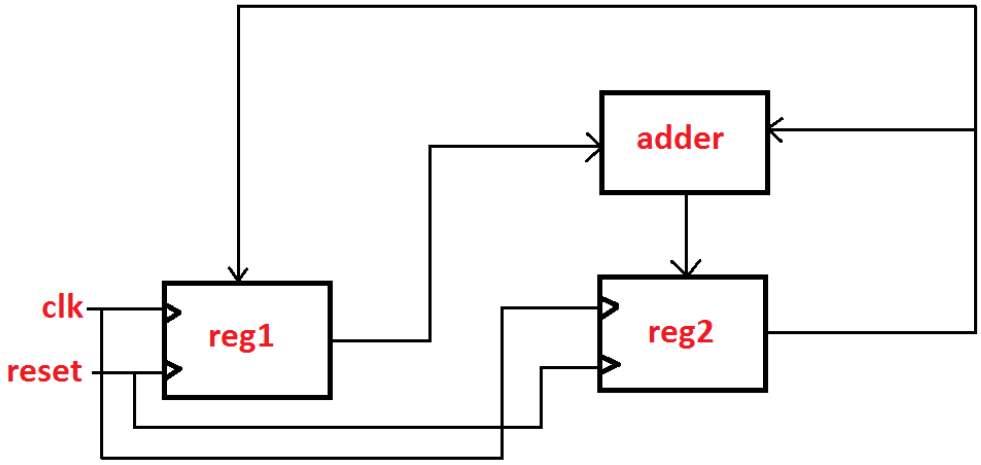
Block διάγραμμα κυκλώματος



Το κύκλωμα αποτελείται από 2 καταχωρητές και έναν αθροιστή. Οι καταχωρητές μπορούν να αλλάξουν τις τιμές τους μόνο στις θετικές ακμές του ρολογιού. Στις άλλες περιπτώσεις κρατάνε αποθηκευμένες τις προηγούμενες τιμές τους. Όταν ενεργοποιήσουμε το reset τότε ο καταχωρητής reg1 αρχικοποιείται σε 0 και ο reg2 σε 1. Μετά τις αλλαγές ο adder υπολογίζει το reg1+reg2. Στις θετικές ακμές του ρολογιού ο reg1 παίρνει την τιμή του reg2 και ο reg2 παίρνει την έξοδο του adder, όπου αυτές οι τιμές θα χρησιμοποιηθούν για να μας παράγουν τον επόμενο όρο της ακολουθίας. Κάθε φορά οι τιμές προλαβαίνουν να κλειδώσουν πριν υπολογιστεί το άθροισμα. Η έξοδος μας είναι ο reg1 και όχι η έξοδος του adder, ώστε να μπορέσουμε να παρατηρήσουμε όλους τους όρους της ακολουθίας Fibonacci κατά την εξομοίωση.

Μέρος πρώτο - εξομοίωση

Ερωτήματα:

1. **Κώδικας του Fibonacci module:**

```
module Fibonacci(reg1,clk,reset);
        output reg [7:0] reg1;
        input clk,reset;
        reg [7:0] reg2;
        wire [7:0] q;
        assign q=reg1+reg2;
        always @(posedge clk or posedge reset)
```

```
        if(reset)
            begin
                reg1<=8'd0;
                reg2<=8'd1;
             end
        else
            begin
                if(reg1===8'd233)
                    begin
                        reg1<=8'd0;
                        reg2<=8'd1;
                    end
                else
                    begin
                        reg1<=reg2;
                        reg2<=q;
                    end
            end
endmodule
```

Ο παραπάνω κώδικας περιγράφει το κύκλωμα που φαίνεται στην εικόνα. Το always ενεργοποιείται κατά τις θετικές ακμές του ρολογιού ή κατά την ενεργοποίηση του reset. Όταν βγει στην έξοδο και ο αριθμός 233 της ακολουθίας τότε το κύκλωμα αρχικοποιείται ξανά και η έξοδος αρχίζει πάλι από το 0.

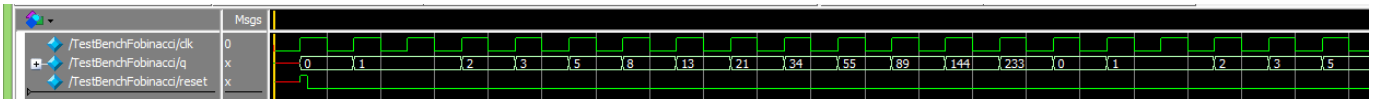
2. Για να εξομοιώσουμε το παραπάνω κύκλωμα γράφουμε το παρακάτω TestBench.

<u>Κώδικας του TestBenchFobinacci module:</u>

```
module TestBenchFobinacci;
   wire [7:0] q;
   reg clk,reset;
        Fibonacci f(q,clk,reset);
        initial
             begin
                clk<=1'b0;
                #10 reset<=1'b1;
```

```
            #3 reset<=~reset;
        end
    always
        #10 clk<=~clk;
endmodule
```

Παρακάτω φαίνονται ένα screenshot που μας δείχνει όλους τους όρους της ακολουθίας από 0 έως 233 που προκύπτουν από την εξομοίωση του κυκλώματος.



Παρατηρούμε πως η έξοδος αλλάζει μόνο κατά τις θετικές ακμές του ρολογιού και πως το 1 διαρκεί για δύο κύκλους επειδή στην πραγματικότητα εμφανίζεται 2 φορές στην έξοδο.

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

```
module FABehav(input a,b,cin,output sum,cout);
  assign {cout,sum} = a + b + cin;
endmodule
```

<u>Κώδικας του FourBitRippleCarryAdder module:</u>

```
module FourBitRippleCarryAdder(a,b,cstart,s);
  output [4:0] s ;
  input [3:0] a,b;
  input cstart;
  wire fa1_Cout,fa2_Cout,fa3_Cout;
  FABehav fa1(a[0], b[0], cstart, s[0], fa1_Cout);
  FABehav fa2(a[1], b[1], fa1_Cout, s[1], fa2_Cout);
  FABehav fa3(a[2], b[2], fa2_Cout, s[2], fa3_Cout);
  FABehav fa4(a[3], b[3], fa3_Cout, s[3], s[4]);
endmodule
```

Κώδικας του EightBitRippleCarryAdder module:

```
module EightBitRippleCarryAdder(a,b,cstart,s);
  output [8:0] s;
  input [7:0] a,b;
  input cstart;
  wire [9:0] c;
  FourBitRippleCarryAdder fbrca1(a[3:0],b[3:0],cstart,c[4:0]);
  FourBitRippleCarryAdder fbrca2(a[7:4],b[7:4],c[4],c[9:5]);
  assign s[3:0]=c[3:0];
  assign s[8:4]=c[9:5];
endmodule
```

Κώδικας του RipleCarryAdderFibonacci module:

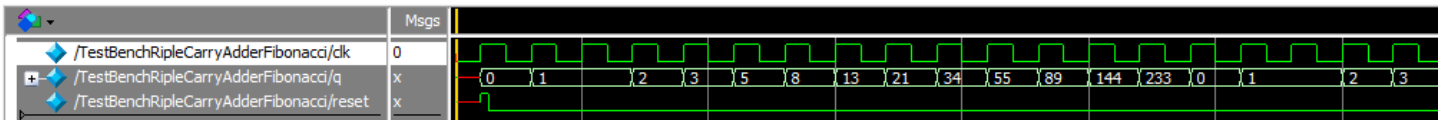
```
module RipleCarryAdderFibonacci(reg1,clk,reset);
        output reg [7:0] reg1;
        input clk,reset;
        reg [7:0] reg2;
        wire [7:0] q;
        EightBitRippleCarryAdder ebrca0(reg1,reg2,1'b0,q);
        always @(posedge clk or posedge reset)
          if(reset)
            begin
              reg1<=8'd0;
              reg2<=8'd1;
            end
          else
            begin
              if(reg1===8'd233)
                begin
                  reg1<=8'd0;
                  reg2<=8'd1;
                end
              else
                begin
                  reg1<=reg2;
                  reg2<=q;
                end
```

```
        end
endmodule
```

Κώδικας του TestBenchRipleCarryAdderFibonacci module:

```
module TestBenchRipleCarryAdderFibonacci;
   wire [7:0] q;
   reg clk,reset;
       RipleCarryAdderFibonacci rcaf0(q,clk,reset);
       initial
           begin
              clk<=1'b0;
              #10 reset<=1'b1;
              #3 reset<=~reset;
            end
       always
           #10 clk<=~clk;
endmodule
```

Παρακάτω φαίνεται ένα screenshot από την εξομοίωση του κυκλώματος το οποίο μας δείχνει όλους τους όρους της ακολουθίας Fibonacci από 0 έως 233.



Μέρος δεύτερο – επιβεβαίωση του σχεδιασμού στην πλακέτα

Κώδικας του Fibonacci module το οποίο εξομοιώθηκε στην πλακέτα:

```
module Fibonacci(HEX0,HEX1,HEX2,clk,reset);
output reg [6:0] HEX0,HEX1,HEX2;
input wire clk,reset;
reg [7:0] reg2,reg1;
wire [7:0] q;
wire enable;
```

```
reg [24:0] delay_counter;
assign q=reg1+reg2;
assign enable = (delay_counter == 25'd24999999) ? 1'b1 : 1'b0;
always @(posedge clk or posedge reset)
begin
   if(reset)
       begin
                reg1<=7'd0;
                reg2<=7'd1;
                delay_counter<=25'd0;
       end
       else if(enable)
       begin
                reg1<=reg2;
                reg2<=q;
                delay_counter<=25'd0;
       end
   else
         delay_counter<=delay_counter+1'b1;
end

always @(reg1)
       begin
         case (reg1)
              8'd0:begin//periptwsh apeikonishs tou 0
                                  HEX2<=7'b1111111;
                                  HEX1<=7'b1111111;
                                  HEX0<=7'b1000000;
                             end
              8'd1:begin//periptwsh apeikonishs tou 1
                                  HEX2<=7'b1111111;
                                  HEX1<=7'b1111111;
                                  HEX0<=7'b1111001;
                             end
              8'd2:begin//periptwsh apeikonishs tou 2
                                  HEX2<=7'b1111111;
                                  HEX1<=7'b1111111;
```

```
                HEX0<=7'b0100100;
            end
8'd3:begin//periptwsh apeikonishs tou 3
                HEX2<=7'b1111111;
                HEX1<=7'b1111111;
                HEX0<=7'b0110000;
            end
8'd5:begin//periptwsh apeikonishs tou 5
                HEX2<=7'b1111111;
                HEX1<=7'b1111111;
                HEX0<=7'b0010010;
            end
8'd8:begin//periptwsh apeikonishs tou 8
                HEX2<=7'b1111111;
                HEX1<=7'b1111111;
                HEX0<=7'b0000000;
            end
8'd13:begin//periptwsh apeikonishs tou 13
                HEX2<=7'b1111111;
                HEX1<=7'b1111001;
                HEX0<=7'b0110000;
            end
8'd21:begin//periptwsh apeikonishs tou 21
                HEX2<=7'b1111111;
                HEX1<=7'b0100100;
                HEX0<=7'b1111001;
            end
8'd34:begin//periptwsh apeikonishs tou 34
                HEX2<=7'b1111111;
                HEX1<=7'b0110000;
                HEX0<=7'b0011001;
            end
8'd55:begin//periptwsh apeikonishs tou 55
                HEX2<=7'b1111111;
                HEX1<=7'b0010010;
                HEX0<=7'b0010010;
            end
```

```
                8'd89:begin//periptwsh apeikonishs tou 89
                                HEX2<=7'b1111111;
                                HEX1<=7'b0000000;
                                HEX0<=7'b0010000;
                            end
                8'd144:begin//periptwsh apeikonishs tou 144
                                HEX2<=7'b1111001;
                                HEX1<=7'b0011001;
                                HEX0<=7'b0011001;
                            end
                8'd233:begin//periptwsh apeikonishs tou 233
                                HEX2<=7'b0100100;
                                HEX1<=7'b0110000;
                                HEX0<=7'b0110000;
                            end
                default:begin//mh enammenomenh timh emfanizoume to EEE
                                HEX2<=7'b0000110;
                                HEX1<=7'b0000110;
                                HEX0<=7'b0000110;
                            end
            endcase
    end
endmodule
```

Σε αυτό το module που εξομοιώθηκε στην πλακέτα όταν η έξοδος πάρει την τιμή 233, τότε δεν αρχικοποιείται ξανά, αλλά εμφανίζεται σαν έξοδος η default τιμή της case. Αν θέλουμε να αρχίσει το κύκλωμα από την αρχή μπορούμε να το αρχικοποιήσουμε ξανά χρησιμοποιώντας το sw[17], το οποίο λειτουργεί σαν reset για το συγκεκριμένο κύκλωμα μας.