2η Εργαστηριακή Άσκηση Σχεδιασμός Συστημάτων VLSI

Κόλλιας Ιωάννης 1084578

Ποδηματά Θεοδώρα Παναγιώτα 1090031

4ο Έτος 2023-2024

Ερώτημα 1

Αρχικά, δημιουργήσαμε έναν συγκριτή.

```
module comparator #(parameter n=4, parameter m=3)(
        input [n+m-1:0] in1,
        input [n+m-1:0] in2,
        output [n+m-1:0] out
        );

wire signed [n-1:0] c1;
wire signed [n-1:0] c2;

assign c1=in1[n-1:0];
assign c2=in2[n-1:0];
assign out=(c1>c2)?in1:in2;

endmodule
```

Στη συνέχεια, δημιουργήσαμε το parallelargmax όπου εφαρμόζει έναν παράλληλο αλγόριθμο argmax για να βρει το δείκτη(output) της μέγιστης τιμής μεταξύ πολλαπλών δεδομένων. Η λογική του κώδικά μας είναι οι συγκρίσεις των δεδομένων να γίνονται με δεντρική δομή, για να έχουμε παράλληλες συγκρίσεις. Εάν το k είναι περιττό, το αποτέλεσμα σύγκρισης μεταξύ της ρίζας και του τελευταίου κόμβου φύλλου καθορίζει τον δείκτη μέγιστης τιμής. Εάν το k είναι άρτιο, το αποτέλεσμα προκύπτει απευθείας από τη ρίζα του δυαδικού δέντρου.

```
module parallelargmax #(parameter k=10, parameter n=4, parameter m=$clog2(k))(

        input [k*n-1:0] data,
        output [m-1:0] result);
```

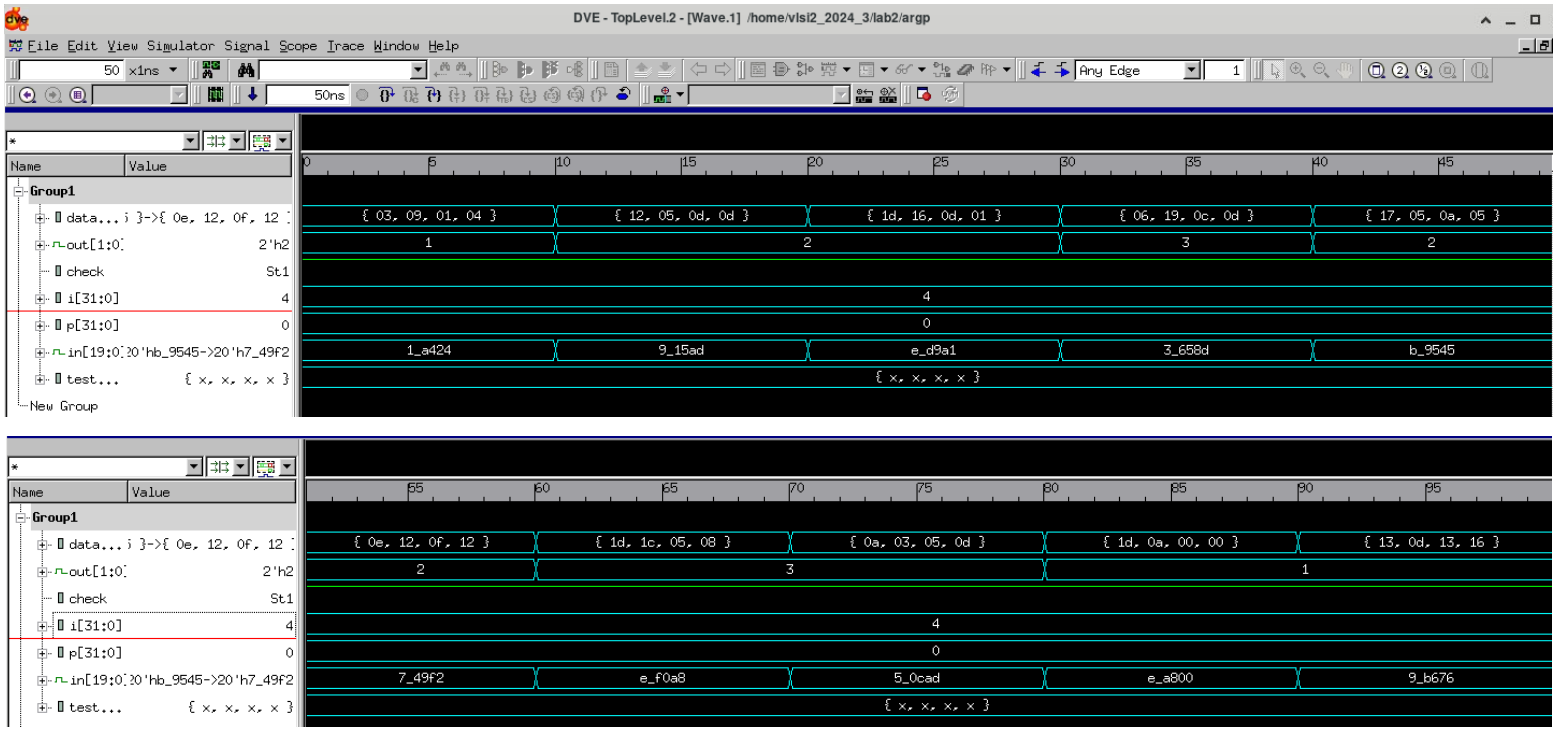
```
wire [n+m-1:0] data2 [k-1:0];
wire [n+m-1:0] tree [k-1:1];
wire [n+m-1:0] last;
genvar gi;

generate
for(gi=0;gi<k;gi=gi+1)begin:gen
        wire [m-1:0] num= k-1-gi;
        wire [n-1:0] value = data[gi*n+n-1:gi*n];
        assign data2[gi]={num,value};
end

for(gi=k/2;gi<=k-1;gi=gi+1)begin:gen2
        if(gi!=k-1||k%2==0)
        comparator #(.n(n),.m(m)) compare(.in1(data2[(gi-k/2)*2]),.in2(data2[(gi-
k/2)*2+1]),.out(tree[gi]));
        if(gi==k-1&&k%2==1)
        assign tree[gi]=data2[k-1];
end

for(gi=k/2-1;gi>0;gi=gi-1)begin:gen3
        comparator #(.n(n),.m(m))
compare(.in1(tree[2*gi]),.in2(tree[2*gi+1]),.out(tree[gi]));
end

if(k%2==1)
        begin
                comparator #(.n(n),.m(m)) compare(.in1(tree[1]),.in2(tree[k-
1]),.out(last));
                assign result=last[n+m-1:n];
        end
```

```
else
        begin
                assign last=tree[1];
                assign result=last[n+m-1:n];
        end
endgenerate
endmodule
```

Testbench

```
module argptb;

reg check;
reg signed [4:0] data [3:0];
reg test [3:0];
wire  [19:0] in;
wire [1:0] out;

assign in={data[3],data[2],data[1],data[0]};
integer i,p=0;

parallelargmax #(.k(4),.n(5)) argp(.data(in),.result(out));

initial
begin

        repeat(10) begin
                for(i=0;i<4;i=i+1)begin
                        data[i]=$random;
                        end
```

```
                check=1'b1;
        #1      for(i=1;i<4;i=i+1)begin
                        if(data[i]>data[out])
                        check=1'bx;
                        end
        end
end
endmodule
```



Ερώτημα 2

Όπως γνωρίζουμε ένας shift register αποτελείται από flip-flop συνδεδεμένα στη σειρά, όπου μετατοπίζουν ένα bit προς το λιγότερο σημαντικό ψηφίο, με βάση την εκφώνηση. Η ενεργοποίηση της ολίσθησης γίνεται μέσω του σήματος en και η είσοδος φορτώνεται μέσω του σήματος ενεργοποίησης παράλληλης φόρτωσης pl (όταν pl=1). Το πιο σημαντικό bit του καταχωρητή φορτώνεται με τη σειριακή είσοδο si. Αν το pl=0 και το en=0, ο καταχωρητής shift register διατηρεί την τρέχουσα κατάστασή του χωρίς καμία αλλαγή στο περιεχόμενό του.

module shift_register # (parameter n=8)(input si,pl,en,clk,rstn, input [n-1:0] din, output reg so);

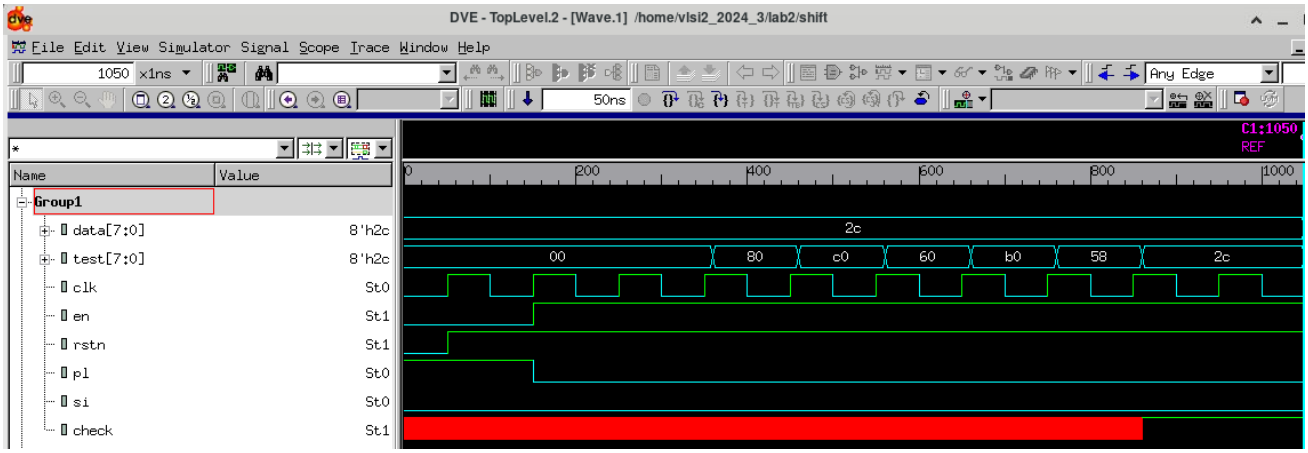
```
        reg [n-1:0] register;


        always @(posedge clk or negedge rstn) begin
                if(rstn==0)begin
                register<=0;
                end
        else if(pl==1)begin
                register<=din;
                end
        else if(en==1)begin
                so<=register[0];
                register<=register>>1;
                register[n-1]<=si;
        end
        end
endmodule
```

Testbench

Στο testbench μας ελέγχουμε την λειτουργία του παραπάνω shift register. Για ένα shift register των 8-bit αντιστοιχούμε τα σήματα του κώδικά μας με του testbench. Ορίζουμε ένα ρολόι που εναλλάσσεται κάθε 5 χρονικές μονάδες, δημιουργώντας ένα σήμα ρολογιού με περίοδο 10 μονάδων χρόνου. Αρχικοποιούμε τα σήματα με non-blocking για τεθούν ταυτόχρονα, θέτουμε μια τιμή στα δεδομένα (data) και ορίζουμε την μεταβλητή test για να μπορέσουμε να ελέγξουμε αν είναι ορθή η λύση μας. Εντός του βρόχου, το test μετατοπίζεται δεξιά κατά 1 bit και η έξοδος του καταχωρητή μετατόπισης (so) εκχωρείται στο πιο σημαντικό bit (test[7]). Μετά τη διαδικασία δοκιμής, ο κωδικός ελέγχει εάν τα δεδομένα είναι ίσα με το test. Εάν είναι ίσα, ο έλεγχος εκχωρείται 1. Διαφορετικά, εκχωρείται 1'bx, υποδεικνύοντας άγνωστη τιμή.

```
module shifttb;

        reg [7:0] data, test;
```

```
	reg clk, en, rstn, pl, si;
	reg check;
	wire so;

	shift_register #(8) shift (.si(si),.pl(pl),.en(en),.clk(clk),.rstn(rstn),.din(data),.so(so));

	always #5 clk=~clk;

	initial begin
	clk<=0;
	rstn<=0;
	en<=0;
	pl<=1;
	si<=0;

	data<=8'h2c;
	test<=8'h00;

	#5 rstn=1;
	#10 pl=0;
	en=1;

	repeat(8)@(posedge clk) begin
		#1 test=test>>1;
		test[7]<=so;
	end

	if(data==test)
		check<=1;
	else
		check<=1'bx;
	end
endmodule
```



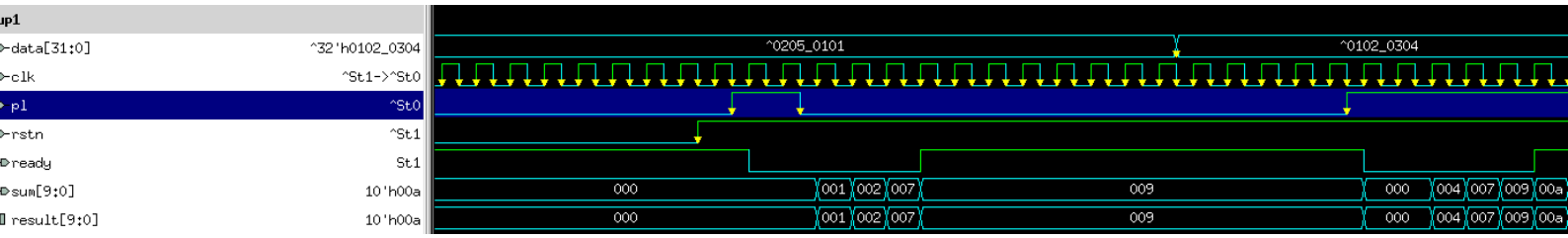
Ερώτημα 3

Ο παρακάτω κώδικας υλοποιεί έναν accumulator όπου δέχεται μια παράλληλη είσοδο από αριθμούς. Όταν το pl=1 τότε οι τιμές περνάνε στον καταχωρητή και ο accumulator υπολογίζει το άθροισμα ανά χτύπο ρολογιού (LSB->MSB). Όταν το άθροισμα είναι έτοιμο το σήμα ready=1. Το add ελέγχει εάν πρέπει να προστεθούν νέα δεδομένα στον καταχωρητή (1) ή να μετατοπιστούν (0).

```
module accumulator #(parameter k=4, parameter n=8, parameter m=$clog2(k))(input [k*n-1:0] data, input pl, input clk, input rstn, output ready, output [n+m-1:0] sum);

genvar gi;

reg [k*n-1:0] register;

wire add;

reg [n+m-1:0] result;

reg [m:0] counter;

reg r;


always @(posedge clk or negedge rstn)begin
        if(rstn==0)
                register<=0;
        else begin
                if(add)
                        register<=register>>n;
                else begin
```

```
                if(pl)
                register<=data;
            end
        end
end

always @(posedge clk or negedge rstn)begin
        if(rstn==0||(pl==1&&counter==0))
            result<=0;
        else begin
            if(add)
                result=result+register[n-1:0];
        end
end

always @(posedge clk or negedge rstn)begin
        if(rstn==0)
            counter<=0;
        else begin
            if(counter!=0&&add==1)
                counter<=counter-1;
            else if(pl)
                counter<=k;
        end
end

always @(posedge clk or negedge rstn)begin
        if(rstn==0)
            r<=0;
```

```
        else begin
        if(counter==0)
                r<=0;
        else
                r<=1;
        end
end

assign add=r;
assign ready=(counter==0)?1:0;
assign sum=result;

endmodule
```



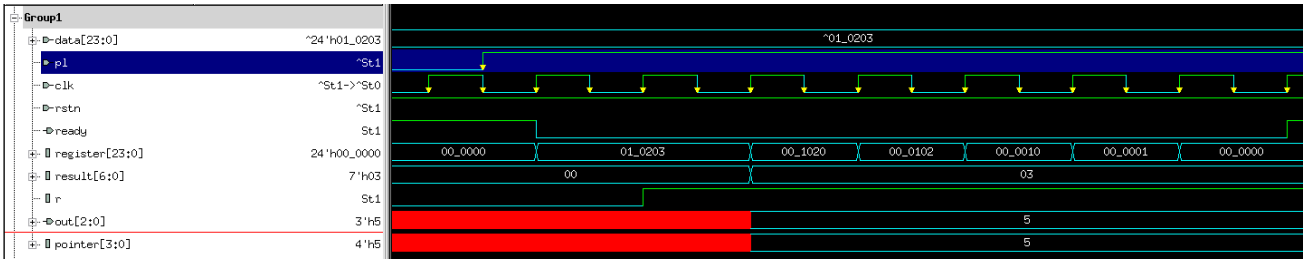
Ερώτημα 4

Το argmax_serial βρίσκει τον δείκτη της μέγιστης τιμής των δεδομένων. Ο κώδικας συγκρίνει συνεχώς το τρέχον σύνολο δεδομένων με την προηγουμένως αποθηκευμένη μέγιστη τιμή. Εάν το τρέχον σύνολο δεδομένων είναι μεγαλύτερο από την αποθηκευμένη μέγιστη τιμή, ενημερώνει το αποτέλεσμα με τη νέα μέγιστη τιμή και ενημερώνει τον δείκτη με την τρέχουσα τιμή του μετρητή (που υποδεικνύει το δείκτη της μέγιστης τιμής). Το compare ελέγχει εάν πρέπει να προστεθούν νέα δεδομένα στον καταχωρητή (1) ή να μετατοπιστούν (0). Όταν pl=1 και counter=0 το κύκλωμά μας διαβάζει τις νέες εισόδους και ξεκινά ο νέος υπολογισμός.

```
module argmax_serial #(parameter k=6, parameter n=4, parameter
m=$clog2(k))(input [k*n-1:0] data, input pl, input clk, input rstn, output ready,
output [m-1:0] out);
        genvar gi;
        reg [k*n-1:0] register;
        wire compare;
```

```
reg [n+m-1:0] result;
reg [m:0] counter;
reg r;
reg [m:0] pointer;

always @(posedge clk or negedge rstn)begin
        if(rstn==0)
                register<=0;
        else
                begin
                if(compare)
                        register<=register>>n;
                else begin
                        if(pl)
                                register<=data;
                end
        end
end

always @(posedge clk or negedge rstn)begin
        if(rstn==0||(pl==1&&counter==0))
                result<=0;
        else begin
                if(compare)
                        if(result<register[n-1:0])begin
                                result<=register[n-1:0];
                                pointer<=counter-1;
                        end
        end
end
```

```
        always @(posedge clk or negedge rstn)begin
                if(rstn==0)
                        counter<=0;
                else begin
                if(counter!=0&& compare==1)
                        counter<=counter-1;
                else if(pl)
              counter<=k;
                end
         end

         always @(posedge clk or negedge rstn)begin
                if(rstn==0)
                        r<=0;
                else begin
                        if(counter==0)
                        r<=0;
                else
                        r<=1;
                end
         end

assign compare=r;
assign ready=(counter==0)?1:0;
assign out=pointer;

endmodule
```



Testbench(δεν δουλεύει καλά)

```
module argptb;


reg check;

reg [19:0] data;

reg test [3:0];

wire  [19:0] in;

wire [1:0] out;

reg clk,pl,rstn;

wire ready;

reg [4:0] data2 [3:0];

wire [19:0] c;


assign c = {data2[0],data2[1],data2[2],data[3]};

assign in = data;

integer i,p=0;


argmax_serial #(.k(4),.n(5)) args(.data(in),.pl(pl),.clk(clk),.rstn(rstn),.ready(ready),.out(out));


always #5 clk=~clk;


initial

begin


data<=c;

rstn<=0;

clk<=0;

#5 rstn=1;
```

```
pl<=1;
#10 pl<=0;

        repeat(10) begin
                        for(i=0;i<4;i=i+1)begin
                        data[i]=$random;
                        end

                check=1'b1;
        @(posedge ready) begin
        #1
                        if(data[0]>data[out])
                        check=1'bx;

/*                      pl<=1;
                        #10 pl<=0;
                for(i=0;i<4;i=i+1)begin


                        data[i+4:i]=$random;
                        end
                in<= */
        end
end
end
endmodule
```