Floating-Point Division Report

1. Introduction to Floating-Point Arithmetic

1.1 Definition and Importance

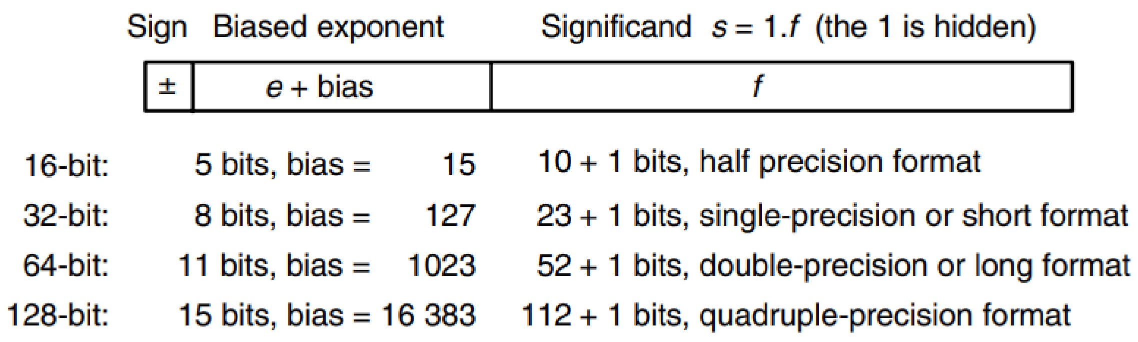
Floating-point arithmetic is essential in computer systems for representing real numbers with a wide dynamic range. Unlike fixed-point representation, floating-point numbers enable efficient handling of very large and very small values.

1.2 IEEE 754 Standard

The IEEE 754 standard defines the representation and operations of floating-point numbers. It includes:

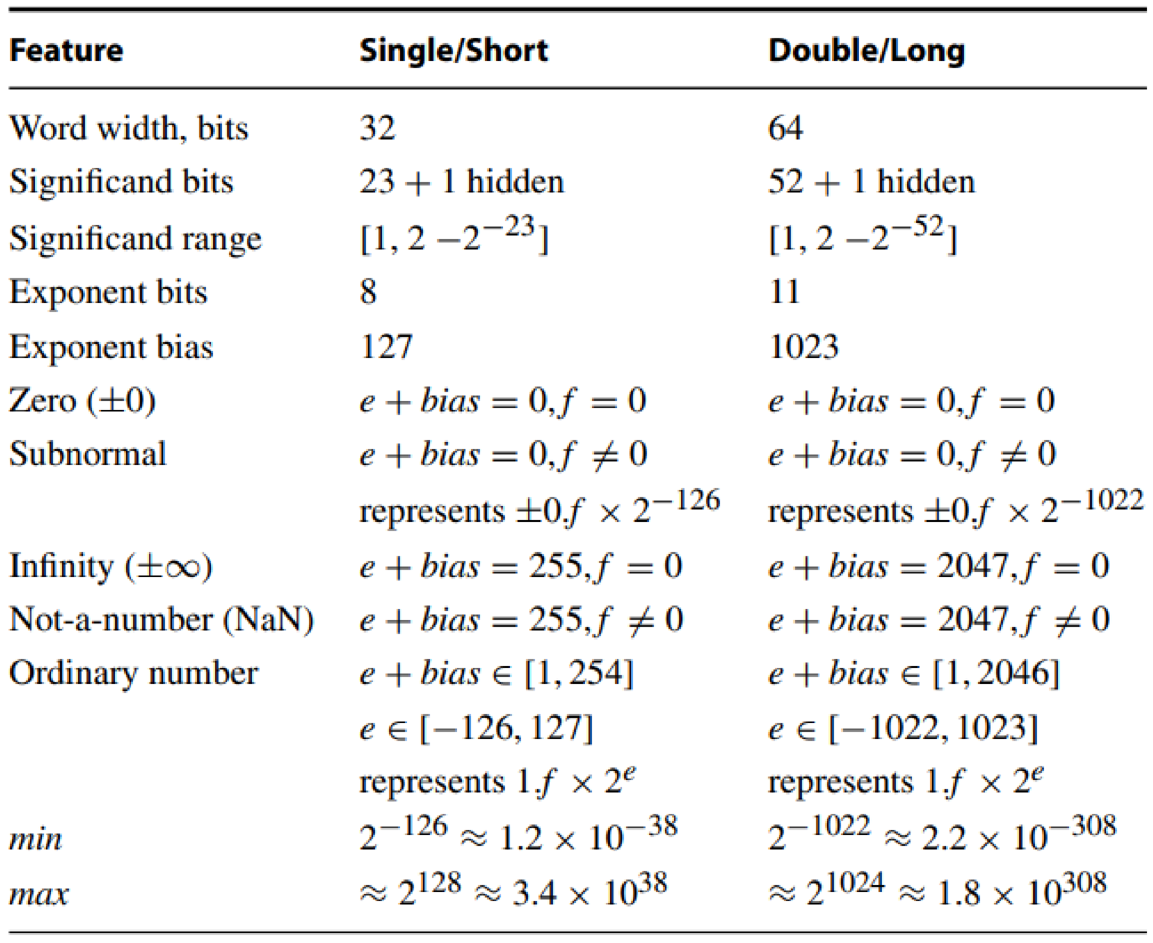
- **Single Precision (32-bit)**: 1-bit sign, 8-bit exponent, 23-bit fraction.
- **Double Precision (64-bit)**: 1-bit sign, 11-bit exponent, 52-bit fraction.
- **Operations**: Addition, subtraction, multiplication, and division.

IEEE 754 Floating-Point Format



16-bit:	5 bits, bias = 15	10 + 1 bits, half precision format
32-bit:	8 bits, bias = 127	23 + 1 bits, single-precision or short format
64-bit:	11 bits, bias = 1023	52 + 1 bits, double-precision or long format
128-bit:	15 bits, bias = 16 383	112 + 1 bits, quadruple-precision format

Table 17.1 Some features of the IEEE 754-2008 short and long floating-point formats



Feature	Single/Short	Double/Long
Word width, bits	32	64
Significand bits	23 + 1 hidden	52 + 1 hidden
Significand range	$[1, 2 - 2^{-23}]$	$[1, 2 - 2^{-52}]$
Exponent bits	8	11
Exponent bias	127	1023
Zero (± 0)	$e + bias = 0, f = 0$	$e + bias = 0, f = 0$
Subnormal	$e + bias = 0, f \neq 0$	$e + bias = 0, f \neq 0$
	represents $\pm 0.f \times 2^{-126}$	represents $\pm 0.f \times 2^{-1022}$
Infinity ($\pm\infty$)	$e + bias = 255, f = 0$	$e + bias = 2047, f = 0$
Not-a-number (NaN)	$e + bias = 255, f \neq 0$	$e + bias = 2047, f \neq 0$
Ordinary number	$e + bias \in [1, 254]$	$e + bias \in [1, 2046]$
	$e \in [-126, 127]$	$e \in [-1022, 1023]$
	represents $1.f \times 2^e$	represents $1.f \times 2^e$
min	$2^{-126} \approx 1.2 \times 10^{-38}$	$2^{-1022} \approx 2.2 \times 10^{-308}$
max	$\approx 2^{128} \approx 3.4 \times 10^{38}$	$\approx 2^{1024} \approx 1.8 \times 10^{308}$

2. Floating-Point Division

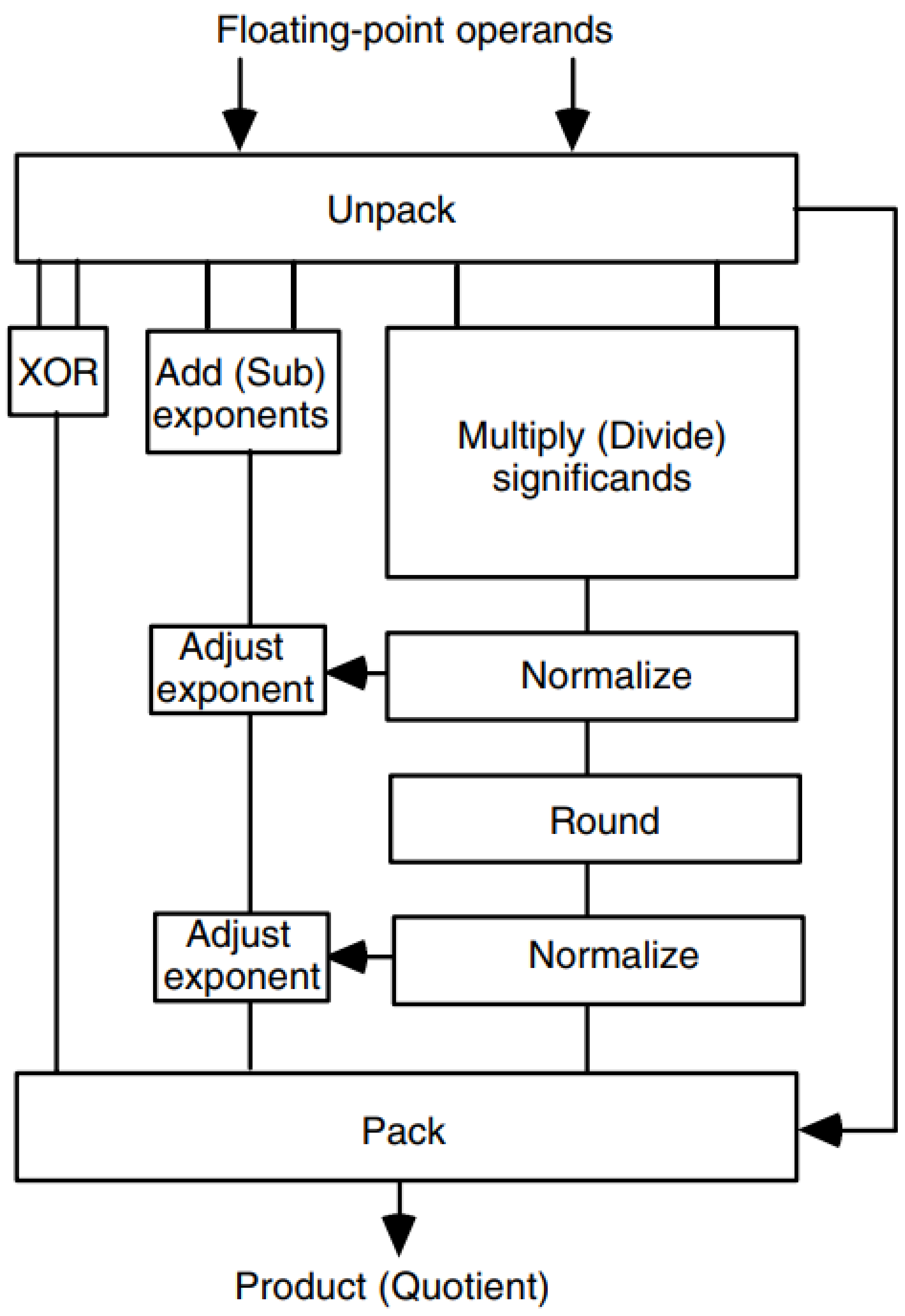
2.1 Algorithm and Implementation

Floating-point division follows these steps:

1. **Extract Components**: Sign, exponent, and mantissa.
2. **Compute Sign**: XOR the signs of the numerator and denominator.
3. **Exponent Calculation**: Subtract the denominator exponent from the numerator exponent.
4. **Mantissa Division**: Perform division using normalized mantissas.
5. **Normalization**: Adjust results to maintain proper IEEE 754 format.

Challenges in Floating-Point Division

- Precision loss due to rounding.
- Handling special cases like division by zero, infinity, and NaN.
- Latency issues in hardware implementations.



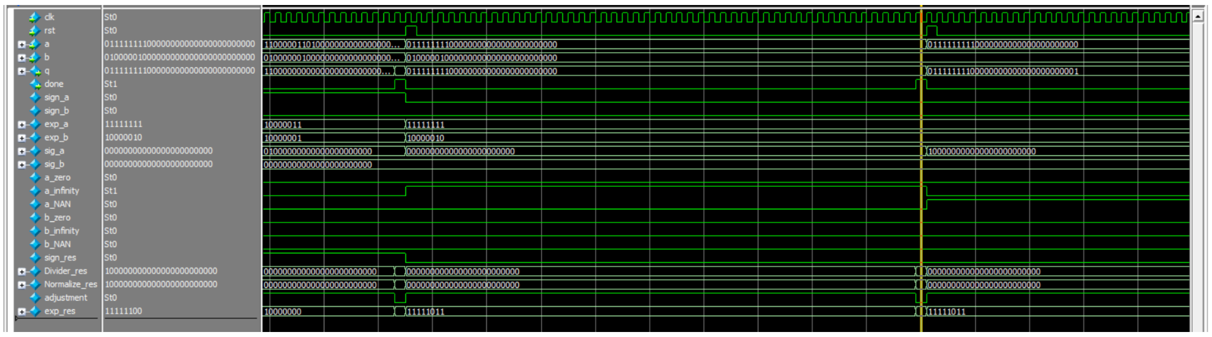
3. Simulation and Implementation

3.1 ModelSim Results

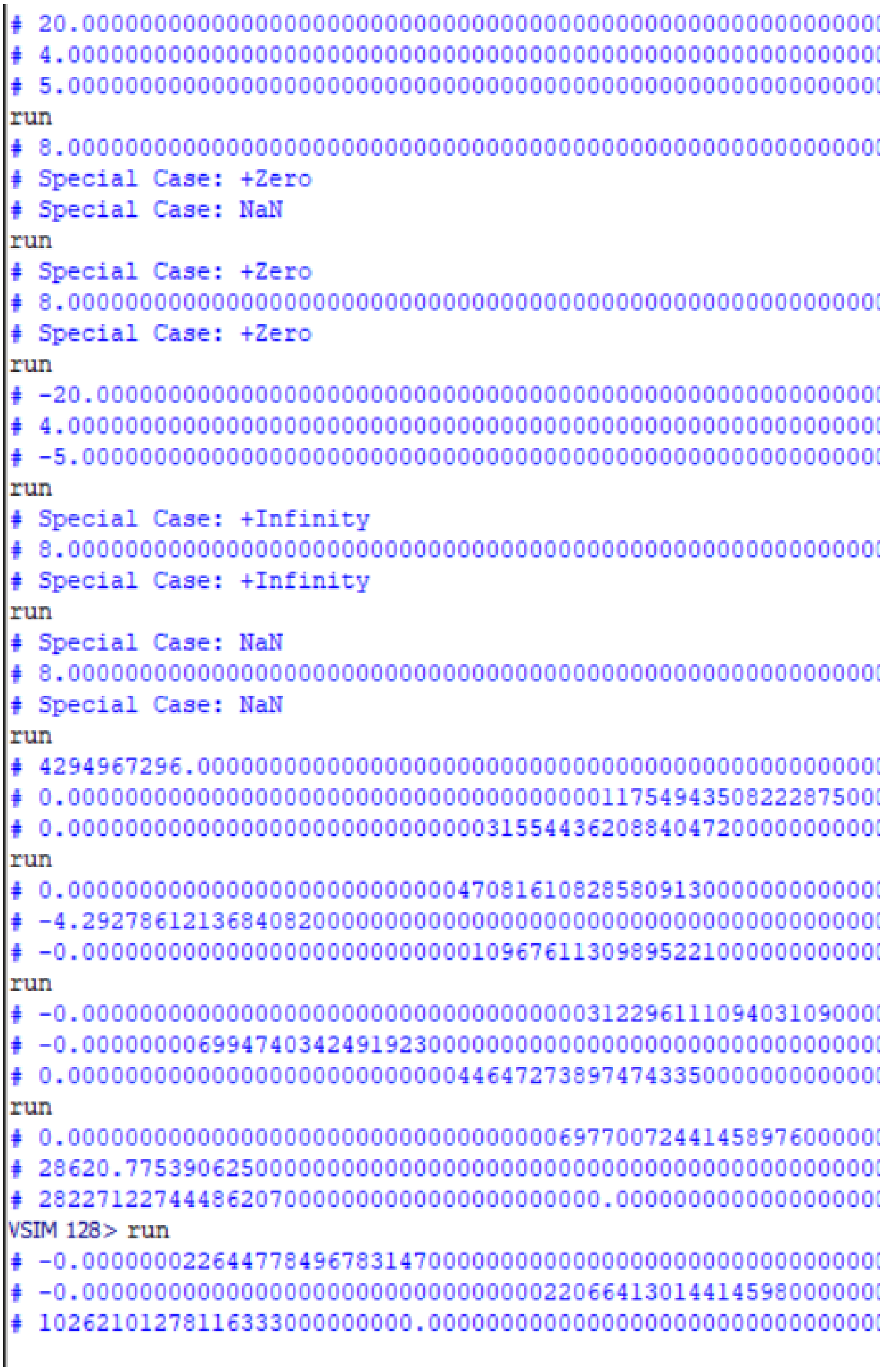
Simulation in **ModelSim** verifies the floating-point division logic.

- **Testbench setup**: Inputs are provided as IEEE 754 formatted numbers.
- **Waveform Analysis**: Results are compared against expected values.

  Screenshots and Explanation:



```
clk            St0
rst            St0
a              01111111100000000000000000000000
b              01000001000000000000000000000000
q              01111111100000000000000000000000
done           St1
sign_a         St0
sign_b         St0
exp_a          11111111
exp_b          10000010
sig_a          00000000000000000000000
sig_b          00000000000000000000000
a_zero         St0
a_infinity     St1
a_NAN          St0
b_zero         St0
b_infinity     St0
b_NAN          St0
sign_res       St0
Divider_res    100000000000000000000000
Normalize_res  100000000000000000000000
adjustment     St0
exp_res        11111100
```



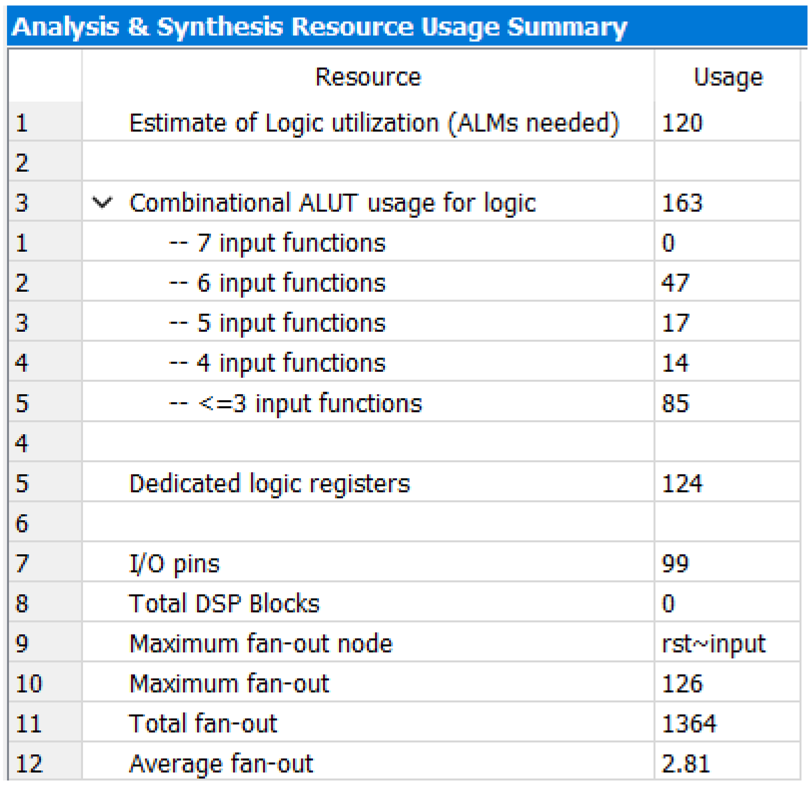
```
# 20.0000000000000000000000000000000000000000000000000000000
# 4.00000000000000000000000000000000000000000000000000000000
# 5.00000000000000000000000000000000000000000000000000000000
run
# 8.00000000000000000000000000000000000000000000000000000000
# Special Case: +Zero
# Special Case: NaN
run
# Special Case: +Zero
# 8.00000000000000000000000000000000000000000000000000000000
# Special Case: +Zero
run
# -20.000000000000000000000000000000000000000000000000000000
# 4.00000000000000000000000000000000000000000000000000000000
# -5.0000000000000000000000000000000000000000000000000000000
run
# Special Case: +Infinity
# 8.00000000000000000000000000000000000000000000000000000000
# Special Case: +Infinity
run
# Special Case: NaN
# 8.00000000000000000000000000000000000000000000000000000000
# Special Case: NaN
run
# 4294967296.00000000000000000000000000000000000000000000000
# 0.00000000000000000000000000000000000011754943508222875000
# 0.00000000000000000000000000003155443620884047200000000000
run
# 0.00000000000000000000000000470816108285809130000000000000
# -4.292786121368408200000000000000000000000000000000000000
# -0.00000000000000000000000000109676113098952210000000000000
run
# -0.0000000000000000000000000000000000031229611109403109000
# -0.0000000699474034249192300000000000000000000000000000000
# 0.00000000000000000000000000044647273897474335000000000000
run
# 0.00000000000000000000000000000000006977007244145897600000
# 28620.7753906250000000000000000000000000000000000000000000
# 282271227444862070000000000000000000000.000000000000000000
VSIM 128> run
# -0.0000000226447784967831470000000000000000000000000000000
# -0.0000000000000000000000000000000002206641301441459800000
# 10262101278116333000000000.0000000000000000000000000000000
```

3.2 Quartus Results

Implementation in **Quartus** targets an FPGA.

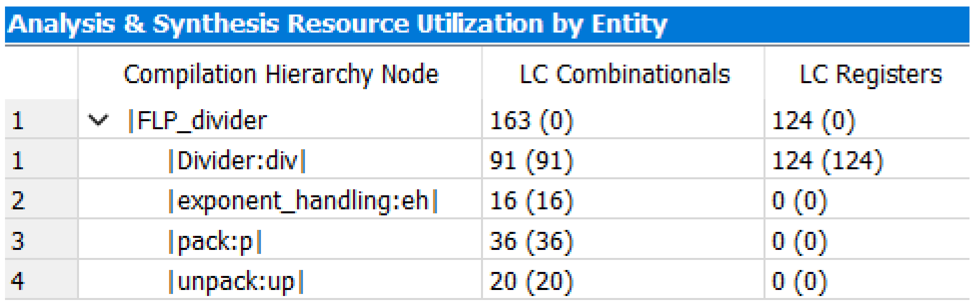
- **Synthesis Report**: Area utilization and timing analysis.

Diagrams and Observations:



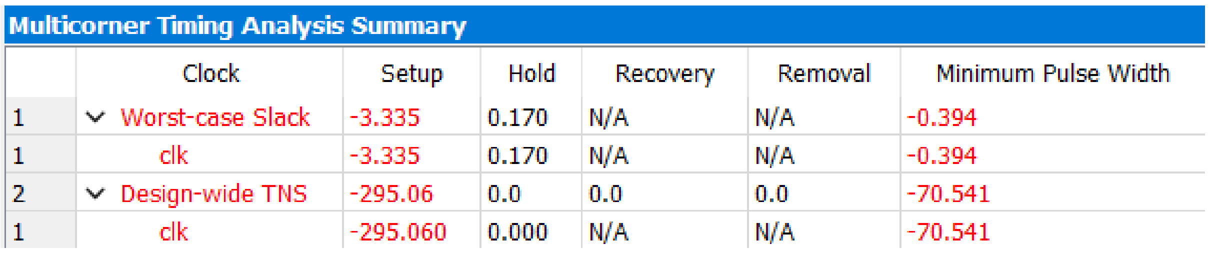
Analysis & Synthesis Resource Usage Summary

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	120
2		
3	Combinational ALUT usage for logic	163
1	-- 7 input functions	0
2	-- 6 input functions	47
3	-- 5 input functions	17
4	-- 4 input functions	14
5	-- <=3 input functions	85
4		
5	Dedicated logic registers	124
6		
7	I/O pins	99
8	Total DSP Blocks	0
9	Maximum fan-out node	rst~input
10	Maximum fan-out	126
11	Total fan-out	1364
12	Average fan-out	2.81

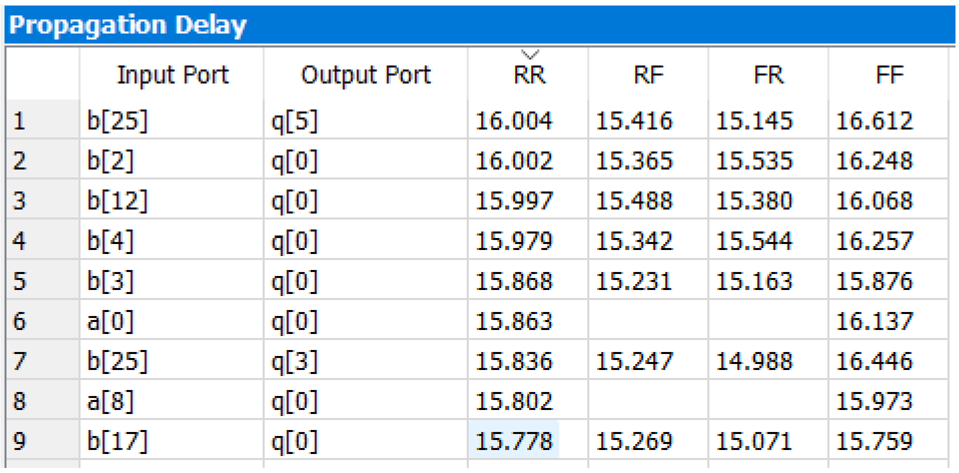


Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	LC Combinationals	LC Registers
1	|FLP_divider	163 (0)	124 (0)
1	|Divider:div|	91 (91)	124 (124)
2	|exponent_handling:eh|	16 (16)	0 (0)
3	|pack:p|	36 (36)	0 (0)
4	|unpack:up|	20 (20)	0 (0)


Multicorner Timing Analysis Summary

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	-3.335	0.170	N/A	N/A	-0.394
1	clk	-3.335	0.170	N/A	N/A	-0.394
2	Design-wide TNS	-295.06	0.0	0.0	0.0	-70.541
1	clk	-295.060	0.000	N/A	N/A	-70.541


Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	b[25]	q[5]	16.004	15.416	15.145	16.612
2	b[2]	q[0]	16.002	15.365	15.535	16.248
3	b[12]	q[0]	15.997	15.488	15.380	16.068
4	b[4]	q[0]	15.979	15.342	15.544	16.257
5	b[3]	q[0]	15.868	15.231	15.163	15.876
6	a[0]	q[0]	15.863			16.137
7	b[25]	q[3]	15.836	15.247	14.988	16.446
8	a[8]	q[0]	15.802			15.973
9	b[17]	q[0]	15.778	15.269	15.071	15.759

r4. Conclusion and Observations

4.1 Performance Analysis

The floating-point division implementation was evaluated in terms of:

- **Accuracy and Precision**: The results were verified against expected floating-point division values, demonstrating the correctness of the implementation.
- **Hardware Resource Utilization**: The synthesis reports from Quartus provided insights into the area and power consumption, indicating the feasibility of the design.
- **Latency Considerations**: The number of clock cycles required to complete a division operation was analyzed, showcasing the trade-offs between accuracy and speed.

4.2 Key Findings and Future Improvements

- **Strengths**: The division algorithm efficiently handles normalization and rounding while maintaining IEEE 754 compliance.
- **Limitations**: Certain edge cases, such as very small numbers (denormals) and rounding errors, can impact precision.

- **Potential Improvements**: Future optimizations could include:
  - Implementing a faster division algorithm to reduce latency.
  - Exploring alternative hardware implementations for better resource efficiency.
  - Enhancing precision handling for subnormal numbers and rounding modes.

○

○ **Thank you**