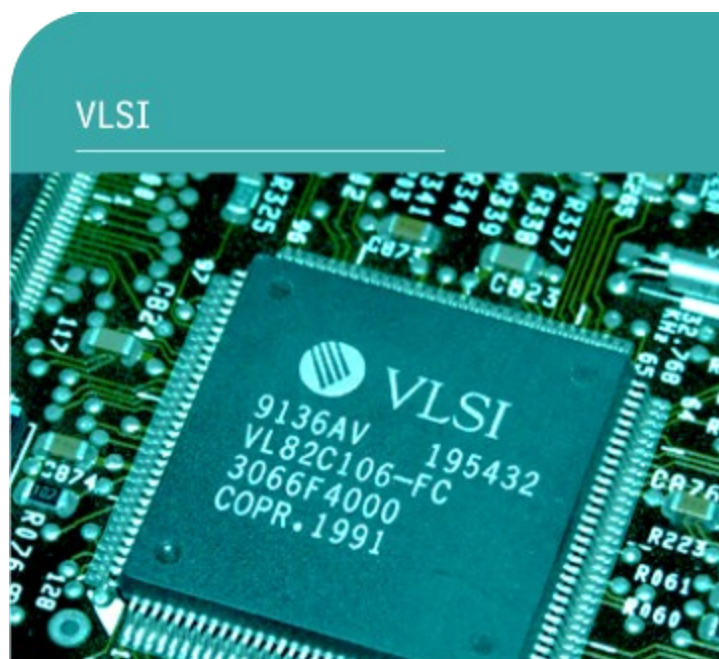


7η Εργαστηριακή άσκηση



Σχεδίαση VLSI με τη γλώσσα περιγραφής υλικού Verilog HDL

Τζανάκη Βασιλική (03108062)
Τζίμα Σοφία (03108052)
31/01/2012

3. Η 1^η εργαστηριακή άσκηση

Κατασκευή ενός απλού 3-bit counter

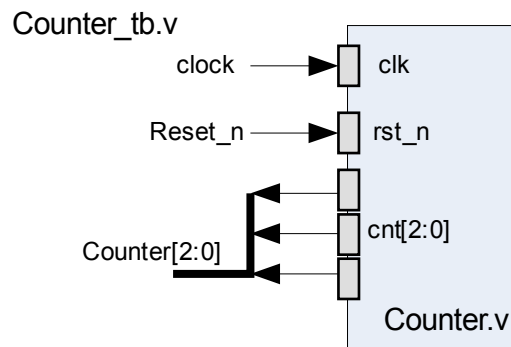
Βήμα 1

Φτιάχνουμε ένα αρχείο Counter.v με τον παρακάτω κώδικα, που υλοποιεί τη συμπεριφορά ενός απλού μετρητή και το περνάμε από compilation με τη χρήση του Modelsim.

```
module counter (clk, rst_n, cnt);  
  
    // This is the clock of the counter  
    input clk;  
    // This is the reset signal of the  
    counter (Active low)  
    input rst_n;  
    // This is the counter bits  
    output [2:0] cnt;  
  
    reg [2:0] cnt;  
  
    always @(posedge clk or negedge rst_n)  
    begin  
        if (!rst_n)  
            cnt <= 3'b0;  
        else  
            cnt <= cnt + 3'b001;  
    end  
endmodule
```

Βήμα 2

Σ' αυτό το βήμα φτιάχνουμε το αρχείο Counter_tb.v, το οποίο θα ελέγχει τις εισόδους του κυκλώματός μας, δηλαδή τα σήματα clk και rst_n, και θα διαβάζει τις εξόδους του Counter για να μπορεί να τις απεικονίσει στον εξομοιωτή. Στο παρακάτω σχήμα παρουσιάζεται η σχέση του αρχείου Testbench με τον μετρητή :



Ακολουθεί ο κώδικας του αρχείου Testbench.

```
`timescale 1ns / 1ps

module counter_tb;
//-----
----
// Various signals declaration
reg reset_n;
reg clock;
wire [2:0] counter;




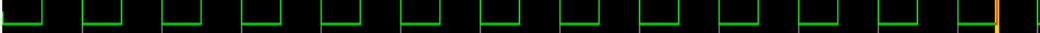

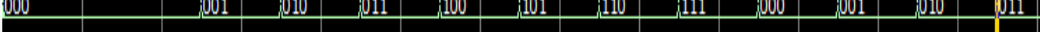
//-----
----
// Various initializations
initial
begin
    clock <= 1'b0;
    reset_n <= 1'b0;
    #40
    reset_n <= 1'b1;
end

//-----
----
// This is where I call my counter ...
counter DUT (
    .clk(clock),
    .rst_n(reset_n),
    .cnt(counter)
);

//-----
----
// Clock Generation
always begin
    #10 clock <= ~clock;
end

endmodule
```

Αφού περάσουμε και αυτό το αρχείο από compilation με τη βοήθεια του Modelsim, συνεχίζουμε με την εξομοίωση του κυκλώματος και την απεικόνιση όλων των σημάτων στο γραφικό περιβάλλον. Τα σήματα αυτά φαίνονται στο σχήμα που ακολουθεί, όπου και επιβεβαιώνουμε τη σωστή λειτουργία του προγράμματός μας.

	Msgs	
 /counter_tb/reset_n	1	
 /counter_tb/clock	1	
 /counter_tb/counter	011	

Βήμα 3

Σ' αυτό το βήμα θα κάνουμε αλλαγές στα παραπάνω αρχεία, ώστε να υλοποιήσουμε έναν Up/Down μετρητή των 4-bits.

Ειδικότερα:

1. Στο αρχείο Counter.v προσθέτουμε μία ακόμα είσοδο με όνομα up_down, αλλάζουμε το εύρος του διανύσματος του μετρητή από 3 σε 4 bits, ώστε να μετράει μέχρι και το 15 και ανάλογα με το αν η τιμή του σήματος up_down είναι 1 ή 0, έχουμε μέτρηση προς τα πάνω ή μέτρηση προς τα κάτω αντίστοιχα.

2. Στο αρχείο Counter_tb.v αλλάζουμε τον αριθμό των bits του σήματος counter, την κλήση του counter, αφού πλέον έχουμε 1 παραπάνω pin και να προσθέσουμε τον έλεγχο για το νέο σήμα up_down.

Οι αλλαγές αυτές φαίνονται στα αρχεία που ακολουθούν:

Counter2.v

```

module counter2 (clk, rst_n, up_down, cnt);
// This is the clock of the counter
input clk;
// This is the reset signal of the counter (Active low)
input rst_n;
// Choice count up or down
input up_down;
// This is the counter bits
output [3:0] cnt;

reg [3:0] cnt;

always @(posedge clk or negedge rst_n)
begin
    if (!rst_n)
        cnt <= 4'b0;
    else if (up_down)
        cnt <= cnt + 4'b0001;
    else
        cnt <= cnt - 4'b0001;
end
endmodule

```

Counter2_tb.v

```
`timescale 1ns / 1ps

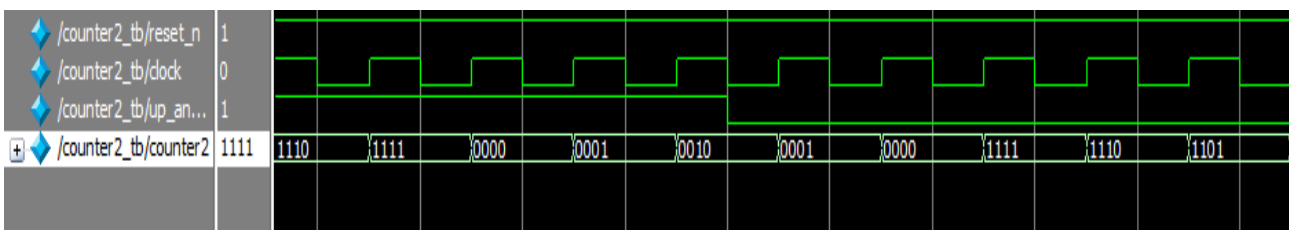
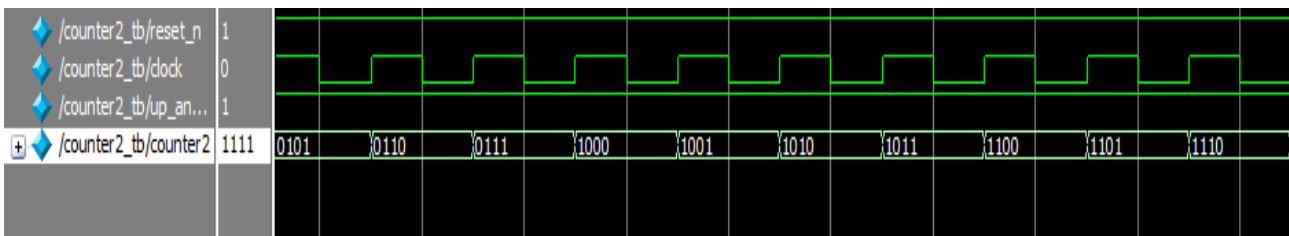
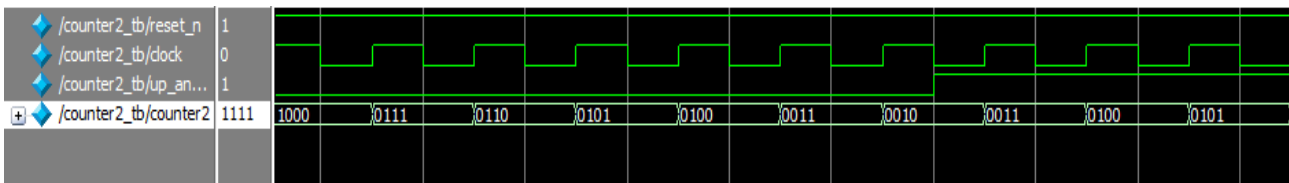
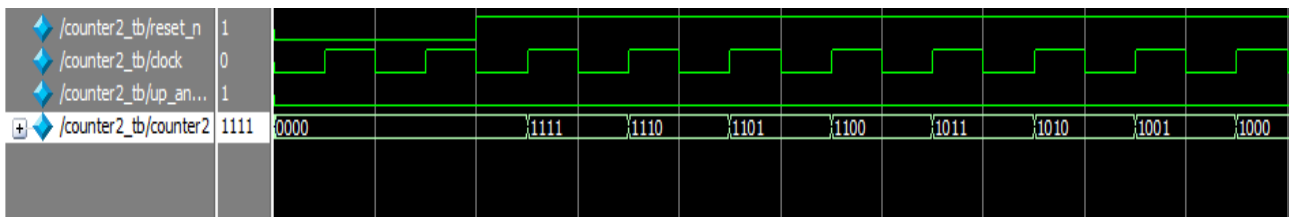
module counter2_tb;
//-----
// Various signals declaration
reg reset_n;
reg clock;
reg up_and_down;
wire [3:0] counter2;

//-----
// Various initializations
initial
begin
    clock <= 1'b0;
    up_and_down <= 1'b0;
    reset_n <= 1'b0;
    #40
    reset_n <= 1'b1;
end

//-----
// This is where I call my counter...
counter2 DUT (
    .clk(clock),
    .rst_n(reset_n),
    .up_down(up_and_down),
    .cnt(counter2)
);

//-----
// Clock generation
always
begin
    #10 clock <= ~clock;
end
always
begin
    #320 up_and_down <= ~up_and_down;
end
endmodule
```

Τα αποτελέσματα της προσομοίωσης φαίνονται στα σχήματα που ακολουθούν.



Όπως βλέπουμε από τα παραπάνω σχήματα αρχικά το σήμα reset είναι μηδέν και θα πάρει την τιμή του (1) μετά 40ns simulation time. Το ρολόι clock έχει περίοδο 20ns, ενώ το σήμα update το έχουμε βάλει να εναλλάσσεται από 0 σε 1 κάθε 320ns. Η αρχική τιμή του μετρητή είναι 0000, ενώ το update έχει τιμή 0. Έτσι ο μετρητής μας αρχίζει να μετράει προς τα κάτω, οπότε παίρνει τις τιμές:

0000->1111->1110->1101->1100->1011->1010->1001->1000->0111->
0110->0101->0100->0011->0010.

Έπειτα, επειδή στο χρόνο που έχουμε ορίσει το σήμα update παίρνει την τιμή ένα, ο μετρητής αρχίζει να μετράει προς τα πάνω, οπότε συνεχίζει παίρνοντας τις εξής τιμές:

0011->0100->0101->0110->0111->1000->1001->1010->1011->1100->
1101->1110->1111->0000->0001->0010.

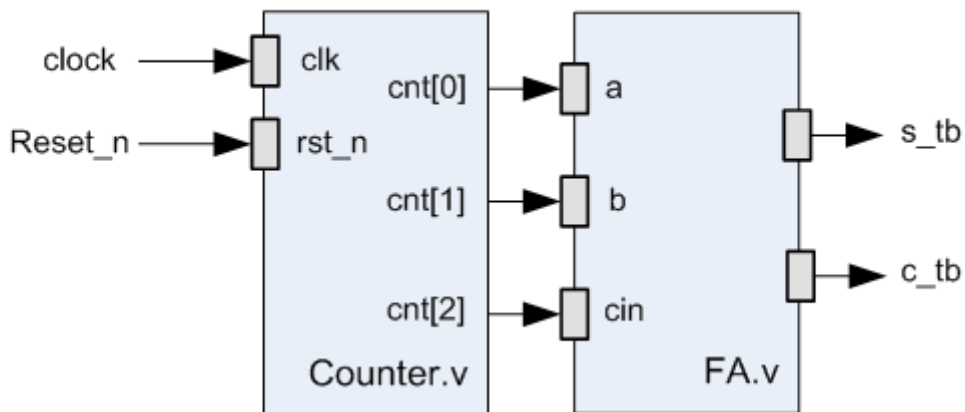
Στη συνέχεια αφού και πάλι το σήμα update άλλαξε τιμή από ένα σε μηδέν έχουμε και πάλι μέτρηση προς τα κάτω, δηλαδή 0001->0000 κ.ο.κ.

Μέρος 2

Γι' αυτό το μέρος της άσκησης θα συνδέσουμε τα κυκλώματα Counter και Full Adder για να διαπιστώσουμε την ορθή λειτουργία τους. Συγκεκριμένα οι έξοδοι του μετρητή δίνονται ως είσοδοι του πλήρη αθροιστή (οι δύο πρώτες ως είσοδοι και η τελευταία ως κρατούμενο εισόδου). Έτσι στην έξοδο του αθροιστή παράγεται το αποτέλεσμα S και Cout.

Τα παραπάνω φαίνονται καθαρά στο σχήμα ενώ οι κώδικες verilog των δύο κυκλωμάτων φαίνονται στους πίνακες:

FA_tb.v



```
module fa(a,b,cin,s,cout);
input a,b;
input cin;
output s;
output cout;
assign cout= (a & b) | ( cin & (a ^b) );
assign s=a^b^cin;
endmodule
```

```
module counter (clk, rst_n, cnt);
// This is the clock of the counter
input clk;
// This is the reset signal of the counter (Active low)
input rst_n;
// This is the counter bits
output [2:0] cnt;

reg [2:0] cnt;

always @(posedge clk or negedge rst_n)
begin
    if (!rst_n)
        cnt <= 3'b0;
    else
        cnt <= cnt + 3'b001;
    end
endmodule
```

Έτσι θα φτιάξουμε το αρχείο testbencher fa_tb, που θα λειτουργεί ως εξής:

- Παίρνει τις εξόδους του counter [0-2] και τις δίνει μια προς μία ως εισόδους στον fa, a,b και cin.
- Παίρνει τις εξόδους του fa [0-1], που αντιστοιχούν στα S και Cout.
- Καλεί τα δύο κυκλώματα fa και counter.

Ο κώδικας που υλοποιεί όλα τα παραπάνω είναι:

```
`timescale 1ns / 1ps

module FA_tb;
//-----
// Various signals declaration
reg reset_n;
reg clock;
wire [2:0] counter;
wire [1:0] FA;

//-----
// Various initializations
initial
begin
    clock <= 1'b0;
    reset_n <= 1'b0;
    #40
    reset_n <= 1'b1;
end

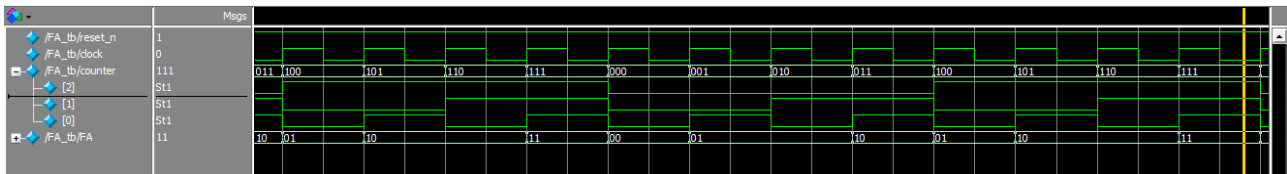
//-----
// This is where I call my counter ...
counter DUT1 (
    .clk(clock),
    .rst_n(reset_n),
    .cnt(counter)
);

fa DUT2 (
    .a(counter[0]),
    .b(counter[1]),
    .cin(counter[2]),
    .s(FA[0]),
    .cout(FA[1])
);

//-----
// Clock Generation
always begin
    #10    clock <= ~clock;
end

endmodule
```


Αφού κάνουμε compile προσομοιώνουμε με το Modelsim και λαμβάνουμε την εξής έξοδο στο wave:



όπου μπορούμε να διαπιστώσουμε την ορθή λειτουργία του προγράμματος, σύμφωνα με τις εσόδους και τις εξόδους του full adder.