6 reg y;  
7  
8 always @ (a or b or c or d or sel)  
9 case (sel)  
10 0: y = a;  
11 1: y = b;  
12 2: y = c;  
13 3: y = d;  
14 default: $display("Error in SEL");  
15 endcase  
16  
17 endmodule
```

Sel για 2 κανάλια φτάνει να είναι 1 bit

Π.χ.

όταν είναι 0 περνάει το a

όταν είναι 1 περνάει το b



To **default**
εκτελείται όταν
καμία άλλη
περίπτωση δεν
ταιριάζει

Συνθήκη με περιπτώσεις - Case

Η **δήλωση περιπτώσεων (case)** συγκρίνει μια έκφραση με μια σειρά περιπτώσεων και εκτελεί κώδικα που αντιστοιχεί στην πρώτη περίπτωση που ταιριάζει:

Υποστηρίζει μεμονωμένες ή πολλαπλές περιπτώσεις.

Ομαδοποιεί πολλές δηλώσεις χρησιμοποιώντας λέξεις-κλειδιά έναρξης και λήξης.

Η **σύνταξη του case φαίνεται παρακάτω:**

case (expression)

S

< case1 > : < statement >

< case2 > : < statement >

.....

default : < statement >

endcase

Παράδειγμα

```
1 module mux (a,b,c,d,sel,y);
2 input a, b, c, d;
3 input [1:0] sel;
4 output y;
5
6 reg y;
7
8 always @ (a or b or c or d or sel)
9 case (sel)
10 0: y = a;
11 1: y = b;
12 2: y = c;
13 3: y = d;
14 default: $display("Error in SEL");
15 endcase
16
```

Το **default**
εκτελείται όταν
καμία άλλη
περίπτωση δεν
ταιριάζει

Sel για N κανάλια ??

φτάνει να είναι $\log_2(N)$ bits

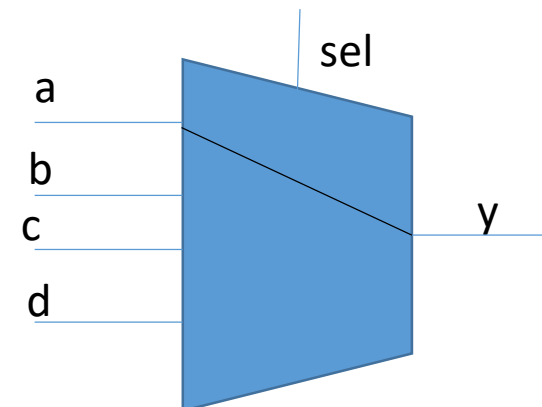
Π.χ. για $N=4$ θέλουμε $\log_2(4)=2$

όταν είναι sel=00 περνάει το a

όταν είναι sel=01 περνάει το b

όταν είναι sel=10 περνάει το c

όταν είναι sel=11 περνάει το d



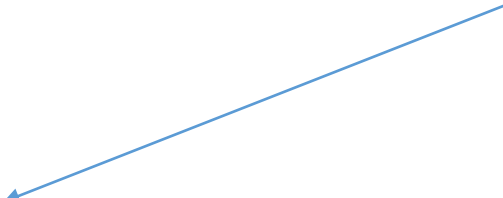
Πολλαπλές περιπτώσεις χωρίς default

Σε μια περίπτωση μπορούν να συγχωνευθούν πολλές περιπτώσεις που αντιστοιχίζονται για τις οποίες ο ίδιος κώδικας θέλουμε να εκτελεστεί

Παράδειγμα

```
1 module mux_without_default (a,b,c,d,sel,y);
2 input a, b, c, d;
3 input [1:0] sel;
4 output y;
5
6 reg y;
7
8 always @ (a or b or c or d or sel)
9 case (sel)
10 0: y = a;
11 1: y = b;
12 2: y = c;
13 3: y = d;
14 2'bxx,2'bx0,2'bx1,2'b0x,2'b1x,
15 2'bzz,2'bz0,2'bz1,2'b0z,2'b1z : $display("Error in SEL");
16 endcase
17
18 endmodule
```

Εδώ έχουμε γράψει όλες τις περιπτώσεις σε μία περίπτωση του case χωρίς να κάνουμε χρήση του default



Ειδικές περιπτώσεις case

Case με x και z (μη αδιάφορα)

```
1 module case_xz(enable);  
2 input enable;  
3  
4 always @ (enable)  
5 case(enable)  
6     1'bz : $display ("enable is floating");  
7     1'bx : $display ("enable is unknown");  
8     default : $display ("enable is % b",enable);  
9 endcase  
10  
11 endmodule
```

Ειδικές περιπτώσεις casez, casex

Παράδειγμα Casez (Treats z as don't care)

1??x

```
1 module casez_example();
2 reg [3:0] opcode;
3 reg [1:0] a,b,c;
4 reg [1:0] out;
5
6 always @ (opcode or a or b or c)
7 casez(opcode)
8   4'b1zzx : begin // Don't care about lower 2:1 bit, bit 0 match with x
9       out = a;
10      $display("@%0dns 4'b1zzx is selected, opcode %b", $time, opcode);
11    end
12   4'b01?? : begin
13       out = b; // bit 1:0 is don't care
14      $display("@%0dns 4'b01?? is selected, opcode %b", $time, opcode);
15    end
16   4'b001? : begin // bit 0 is don't care
17       out = c;
18      $display("@%0dns 4'b001? is selected, opcode %b", $time, opcode);
19    end
20 default : begin
21      $display("@%0dns default is selected, opcode %b", $time, opcode);
22    end
23 endcase
24
25 // Testbench code goes here
26 always #2 a = $random;
27 always #2 b = $random;
28 always #2 c = $random;
29
30 initial begin
31   opcode = 0;
32   #2 opcode = 4'b101x;
33   #2 opcode = 4'b0101;
34   #2 opcode = 4'b0010;
35   #2 opcode = 4'b0000;
36   #2 $finish;
37 end
38
39 endmodule
```

```
@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
```

Παράδειγμα Casex: Treats x and z as don't care.

```
1 module casex_example();
2 reg [3:0] opcode;
3 reg [1:0] a,b,c;
4 reg [1:0] out;
5
6 always @ (opcode or a or b or c)
7 casex(opcode)
8   4'b1zzx : begin // Don't care 2:0 bits
9       out = a;
10      $display("@%0dns 4'b1zzx is selected, opcode %b", $time, opcode);
11    end
12   4'b01?? : begin // bit 1:0 is don't care
13       out = b;
14      $display("@%0dns 4'b01?? is selected, opcode %b", $time, opcode);
15    end
16   4'b001? : begin // bit 0 is don't care
17       out = c;
18      $display("@%0dns 4'b001? is selected, opcode %b", $time, opcode);
19    end
20 default : begin
21      $display("@%0dns default is selected, opcode %b", $time, opcode);
22    end
23 endcase
24
25 // Testbench code goes here
26 always #2 a = $random;
27 always #2 b = $random;
28 always #2 c = $random;
29
30 initial begin
31   opcode = 0;
32   #2 opcode = 4'b101x;
33   #2 opcode = 4'b0101;
34   #2 opcode = 4'b0010;
35   #2 opcode = 4'b0000;
36   #2 $finish;
37 end
38
39 endmodule
```

```
@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
```


Σύγκριση case, casez, casex

```
1 module case_compare;
2
3 reg sel;
4
5 initial begin
6   #1 $display ("Driving 0");
7   sel = 0;
8   #1 $display ("Driving 1");
9   sel = 1;
10  #1 $display ("Driving x");
11  sel = 1'bx;
12  #1 $display ("Driving z");
13  sel = 1'bz;
14  #1 $finish;
15 end
16
17 always @ (sel)
18 case (sel)
19   1'b0 : $display("Normal : Logic 0 on sel");
20   1'b1 : $display("Normal : Logic 1 on sel");
21   1'bx : $display("Normal : Logic x on sel");
22   1'bz : $display("Normal : Logic z on sel");
23 endcase
24
25 always @ (sel)
26 casex (sel)
27   1'b0 : $display("CASEX : Logic 0 on sel");
28   1'b1 : $display("CASEX : Logic 1 on sel");
29   1'bx : $display("CASEX : Logic x on sel");
30   1'bz : $display("CASEX : Logic z on sel");
31 endcase
32
33 always @ (sel)
34 casez (sel)
35   1'b0 : $display("CASEZ : Logic 0 on sel");
36   1'b1 : $display("CASEZ : Logic 1 on sel");
37   1'bx : $display("CASEZ : Logic x on sel");
38   1'bz : $display("CASEZ : Logic z on sel");
39 endcase
40
41 endmodule
```

Simulation Output

Driving 0

Normal : Logic 0 on sel
CASEX : Logic 0 on sel
CASEZ : Logic 0 on sel

Driving 1

Normal : Logic 1 on sel
CASEX : Logic 1 on sel
CASEZ : Logic 1 on sel

Driving x

Normal : Logic x on sel
CASEX : Logic 0 on sel
CASEZ : Logic x on sel

Driving z

Normal : Logic z on sel
CASEX : Logic 0 on sel
CASEZ : Logic 0 on sel

Δηλώσεις Επανάληψης

forever

Ο βρόχος forever εκτελείται συνεχώς.
Χρησιμοποιείτε συνήθως σε αρχικά μπλοκ.

σύνταξη: forever <δήλωση>

Πρέπει να είμαστε προσεκτικοί στη δήλωση αυτή, γιατί εάν δεν υπάρχει κάποια δήλωση χρονισμού, η προσομοίωση θα μπορούσε να κολλήσει.

Ο διπλανός κώδικας είναι μια τέτοια εφαρμογή, όπου μια δήλωση χρονισμού περιλαμβάνεται σε μια δήλωση forever.

Παράδειγμα – γεννήτρια ρολογιού

```
1 module forever_example ();
2
3 reg clk;
4
5 initial begin
6     #1 clk = 0;
7     forever begin
8         #5 clk = !clk;
9     end
10 end
11
12 initial begin
13     $monitor ("Time = %d clk = %b", $time, clk);
14     #100 $finish;
15 end
16
17 endmodule
```

Repeat

Ο επαναληπτικός βρόχος repeat εκτελεί την <δήλωση> σταθερό <αριθμό> φορές.

σύνταξη: repeat (<αριθμός>) <δήλωση>

Παράδειγμα (αναγνώριση opcode για αριστερή ολίσθηση 8 θέσεων)

```
1 module repeat_example();
2 reg [3:0] opcode;
3 reg [15:0] data;
4 reg      temp;
5
6 always @ (opcode or data)
7 begin
8     if (opcode == 10) begin
9         // Perform rotate
10        repeat (8) begin
11            #1 temp = data[15];
12            data = data << 1;
13            data[0] = temp;
14        end
15    end
16 end
17 // Simple test code
18 initial begin
19     $display (" TEMP DATA");
20     $monitor (" %b  %b ",temp, data);
21     #1 data = 18'hF0;
22     #1 opcode = 10;
23     #10 opcode = 0;
24     #1 $finish;
25 end
26
27 endmodule
```

While-loop

Ο βρόχος while εκτελείται όσο η <έκφραση> είναι αληθής. Όπως σε κάθε γλώσσα προγραμματισμού.

σύνταξη: while (<έκφραση>) <δήλωση>

Παράδειγμα (εντοπισμός πρώτης τιμής λογικό-`1` σε έναν καταχωρητή των 8 ψηφίων)

```
1 module while_example();
2
3 reg [5:0] loc;
4 reg [7:0] data;
5
6 always @ (data or loc)
7 begin
8     loc = 0;
9     // If Data is 0, then loc is 32 (invalid value)
10    if (data == 0) begin
11        loc = 32;
12    end else begin
13        while (data[0] == 0) begin
14            loc = loc + 1;
15            data = data >> 1;
16        end
17    end
18    $display ("DATA = %b  LOCATION = %d",data,loc);
19 end
20
21 initial begin
22     #1 data = 8'b11;
23     #1 data = 8'b100;
24     #1 data = 8'b1000;
25     #1 data = 8'b1000_0000;
26     #1 data = 8'b0;
27     #1 $finish;
28 end
29
30 endmodule
```

for-loop

Το for loop, όπως και σε μια γλώσσα προγραμματισμού, εκτελεί μια <αρχική ανάθεση> μία φορά στην αρχή του βρόχου.

Εκτελεί το βρόχο όσο ένα <έκφραση> αξιολογείται ως αληθή.

Εκτελεί μια <εκχώρηση βήματος> στο τέλος κάθε διέλευσης μέσω του βρόχου.

σύνταξη:

for (<αρχική ανάθεση>; <έκφραση>, <βήμα ανάθεση>)
<δήλωση>

Σημείωση: στην verilog δεν υπάρχει ο τελεστής ++ όπως στην περίπτωση της γλώσσας C.

Παράδειγμα (μηδενισμός των στοιχείων μιας μνήμης RAM)

```
1 module for_example();
2
3 integer i;
4 reg [7:0] ram [0:255];
5
6 initial begin
7     for (i = 0; i < 256; i = i + 1) begin
8         #1 $display(" Address = %g Data = %h",i,ram[i]);
9         ram[i] <= 0; // Initialize the RAM with 0
10        #1 $display(" Address = %g Data = %h",i,ram[i]);
11    end
12    #1 $finish;
13 end
14
15 endmodule
```

RAM	
Address	: data
00000000	: 00000000
00000001	: 00000000
00000010	: 00000000
.	.
.	.
.	.
11111111	: 00000000

Δηλώσεις συνεχούς ανάθεσης

Δηλώσεις συνεχούς ανάθεσης

Μια δήλωση συνεχούς ανάθεσης χρησιμοποιείται για να οδηγήσει Nets/Wires (τύπος καλωδίου δεδομένων). Αποτελούν τις δομικές συνδέσεις.

Χρησιμοποιούνται για μοντελοποίηση buffer τριών καταστάσεων (Tri-State)

Πολύ χρήσιμη δομή όπως θα δούμε για διαύλους επικοινωνίας (Buses)

Μπορούν να χρησιμοποιηθούν για μοντελοποίηση συνδυαστικής λογικής.

Βρίσκονται εκτός των διαδικαστικών μπλοκ (και εκτός από τα always και εκτός από τα initial μπλοκ).

Η συνεχής εκχώρηση αντικαθιστά τυχόν διαδικαστικές αναθέσεις.

Η αριστερή πλευρά μιας συνεχούς ανάθεσης πρέπει να είναι καθαρός τύπος δεδομένων.

σύνταξη: assign (δύναμη, ισχύς) # (καθυστέρηση) net = έκφραση;

Απλή μορφή:

assign net = έκφραση;

Παράδειγμα δηλώσεως συνεχούς ανάθεσης – πλήρης αθροιστής

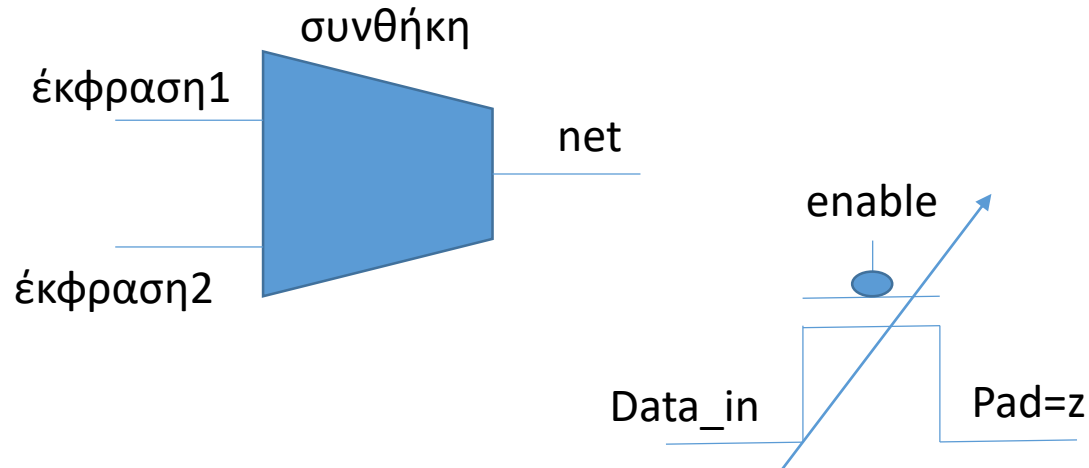
```
1 module adder_using_assign ();
2 reg a, b;
3 wire sum, carry;
4
5 assign #5 {carry,sum} = a+b;
6
7 initial begin
8     $monitor (" A = %b B = %b CARRY = %b SUM = %b",a,b,carry,sum);
9     #10 a = 0;
10    b = 0;
11    #10 a = 1;
12    #10 b = 1;
13    #10 a = 0;
14    #10 b = 0;
15    #10 $finish;
16 end
17
18 endmodule
```

Δήλωση συνεχούς ανάθεσης με συνθήκη

Η συνεχής ανάθεση μπορεί να γίνει και υπο συνθήκη σύμφωνα με την **σύνταξη**:

```
assign net = (συνθήκη) ? έκφραση1: έκφραση2;
```

Στην περίπτωση αυτή, όταν η συνθήκη είναι αληθής τότε γίνεται ανάθεση της έκφρασης 1, αλλιώς γίνεται ανάθεση της έκφρασης 2.



Παράδειγμα – tri-state buffer

```
module tri_buf_using_assign();  
  reg data_in, enable;  
  wire pad;
```

```
  assign pad = (enable) ? 1'bz: data_in;
```

```
  initial begin
```

```
    $monitor ("TIME = %g ENABLE = %b DATA : %b PAD %b",  
              $time, enable, data_in, pad);
```

```
    #1 enable = 0;
```

```
    #1 data_in = 1;
```

```
    #1 enable = 1;
```

```
    #1 data_in = 0;
```

```
    #1 enable = 0;
```

```
    #1 $finish;
```

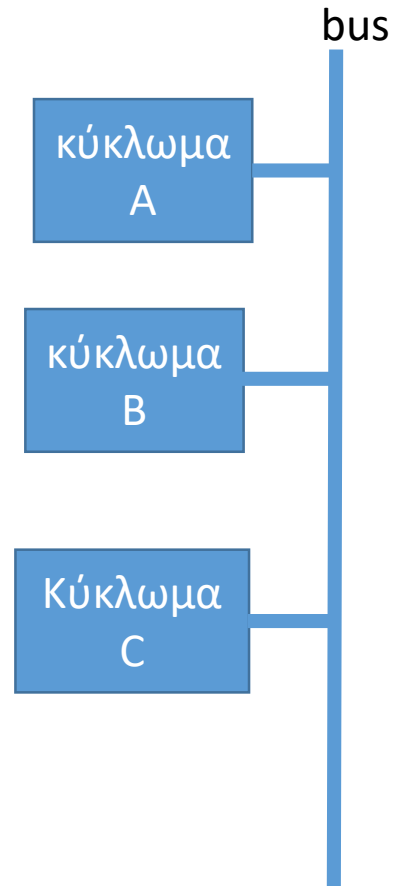
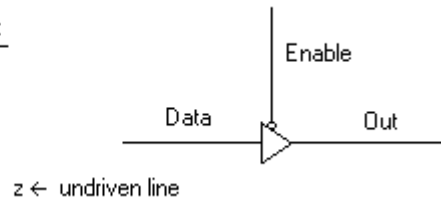
```
  end
```

```
endmodule
```

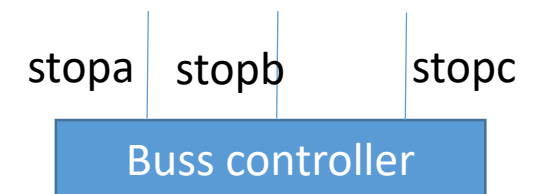
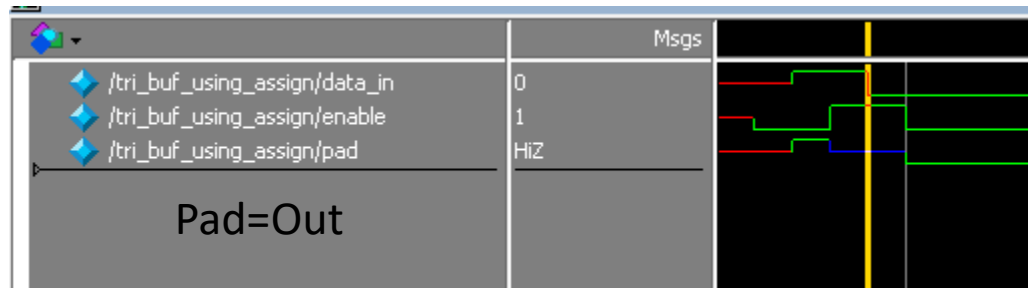
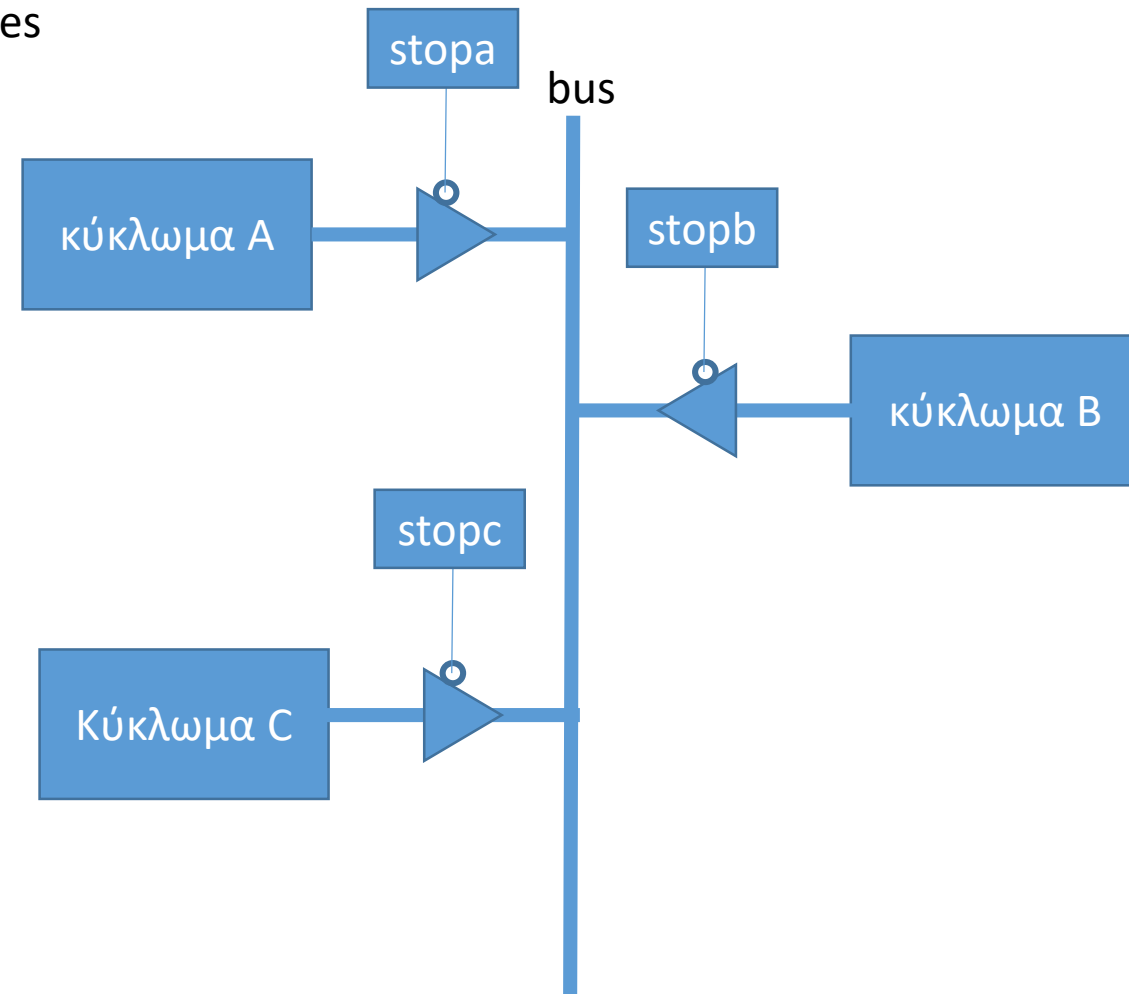
Tri-state buffer

Πίνακας Αληθείας tri-state buffer

Enable	Data	Out
0	0	0
0	1	1
1	0	z
1	1	z



Χρησιμοποιείτε για σηματοδότηση σε buses



Tasks/Διεργασίες και
functions/συναρτήσεις

Tasks

Τα tasks θυμίζουν τις ρουτίνες των γλωσσών προγραμματισμού

Οι γραμμές κώδικα περικλείονται στο task. Τα δεδομένα μεταβιβάζονται στο task, το οποίο εκτελεί την επεξεργασία του και επιστρέφει το αποτέλεσμα. Πρέπει να κληθούν με συγκεκριμένα δεδομένα εισόδου/εξόδου.

Περιλαμβάνονται στο κύριο σώμα του κώδικα, μπορούν να κληθούν πολλές φορές, μειώνοντας την επανάληψη κώδικα.

Τα tasks ορίζονται στην ενότητα στην οποία χρησιμοποιούνται. Είναι δυνατό να ορίσετε ένα task σε ένα ξεχωριστό αρχείο και να χρησιμοποιήσετε την εντολή μεταγλώττισης `include` για να συμπεριλάβετε το αρχείο με το task.

ένα task μπορεί να περιλαμβάνει καθυστερήσεις χρονισμού, όπως `posedge`, `negedge`, `#` καθυστέρηση και αναμονή.

ένα task μπορεί να έχει οποιοδήποτε αριθμό εισόδων και εξόδων.

Οι μεταβλητές που δηλώνονται εντός ενός task είναι τοπικές. Η σειρά δήλωσης καθορίζει τον τρόπο με τον οποίο χρησιμοποιούνται οι μεταβλητές που μεταβιβάστηκαν από τον καλούνται.

ένα task μπορεί να οδηγήσει και να δημιουργήσει καθολικές μεταβλητές, όταν δεν χρησιμοποιούνται τοπικές μεταβλητές.

Ένα task μπορεί να καλέσει ένα άλλο task.

Δήλωση Tasks

Ένα task ξεκινά με μια λέξη-κλειδί και τελειώνει με μια λέξη-κλειδί

Οι είσοδοι και οι έξοδοι δηλώνονται μέσα στο task.

Οι τοπικές μεταβλητές δηλώνονται μετά τη δήλωση εισόδων/εξόδων.

Παράδειγμα – task με τοπικές μεταβλητές

```
module simple_module_with_task();  
  reg in1, out1;  
  task convert;  
    input [7:0] temp_in;  
    output [7:0] temp_out;  
    begin  
      temp_out = (9/5) * ( temp_in + 32)  
    end  
  endtask  
  always  
  begin  
    convert(in1 out1);  
  end  
endmodule
```

Παράδειγμα – task με global μεταβλητές

```
module task_global();  
  
  reg [7:0] temp_out;  
  reg [7:0] temp_in;  
  
  task convert;  
    begin  
      temp_out = (9/5) * ( temp_in + 32);  
    end  
  endtask  
  
endmodule
```

Κλήση Tasks

Ας υποθέσουμε ότι το task από το προηγούμενο παράδειγμα αποθηκεύεται σε ένα αρχείο που ονομάζεται mytask.v. Το πλεονέκτημα της κωδικοποίησης μιας εργασίας σε ξεχωριστό αρχείο, είναι ότι μπορεί να χρησιμοποιηθεί σε πολλές ενότητες

```
module task_calling (temp_a, temp_b, temp_c, temp_d);  
input [7:0] temp_a, temp_c;  
output [7:0] temp_b, temp_d;  
reg [7:0] temp_b, temp_d;  
`include "mytask.v"  
  
always @ (temp_a)  
begin  
    convert (temp_a, temp_b);  
end  
  
always @ (temp_c)  
begin  
    convert (temp_c, temp_d);  
end  
  
endmodule
```

Παράδειγμα - CPU Write / Read Task

Παρακάτω είναι η κυματομορφή που χρησιμοποιείται για την εγγραφή στη μνήμη και την ανάγνωση από τη μνήμη. Υποθέτουμε ότι υπάρχει ανάγκη χρήσης αυτής της διεπαφής από πολλά κυκλώματα που διαβάζουν και γράφουν στη μνήμη. Γιο το λόγο αυτό γράφουμε την ανάγνωση / εγγραφή ως tasks.

Παράδειγμα – Read task

```
28 // CPU Read Task
29 task cpu_read;
30 input [7:0] address;
31 output [7:0] data;
32 begin
33     $display ("%g CPU Read task with address : %h", $time, address);
34     $display ("%g -> Driving CE, RD and ADDRESS on to bus", $time);
35     @ (posedge clk);
36     addr = address;
37     ce = 1;
38     rd = 1;
39     @ (negedge clk);
40     data = data_rd;
41     @ (posedge clk);
42     addr = 0;
43     ce = 0;
44     rd = 0;
45     $display ("%g CPU Read data : %h", $time, data);
46     $display ("=====");
47 end
48 endtask
```

Η μνήμη

```
71 // Memory model for checking tasks
72 reg [7:0] mem [0:255];
73
74 always @ (addr or ce or rd or wr or data_wr)
75 if (ce) begin
76     if (wr) begin
77         mem[addr] = data_wr;
78     end
79     if (rd) begin
80         data_rd = mem[addr];
81     end
82 end
83
84 endmodule
```

Παράδειγμα – write task

```
49 // CU Write Task
50 task cpu_write;
51 input [7:0] address;
52 input [7:0] data;
53 begin
54     $display ("%g CPU Write task with address : %h Data : %h",
55             $time, address, data);
56     $display ("%g -> Driving CE, WR, WR data and ADDRESS on to bus",
57             $time);
58     @ (posedge clk);
59     addr = address;
60     ce = 1;
61     wr = 1;
62     data_wr = data;
63     @ (posedge clk);
64     addr = 0;
65     ce = 0;
66     wr = 0;
67     $display ("=====");
68 end
69 endtask
```

Simulation Output

```
1 CPU Write task with address : 11 Data : aa
1 -> Driving CE, WR, WR data and ADDRESS on to bus
=====
4 CPU Read task with address : 11
4 -> Driving CE, RD and ADDRESS on to bus
7 CPU Read data : aa
=====
8 CPU Write task with address : 12 Data : ab
8 -> Driving CE, WR, WR data and ADDRESS on to bus
=====
12 CPU Read task with address : 12
12 -> Driving CE, RD and ADDRESS on to bus
15 CPU Read data : ab
=====
16 CPU Write task with address : 13 Data : 0a
16 -> Driving CE, WR, WR data and ADDRESS on to bus
=====
20 CPU Read task with address : 13
20 -> Driving CE, RD and ADDRESS on to bus
23 CPU Read data : 0a
=====
```

```
RAM
Address : data
00000000: 00000000
00000001: 00000000
00000010: 00000000
.
.
.
11111111: 00000000
```

```
1 module bus_wr_rd_task();
2
3 reg clk, rd, wr, ce;
4 reg [7:0] addr, data_wr, data_rd;
5 reg [7:0] read_data;
6
7 initial begin
8     clk = 0;
9     read_data = 0;
10    rd = 0;
11    wr = 0;
12    ce = 0;
13    addr = 0;
14    data_wr = 0;
15    data_rd = 0;
16    // Call the write and read tasks here
17    #1 cpu_write(8'h11, 8'hAA);
18    #1 cpu_read(8'h11, read_data);
19    #1 cpu_write(8'h12, 8'hAB);
20    #1 cpu_read(8'h12, read_data);
21    #1 cpu_write(8'h13, 8'h0A);
22    #1 cpu_read(8'h13, read_data);
23    #100 $finish;
24 end
25 // Clock Generator
26 always
27    #1 clk = ~clk;
```


Arithmetic logic unit ενός bit

alu1bit

```
module alu1bit(opcode,r1,r2,out);
input [1:0] opcode;

input r1, r2;
output [1:0] out;

wire r1, r2;
reg [1:0] out;

always @ (opcode or r1 or r2)
case(opcode)
  2'b00 : begin // Don't care about lower 2:1 bit, bit 0 match with x
    out = r1&r2;
  end
  2'b01 : begin
    out = r1|r2;
  end
  2'b10 : begin // bit 0 is don't care
    out = r1^r2;
  end
  default : begin
    out = r1+r2;
  end
endcase

endmodule
```

Testbench alu1bit

```
module alu1bittb();

reg [1:0] opcode;
reg r1, r2;
wire [1:0] out;

alu1bit aluinst(opcode, r1, r2, out);

// Testbench code goes here
always #2 r1 = $random;
always #2 r2 = $random;
always #2 opcode[0] = $random;
always #2 opcode[1] = $random;

initial begin
  $display("@%0dns default is selected, opcode %b",
    $time,opcode);
end

endmodule
```

βοηθητικές συναρτήσεις για αρχεία \$fopen, \$fdisplay, \$fstrobe \$fmonitor and \$fwrite

Με τις παρακάτω εντολές μπορούμε να γράψουμε και να φορτώσουμε αρχεία

\$fopen: ανοίγει ένα αρχείο και επιστρέφει τον file descriptor.

\$fclose: κλείνει ένα αρχείο.

\$fdisplay και \$fwrite γράφουν μορμαρισμένα δεδομένα στο αρχείο. Είναι παρόμοιες μόνο που η \$fdisplay γράφει ένα σύμβολο νέας γραμμής στο τέλος κάθε εγγραφής ενώ η \$write όχι.

\$strobe περιμένει να τελειώσουν όλες οι άλλες πράξεις μέσα στο βήμα χρόνο που εκτελείται και τελευταία γράφει στο αρχείο.

Έτσι για παράδειγμα:

```
initial #1 a=1; b=0; $strobe(f, a,b); b=1;
```

Θα γράψει 1 1 for a and b.

\$monitor γράφει στο αρχείο όποτε κάποιο από τα ορίσματα αλλάζει.

Functions / Συναρτήσεις

Μια συνάρτηση Verilog είναι παρόμοια με ένα task, με πολύ μικρές διαφορές, όπως το ότι η συνάρτηση δεν μπορεί να οδηγήσει περισσότερες από μία εξόδους, και δεν μπορεί να περιέχει καθυστερήσεις.

Οι συναρτήσεις ορίζονται στη λειτουργική μονάδα στην οποία χρησιμοποιούνται. Είναι δυνατόν να ορίσετε τις λειτουργίες σε ξεχωριστά αρχεία και να χρησιμοποιήσετε την εντολή `include` για να συμπεριλάβετε τη συνάρτηση στο αρχείο που δημιουργεί την εργασία.

Οι συναρτήσεις δεν μπορούν να περιλαμβάνουν καθυστερήσεις χρονισμού, όπως `posedge`, `negedge`, `# delay`, πράγμα που σημαίνει ότι οι συναρτήσεις πρέπει να εκτελούνται σε χρονική καθυστέρηση "μηδέν".

Οι συναρτήσεις μπορούν να έχουν οποιοδήποτε αριθμό εισόδων, αλλά μόνο μία έξοδο.

Οι μεταβλητές που δηλώνονται μέσα στη συνάρτηση είναι τοπικές σε αυτήν τη συνάρτηση. Η σειρά δήλωσης εντός της συνάρτησης καθορίζει τον τρόπο με τον οποίο χρησιμοποιούνται οι μεταβλητές που μεταβιβάζονται στη συνάρτηση από τον καλούντα.

Οι συναρτήσεις μπορούν να λαμβάνουν, να οδηγούν και να δημιουργούν καθολικές μεταβλητές, όταν δεν χρησιμοποιούνται τοπικές μεταβλητές. Όταν χρησιμοποιούνται τοπικές μεταβλητές, τότε η έξοδος εκχωρείται μόνο στο τέλος της εκτέλεσης της συνάρτησης.

Οι λειτουργίες μπορούν να καλέσουν άλλες λειτουργίες, αλλά δεν μπορούν να καλέσουν εργασίες.

Παραδείγματα Functions / Συναρτήσεις

Παράδειγμα - Δήλωση συνάρτησης

```
module simple_function();

function myfunction;
input a, b, c, d;
begin
    myfunction = ((a+b) + (c-d));
end
endfunction

endmodule
```

Παράδειγμα – Κλήση συνάρτησης

```
module function_calling(a, b, c, d, e, f);

input a, b, c, d, e ;
output f;
wire f;
`include "myfunction.v"

assign f = (myfunction (a,b,c,d)) ? e :0;

endmodule
```

Βοηθητικά tasks/functions

Βοηθητικά Tasks και Functions

Υπάρχουν διεργασίες/tasks και συναρτήσεις/functions που χρησιμοποιούνται για τη δημιουργία εισόδου και εξόδου κατά την προσομοίωση. Τα ονόματά τους ξεκινούν με το σύμβολο του δολαρίου (\$). Τα εργαλεία σύνθεσης αναλύουν και αγνοούν τις λειτουργίες του συστήματος, και ως εκ τούτου μπορούν να συμπεριληφθούν ακόμη και σε μοντέλα συνθέσιμα.

\$display, \$strobe, \$monitor

Αυτές οι εντολές έχουν την ίδια σύνταξη και εμφανίζουν κείμενο στην οθόνη κατά τη διάρκεια της προσομοίωσης.

Είναι πολύ λιγότερο βολικά από τα εργαλεία εμφάνισης κυματομορφής όπως το GTKWave. ή Undertow ή Debussy. \$display και \$strobe display μία φορά κάθε φορά που εκτελούνται, ενώ το \$ monitor εμφανίζεται κάθε φορά που μία από τις παραμέτρους της αλλάζει. Η διαφορά μεταξύ \$ display και \$ strobe είναι ότι το \$ strobe εμφανίζει τις παραμέτρους στο τέλος της τρέχουσας μονάδας χρόνου προσομοίωσης και όχι ακριβώς όταν εκτελείται.

Η συμβολοσειρά μορφής είναι παρόμοια στο C / C ++ και μπορεί να περιέχει χαρακτήρες μορφής. Οι χαρακτήρες μορφής περιλαμβάνουν:

- % d (δεκαδικό),
- % h (δεκαεξαδικό),
- % b (δυαδικό),
- % c (χαρακτήρας),
- % s (συμβολοσειρά) και
- % t (χρόνος),
- % m (επίπεδο ιεραρχίας).

Τα % 5d, % 5b κ.λπ. θα έδιναν ακριβώς 5 κενά για τον αριθμό αντί για τον απαιτούμενο χώρο. Προσάρτηση b, h, o στο όνομα της εργασίας για να αλλάξετε την προεπιλεγμένη μορφή σε δυαδικό, οκταδικό ή δεκαεξαδικό.

Σύνταξη \$display, \$strobe, \$monitor

```
$display ("format_string", par_1, par_2, ... );  
$strobe ("format_string", par_1, par_2, ... );  
$monitor ("format_string", par_1, par_2, ... );  
$displayb (as above but defaults to binary..);  
$strobeh (as above but defaults to hex..);  
$monitoro (as above but defaults to octal..);
```


Και άλλες βοηθητικές συναρτήσεις

\$time, \$stime, \$realtime

επιστρέφουν τον τρέχοντα χρόνο προσομοίωσης ως ακέραιος αριθμός 64-bit, ακέραιος αριθμός 32-bit και πραγματικός αριθμός, αντίστοιχα.

Το \$reset επαναφέρει την προσομοίωση στο χρόνο 0;

Το \$stop σταματά τον προσομοιωτή και το θέτει σε διαδραστική λειτουργία όπου ο χρήστης μπορεί να εισάγει εντολές.

Το \$finish τερματίζει την προσομοίωση.

Το \$scope, \$showscope (hierarchy_name) ορίζει το τρέχον ιεραρχικό εύρος σε hierarchy_name. Το \$showscopes (n) παραθέτει όλες τις ενότητες, εργασίες και ονόματα μπλοκ στο (και παρακάτω, εάν το n έχει οριστεί σε 1) στο τρέχον πεδίο. Συνεργάζεται με την \$dumpnvc

Το \$random δημιουργεί έναν τυχαίο ακέραιο κάθε φορά που καλείται. Εάν η ακολουθία πρόκειται να επαναληφθεί, την πρώτη φορά που κάποιος επικαλεστεί τυχαία δίνοντάς του ένα αριθμητικό όρισμα (σπόρος). Διαφορετικά, ο σπόρος προέρχεται από το ρολόι του υπολογιστή.

βοηθητικές συναρτήσεις για αρχεία \$fopen, \$fdisplay, \$fstrobe \$fmonitor and \$fwrite

Με τις παρακάτω εντολές μπορούμε να γράψουμε και να φορτώσουμε αρχεία

\$fopen: ανοίγει ένα αρχείο και επιστρέφει τον file descriptor.

\$fclose: κλείνει ένα αρχείο.

\$fdisplay και \$fwrite γράφουν μορμαρισμένα δεδομένα στο αρχείο. Είναι παρόμοιες μόνο που η \$fdisplay γράφει ένα σύμβολο νέας γραμμής στο τέλος κάθε εγγραφής ενώ η \$write όχι.

\$strobe περιμένει να τελειώσουν όλες οι άλλες πράξεις μέσα στο βήμα χρόνο που εκτελείται και τελευταία γράφει στο αρχείο.

Έτσι για παράδειγμα:

```
initial #1 a=1; b=0; $strobe("%b %b", a,b); b=1;
```

Θα γράψει 1 1 for a and b.

\$monitor γράφει στο αρχείο όποτε κάποιο από τα ορίσματα αλλάζει.