CASEST

हैदराबाद विश्वविद्यालय
University of Hyderabad

Lab Assignment 3 a

Digital Signal Processing

Submitted by:

Full Name	**Enrollment No.**
Abhinav M	23PMMT11(@uohyd.ac.in)

Contents

DA dot product

Centre for Advanced Studies in Electronics Science and Technology

Objectives

Design a DA architecture for inner product of any length and bit-length, here I chose 3 and 3.

Logic

First I declare a memory to store all possible combinations of partial products, then according to the value of the input vector B, Then I will use a mux to select the pre-computed result and send them to the accumulator.

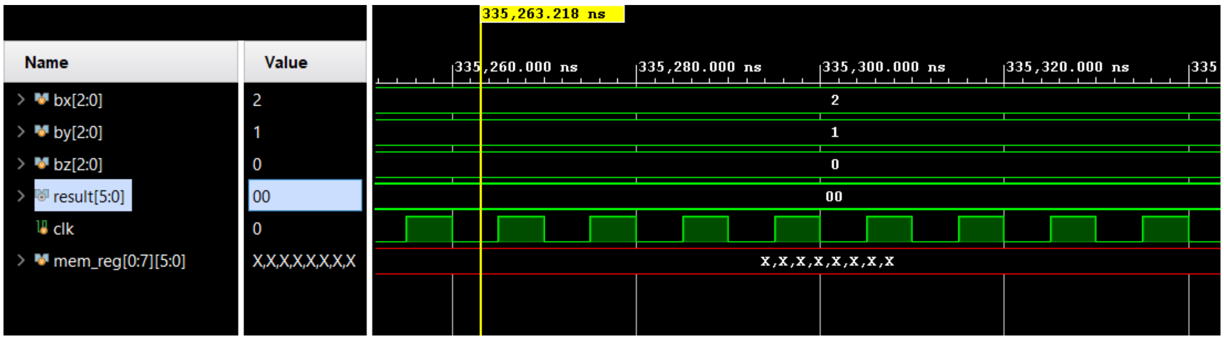
Code

```
module distributed_arithmetic (
    input [2:0] bx, by, bz, // Input elements of vector 2
    output reg [5:0] result // Output result
);
// Define memory for storing precomputed values
// Each entry represents the product of the corresponding elements of
vector1 and vector2
// memory size: 2^N
reg [5:0] mem_reg [0:7];

// Hardcode values for vector 1
parameter [2:0] ax = 3'b011; // 3
parameter [2:0] ay = 3'b100; // 4
parameter [2:0] az = 3'b101; // 5
// Precompute partial products and store them in memory
initial begin
    mem_reg[0] = ax * bx;
    mem_reg[1] = ax * by;
    mem_reg[2] = ax * bz;
    mem_reg[3] = ay * bx;
    mem_reg[4] = ay * by;
    mem_reg[5] = ay * bz;
    mem_reg[6] = az * bx;
    mem_reg[7] = az * by;
end
// Mux to select partial products based on 'b' coefficients
reg [5:0] selected_partial_product;
always @* begin
    case ({bx, by, bz})
        3'b000: accumulated_product = 6'd0; // No partial product selected
        3'b001: accumulated_product = mem_reg[1];
        3'b010: accumulated_product = mem_reg[2];
        3'b011: accumulated_product = mem_reg[1] + mem_reg[2];
        3'b100: accumulated_product = mem_reg[3];
        3'b101: accumulated_product = mem_reg[1] + mem_reg[3];
        3'b110: accumulated_product = mem_reg[2] + mem_reg[3];
        3'b111: accumulated_product = mem_reg[1] + mem_reg[2] + mem_reg[3];
        default: accumulated_product = 6'd0; // Default value, if needed
    endcase
end

// Accumulate selected partial products to compute dot product
always @* begin
    result = selected_partial_product[5:0]; // Initialize result with
selected partial product
end
endmodule
```

Test bench

```
module distributed_arithmetic_tb;

    localparam CLK_PERIOD = 10;

    reg [2:0] bx, by, bz; // Input elements of vector 2
    wire [5:0] result; // Output result

    distributed_arithmetic uut (
        .bx(bx),
        .by(by),
        .bz(bz),
        .result(result)
    );

    // Clock generation
    reg clk = 0;
    always #((CLK_PERIOD)/2) clk = ~clk;

    // Stimulus
    initial begin
        // Test case 1: a = {3, 4, 5}, b = {2, 1, 0}, expected result = 10
        bx = 2; by = 1; bz = 0;
        #100;
        $finish;
    end

    // Monitor
    always @(posedge clk) begin
        $display("Time = %0t, Result = %d", $time, result);
    end

endmodule
```

Waveform



```
Name              Value
bx[2:0]           2                 2
by[2:0]           1                 1
bz[2:0]           0                 0
result[5:0]       00                00
clk               0
mem_reg[0:7][5:0] X,X,X,X,X,X,X,X   X,X,X,X,X,X,X,X
```

The logic of my code seems to be ok, but for some reason I am getting XX values in the memory register, I am unable to figure out the issue even after few hours of debugging attempts.

I accidentally did the below report, it was not part of the question

CASEST

हैदराबाद विश्वविद्यालय
University of Hyderabad

DSP Lab Assignment

3- Tap FIR filter using Distributed Arithmetic

Submitted by:

Full Name	Enrolment No.
Abhinav M	23PMMT11(@uohyd.ac.in)

08-05-2024

Centre for Advanced Studies in Electronics Science and Technology

3-tap FIR using Distributed Arithmetic

In distributed arithmetic, instead of performing multiplications and additions directly with the input data and coefficients, the multiplications are replaced with table lookups followed by summation. In my code I used lookup tables to store the precomputed products and then fetched the values and added them to produce the output.

This approach saves computational resources compared to performing individual multiplications for each tap.

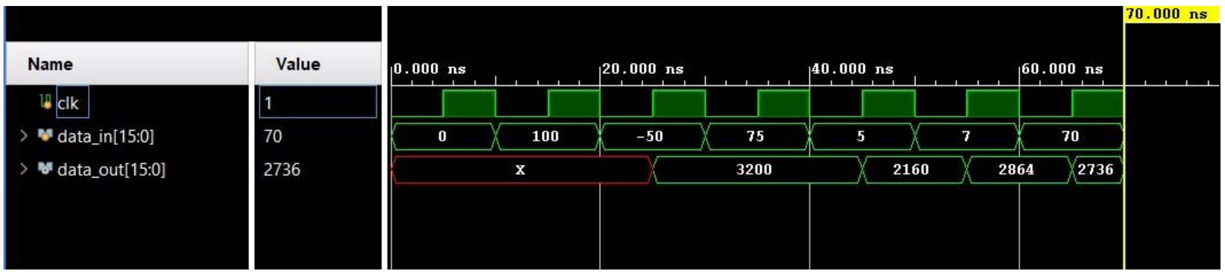
Code

```
module fir_filter_DA (
    input clk,
    input [15:0] data_in,
    output [15:0] data_out
);

// Coefficients for the FIR filter
parameter COEFF_0 = 8'h20;
parameter COEFF_1 = 8'h30;
parameter COEFF_2 = 8'h20;

// ROM for coefficient storage
reg signed [15:0] mem_reg [0:2];
initial begin
    mem_reg[0] = COEFF_0;
    mem_reg[1] = COEFF_1;
    mem_reg[2] = COEFF_2;
end

// Registers to hold input samples
reg [15:0] sample_mem [0:2];

// Register to hold output
reg [31:0] output_reg;

// Update memory with new input samples
always @(posedge clk) begin
    sample_mem[0] <= data_in;
    sample_mem[1] <= sample_mem[0];
    sample_mem[2] <= sample_mem[1];
end

// Distributed arithmetic implementation
always @* begin
    output_reg = (sample_mem[0] * mem_reg[0]) +
                 (sample_mem[1] * mem_reg[1]) +
                 (sample_mem[2] * mem_reg[2]);
end

assign data_out = output_reg[15:0]; // Output data is truncated to 16 bits

endmodule
```

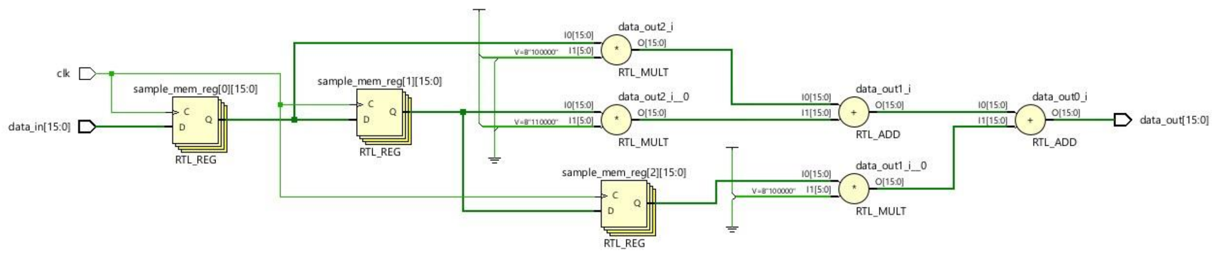
Testbench

```
module fir_filter_tb;

parameter CLK_PERIOD = 10;

// Inputs reg
clk = 0;
reg signed [15:0] data_in = 0;

// Outputs
wire signed [15:0] data_out;

// Instantiate fir_filter
uut (
    .clk(clk),
    .data_in(data_in),
    .data_out(data_out)
);

// Clock
always #(CLK_PERIOD/2) clk = ~clk;
  initial begin
// Initialize
data_in = 0;


    #10 data_in = 100;
    #10 data_in = -50;
    #10 data_in = 75;
    #10 data_in = 5;
    #10 data_in = 7;
    #10 data_in = 70;


    #10 $finish;
end   endmodule
```

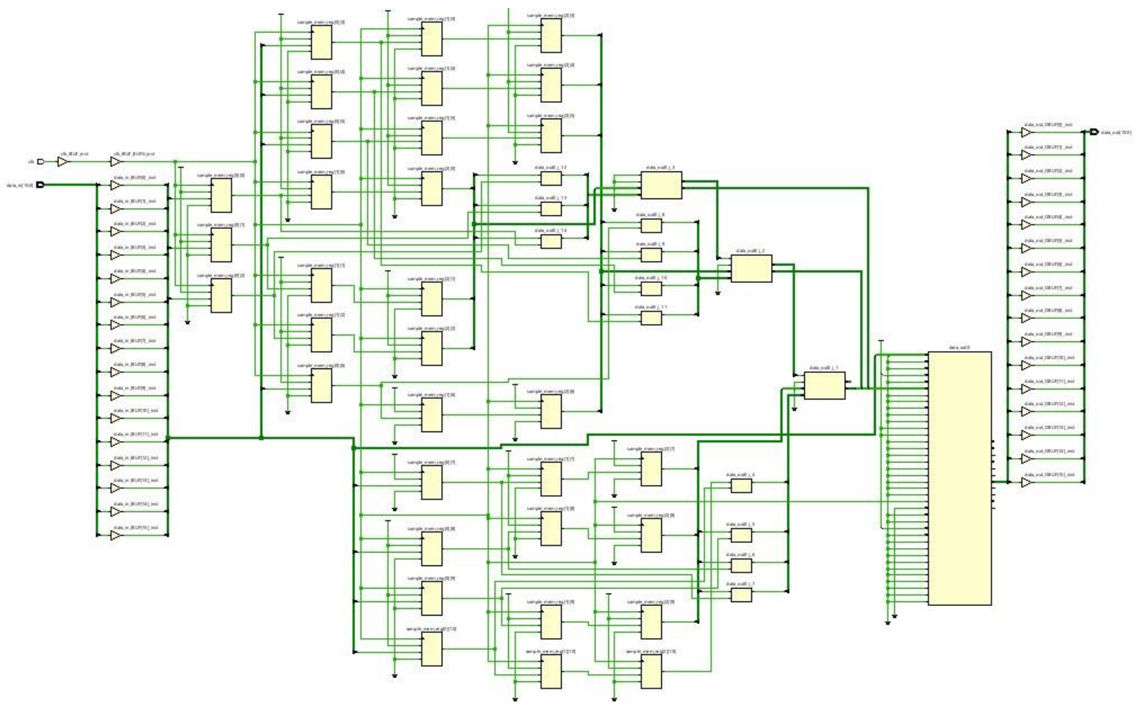
Simulation Result



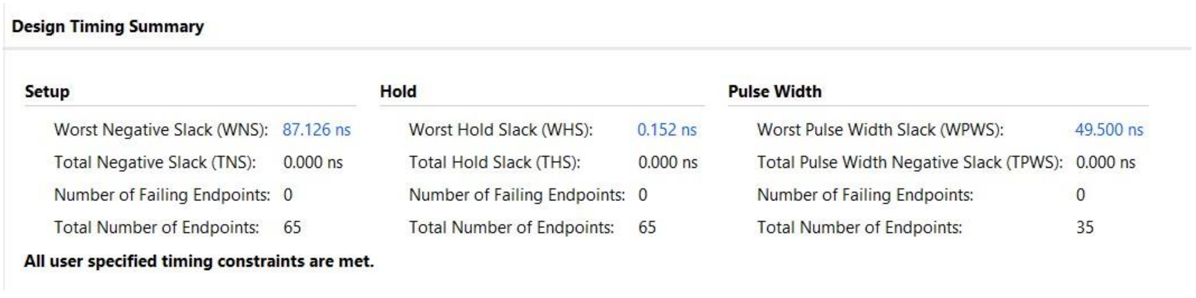
Elaborated Design



Synthesized Design



Timing Report (100ns clock)

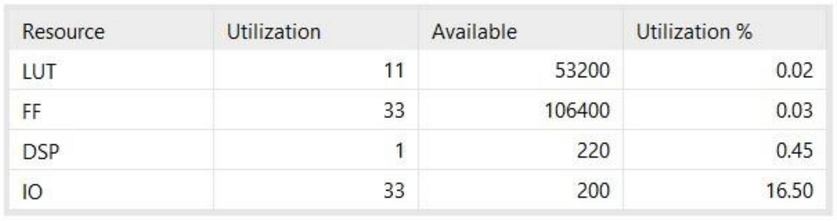


Design Timing Summary

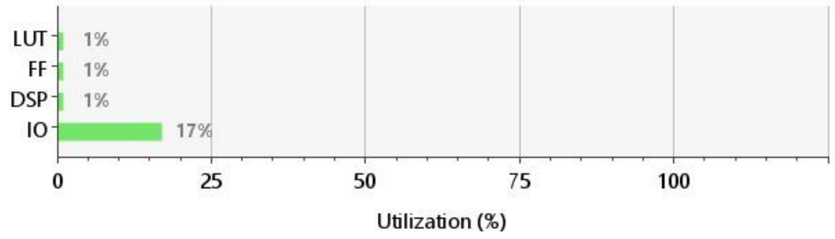
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	87.126 ns	Worst Hold Slack (WHS):	0.152 ns	Worst Pulse Width Slack (WPWS):	49.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	65	Total Number of Endpoints:	65	Total Number of Endpoints:	35

All user specified timing constraints are met.

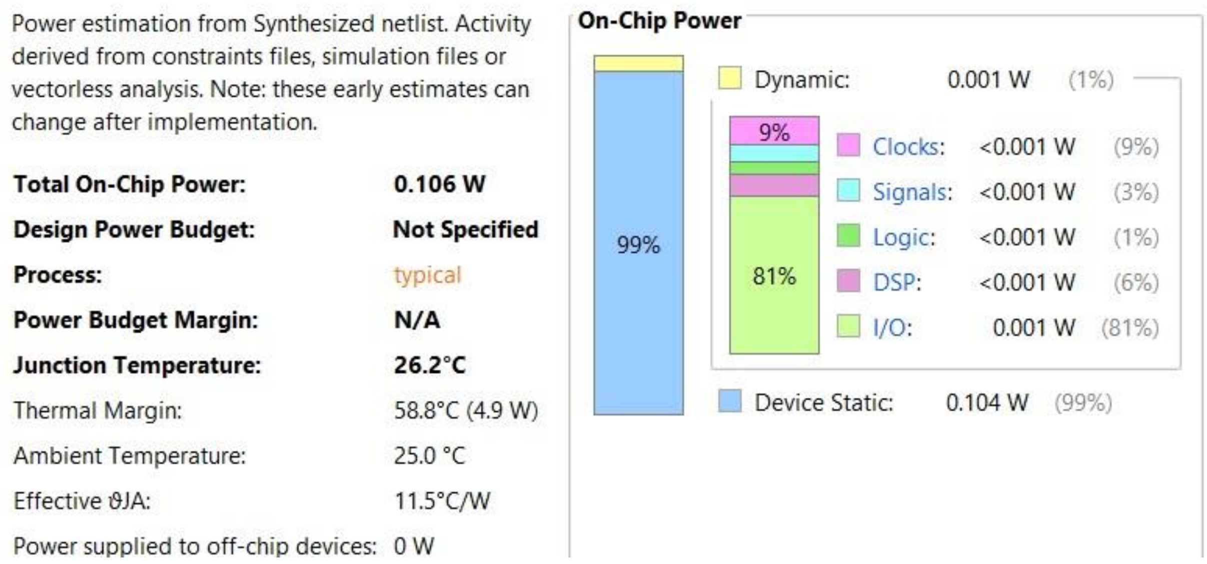
Utilization



Resource	Utilization	Available	Utilization %
LUT	11	53200	0.02
FF	33	106400	0.03
DSP	1	220	0.45
IO	33	200	16.50



Power Report



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.106 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**26.2°C**
Thermal Margin:	58.8°C (4.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W

On-Chip Power

🟨	Dynamic:	0.001 W	(1%)
🟪	Clocks:	<0.001 W	(9%)
🟦	Signals:	<0.001 W	(3%)
🟩	Logic:	<0.001 W	(1%)
🟪	DSP:	<0.001 W	(6%)
🟩	I/O:	0.001 W	(81%)
🟦	Device Static:	0.104 W	(99%)