Grenze International Journal of Engineering and Technology, Jan Issue

Design and Synthesizing of Floating Point Adder and Multiplier using Cadence RTL Compiler

Mr. Chandra Shekar P[1], Kiran A[2], Sougandha G[3], Mamatha[4] and Ramya N[5]
[1-5]ATME College of Engineering, Mysuru, India
Email: cspatmece@gmail.com

***Abstract*—Floating point arithmetic is widely used in many areas. IEEE Standard 754 floating point is the most common representation today for real numbers on computers. IEEE standards specify a set of floating-point formats for single precision and double precision. The representation of very large or very small values, is carried out using the IEEE-754 floating-point standard. This work intends to develop an Verilog implementation of a floating-point arithmetic unit which can perform addition, subtraction and multiplication operations on 32-bit operands using IEEE 754-2008 representations. Low power consumption and smaller area are some of the most important criteria for the fabrication high performance systems. The algorithms will be model in Verilog and synthesized using Cadence RTL complier using Cadence 45nm GPDK.**

***Index Terms*— IEEE 754-2008 standard, Floating point multiplication, adder, cadence RTL compiler, low power.**

I. INTRODUCTION

Floating point number system is a common choice for many scientific computations due to its wide dynamic range feature. For instance, floating point arithmetic is widely used in many areas, especially in scientific
The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, that is, the decimal point or binary point can float. There are also representations in which the number of digits before and after the decimal or binary point is fixed, called fixed-point representations.
The work is to provide Hardware support for floating point arithmetic. Because we already have hardware support for integer base operation but we don't have hardware support for Floating point arithmetic.
Comprehending how to represent floating point numbers in the processor. Like how does exactly processor handle such floating numbers. and how do perform arithmetic with floating point numbers, like how do we addition, multiplication etc. using floating point arithmetic numbers IEEE Floating point number consists of three fields:

$$Z = (-1)^{s} * 2^{(E^{-}b\square aS)} * (1 \cdot M)$$

Sign (S): It used to denote the sign of the number i.e.,0 represent positive number and 1 represent negative number.
Mantissa (M): Mantissa is part of a floating-point number which represents the magnitude of the number.
Exponent (E): Exponent is part of the floating-point number that represents the number of places that the ++ Xdrf + decimal point (or binary point) is to be moved.

Grenze ID: 01.GIJET.9.1.658

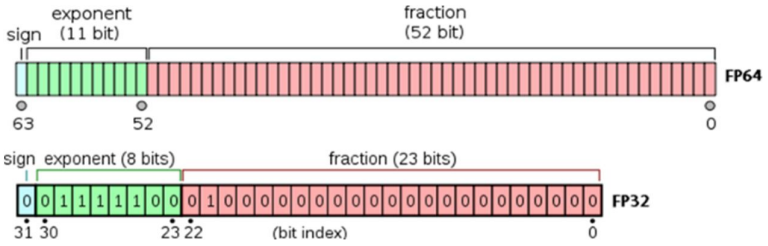

Figure 1 IEEE Double and Single Precision floating point format

This is called a normalization representation, because we don't have any leading zeros as we mentioned.
There are 2 types of IEEE 754 standard representation

1. Single precision (32bit)
2. Double precision (64 bit)

TABLE I. REPRESENTATION OF FLOATING-POINT FORMAT

	Sign	Exponent	Fraction	Bias
Single precision	1[31]	8[30-23]	23[22-00]	127
Double precision	1[63]	11[62-52]	52[51-00]	1023

II. PROBLEM STATEMENT

Floating point numbers are limited in size, so they can theoretically only represent certain numbers. Everything that is in between has to be rounded to be rounded to be closest possible number. This can cause errors in a number that it stored.
Therefore, the result of a floating point calculation must often be rounded in order to fit back into its finite representation. Many numbers can't be represented by a sum of finite number of those inverse powers. Using more bit values will increase the precision of the representation, but never get it exactly because it only has a limited number of bits. The work carried out is intended to implement a Verilog implementation of a 32-bit binary floating-point adder.

III. OBJECTIVE

The objective of this project is to implement a 32-bit binary floating-point arithmetic unit which includes floating point adder, subtractor and multiplier. Pipelined architecture will be used in order to increase the performance and to increase the operating frequency. The main objective of this work will be.

- Development of a Verilog behavioral implementation of a floating-point arithmetic unit
- Development of a Verilog structural implementation of a floating-point adder with required constituent blocks
- A comparison of performance of the above implementations at the RTL-level.

At the same time, the related logic includes both normalizer and the rounder according to the inexact mantissa and exponent parts. Floating point arithmetic is handled by the FP add, FP sub, FP mull. FP add adds the value in the floating-point accumulator to the floating-point accumulator.

IV. LITERATURE REVIEW

The implementation of Low Power VLSI Architectures in the area of Digital Image Processing applications, Matrix Multiplication is a key arithmetic operation. To construct VLSI architectures with Low Power, High Speed and Low area, Matrix Multiplication design becomes complex. In this paper, a simple novel VLSI architecture for FPMM is presented [1]. To presents design of high speed floating point unit using reversible logic. In recent nanotechnology, Programmable reversible logic design is trending as a prospective logic design style for implementation and quantum computing with low impact on circuit heat generation. Floating-point operations are used very frequently in nearly all computing disciplines. The analysis of proposed reversible circuit can be done in terms of quantum cost, garbage outputs, constant inputs, power consumption, speed and area. [2]. The Peres Full Adder Gate (PFAG) is used in reversible ALU design and HNG gate is used as an adder

logic circuit in the proposed ALU design 2. Both proposed designs are analysed and compared in terms of number of gates count, garbage output, quantum cost and propagation delay. The simulation results show that the proposed reversible ALU design 2 outperforms the proposed reversible ALU design 1 and conventional ALU design. [3]. The new reversible design of single precision floating point multiplier has been proposed based on operand decomposition approach. Furthermore, a new reversible design of the 8x8 bit Wallace tree multiplier has proposed that is optimized in terms of quantum cost, delay, and number of garbage outputs. Wallace tree multiplication consists of three conceptual stages: Partial product generation, partial product compression using 4:2 compressors, full adders, and half adders, and then the final addition stage to generate the product. In this work we perform optimization at each of these three stages [4]. The Reversible gates namely TSG gate performs 1-bit addition with carry. This is the first reversible gate which alone can acts as full adder. Gate is used to perform logical operations like AND, OR. In this works, designing 1-bit alum has also been presented using pass transistor with virtuoso tool of cadence. Based on analysis of the result, this design using reversible gates is better than that using the irreversible gates [5].

V. IMPLEMENTATION

The two 32-bit floating-point numbers are given as inputs to adders (subtractors). As shown in the figure the mantissa, exponential and sign parts are ca loculate according to the flow Architecture.

A. Floating-point Adder Architecture

Digital addition operation plays an important role in recent embedded processors and the performance of adder unit basically decides the performance of ALU in CPU and GPU. The performance of such complex system is completely depending upon the efficient design of adders and multipliers. Out of that the most fundamental component is the adders which are incredibly often present inside the building blocks of various systems like controllers and processing chips, FIR filtering, communication, and cryptography etc. Figure 2 illustrated the design process of the floating-point addition and subtraction.

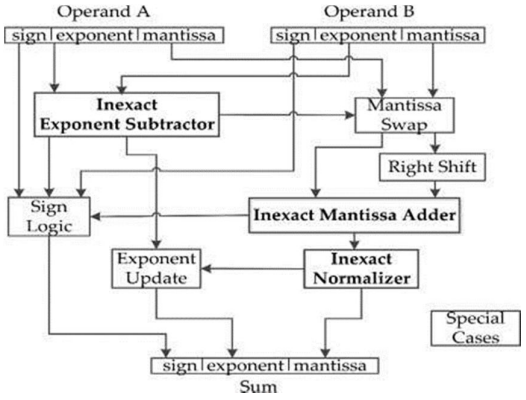

Figure 2- Floating point adder architecture

Traditional designs apply fully accurate computing to all types of applications; however, error tolerant applications involving human intervention do not require full accuracy. So, it is possible to perform computation with inexact circuits; in these cases, inexact computing is an attractive approach to save power and area, it replaces the mantissa swap circuit with the 2x1 multiplexers. so, the number of gates required for the inexact is very less and power consumption also very small and the Accurate floating-point adder (FPA) has a rounder to round off the value at the end but in proposed inexact floating-point adder (FPA) the rounder block was eliminated. difference will contain 8 bits. The last 4 bits i.e., Least significant bits (LSB) will act as a selector signal for the shifter which is designed with twenty-three 4×1 multipliers of 4 stage. This shifter will shift up to 15 bits in the mantissa. The 8 bits in the difference will get added by using OR gate and changed as a single control signal which will act as a selector for those two 4x1 multiplexer in the proposed architecture.

B. Floating Point Multiplier Architecture

The algorithm for floating point multiplication is explained through the flow chart in Figure 3. Let N1 and N2 are normalized operands represented by S1, M1, E1 and S2, M2, E2 as their respective sign bit, mantissa (significand) and exponent. Basically, the following four steps are used for floating point multiplication.

Multiply significands add exponents and determine sign
M=M1*M2
E=E1=E2-Bias
S=S1 XOR S2

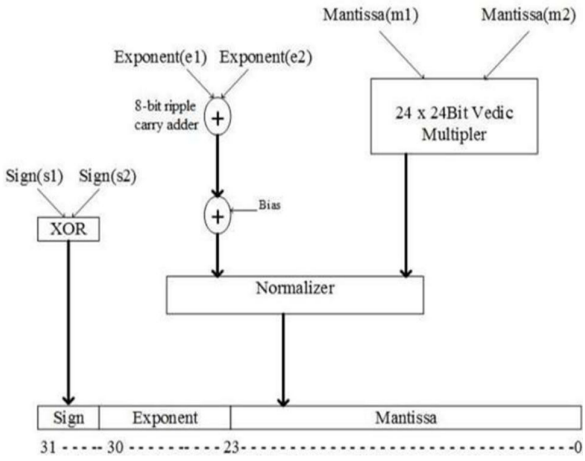

Figure 3 -Floating Point Multiplier Architecture

1. Normalize Mantissa M (Shift left or right by 1) and update exponent E
2. Rounding the result to fit in the available bits

Determine exception flags and special values for overflow and underflow. Sign Bit Calculation: The result of multiplication is a negative sign if one of the multiplied numbers is of a negative value and that can be obtained by XORing the sign of two inputs. Exponent Addition is done through unsigned adder for adding the exponent of the first input to the exponent of the second input and after that subtract the Bias (127) from the addition result (i.e., E1+E2-Bias). The result of this stage can be called an intermediate exponent. Significand Multiplication is done for multiplying the unsigned significand and placing the decimal point in the multiplication product. The result of significand multiplication can be called as intermediate product (IP). The unsigned significand multiplication is done on 24 bits.

VI. RESULT

RTL Schematic

The Synthesis of the below Floating-Point Multiplier is done using Cadence RTL Compiler in 45nm technology. The area, leakage power, net power, timing (worst path delay) and power delay product has been calculated for Floating Point Multiplier.

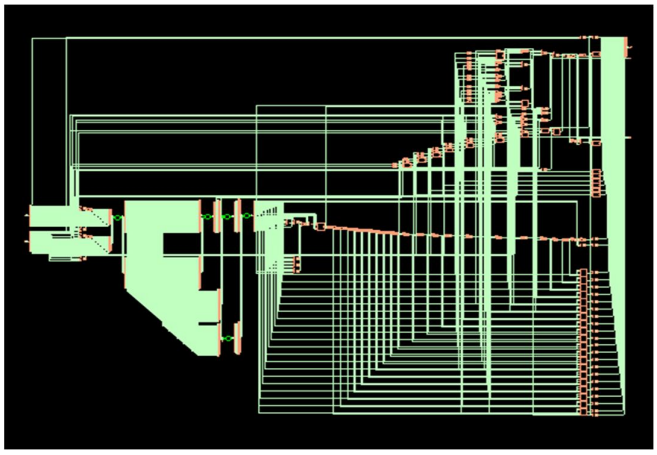

Figure 4: RTL Schematic of Floating Point Multiplier

TABLE II. FLOATING-POINT ADDER

Instance	Cells	Leakage power (nw)	Dynamic power (nw)	Total power (nw)
Multiplication	1431	0.680	24989 86.627	24989 87.307
Csa_tree_mul_17_23	1081	0.417	16429 97.719	162449 98.136
Final_adder_ Mul_17_23	49	0.048	24641 5.785	24641 5.833
Csa_mux_A_mux_20_29	48	0.047	10756 2.175	10756 2.222
Final_adder_Mux_20_29	47	0.047	27702 6.944	27702 6.991
Csa_mux_B_mux_20_29	48	0.046	6125 6.840	6125 6.886

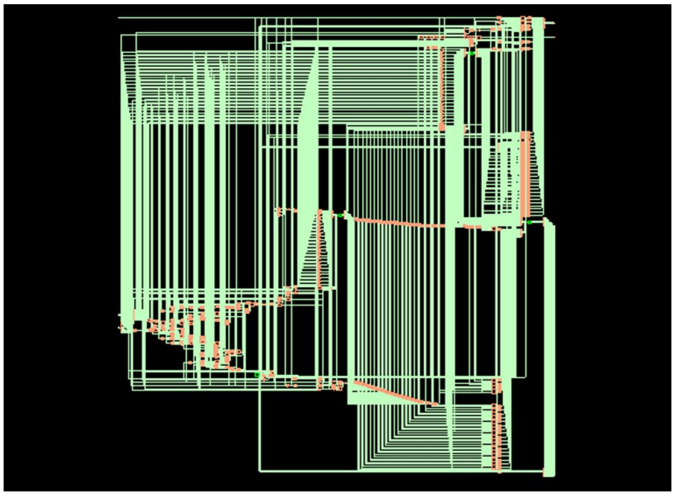

Figure 5: RTL Schematic of Floating Point Adder

The Synthesis of the below Floating Point Adder is done using Cadence RTL Compiler in 45nm technology. The area, leakage power, net power, timing(worst path delay) and power delay product has been calculated for Floating Point Adder.

TABLE III. FLOATING-POINT ADDER

Instance	Cells	Leakage Power (nW)	Dynamic Power (nW)	Total Power (nW)
Addition	512	0.270	29793 0.360	29793 0.630
add_49_81	24	0.024	2309 5.352	2309 5.376
add_58_52	24	0.024	1669 0.031	1669 0.0055
srl_42_47	109	0.023	8205 7.371	8205 7.395

III. CONCLUSIONS

This work intends to develop an HDL implementation of the floating point arithemetic unit which performed addition and multiplication operations on 32-bit operands. Its specifics a set of floating-point formats for single precision and double precision. The multiplier on the other hand has a combinational area which is almost twice as large as the multiplier. The combinational area of a high radix floating point adder is shown to be around 30% smaller than the combinational area of floating-point adder. We are implemented adder and multiplier using standard cell in 45nm. We synthesis the Verilog code for adder and multiplier and generate the physical design of adder and multiplier.

REFERENCES

[1] Lakshmi kiran Mukkara et al. for implementation of Low Power VLSI Architectures in the area of Digital Image Processing applications.

[2] IEEE 754 single-precision (or IEE 754-2008 "binary32") (ubiquitous). In the IEEE 754- 2008 standard, the 32-bit base-2 format is officially referred to as binary32.

[3] IEEE 754 double-precision (or IEE 754-2008 "binary64") (nearly ubiquitous). In the IEEE 754-2008 standard, the 64-bit base-2 format is officially referred to as binary64. Eclair, Andreas. "Area efficient floating-point adder and multiplier with IEEE-754 compatible semantics." 2014 International Conference on Field-Programmable Technology (FPT). IEEE, 2014.

[4] Weste, Neil HE, and David Harris. CMOS VLSI design: a circuits and systems perspective. Pearson Education India, 2015.

[5] IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, New York: IEEE, Inc., Aug. 29, 2008.

[6] IEEE Standard for Floating-Point Arithmetic, pp. 1 58, 2008.

[7] Lang and J. D. Bruguera, "Floating-point fused multiply-add with reduced latency," IEEE Trans. Compute., vol. 53, no. 8, pp. 988–1003, Aug. 2004.

[8] T. Lang and J. D. Bruguera, "Floating-point fused multiply-add with reduced latency," IEEE Trans. Compute., vol. 53, no. 8, pp. 988–1003, Aug. 2004.

[9] E. Hokenek, R. K. Montoye, and P. W. Cook, "Second-generation RISC floating point with multiply-