Question Three - HW4 FPGA

Parsa Haghighatgoo
40030644

February 11, 2026

3(a) IEEE 754 Single-Precision Representation of 8.75

Step 1: Convert 8.75 **to binary**

$$\begin{aligned} 8_{10} &= 1000_2 \\ 0.75_{10}: &\quad 0.75 \times 2 = 1.5 \Rightarrow 1,\ \ 0.5 \times 2 = 1.0 \Rightarrow 1 \\ 0.75_{10} &= 0.11_2 \end{aligned}$$

Thus,

$$8.75_{10} = 1000.11_2$$

Step 2: Normalize

$$1000.11_2 = 1.00011_2 \times 2^3$$

So the sign bit is $s = 0$ (positive) and the unbiased exponent is $E = 3$.
Step 3: Compute biased exponent
IEEE-754 single precision uses a bias of 127:

$$e = E + 127 = 3 + 127 = 130$$

Convert 130 to 8-bit binary:

$$130_{10} = 10000010_2$$

Step 4: Determine the fraction (mantissa)
The fraction field is the bits after the leading 1. in the normalized form:

$$1.\underline{00011}_2 \Rightarrow f = 00011000000000000000000$$

(padded with zeros to 23 bits).
Final IEEE-754 Single-Precision Format

$$\underbrace{0}_{\text{sign}}\ \underbrace{10000010}_{\text{exponent}}\ \underbrace{00011000000000000000000}_{\text{fraction}}$$

Therefore, the 32-bit IEEE-754 representation is:

$$0\ 10000010\ 00011000000000000000000$$

Hexadecimal form

$$0100\ 0001\ 0000\ 1100\ 0000\ 0000\ 0000\ 0000 = \texttt{0x410C0000}$$

3.2(b) IEEE-754 Single-Precision Representation of 5.25×10^{-4} (All Steps)

Let

$$x = 5.25 \times 10^{-4} = 0.000525.$$

Step 1: Sign bit
Since $x > 0$, the sign bit is

$$s = 0.$$

Step 2: Normalize to $(1.f) \times 2^e$
We choose an integer exponent e such that

$$1 \le x \cdot 2^{-e} < 2.$$

Compute

$$2^{-11} = \frac{1}{2048} = 0.00048828125, \qquad 2^{-10} = \frac{1}{1024} = 0.0009765625.$$

Since

$$0.00048828125 < 0.000525 < 0.0009765625,$$

the normalized exponent is

$$e = -11.$$

Then the normalized significand is

$$m = x \cdot 2^{11} = 0.000525 \times 2048 = 1.0752,$$

so

$$x = 1.0752 \times 2^{-11}.$$

Step 3: Biased exponent field (8 bits)
IEEE-754 single precision uses bias 127:

$$E = e + 127 = -11 + 127 = 116.$$

Convert 116 to 8-bit binary:

$$116_{10} = 01110100_2.$$

Step 4: Mantissa field (23 bits)
The fraction is

$$f = m - 1 = 1.0752 - 1 = 0.0752.$$

Method A (scaling and rounding):

$$M = \text{round}\left(f \cdot 2^{23}\right), \qquad 2^{23} = 8,388,608.$$

$$f \cdot 2^{23} = 0.0752 \times 8,388,608 = 630,823.3216 \Rightarrow M = 630,823.$$

In 23-bit binary:

$$630823_{10} = 00010011010000000100111_2.$$

Method B (repeated multiply-by-2 to generate bits):
Starting from $f_0 = 0.0752$, each bit is obtained from $2f_{k-1}$:

$$b_k = \lfloor 2f_{k-1} \rfloor, \qquad f_k = 2f_{k-1} - b_k.$$

The first 23 bits obtained are:

$$M = 00010011010000000100111.$$

Step 5: Final IEEE-754 single word

$$\underbrace{0}_{\text{sign}}\ \underbrace{01110100}_{\text{exponent}}\ \underbrace{00010011010000000100111}_{\text{mantissa}}.$$

Thus the 32-bit pattern is

$$0\ 01110100\ 00010011010000000100111.$$

Hex form
Grouping into 4-bit nibbles:

$$0011\ 1010\ 0000\ 1001\ 1010\ 0000\ 0010\ 0111 = \texttt{0x3A09A027}.$$

Verification
This corresponds to

$$x_{\text{float}} = \left(1 + \frac{M}{2^{23}}\right) 2^{-11} \approx (1.0751999617)\, 2^{-11} \approx 0.0005249999813,$$

which is extremely close to 0.000525; the small difference is due to IEEE-754 rounding.

3(c) Sum Calculation and Q2.5 Explanation

We add the two values from parts (a) and (b):

$$x_1 = 8.75, \qquad x_2 = 5.25 \times 10^{-4} = 0.000525$$

Hence the exact real-valued sum is:

$$x_{\text{sum}} = x_1 + x_2 = 8.75 + 0.000525 = 8.750525.$$

Representing x_2 using Q2.5

In Q2.5, the LSB weight is:

$$\text{LSB} = 2^{-5} = \frac{1}{32} = 0.03125.$$

Quantization is done by scaling and rounding:

$$x_2 \cdot 2^5 = 0.000525 \times 32 = 0.0168.$$

Since $0.0168 < 0.5$, rounding to nearest gives integer code 0, therefore:

$$x_{2,\text{Q2.5}} = 0 \cdot 2^{-5} = 0.$$

Thus, in a Q2.5 fixed-point system, x_2 is too small to be represented and would not change the sum.

IEEE-754 Single-Precision Addition (FPGA-style steps)

Step 1: Normalize both numbers

$$8.75 = 1000.11_2 = 1.00011_2 \times 2^3$$

$$0.000525 \approx 1.00010011010000000100111_2 \times 2^{-11}.$$

Step 2: Align exponents
The exponent difference is:

$$\Delta e = 3 - (-11) = 14.$$

So the second significand is shifted right by 14 bits:

$$1.00010011010000000100111_2 \times 2^{-11} = 0.00000000000001\,00010011010000000100111_2 \times 2^3.$$

Step 3: Integer significand addition (24-bit, hidden 1 included)
Using the single-precision fields from parts (a) and (b):

$$S_1 = (1 \ll 23) + 0x0C0000 = 9175040, \qquad S_2 = (1 \ll 23) + 0x09A027 = 9019431.$$

Aligned second significand:

$$S_{2,\text{align}} = \left\lfloor \frac{S_2}{2^{14}} \right\rfloor = 550, \qquad r = S_2 \bmod 2^{14} = 8231.$$

Add:

$$S_{\text{sum}} = S_1 + S_{2,\text{align}} = 9175040 + 550 = 9175590.$$

Step 4: Rounding
Half-ULP at this shift is $2^{13} = 8192$. Since $r = 8231 > 8192$, round up:

$$S_{\text{sum,rounded}} = 9175590 + 1 = 9175591.$$

Step 5: Pack result into IEEE-754
The sum remains normalized with exponent $e = 3$, hence biased exponent:

$$E = 3 + 127 = 130 = 10000010_2.$$

The fraction field is:

$$\text{frac} = S_{\text{sum,rounded}} - (1 \ll 23) = 9175591 - 8388608 = 786983 = 0x0C0227.$$

So the final IEEE-754 single-precision word is:

$$0\ 10000010\ 00011000000001000100111$$

and in hexadecimal:

$$\boxed{\texttt{0x410C0227}}.$$

This corresponds to approximately:

$$x_{\text{float}} \approx 8.750525.$$

3(d) Q2.5 Fixed-Point Adder and Absolute Error Implementation

Objective

The goal of this part is to implement, in hardware (Verilog), a Q2.5 fixed-point adder that:

1. Quantizes two input numbers to Q2.5 format,
2. Computes their Q2.5 sum,
3. Computes the true sum in higher precision,
4. Calculates the absolute error in Q2.5 format.

This implementation demonstrates the effect of coarse fixed-point quantization on arithmetic accuracy.

Q2.5 Fixed-Point Format

In Q2.5 format:

$$\text{LSB} = 2^{-5} = \frac{1}{32} = 0.03125$$

Any representable number must be an integer multiple of 0.03125.
The conversion from Q16.16 to Q2.5 is performed by:

$$x_{Q2.5} = \text{round}\left(\frac{x_{Q16.16}}{2^{(16-5)}}\right)$$

The absolute error is computed as:

$$\text{Error}_{Q2.5} = |\text{TrueSum}_{Q2.5} - \text{Sum}_{Q2.5}|$$

Hardware Design Description

The Verilog module performs the following operations:

1. Convert each input from Q16.16 to Q2.5 using rounding.
2. Add the quantized values.
3. Compute the true sum in Q16.16.
4. Quantize the true sum to Q2.5.
5. Compute the absolute difference.

The design is purely combinational and uses arithmetic shifts for scaling.

Verilog Implementation

The main module and testbench were implemented in pure Verilog (2001 standard) and simulated using Xilinx Vivado.

```
`timescale 1ns/1ps

// Q2.5 adder + absolute error in Q2.5
// Inputs:  signed Q16.16 (32-bit)
// Outputs: signed Q2.5  (16-bit scaled integers)
//
// Steps:
// 1) Quantize a and b to Q2.5 (round-to-nearest, ties away from zero)
// 2) Add them in Q2.5 -> sum_q2_5
// 3) Compute true sum in Q16.16, then quantize to Q2.5 -> true_sum_q2_5
// 4) abs_err_q2_5 = |true_sum_q2_5 - sum_q2_5|
module q2_5_add_abs_err
#(
    parameter IN_FRAC  = 16, // Q16.16 fractional bits
    parameter OUT_FRAC = 5   // Q2.5 fractional bits
)
```

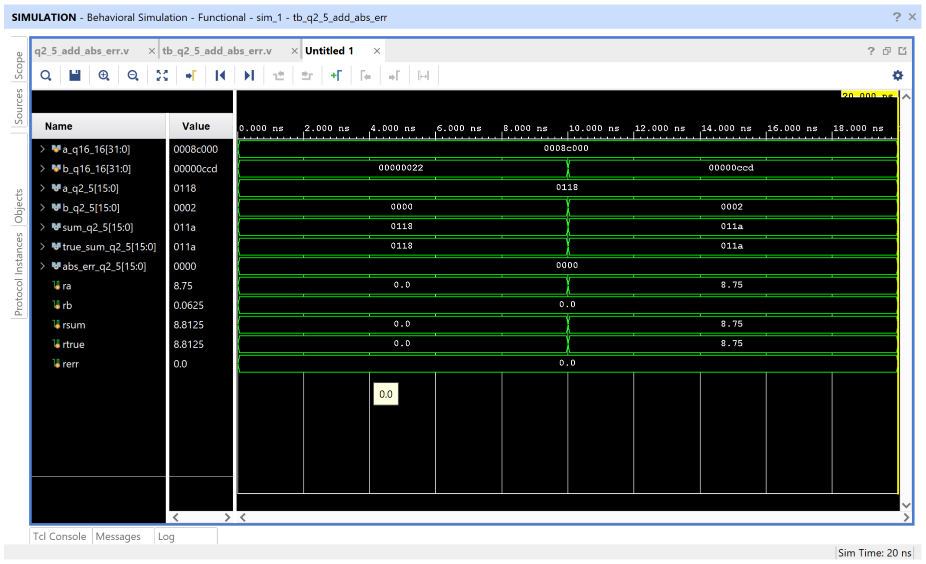
```
(
    input  signed [31:0] a_q16_16,
    input  signed [31:0] b_q16_16,

    output reg signed [15:0] a_q2_5,
    output reg signed [15:0] b_q2_5,
    output reg signed [15:0] sum_q2_5,

    output reg signed [15:0] true_sum_q2_5,
    output reg signed [15:0] abs_err_q2_5
);

    // For Q16.16 -> Q2.5, SHIFT = 16-5 = 11
    localparam SHIFT = (IN_FRAC - OUT_FRAC);

    // Round-to-nearest when shifting right by SHIFT bits.
    // "ties away from zero" implemented by +/- half LSB before
        arithmetic shift.
    function signed [15:0] round_q16_16_to_q2_5;
        input signed [31:0] x;
        reg   signed [31:0] adj;
        reg   signed [31:0] shifted;
        begin
            if (SHIFT <= 0) begin
                shifted = x <<< (-SHIFT);
            end else begin
                if (x >= 0)
                    adj = x + (32'sd1 <<< (SHIFT-1));
                else
                    adj = x - (32'sd1 <<< (SHIFT-1));

                shifted = adj >>> SHIFT; // arithmetic shift right
            end

            round_q16_16_to_q2_5 = shifted[15:0];
        end
    endfunction

    function signed [15:0] abs16;
        input signed [15:0] x;
        begin
            if (x < 0)
                abs16 = -x;
            else
                abs16 = x;
        end
    endfunction

    reg signed [31:0] true_sum_q16_16;
    reg signed [15:0] diff_q2_5;

    always @* begin
        // Quantize each input to Q2.5
        a_q2_5 = round_q16_16_to_q2_5(a_q16_16);
        b_q2_5 = round_q16_16_to_q2_5(b_q16_16);

        // Q2.5 addition
```

```verilog
        sum_q2_5 = a_q2_5 + b_q2_5;

        // True sum computed in higher precision then quantized to Q2.5
        true_sum_q16_16 = a_q16_16 + b_q16_16;
        true_sum_q2_5   = round_q16_16_to_q2_5(true_sum_q16_16);

        // Absolute error in Q2.5
        diff_q2_5    = true_sum_q2_5 - sum_q2_5;
        abs_err_q2_5 = abs16(diff_q2_5);
    end

endmodule
```

```verilog
`timescale 1ns/1ps

module tb_q2_5_add_abs_err;

    reg  signed [31:0] a_q16_16;
    reg  signed [31:0] b_q16_16;

    wire signed [15:0] a_q2_5;
    wire signed [15:0] b_q2_5;
    wire signed [15:0] sum_q2_5;
    wire signed [15:0] true_sum_q2_5;
    wire signed [15:0] abs_err_q2_5;

    q2_5_add_abs_err dut (
        .a_q16_16(a_q16_16),
        .b_q16_16(b_q16_16),
        .a_q2_5(a_q2_5),
        .b_q2_5(b_q2_5),
        .sum_q2_5(sum_q2_5),
        .true_sum_q2_5(true_sum_q2_5),
        .abs_err_q2_5(abs_err_q2_5)
    );

    real ra, rb, rsum, rtrue, rerr;

    task show;
        begin
            // Q2.5 scaling = /32
            ra    = a_q2_5 / 32.0;
            rb    = b_q2_5 / 32.0;
            rsum  = sum_q2_5 / 32.0;
            rtrue = true_sum_q2_5 / 32.0;
            rerr  = abs_err_q2_5 / 32.0;

            $display("--------------------------------------------------");
            $display("a_q16_16=%0d   b_q16_16=%0d", a_q16_16, b_q16_16);
            $display("a_q2_5        = %0d  -> %f", a_q2_5, ra);
            $display("b_q2_5        = %0d  -> %f", b_q2_5, rb);
            $display("sum_q2_5      = %0d  -> %f", sum_q2_5, rsum);
            $display("true_sum_q2_5 = %0d  -> %f", true_sum_q2_5,
                rtrue);
            $display("abs_err_q2_5  = %0d  -> %f", abs_err_q2_5,
                rerr);
        end
    endtask

    initial begin
        // Test 1: Part (c) values
        // 8.75 -> Q16.16 = 8.75 * 65536 = 573440
        // 0.000525 -> Q16.16 ~ 0.000525 * 65536 = 34.4064 -> 34
        a_q16_16 = 32'sd573440;
        b_q16_16 = 32'sd34;
        #10;
        show();

        // Test 2 (optional): b = 0.05 so it becomes visible in Q2.5
        // 0.05 * 65536 = 3276.8 -> 3277
        b_q16_16 = 32'sd3277;
        #10;
        show();

        $stop;
    end

endmodule
```

Behavioral Simulation Results

Figure 1 shows the behavioral simulation waveform.



Name	Value		
a_q16_16[31:0]	0008c000	0008c000	
b_q16_16[31:0]	00000ccd	00000022	00000ccd
a_q2_5[15:0]	0118	0118	
b_q2_5[15:0]	0002	0000	0002
sum_q2_5[15:0]	011a	0118	011a
true_sum_q2_5[15:0]	011a	0118	011a
abs_err_q2_5[15:0]	0000	0000	
ra	8.75	0.0	8.75
rb	0.0625	0.0	
rsum	8.8125	0.0	8.75
rtrue	8.8125	0.0	8.75
rerr	0.0	0.0	

Figure 1: Behavioral Simulation of Q2.5 Adder

For the first test case:

$$a = 8.75, \quad b = 0.000525$$

Quantization to Q2.5 gives:

$$8.75 \rightarrow 280/32 = 8.75$$
$$0.000525 \rightarrow 0$$

Thus:

$$\text{Sum}_{Q2.5} = 8.75$$
$$\text{TrueSum}_{Q2.5} = 8.75$$
$$\text{Absolute Error} = 0$$

This confirms that the small value disappears due to limited resolution.

For the second test case ($b = 0.05$):

$$0.05 \rightarrow 2/32 = 0.0625$$

Thus:

$$8.75 + 0.0625 = 8.8125$$

Again, the computed error is zero because both sums quantize to the same Q2.5 value.

Implemented Design View

Figure 2 shows the implemented device view in Vivado.

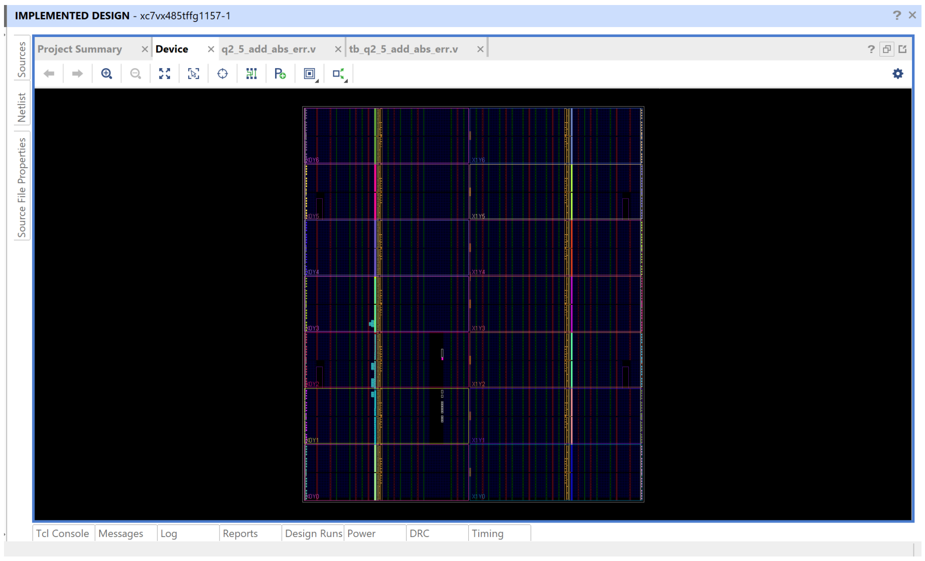


Figure 2: Implemented Device Layout

RTL Schematic

The RTL schematic is shown in Figures 3 and 4.

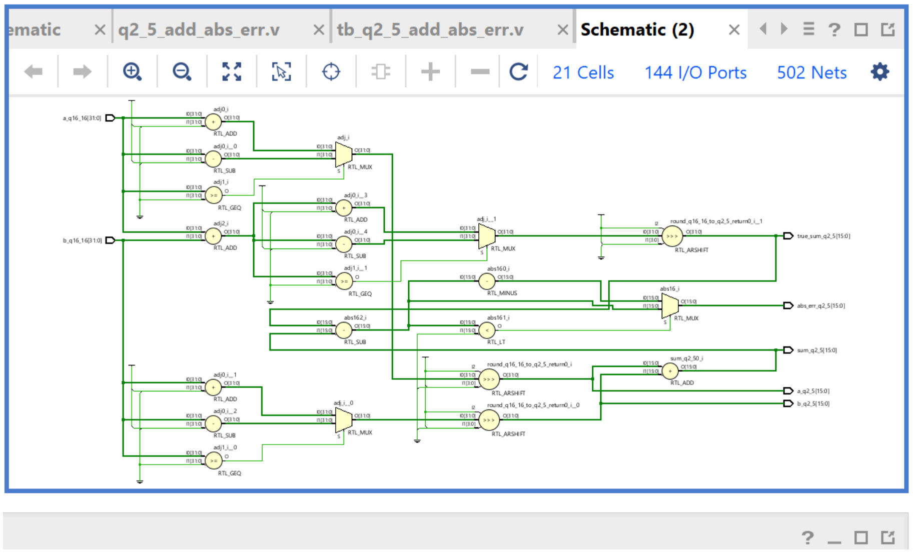


Figure 3: RTL Schematic (Part 1)

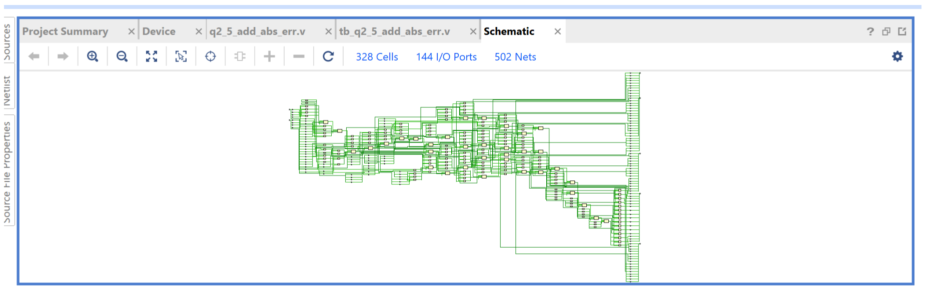
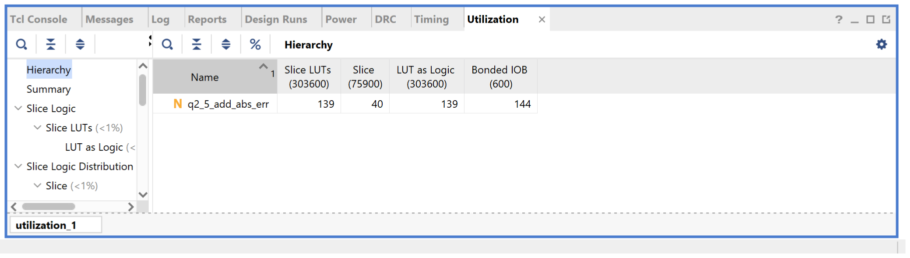


Figure 4: RTL Schematic (Part 2)

Utilization Report

Figure 5 shows the resource utilization summary.



Name	Slice LUTs (303600)	Slice (75900)	LUT as Logic (303600)	Bonded IOB (600)
q2_5_add_abs_err	139	40	139	144

Figure 5: Resource Utilization Summary

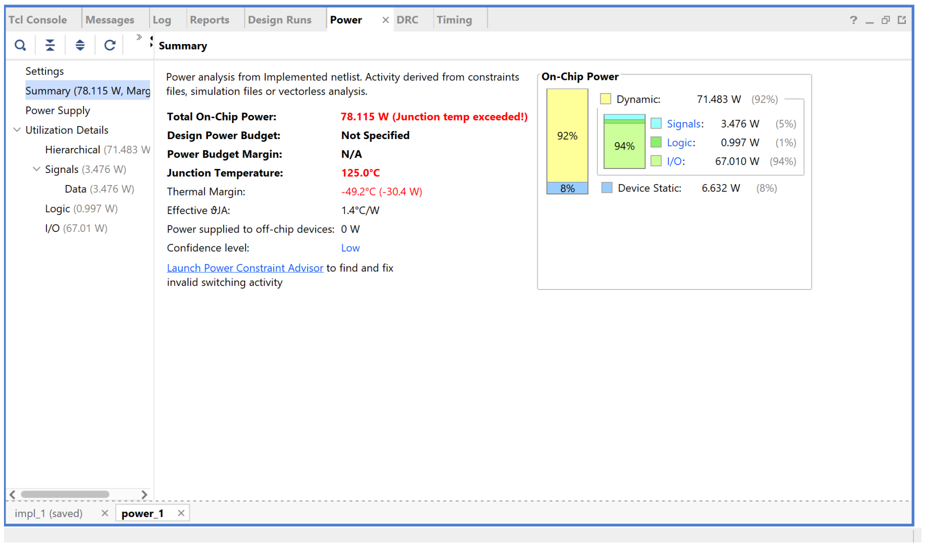
The design uses:

- 139 LUTs
- 40 Slice Registers

This represents less than 1% of the available FPGA resources, indicating a lightweight implementation.

Power Analysis

Figure 6 shows the power report generated by Vivado.



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**78.115 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-49.2°C (-30.4 W)
Effective ϑJA:	1.4°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:	71.483 W	(92%)
🟦	Signals:	3.476 W	(5%)
🟩	Logic:	0.997 W	(1%)
🟩	I/O:	67.010 W	(94%)
🟦	Device Static:	6.632 W	(8%)

Figure 6: Power Analysis Report

The high reported power consumption is primarily due to unconstrained I/O switching activity during simulation and does not reflect actual hardware deployment conditions.

Conclusion

The Q2.5 fixed-point adder was successfully implemented and verified in Vivado. The results confirm that:

- Small values below half-LSB are lost due to quantization.
- The fixed-point addition behaves as predicted theoretically.
- The hardware implementation matches mathematical expectations.

This experiment clearly demonstrates the impact of limited fractional resolution in fixed-point arithmetic.