Μέρος πρώτο - εξομοίωση

Ερωτήματα:

1. Κώδικας του RippleCounter module:

```
//4bit RippleCounter
module RippleCounter(q,clk,reset);
    output [3:0] q;
    input clk,reset;
    wire t_start,Q0,Q1,Q2,Q3;
    buf(t_start,1'b1);
    T_FF t_ff0(q[0],t_start,clk,reset);
    T_FF t_ff1(q[1],t_start,q[0],reset);
    T_FF t_ff2(q[2],t_start,q[1],reset);
    T_FF t_ff3(q[3],t_start,q[2],reset);
endmodule
```

2. Κώδικας του TestBenchRippleCounter module:

```
//4bit RippleCounter - TestBench
module TestBenchRippleCounter;
        reg clk,reset;
        wire [3:0] q;
        RippleCounter rc0(q,clk,reset);
        initial begin
                clk<=1'b0;
                #19 reset<=1'b1;
                #20 reset<=~reset;
        end
        always
                #10 clk <= ~clk;
endmodule
```

Στο initial αρχικοποιούμε το ρολόι στο 0 και τον μετρητή δίνοντας ένα παλμό στην είσοδο reset. Στο always με καθυστέρηση 10 χρονικές μονάδες αλλάζουμε την τιμή του ρολογιού.

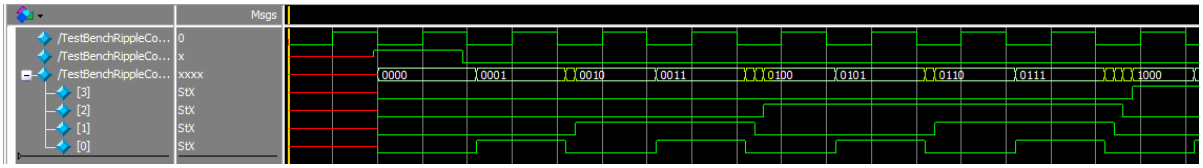
<u>Screenshots από την εξομοίωση του 4-bit Ripple Counter, όπου φαίνεται το rippling</u>:

Παρατηρούμε το rippling με κίτρινο χρώμα. Αυτό το φαινόμενο συμβαίνει λόγω του ότι το κάθε flip-flop παίρνει σαν ρολόι την έξοδο του προηγούμενου flip-flop, εκτός από το πρώτο.

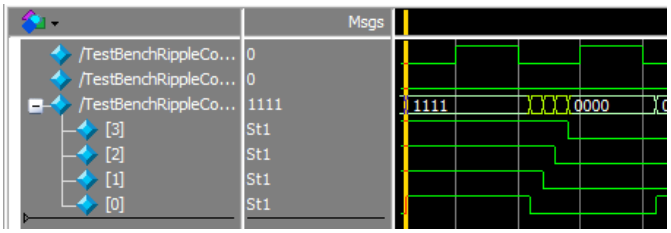
Μεταβολή του μετρητή από το $7(0111)_2$ στο $8(1000)_2$.

Δεκαδικό	**q[3]**	**q[2]**	**q[1]**	**q[0]**
7	0	1	1	1
8	1	0	0	0

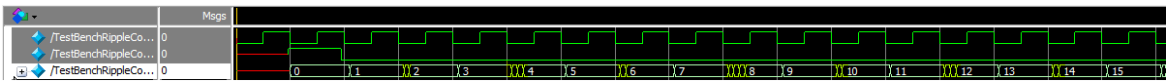
Ενδεικτική περίπτωση είναι η μεταβολή του μετρητή από το $7(0111)_2$ στο $8(1000)_2$, όπου το λιγότερο σημαντικό ψηφίο q[0] αλλάζει από 1 σε 0 και λόγω του ότι το flip-flop μας είναι αρνητικής ακμοπυροδότησης προκαλεί αλλαγή στο προηγούμενο ψηφίο q[1] από 1 σε 0, το οποίο προκαλεί επίσης αλλαγή στο προηγούμενο ψηφίο q[2] από 1 σε 0, το οποίο με την σειρά του προκαλεί αλλαγή στο περισσότερο σημαντικό ψηφίο q[3] από 0 σε 1 και έτσι ο μετρητής μας έχει μεταβεί από το 7 στο 8. Το rippling είναι η καθυστέρηση ανάμεσα στην αλλαγή των καταστάσεων 7 και 8, μιλώντας για την συγκεκριμένη περίπτωση.



Άλλη μια περίπτωση που φαίνεται έντονα το rippling είναι η μεταβολή από το $15(1111)_2$ ξανά στην αρχή, δηλαδή στο $0(0000)_2$. Η διαδικασία αλλαγής είναι η ίδια που περιγράψαμε παραπάνω.



Παρακάτω φαίνονται οι αριθμοί στο δεκαδικό από την εξομοίωση.



Κάθε T flip-flop πρέπει να περιμένει το προηγούμενο του να βγάλει την έξοδο του, ώστε να το χρησιμοποιήσει σαν ρολόι. Έτσι η μέγιστη καθυστέρηση που έχουμε εδώ είναι 8 χρονικές μονάδες από την στιγμή που το ρολόι θα δείξει από 1 σε 0 μέχρι την στιγμή που θα δούμε την έξοδο του μετρητή να αλλάζει στην επόμενη κατάσταση του. Αυτή η καθυστέρηση εμφανίζεται στις περιπτώσεις που έχουμε τις περισσότερες αλλαγές, δηλαδή στην μεταβολή της κατάστασης του από 7 σε 8 και από 15 σε 0.

3. **<u>Κώδικας του SynchronousCounter module</u>:**

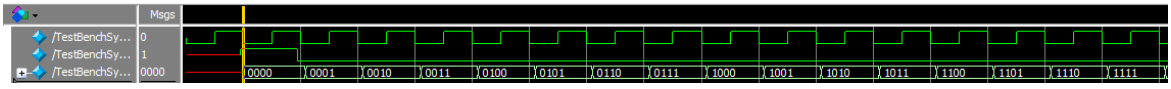
```
//4bit Synchronous Counter
module SynchronousCounter(q,clk,reset);
        output [3:0] q;
        input clk,reset;
        wire t2,t3;
        T_FF t_ff0(q[0],1'b1,clk,reset);
        T_FF t_ff1(q[1],q[0],clk,reset);
        and(t2,q[0],q[1]);
        T_FF t_ff2(q[2],t2,clk,reset);
        and(t3,t2,q[2]);
        T_FF t_ff3(q[3],t3,clk,reset);
endmodule
```

4. **<u>Κώδικας του TestBenchSynchronousCounter module</u>:**

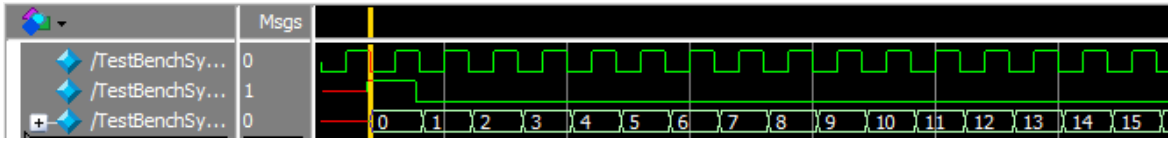
```
//4bit SynchronousCounter - TestBench
module TestBenchSynchronousCounter;
        reg clk,reset;
        wire [3:0] q;
        SynchronousCounter sc0(q,clk,reset);
        initial begin
                clk<=1'b0;
                #19 reset<=1'b1;
                #20 reset<=~reset;
        end
        always
           #10 clk<=~clk;
endmodule
```

Στον σύγχρονο μετρητή όλα τα T flip-flop παίρνουν το ίδιο ρολόι. Αυτός είναι ο λόγος που το rippling εξαφανίζεται.

Screenshots από την εξομοίωση του 4-bit Synchronous Counter:



Οι καταστάσεις του μετρητή στο δεκαδικό.



Η μόνη καθυστέρηση που έχουμε εδώ μεταξύ των καταστάσεων του σύγχρονου μετρητή, είναι η καθυστέρηση του T flip-flop για να αλλάξει την τιμή του, η οποία είναι ίση με 2 χρονικές μονάδες.

Μέρος δεύτερο - επιβεβαίωση του σχεδιασμού στην πλακέτα

Παρακάτω φαίνεται ο κώδικας του αρχείου Exe3_part2.v, που χρησιμοποιήσαμε για να συνθέσουμε τον σχεδιασμό μας στο λογισμικό Quartus II.

Κώδικας του Exe3_part2.v αρχείου:

```
//T flip-flop
module T_FF (q, t, clk, reset, enable);
        output q;
        input t, clk, reset, enable;
        reg q;

        always @ (posedge reset or negedge clk)
         if (reset)
           #1 q <= 1'b0;
         else if (t == 1 && enable)
           #2 q <= ~q;
endmodule
```

```
//4bit Synchronous Counter
module SynchronousCounter(q,clk,reset);
	output [3:0] q;
	input clk,reset;
	wire t2,t3,enable;
	reg [24:0] delay_counter;
	assign enable = (delay_counter == 25'd24999999) ? 1'b1 : 1'b0;
	T_FF t_ff0(q[0],1'b1,clk,reset,enable);
	T_FF t_ff1(q[1],q[0],clk,reset,enable);
	and(t2,q[0],q[1]);
	T_FF t_ff2(q[2],t2,clk,reset,enable);
	and(t3,t2,q[2]);
	T_FF t_ff3(q[3],t3,clk,reset,enable);
	always @ (negedge clk or posedge reset)
		if (reset)
		  delay_counter <= 25'd0;
		else if (enable)
		  delay_counter <= 25'd0;
		else
		  delay_counter <= delay_counter + 1'b1;
endmodule
```

Στο module του T flip-flop έχουμε προσθέσει το σήμα enable και έχουμε τροποποιήσει την if δομή, έτσι ώστε το flip-flop να αλλάζει τιμή όταν το t=1 και το enable=1. Στο module του σύγχρονου μετρητή έχουμε προσθέσει τον κώδικα του μετρητή καθυστέρησης ο οποίος μας δίνεται από την εκφώνηση και σκοπό έχει να καθυστερήσει τις αλλαγές μεταξύ των καταστάσεων του, ώστε να μπορέσουμε να τις παρατηρήσουμε με μια καθυστέρηση 0,5 sec. Παρατηρούμε πως με το reset όλα αρχικοποιούνται στο μηδέν. Τώρα όσον αφορά την λειτουργία του μετρητή όσο το delay_counter, δεν είναι ίσο με το 24999999 τότε το enable είναι μηδέν, το οποίο έχει ως αποτέλεσμα να εκτελείται το else της if και να αυξάνει τον delay_counter κατά ένα. Όλο αυτό το χρονικό διάστημα που το delay_counter παίρνει τιμές από 0 έως 24999999, δηλαδή 0,5 sec (αφού η συχνότητα ρολογιού είναι 50 MHz => 50000000Hz έχω 50000000 αλλαγές το 1 δευτερόλεπτο, άρα η μέτρηση από 0 έως 24999999 θα διαρκεί ακριβώς το μισό, άρα 0,5sec)

η κατάσταση στην πλακέτα παραμένει η ίδια. Όταν το delay_counter πάρει την τιμή 24999999, τότε το enable σήμα ενεργοποιείται (τιμή 1), με αποτέλεσμα η κατάσταση του σύγχρονου μετρητή να αλλάζει στην επόμενη και το delay_counter να αρχικοποιείται πάλι στο μηδέν με σκοπό και η νέα κατάσταση του μετρητή να καθυστερήσει 0,5 sec στην πλακέτα, ώστε να μπορέσουμε να την παρατηρήσουμε. Όταν γίνει αυτός ο κύκλος μεταβολών για όλες τις καταστάσεις του σύγχρονου μετρητή, τότε αυτός ξεκινάει να μετράει πάλι από το μηδέν σε ένα αδιάκοπο κύκλο. Επίσης με τον διακόπτη που αντιστοιχίσαμε στο reset μπορούμε να κάνουμε αρχικοποίηση του κυκλώματος όποια στιγμή θέλουμε δίνοντας έναν παλμό στο σχεδιασμό μας, ώστε να ξεκινήσει να μετράει πάλι από το μηδέν.

Η αντιστοίχιση των εισόδων και των εξόδων του σχεδιασμού με την πλακέτα έγινε με την βοήθεια του .csv αρχείου, αλλάζοντας τα ονόματα των I/O στο template που μας δόθηκε και εισάγοντας αυτά του σχεδιασμού μας.

```
clk,Input,PIN_Y2,2,B2_N0,3.3-V LVTTL,
q[3],Output,PIN_E24,7,B7_N1,2.5 V,
q[2],Output,PIN_E25,7,B7_N1,2.5 V,
q[1],Output,PIN_E22,7,B7_N0,2.5 V,
q[0],Output,PIN_E21,7,B7_N0,2.5 V,
reset,Input,PIN_Y23,5,B5_N2,2.5 V,
```