Δημήτρης Καρατζάς **icsd13072**
Νίκος Κατσιώπης **icsd13076**

ΑΝΑΦΟΡΑ 3ΗΣ ΑΣΚΗΣΗΣ ΣΧΕΔΙΑΣΗΣ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

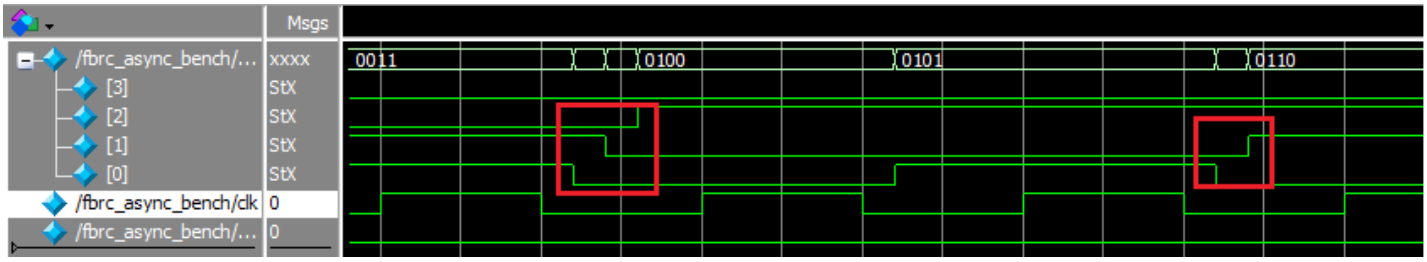
ΜΕΡΟΣ ΠΡΩΤΟ: ΕΞΟΜΟΙΩΣΗ

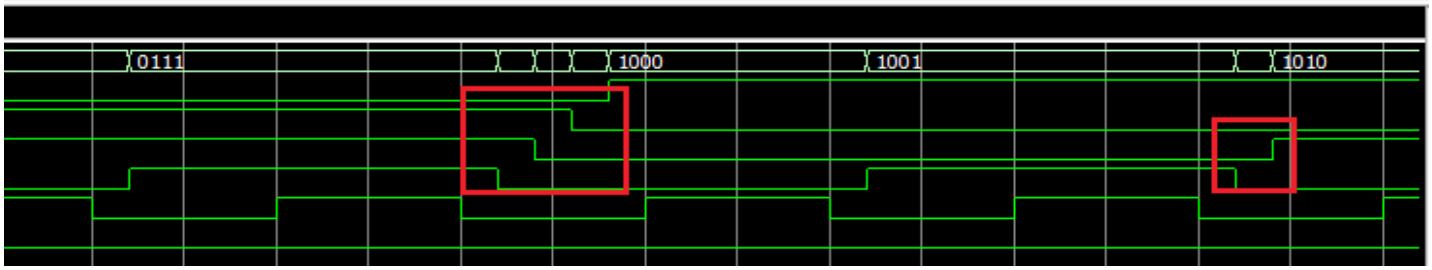
ΕΡΩΤΗΜΑΤΑ 1,2: Υλοποίηση 4bit Ripple Carry Counter (Ασύγχρονος) και simulation

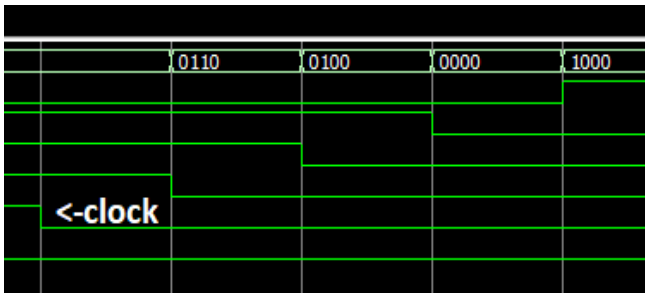
Υλοποιούμε τον 4bit ripple carry counter όπως μας δίνεται στην εκφώνηση. Ο Κώδικας verilog για τον counter, το bench, καθώς και τα αποτελέσματα της εξομοίωσης φαίνονται παρακάτω.

```
module FBRC_Async (
    output [3:0]q,
    input clk,
    input reset
);
reg one = 1'b1;
T_FF t_ff0(q[0], one, clk,  reset);
T_FF t_ff1(q[1], one, q[0], reset);
T_FF t_ff2(q[2], one, q[1], reset);
T_FF t_ff3(q[3], one, q[2], reset);

endmodule
```

```
module fbrc_async_bench;

wire[3:0] out;
reg clk;
reg reset;
initial begin
    reset = 0;
    clk = 0;
    #15 reset = 1;
    #10 reset = 0;
end

FBRC_Async fbrc_a( out, clk, reset);
always begin
   #10 clk= ~clk;
end
endmodule
```







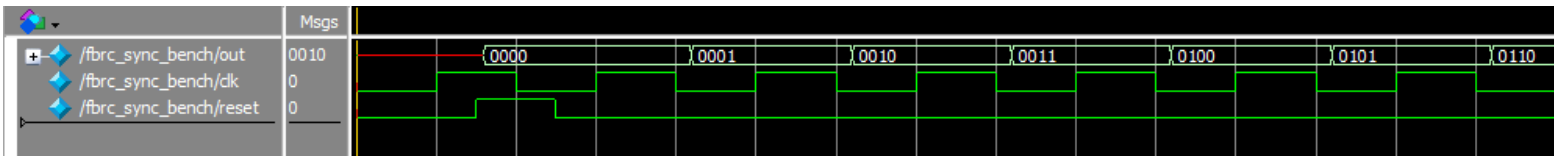
Στις παραπάνω εικόνες του simulation, παρατηρούμε το φαινόμενο του rippling, έχουμε σημειώσει με το κόκκινο ορθογώνιο το πότε εμφανίζεται το φαινόμενο.
Παρατηρούμε πως κάθε φορά που φτάνει η αρνητική ακμή του clock, η τελική έξοδος του κυκλώματος δεν υπολογίζεται ακαριαία αλλά μετά από ένα συγκεκριμένο χρονικό διάστημα (συγκεκριμένα μετά από 2ps, έχουν οριστεί στο module του T Flip Flop).
Στην τρίτη εικόνα έχουμε κάνει zoom στην εναλλαγή **0111 → 1000** ώστε να εξηγήσουμε τι γίνεται.
Στο παραπάνω παράδειγμα η προηγούμενη τιμή είναι η **0111.** Μόλις φτάσει η αρνητική ακμή του ρολογιού, το λιγότερο σημαντικό ψηφίο γίνεται 0, οπότε η ενδιάμεση τιμή της εξόδου είναι **0110.** Το ίδιο ισχύει και για τα επόμενα 2 ψηφία οπότε οι ενδιάμεσες έξοδοι είναι **0100** και **0000.** Τέλος το πιο σημαντικό ψηφίο γίνεται 1 και άρα η τελική έξοδος είναι **1000.** Για να γίνουν όλες αυτές οι εναλλαγές, χρειάστηκαν 2*4 = **8ps**. Αν είχαμε θέσει το clock να είναι μικρότερο των 8ps τότε θα είχαμε πρόβλημα, διότι δε θα προλάβαιναν να βγάλουν το σωστό αποτέλεσμα τα flip flop, λόγω του rippling, ότι δηλαδή το clock των ενδιάμεσων flip flop είναι η έξοδος των προηγούμενων flip flops → καθυστέρηση.

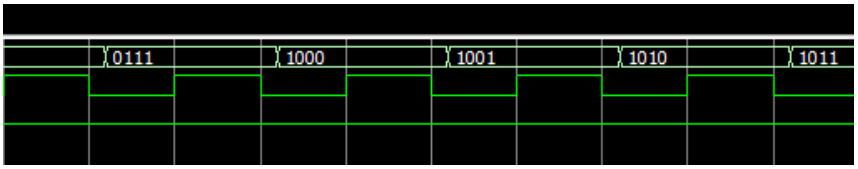
ΕΡΩΤΗΜΑΤΑ 3,4: Υλοποίηση 4bit Ripple Carry Counter (Σύγχρονος) και simulation

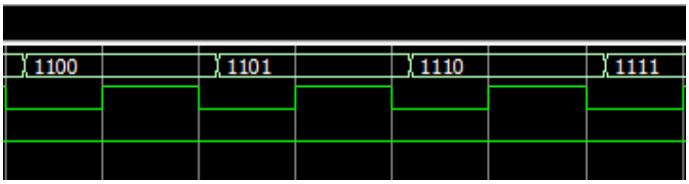
Όπως και με τον ασύγχρονο, έτσι και εδώ θα υλοποιήσουμε το κύκλωμα που μας δίνεται. Στις παρακάτω εικόνες είναι ο κώδικας verilog για τον counter, το test bench (είναι ακριβώς ίδια υλοποίηση με το test bench για τον ασύγχρονο counter, με τη διαφορά ότι εδώ προφανώς κάνουμε instantiate έναν σύγχρονο adder και όχι ασύγχρονο) καθώς και το simulation.

```
module FBRC_Sync (
    output [3:0]q,
    input clk,
    input reset
);
reg one = 1'b1;

wire q0q1;
//q0q1 = q1 and q2
and(q0q1, q[0], q[1]);

wire q2q1q0;
//q2q1q0 = q2 and q1 and q0
and(q2q1q0, q[2], q0q1);

T_FF t_ff0(q[0], one, clk,  reset);
T_FF t_ff1(q[1], q[0], clk, reset);
T_FF t_ff2(q[2], q0q1, clk, reset);
T_FF t_ff3(q[3], q2q1q0 , clk, reset);

endmodule
```

```
module fbrc_sync_bench;

wire[3:0] out;
reg clk;
reg reset;
initial begin
    reset = 0;
    clk = 0;
    #15 reset = 1;
    #10 reset = 0;
end

FBRC_Sync fbrc_a( out, clk, reset);
always begin
   #10 clk= ~clk;
end
endmodule
```







Στις εικόνες σχετικά με τον κώδικα απλώς υλοποιήσαμε το κύκλωμα.
Στις εικόνες του simulation, παρατηρούμε πως δεν υπάρχει το rippling. Σε κάθε αρνητική ακμή του ρολογιού, το κύκλωμα υπολογίζει ακαριαία το αποτέλεσμα. Αυτό συμβαίνει διότι όλα τα flip flops δέχονται το ίδιο clock, και άρα δεν περιμένουν να "τελειώσει" το προηγούμενο flip flop αλλά λειτουργούν παράλληλα.

ΜΕΡΟΣ ΔΕΥΤΕΡΟ: ΣΥΝΘΕΣΗ

Για να γίνει η εξομοίωση, κάνουμε όλες τις αλλαγές που ζητούνται στο αρχείο **.csv**.
Δηλαδή να δώσουμε τα ονόματα των clock, output και reset από τον δικό μας σχεδιασμό (αλλάξαμε το **CLOCK_50** σε **clk,** το **SW[17]** σε **reset** και τα **LEDG[3], LEDG[2], LEDG[1], LEDG[0]** σε **q[3], q[2], q[1], q[0].**
Τροποποιούμε και το όνομα του **.sdc** αρχείου ώστε να είναι ίδιο με του top-level module μας (το clk που έχει είναι ίδιο με το δικό μας όνομα του ρολογιού ("clk") οπότε δε το πειράζουμε.
Επίσης δημιουργούμε ένα νέο αρχείο verilog το οποίο περιλαμβάνει τα τροποποιημένα module για το 4bit ripple carry counter (sync) καθώς επίσης και το τροποποιημένο T Flip Flop, ενώ μετονομάσαμε το όνομα του σύγχρονου μετρητή από FBRC_Sync σε fbrc_sync_simulation για δικιά μας ευκολία. Επίσης προσθέσαμε τον κώδικα καθυστέρησης (τον μετρητή) σε αυτό το module.
Στο module του T Flip Flop απλώς προσθέσαμε στον έλεγχο που ελέγχει για το αν **t==1** , να ελέγχει επιπλέον αν το **enable == 1** (enable γίνεται ίσο με 1 όταν ο μετρητής του σύγχρονου μετρητή τελειώσει και ξανα αρχίσει το loop του).
Ακολουθώντας τις οδηγίες του βίντεο, περάσαμε από σύνθεση τον κώδικα.
Τέλος, στο εργαστήριο τρέξαμε τον κώδικα στο ολοκληρωμένο και δούλεψε κανονικά.

Ο κώδικας verilog για το module φαίνεται παρακάτω

```
module fbrc_sync_simulation (
    output [3:0]q,
    input clk,
    input reset
);
reg one = 1'b1;

wire q0q1;
//q0q1 = q1 and q2
and(q0q1, q[0], q[1]);

wire q2q1q0;
//q2q1q0 = q2 and q1 and q0
and(q2q1q0, q[2], q0q1);

reg [24:0] delay_counter;
wire enable;
assign enable = (delay_counter == 25'd24999999) ? 1'b1: 1'b0;
always @ (negedge clk or posedge reset) begin
    if (reset)
        delay_counter <= 25'd0;
    else if (enable)
        delay_counter <= 25'd0;
    else
        delay_counter <= delay_counter + 1'b1;
end

T_FF t_ff0(q[0], one, clk,  reset, enable);
T_FF t_ff1(q[1], q[0], clk, reset, enable);
T_FF t_ff2(q[2], q0q1, clk, reset, enable);
T_FF t_ff3(q[3], q2q1q0 , clk, reset, enable);

endmodule
```

```
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 == 1)
        #2 q <= ~q;
endmodule
```