Visvesvaraya National Institute of Technology (VNIT), Nagpur

Digital Hardware Design (ECP312)

Lab Report

Submitted by :
Riya Deshpande(BT20ECE029)
Chinmay Patil (BT20ECE074)
Vedan Shet (BT20ECE093)
Valerian Minz(BT20ECE108)
Semester 6

Submitted to :
Dr. Anamika Singh
(Lab Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

Convolution on FPGA

1.1 Theory: Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Although one-time programmable (OTP) FPGAs are available, the dominant types are SRAM based which can be reprogrammed as the design evolves. Convolution is a mathematical operation on two functions (f and g) that produces a third function (f*g) that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reflected about the y-axis and shifted. The choice of which function is reflected and shifted before This can be similarly be applied to discrete sequences the integral does not change the integral result.The integral is evaluated for all values of shift, producing the convolution function.

1.2 Code:

Convolution Code with comments

```
module convolution(data,arraysel,possel,oppossel,Y,on1,on2);
input [3:0]data;
input arraysel;
input [1:0]possel;
input [2:0]oppossel;
output reg [7:0]Y ;
] A[3:0]  ;
reg [3:0] B[3:0]  ;


always @(data,arraysel,possel)  //to enter data in reg
    begin
        if (arraysel == 0) // to enter data in A
            begin
                case(possel)
                    2'b00: A[0] ≤ data;
                    2'b01: A[1] ≤ data;
                    2'b10: A[2] ≤ data;
                    2'b11: A[3] ≤ data;
                endcase
            end
        else    // to enter data in B
            begin
                case(possel)
                    2'b00: B[0] ≤ data;
                    2'b01: B[1] ≤ data;
                    2'b10: B[2] ≤ data;
                    2'b11: B[3] ≤ data;
                endcase
            end
    end

integer i = 0;
reg [1:0] sel1;

always @(negedge on1)
begin
    C[0] = 8'd0;
    C[1] = 8'd0;
    C[2] = 8'd0;
    C[3] = 8'd0;
    C[4] = 8'd0;
    C[5] = 8'd0;
    C[6] = 8'd0;
    for(i=0;i<4;i=i+1)
    begin
        C[i] = A[0]*B[i] + C[i];
        C[i+1] = A[1]*B[i] + C[i+1];
        C[i+2] = A[2]*B[i] + C[i+2];
        C[i+3] = A[3]*B[i] + C[i+3];
    end
end
always @(oppossel)
begin
    case(oppossel)
        3'b000: Y ≤ C[0];
        3'b001: Y ≤ C[1];
        3'b010: Y ≤ C[2];
        3'b011: Y ≤ C[3];
        3'b100: Y ≤ C[4];
        3'b101: Y ≤ C[5];
        3'b110: Y ≤ C[6];
        3'b111: Y ≤ C[3];
    endcase
end
endmodule
```

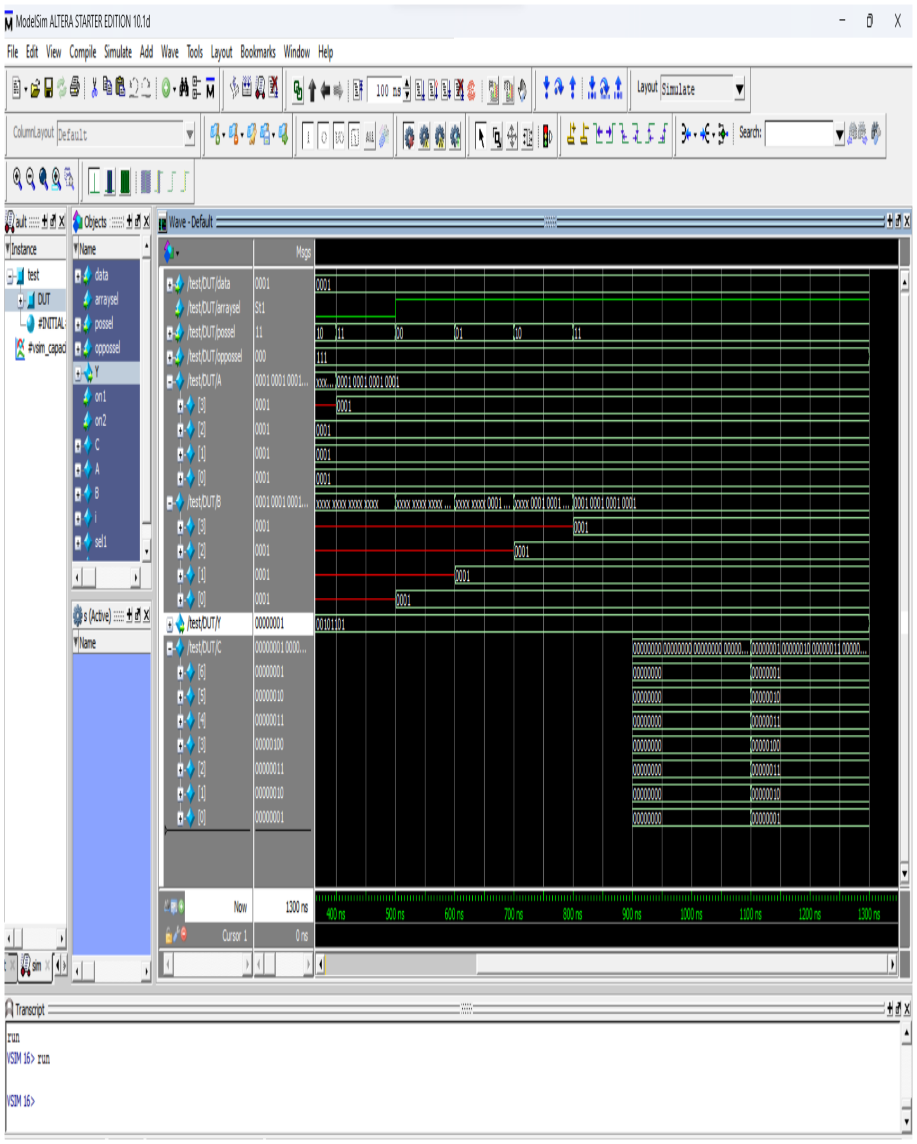
Test Bench Code with comments

```
 `timescale 100ns/1ns
module test;

reg [3:0] data;
reg arraysel;
reg [1:0] possel;
reg [2:0] oppossel;
reg on1;
reg on2;
wire [7:0]Y;
convolution DUT (data,arraysel,possel,oppossel,Y,on1,on2);
initial
    begin
    #1 arraysel=1'b0 ; possel=2'b00 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b0 ; possel=2'b01 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b0 ; possel=2'b10 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b0 ; possel=2'b11 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b1 ; possel=2'b00 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b1 ; possel=2'b01 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b1 ; possel=2'b10 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 arraysel=1'b1 ; possel=2'b11 ; data= 4'b0001 ;
            oppossel = 3'b111;on1=1'b1; on2=1'b1;
    #1 on2 = 1'b0;
    #1 on2 = 1'b1;
    #1 on1 = 1'b0;
    #1 on1 = 1'b1;
    #1 oppossel=3'b000;
    #1 oppossel=3'b001;
    #1 oppossel=3'b010;
    #1 oppossel=3'b011;
    #1 oppossel=3'b100;
    #1 oppossel=3'b101;
    #1 oppossel=3'b110;
    #1 oppossel=3'b111;
    end
endmodule
```

1.3 Conclusion: The program is tested for two arrays of length 4 given by A = 1 1 1 1 and B = 1 1 1 1 which gives the result as C = 1 2 3 4 3 2 1 which is correct and of length 7. This is a purely combinational circuit which makes use of two always block to process the inputs, perform the calculations and display the results in binary with help of LED's. The input is taken with help of switches where 4 are dedicated for the 4 bit data in of each element while 1 switch is for selecting the array and 2 switches for selecting the position of the element in each array. The last 3 switches are dedicated to select the output which is to be displayed.

1.4 Screenshots:



```
run
VSIM 16> run

VSIM 16>
```

Figure 1: Test bench simulation of convolution