Course Code: 19ECE343

Course Name: FPGA Based System Design

Component: Mini Project

Date of Evaluation:

Academic Year: 2024-2025 (Even Semester)

Batch Number:

Name of the Students With Registration no.:

1) Hanu Sharma BL.EN.U4ECE22218

2) J Sai Kesav BL.EN.U4ECE22219

3) Smrithi Banala BL.EN.U4ECE22251

Semester, Branch and Section: 6th Sem, ECE C

Batch: 2022-2026

Faculty In-charge: Dr. C Ramesh

Aim: Design of FFT Algorithm using 32-bit 1EEE 754 single precision standard floating point numbers.

Tool Used: Xilinx Vivado

FPGA Family Used (for Synthesis): Xilinx Zynq

Theory:

1) **Introduction**

Fast Fourier Transform (FFT) is a Digital Signal Processing (DSP) technique to compute Discrete Fourier Transform (DFT) in a faster way by utilizing the properties of the twiddle factor.

FFT can be done in two ways:

(i) Decimation In Time

(ii) Decimation In Frequency (DIF).

DIT FFT reduces the required number of complex multiplications from N^2 to (N*log2 N), whereas, DIF FFT reduces it from N^2 to (N/2*log2 N)

Also, here floating-point arithmetic is used for the computation of FFT. Floating-point numbers are represented by using a significand or mantissa multiplied by an exponent. The base of the exponent is generally 2, 10 or 16. Here 2 is selected as the base.

32-bit floating point numbers represented in IEEE 754 format are composed of three components:

Sign (1 Bit)	Exponent (8 Bits)	Mantissa (23 Bits)

 Sign: it indicates whether the number is positive or negative.

 Mantissa: it holds the main digits

 Exponent: it contains the value of the base power which defines where the decimal point should be placed. Here base value for the exponent is 2.

2) **Block Diagram and/or Logic Circuit**

2-Point FFT Butterfly Diagram

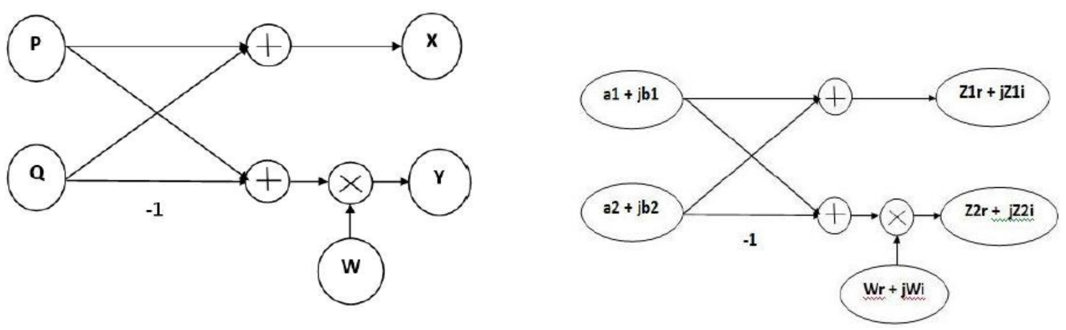


Fig.1 Radix-2 DIF FFT butterfly
Diagram with real and imaginary inputs

$Z1r + jZ1i = (a1 + jb1) + (a2 + jb2) = (a1 + a2) + j(b1 + b2)$
Thus, $Z1r = (a1 + a2)$ and $Z1i = (b1 + b2)$;

$Z2r + jZ2i = [(a1 + jb1) – (a2 + jb2)] * (Wr + jWi)$
$= [a1*Wr – a2*Wr – b1*Wi + b2*Wi] + j[a1*Wi – a2*Wi – b1*Wr + b2*Wr]$

32-Bit Floating Point Adder

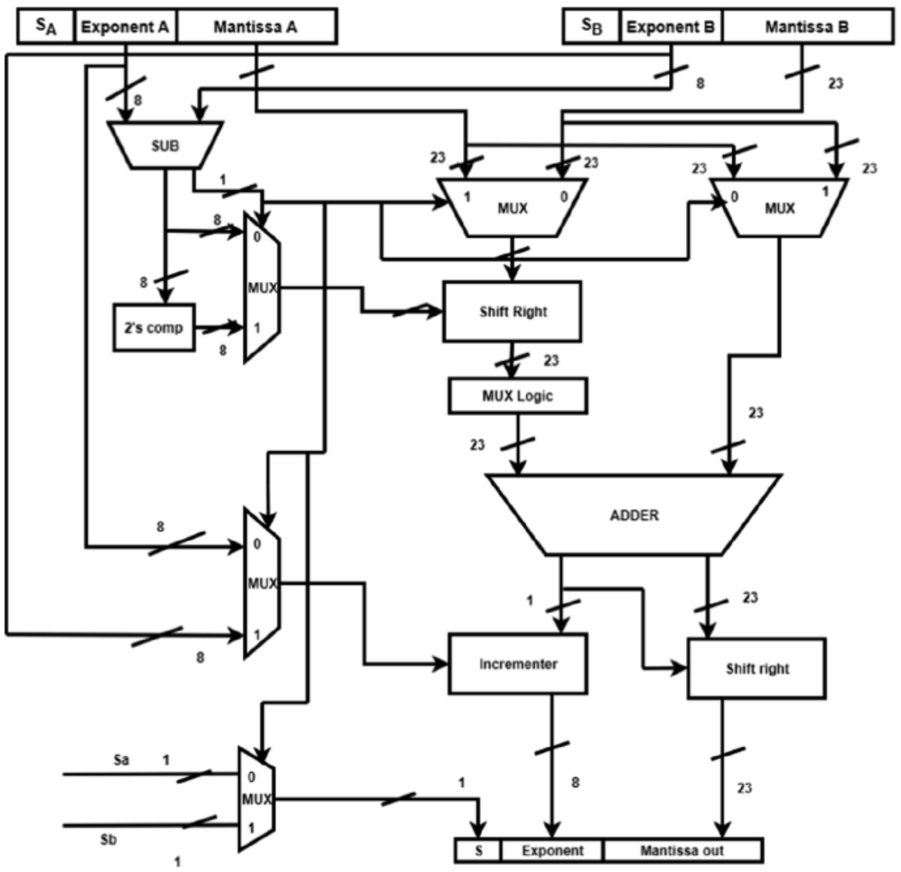


Fig.2 32-bit floating point adder

32-Bit Floating Point Multiplier

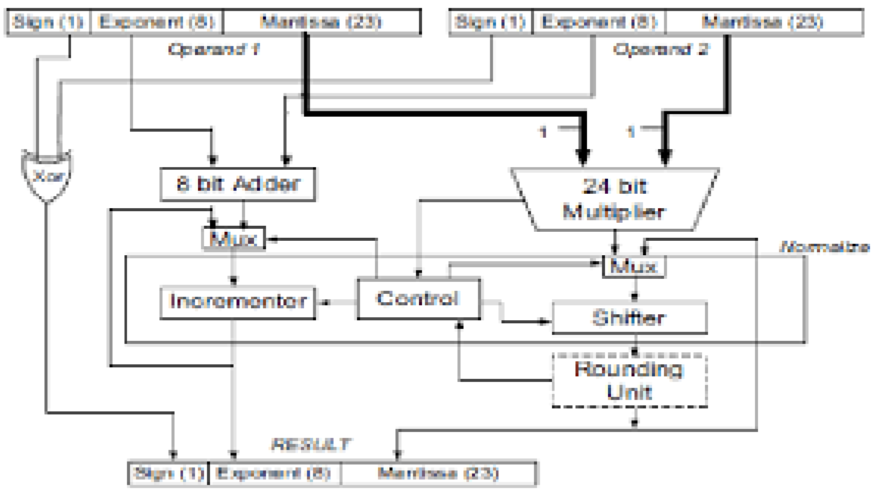
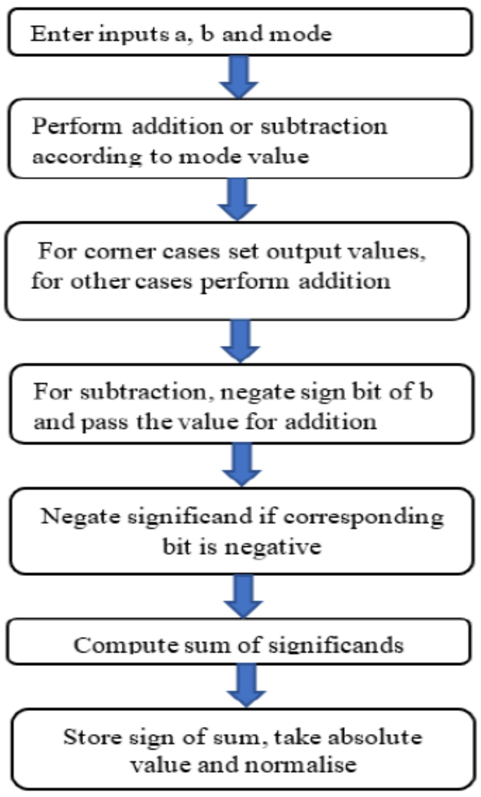


Fig.3 32-bit floating point multiplier

3) Working principle

The working principle of the FFT algorithm includes designing an adder block for the floating-point numbers, then a multiplier, using those to design 2-point FFT Blocks and later a 4-point FFT.

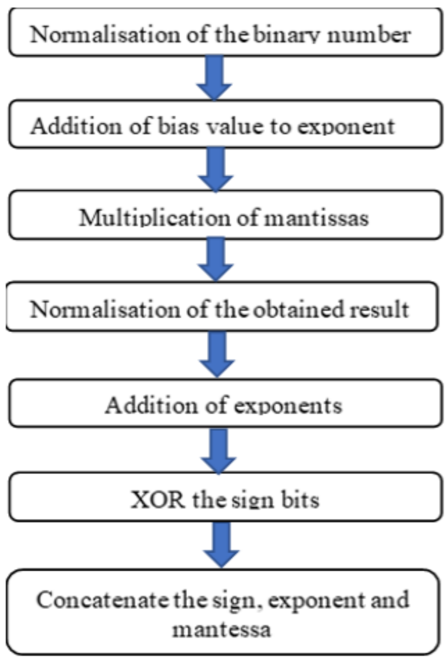
Adder Working Principle



Enter inputs a, b and mode
Perform addition or subtraction according to mode value
For corner cases set output values, for other cases perform addition
For subtraction, negate sign bit of b and pass the value for addition
Negate significand if corresponding bit is negative
Compute sum of significands
Store sign of sum, take absolute value and normalise

Fig. 4. Flow chart of floating-point adder/ subtractor

Multiplier Working Principle



Normalisation of the binary number
Addition of bias value to exponent
Multiplication of mantissas
Normalisation of the obtained result
Addition of exponents
XOR the sign bits
Concatenate the sign, exponent and mantessa

Fig. 5. Flow chart of floating-point multiplier

4 Point FFT Working Principle using 2-point FFT

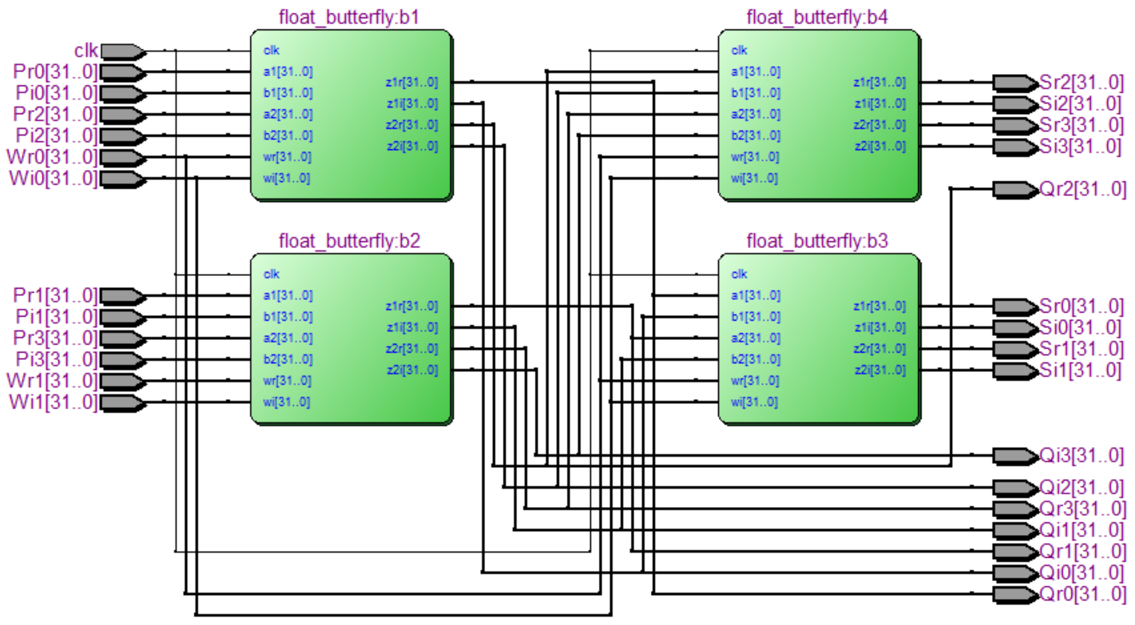


Fig.4 4-point DIF FFT using radix-2

4) Advantages/ Disadvantages

Advantages	Disadvantages
Significantly reduces computational costs compared to the DFT, making it suitable for real-time analysis.	FFT provides detailed frequency information but sacrifices time resolution, as it analyzes the entire signal duration rather than specific time segments.
Require less memory storage compared to direct DFT calculations.	FFT provides frequency information at discrete points, and the resolution is determined by the number of samples, potentially limiting the ability to accurately analyze narrow-band signals.

Verilog Code:

```
`timescale 1ns / 1ps


module fp_multiplier (
    input  [31:0] a,
    input  [31:0] b,
    output        exception,
    output        overflow,
    output        underflow,
    output [31:0] result
);
    wire temp;
    wire sign;
    wire [7:0] exp_a, exp_b;
    wire [23:0] mant_a, mant_b;
    wire [47:0] mant_mult;
    wire [7:0] exp_sum, exp_final;
    wire [22:0] mantissa_out;
    wire [47:0] mant_norm;
    wire guard, round, sticky;
    wire [24:0] rounded_mantissa;
    wire exp_carry;
    assign temp = (a == 0 | b == 0);
    assign sign = a[31] ^ b[31];
    assign exp_a = a[30:23];
    assign exp_b = b[30:23];
    assign mant_a = (exp_a == 8'd0) ? {1'b0, a[22:0]} : {1'b1, a[22:0]};
    assign mant_b = (exp_b == 8'd0) ? {1'b0, b[22:0]} : {1'b1, b[22:0]};
    assign mant_mult = mant_a * mant_b;
    assign exp_sum = exp_a + exp_b - 8'd127;
    assign mant_norm = mant_mult[47] ? mant_mult : mant_mult << 1;
    wire [7:0] exp_adjust = mant_mult[47] ? 8'd1 : 8'd0;
    assign exp_final = exp_sum + exp_adjust;
    assign guard  = mant_norm[23];
    assign round  = mant_norm[22];
    assign sticky = |mant_norm[21:0];
    wire round_bit = guard & (round | sticky | mant_norm[24]);
    assign rounded_mantissa = {1'b0, mant_norm[46:24]} + round_bit;
    assign mantissa_out = rounded_mantissa[24] ? rounded_mantissa[23:1] :
rounded_mantissa[22:0];
    assign exp_carry = rounded_mantissa[24];
    wire [7:0] final_exponent = exp_final + exp_carry;
    assign exception = (&exp_a) | (&exp_b) | temp ;
    assign overflow  = (final_exponent >= 8'hFF);
    assign underflow = (final_exponent <= 8'd0);
    assign result = exception ? 32'h00000000 :
                overflow  ? {sign, 8'hFF, 23'd0} :
                underflow ? {sign, 31'd0} :
                {sign, final_exponent, mantissa_out};
endmodule
module priority_encoder(
                        input [24:0] significand,
                        input [7:0] Exponent_a,
                        output reg [24:0] Significand,
                        output [7:0] Exponent_sub
                        );

reg [4:0] shift;
always @(significand)
begin
casex (significand)
25'b1_1xxx_xxxx_xxxx_xxxx_xxxx_xxxx : begin
Significand = significand;
shift = 5'd0;
end
25'b1_01xx_xxxx_xxxx_xxxx_xxxx_xxxx : begin
Significand = significand << 1;
shift = 5'd1;
end
25'b1_001x_xxxx_xxxx_xxxx_xxxx_xxxx : begin
Significand = significand << 2;
shift = 5'd2;
end
25'b1_0001_xxxx_xxxx_xxxx_xxxx_xxxx : begin
Significand = significand << 3;
shift = 5'd3;
end
25'b1_0000_1xxx_xxxx_xxxx_xxxx_xxxx : begin
Significand = significand << 4;
shift = 5'd4;
end
```

```
25'b1_0000_01xx_xxxx_xxxx_xxxx_xxxx : begin
Significand = significand << 5;
shift = 5'd5;
end
25'b1_0000_001x_xxxx_xxxx_xxxx_xxxx : begin// 24'h020000
Significand = significand << 6;
shift = 5'd6;
end
25'b1_0000_0001_xxxx_xxxx_xxxx_xxxx : begin//24'h010000
Significand = significand << 7;
shift = 5'd7;
end
25'b1_0000_0000_1xxx_xxxx_xxxx_xxxx : begin// 24'h008000
Significand = significand << 8;
shift = 5'd8;
end
25'b1_0000_0000_01xx_xxxx_xxxx_xxxx : begin// 24'h004000
Significand = significand << 9;
shift = 5'd9;
end
25'b1_0000_0000_001x_xxxx_xxxx_xxxx : begin   // 24'h002000
Significand = significand << 10;
shift = 5'd10;
end
25'b1_0000_0000_0001_xxxx_xxxx_xxxx : begin// 24'h001000
Significand = significand << 11;
shift = 5'd11;
end
25'b1_0000_0000_0000_1xxx_xxxx_xxxx : begin// 24'h000800
Significand = significand << 12;
shift = 5'd12;
```

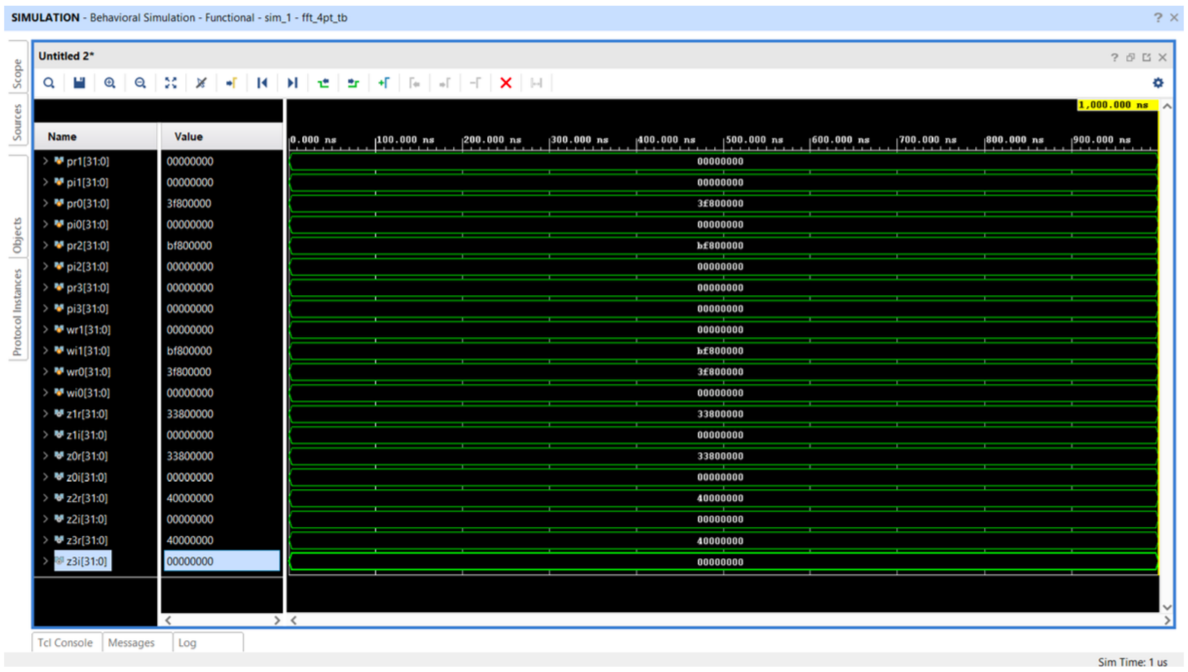
```
end
25'b1_0000_0000_0000_01xx_xxxx_xxxx : begin// 24'h000400
Significand = significand << 13;
shift = 5'd13;
end
25'b1_0000_0000_0000_001x_xxxx_xxxx : begin// 24'h000200
Significand = significand << 14;
shift = 5'd14;
end
25'b1_0000_0000_0000_0001_xxxx_xxxx  :            begin
Significand = significand << 15; shift = 5'd15;
end
25'b1_0000_0000_0000_0000_1xxx_xxxx : begin// 24'h000080
Significand = significand << 16;
shift = 5'd16;
end
25'b1_0000_0000_0000_0000_01xx_xxxx : begin// 24'h000040
Significand = significand << 17;
shift = 5'd17;
end
25'b1_0000_0000_0000_0000_001x_xxxx : begin// 24'h000020
Significand = significand << 18;
shift = 5'd18;
end
25'b1_0000_0000_0000_0000_0001_xxxx : begin// 24'h000010
Significand = significand << 19;
shift = 5'd19;
end
25'b1_0000_0000_0000_0000_0000_1xxx : begin// 24'h000008
Significand = significand << 20;
shift = 5'd20;
```

```
end
25'b1_0000_0000_0000_0000_0000_01xx : begin// 24'h000004
Significand = significand << 21;
shift = 5'd21;
end
25'b1_0000_0000_0000_0000_0000_001x : begin// 24'h000002
Significand = significand << 22;
shift = 5'd22;
end
25'b1_0000_0000_0000_0000_0000_0001 : begin// 24'h000001
Significand = significand << 23;
shift = 5'd23;
end
25'b1_0000_0000_0000_0000_0000_0000 : begin// 24'h000000
Significand = significand << 24;
shift = 5'd24;
end
default :           begin
Significand = (~significand) + 1'b1;
shift = 8'd0;
end
endcase
end
assign Exponent_sub = Exponent_a - shift;
endmodule

module Addition_Subtraction(
input [31:0] a_operand,b_operand,
input AddBar_Sub,
output Exception,
output [31:0] result
);
wire operation_sub_addBar;
wire Comp_enable;
wire output_sign;
wire [31:0] operand_a,operand_b;
wire [23:0] significand_a,significand_b;
wire [7:0] exponent_diff;
wire [23:0] significand_b_add_sub;
wire [7:0] exponent_b_add_sub;
wire [24:0] significand_add;
wire [30:0] add_sum;
wire [23:0] significand_sub_complement;
wire [24:0] significand_sub;
wire [30:0] sub_diff;
wire [24:0] subtraction_diff;
wire [7:0] exponent_sub;
assign {Comp_enable,operand_a,operand_b} = (a_operand[30:0] < b_operand[30:0]) ?
{1'b1,b_operand,a_operand} : {1'b0,a_operand,b_operand};
assign exp_a = operand_a[30:23];
assign exp_b = operand_b[30:23];
assign Exception = (&operand_a[30:23]) | (&operand_b[30:23]);
assign output_sign = AddBar_Sub ? Comp_enable ? !operand_a[31] : operand_a[31] :
operand_a[31] ;
assign operation_sub_addBar = AddBar_Sub ? operand_a[31] ^ operand_b[31] :
~(operand_a[31] ^ operand_b[31]);
assign significand_a = (|operand_a[30:23]) ? {1'b1,operand_a[22:0]} :
{1'b0,operand_a[22:0]};
assign significand_b = (|operand_b[30:23]) ? {1'b1,operand_b[22:0]} :
{1'b0,operand_b[22:0]};
assign exponent_diff = operand_a[30:23] - operand_b[30:23];
assign significand_b_add_sub = significand_b >> exponent_diff;
assign exponent_b_add_sub = operand_b[30:23] + exponent_diff;
assign perform = (operand_a[30:23] == exponent_b_add_sub);
assign significand_add = (perform & operation_sub_addBar) ? (significand_a +
significand_b_add_sub) : 25'd0;

assign add_sum[22:0] = significand_add[24] ? significand_add[23:1] : significand_add[22:0];

assign add_sum[30:23] = significand_add[24] ? (1'b1 + operand_a[30:23]) :
operand_a[30:23];

assign significand_sub_complement = (perform & !operation_sub_addBar) ?
~(significand_b_add_sub) + 24'd1 : 24'd0 ;

assign significand_sub = perform ? (significand_a + significand_sub_complement) : 25'd0;

priority_encoder pe(significand_sub,operand_a[30:23],subtraction_diff,exponent_sub);

assign sub_diff[30:23] = exponent_sub;

assign sub_diff[22:0] = subtraction_diff[22:0];

assign result = Exception ? 32'b0 : ((!operation_sub_addBar) ? {output_sign,sub_diff} :
{output_sign,add_sum});

endmodule

module fft_2pt(

input [31:0]a1, b1, a0, b0, wr, wi,

output [31:0]z1r, z1i, z0r, z0i

   );

wire [31:0]xr, xi;

wire exc1,exc2,exc3,exc4,exc5,exc6;

Addition_Subtraction fa1(a0,a1,1'b0,exc1,z0r);

Addition_Subtraction fa2(b0,b1,1'b0,exc2,z0i);

Addition_Subtraction fs1(a0,a1,1'b1,exc3,xr);

Addition_Subtraction fs2(b0,b1,1'b1,exc4,xi);

wire [4:1]e, u, o;

wire [31:0] y [4:1];

fp_multiplier m1(xr, wr, e[1], u[1], o[1], y[1]);

fp_multiplier m2(xr, wi, e[2], u[2], o[2], y[2]);

fp_multiplier m3(xi, wr, e[3], u[3], o[3], y[3]);

fp_multiplier m4(xi, wi, e[4], u[4], o[4], y[4]);

Addition_Subtraction fs3(y[1],y[4],1,exc5,z1r);

Addition_Subtraction fs4(y[2],y[3],0,exc6,z1i);

endmodule
```

```
module fft_4pt(

input [31:0]pr1, pi1, pr0, pi0,pr2, pi2, pr3, pi3, wr1, wi1, wr0, wi0,

output [31:0]z1r, z1i, z0r, z0i, z2r, z2i, z3r, z3i);

wire [31:0] q0r, q0i, q1r, q1i, q2r, q2i, q3r, q3i;

fft_2pt f2_1(pr2, pi2, pr0, pi0, wr0, wi0, q2r, q2i, q0r, q0i);

fft_2pt f2_2(pr3, pi3, pr1, pi1, wr1, wi1, q3r, q3i, q1r, q1i);

fft_2pt f2_3(q1r, q1i, q0r, q0i, wr0, wi0, z0r, z0i, z1r, z1i);

fft_2pt f2_4(q3r, q3i, q2r, q2i, wr0, wi0, z2r, z2i, z3r, z3i);

endmodule

module fft_4pt_tb;

reg [31:0]pr1, pi1, pr0, pi0,pr2, pi2, pr3, pi3, wr1, wi1, wr0, wi0;

wire [31:0]z1r, z1i, z0r, z0i, z2r, z2i, z3r, z3i;

fft_4pt f41(pr1, pi1, pr0, pi0, pr2, pi2, pr3, pi3, wr1, wi1, wr0, wi0, z1r, z1i, z0r, z0i, z2r, z2i,
z3r, z3i);

initial

begin

pr0 = 32'b00111111100000000000000000000000;

pi0 = 32'b0;

pr1 = 32'b0;

pi1 = 32'b0;

pr2 = 32'b10111111100000000000000000000000;

pi2 = 32'b0;

pr3 = 32'b0;

pi3 = 32'b0;

wr0 = 32'b00111111100000000000000000000000;

wi0 = 32'b0;

wr1 = 32'b0;

wi1 = 32'b10111111100000000000000000000000;

end

endmodule

module fft_2pt_tb;

reg [31:0]pr1, pi1, pr0, pi0,wr0,wi0;
```

```
wire [31:0]z1r, z1i, z0r, z0i,something;
wire exc;
fft_2pt f2_tb(pr1, pi1, pr0, pi0, wr0, wi0, z1r, z1i, z0r, z0i);
Addition_Subtraction a1(pr0,pr1,1'b1,exc,something);
initial
begin
pr0 = 32'b00111111100000000000000000000000;
pi0 = 32'b0;
pr1 = 32'b00111111100000000000000000000000;
pi1 = 32'b00111111100000000000000000000000;
wr0 = 32'b00111111100000000000000000000000;
wi0 = 32'b0;
end
endmodule
```

Simulation Result:


SIMULATION - Behavioral Simulation - Functional - sim_1 - fft_4pt_tb

Untitled 2*

Name	Value
pr1[31:0]	00000000
pi1[31:0]	00000000
pr0[31:0]	3f800000
pi0[31:0]	00000000
pr2[31:0]	bf800000
pi2[31:0]	00000000
pr3[31:0]	00000000
pi3[31:0]	00000000
wr1[31:0]	00000000
wi1[31:0]	bf800000
wr0[31:0]	3f800000
wi0[31:0]	00000000
z1r[31:0]	33800000
z1i[31:0]	00000000
z0r[31:0]	33800000
z0i[31:0]	00000000
z2r[31:0]	40000000
z2i[31:0]	00000000
z3r[31:0]	40000000
z3i[31:0]	00000000

Fig.5 Behavioral simulation

Synthesis Report:

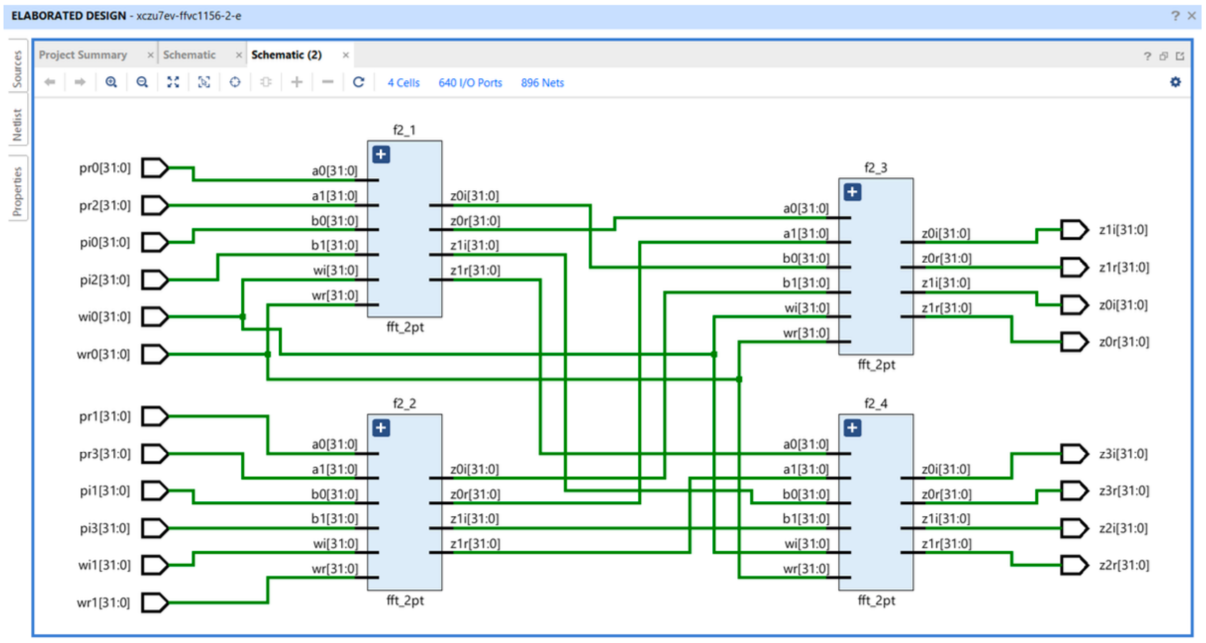
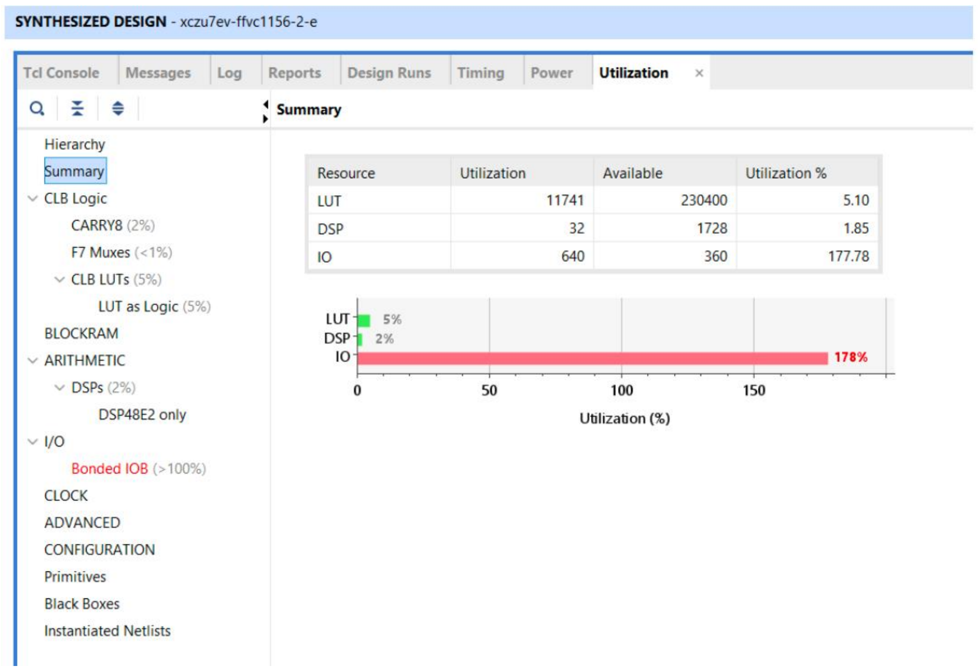


Fig.6 Schematic

i) **Area Report**

Most of the area consumed is by the IO pins. The design includes 32bit complex numbers for calculation, and hence has a lot of IO requirements.

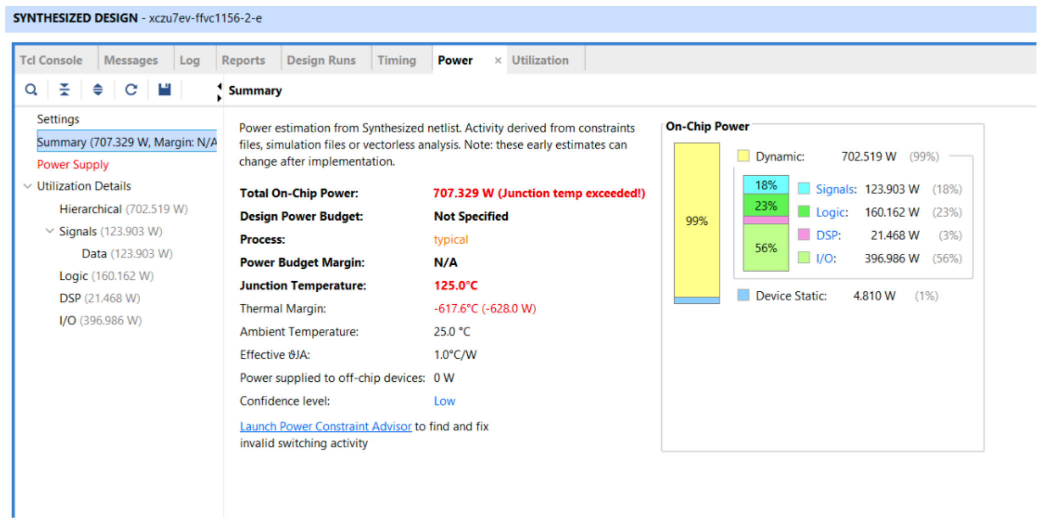


Resource	Utilization	Available	Utilization %
LUT	11741	230400	5.10
DSP	32	1728	1.85
IO	640	360	177.78

Fig.7 Area report

ii) Power Report

As the number of IO pins exceed those that are available on the FPGA Board, the power consumed might go beyond what the capabilities of the board are and might increase the temperatures well above threshold as well.



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:	**707.329 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-617.6°C (-628.0 W)
Ambient Temperature:	25.0 °C
Effective θJA:	1.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

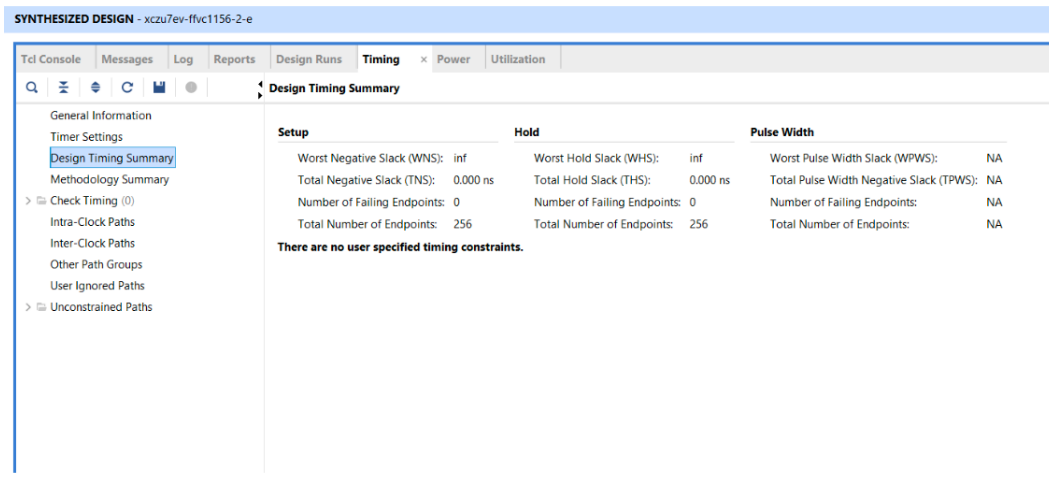
Launch Power Constraint Advisor to find and fix invalid switching activity

Dynamic:		702.519 W	(99%)
	Signals:	123.903 W	(18%)
	Logic:	160.162 W	(23%)
	DSP:	21.468 W	(3%)
	I/O:	396.986 W	(56%)
Device Static:		4.810 W	(1%)

Fig.8 Power report

iii) Timing Report

Most of the Timing Details and Clock Table values are zero. The circuit is strictly combinational in nature and has no timing and delays mentioned.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	256	Total Number of Endpoints:	256	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Fig.9 Timing report

Conclusion:

A 4- DIF FFT was implemented using floating-point adders, subtractors and multipliers in Verilog HDL.

Reference:

Joseph, C., & Prakash, S. S. (2022, February). Design of Efficient Pruning Architecture for FFT Algorithm. In *2022 First International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)* (pp. 1-6). IEEE.