IEEE 754 32-bit Floating Point Multiplier

Lakshit Verma (230959210) Pranav Saini (230959212)

February 3, 2025

Abstract

Floating point arithmetic is widely used in various computing applications, including scientific computations, digital signal processing, and graphics processing. This project focuses on the design and implementation of an IEEE 754 compliant 32-bit floating point multiplier using VLSI design methodologies. The IEEE 754 standard defines a precise format for representing floating point numbers and ensures accuracy in floating point operations.

The proposed design consists of three main stages: exponent addition with bias adjustment, mantissa multiplication with normalization, and rounding to maintain precision. Special cases such as underflow, overflow, and handling of special numbers (`NaN`, `INFINITY`) are also considered. The implementation is carried out using Verilog HDL and synthesized using FPGA-based tools to evaluate the area, power, and timing performance.

The results demonstrate efficient multiplication with optimized resource utilization, making it suitable for high-performance computing applications. This mini-project provides insights into floating point arithmetic and its practical realization in VLSI circuits.

IEEE 754 Format

A 32-bit floating-point number consists of three parts:

```
SEEEEEEE EMMMMMMM MMMMMMMM MMMMMMMM
```

where:

- **`S` (1 bit)** → Sign bit (0 = positive, 1 = negative)
- **`E` (8 bits)** → Exponent (biased by 127)
- **`M` (23 bits)** → Mantissa (fraction part, normalized)

Let's take a sample 32-bit binary representation:

```
0 10000010 11000000000000000000000
```

- **Sign bit (0)** → Positive number
- **Exponent (10000010 in binary = 130 in decimal)** Actual exponent = `130 - 127 = 3`
- **Mantissa (1.110 in binary = 1.75 in decimal, since implicit leading 1)**

Thus, the value is:

$$1.75 \times 2^3 = 14.0$$

Implementation

The proposed floating point multiplier follows three main stages: **exponent addition with bias adjustment**, **mantissa multiplication with normalization**, and **rounding to maintain precision**. Special cases such as **underflow, overflow, NaN, and `INFINITY`** are also handled. Below is a snippet of the Verilog code.

```
module fp_multiplier (
    input [31:0] a, b,  // 32-bit IEEE 754 floating point inputs
    output reg [31:0] result
);
    wire sign_a, sign_b, sign_res;
    wire [7:0] exp_a, exp_b, exp_res;
    wire [23:0] mant_a, mant_b;
    wire [47:0] mant_res;
    reg [22:0] mant_final;
    reg [7:0] exp_final;
.
.
.
endmodule
```

The steps of the code are as follows:

1. **Extraction:** The sign, exponent, and mantissa are extracted from the IEEE 754 32-bit inputs.
2. **Exponent Addition:** Exponents are added with bias adjustment (-127).
3. **Mantissa Multiplication:** The mantissas are multiplied (with implicit leading 1).
4. **Normalization:** If necessary, the result is shifted to maintain proper format.
5. **Special Cases:** Handles `NaN`, `INFINITY`, Underflow, and Overflow cases.
6. **Final Assembly:** The result is reconstructed into IEEE 754 format.

Power, Area and Timing Analysis

Once the Verilog code and testbench are synthesized in an FPGA tool, the area, power, and timing performance can be analyzed using reports.

To **analyze power, area, and timing**, we may use **RTL Compiler**:

1. **Launch RTL Compiler**

```
rc -gui &
```

2. **Load the Design**

```
read_hdl fp_multiplier.v
read_hdl fp_multiplier_tb.v
elaborate
```

3. **Synthesize the Design**

```
compile_ultra
```

4. **Generate Reports**

```
report_area > area_report.txt
report_power > power_report.txt
report_timing > timing_report.txt
```

Performance Metrics

After running the above steps, we check:

- **Power (mW):** Dynamic and static power consumption.
- **Area (µm²):** Standard cells and logic gate usage.
- **Timing (ns):** Maximum operating frequency (F_{max}).

Keywords: IEEE 754, Floating Point Arithmetic, Multiplier, Verilog, VLSI, FPGA