Middle-East Journal of Scientific Research 22 (11): 1652-1657, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.22.11.21536

Efficient Floating Point Multiplier Implementation via Carry Save Multiplier

Padala Siva Kumar, Anurag Verma,
Chirag Patel and S. Sivanantham

VIT University, Vellore, Tamil Nadu, India

Abstract: The demand for floating point multiplication is increasing day by day since it supports wide range of multimedia applications. So floating point multiplication becomes essential in the domain of image processing, data extraction and digital signal processing. In this proposed paper, a single precision floating point multiplier is designed. The design of integrated chip in term of power, area and speed has a challenging problem now a days. The demand for low power VLSI circuits is increasing day by day. So, the main contribution of this paper is to reduce the power consumption. This has been achieved by designing a carry save multiplier. Here by reducing the number of adders required for partial product addition Carry look ahead adder is used for the addition of exponents in order to reduce the overall timing delay of design.

Key words: IEEE 754 Standard · Carry Save Multiplier · Low power multiplier · Carry look ahead Adder

INTRODUCTION

As integrated circuits (ICs) demand is keep on growing, large numbers of active and passive components are fabricated on the chip. So in case of signal processing and multimedia applications, usage of floating point multiplication is more which consume more power and area due to which the speed is slow. So there is need to reduce power dissipation and area but speed should be increased. The proposed single precision low power floating power multiplier conforms to IEEE 754 standard. In this standard, single precision floating point multiplier number is represented as a 32 bit number consisting of 1 bit for sign, 8 bits for exponent and remaining 23 bits for mantissa representation. The overall explanation of this standard is presented in section(II). In this design, we take two operands of floating point type and unpack these operands in order to multiply mantissas, addition of exponents and xoring of sign bits separately. All these operations execute concurrently using clock invocation. After this, overall mantissa is normalised, truncated to get accurate 32 bit result [1]. The remaining of this paper contains detailed explanation about the floating point standard and the design architecture, functioning modules of the design. Finally results and comparisons are presented [2-5].

IEEE 754 Standard Binary Floating Point Number Representation: Floating point numbers are the real numbers represented in binary format. The floating point numbers used in this design conform to IEEE-754 standard [6]. This standard gives formats and methods for doing floating point operations in computer systems. The formats are single, double, extended and extendable precision. The single precision numbers are 32 bit long, double precision numbers are 64 bits long are as shown in Fig. 1. First we represent the general form of decimal systems then relate this to comparable binary representation,

This number can be represented in Single precision format of IEEE standard as shown. A number is called to be a normalized number, it should consist of ’1’ in the MSB of the significant and exponent is greater than 0 and smaller than 127.

Design Architecture: In the design of floating point multiplier, the arithmetic operations are performed in their respective fields i.e. the 24 bit mantissa part 'M1' of the first number is multiplied with the 24 bit mantissa 'M2' of the second number using carry save multiplier shown in Fig. 2. The 8 bit exponent 'E1' of first number is added to the 8 bit exponent 'E2' of the second number using carry look ahead adder. The sign bit is obtained by XOR

Corresponding Author: Padala Siva Kumar, VIT University, Vellore, Tamil Nadu, India.

Figl(a). Single precision

Figl(b). Double precision

1	110111100	00011101011100010101010

Fig. 1: IEEE Formats

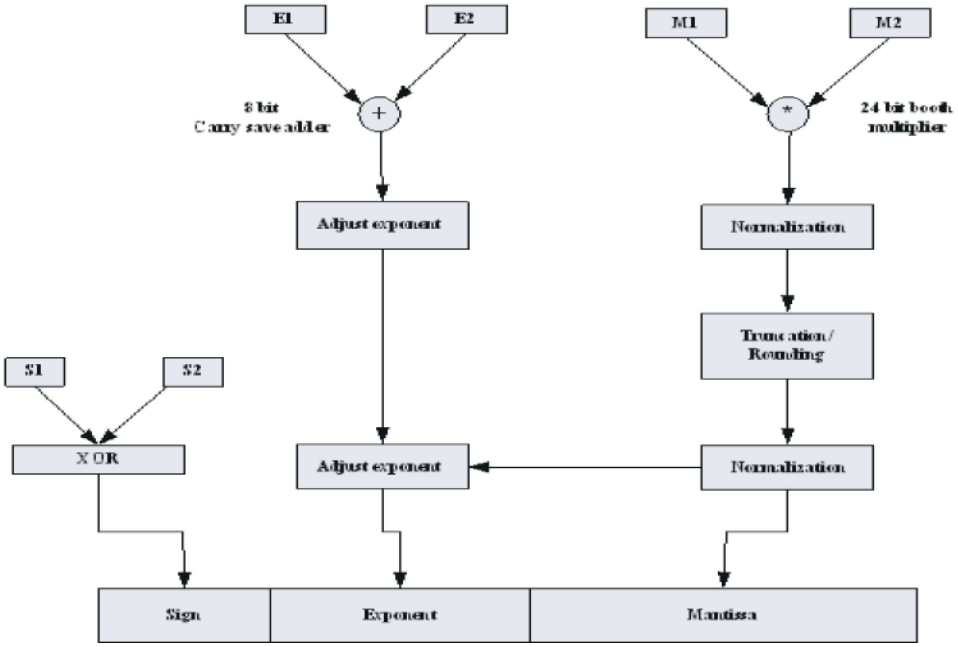


Fig. 2: Architecture

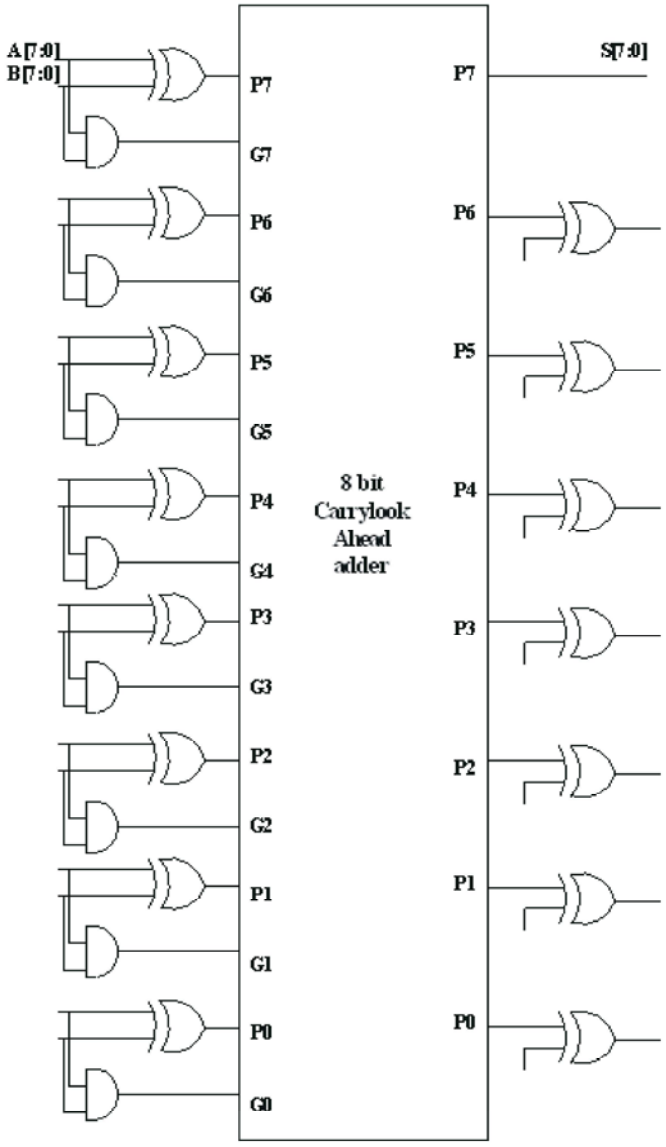


Fig. 3: Carry Look Ahead Adder

operation of respective sign bits of operands. The result obtained from carry save multiplier is normalized and the result of carry look ahead adder is adjusted. This normalized result of carry save multiplier is truncated to 23 bits. Again the normalization on mantissa takes place and exponent is adjusted. The overflow and underflow process is presented. The above architecture is drawn in reference with Ref [1, 4]

Carrylook Ahead Adder: In ripple carry adders for adding two bit numbers requires O(n)time whereas for the carry look ahead adder it requires O(1) shown in Fig. 3. To reduce the computation time, here are faster ways to add two binary numbers by using carry look ahead adders. They work by creating two signals P and G known to be carry propagator and carry generator [2, 3]. The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry regardless of input carry.

The number of gate levels for the carry propagation can be from the circuit of full adder. The signal from input carry Cin to output carry Cout requires AND gate and OR gate, which constitutes two gate levels. Similarly for n-bit parallel adders there are 2n gate levels to propagate through.

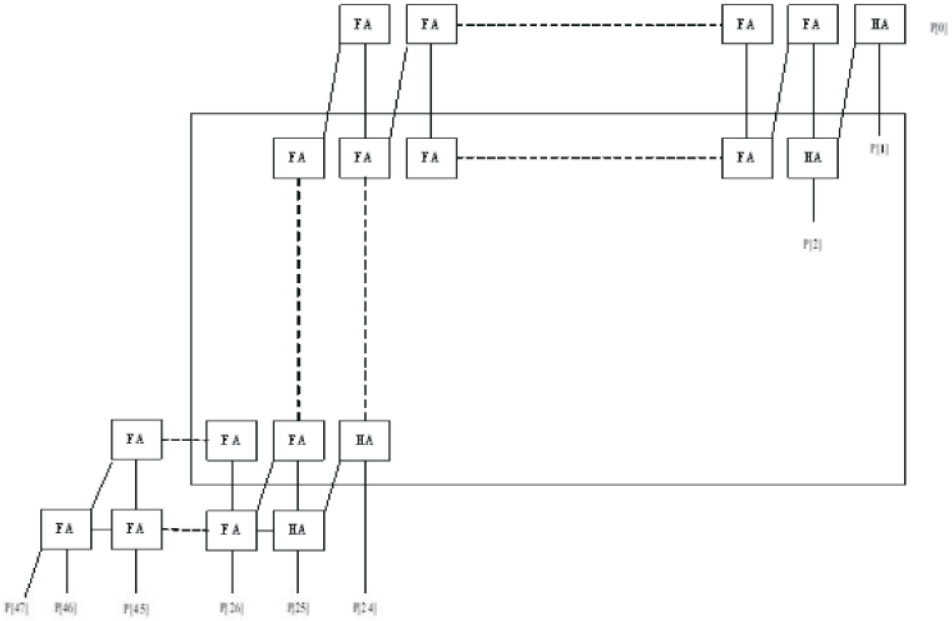


Fig. 4: Carry Save Multiplier

Carry Save Multiplier: In carry save multiplier, we are using 24 half adders and 22x22 full adders shown in Fig. 4. So, it consumes less power when compared to normal multiplication [5], there it requires '2n+1' time levels for 'n' bit operand. By using less number of adders we are reducing much power. The architecture is shown in Fig. 4. If the adder is required to add two numbers and produce a result, carry-save addition is useless, since the result still has to be converted back into binary and this still means that carries have to propagate from right to left. But in large-integer arithmetic addition is a very rare operation and adders are mostly used to accumulate partial sum in a multiplication.

Consider the addition of partial products PP0, PP1and PP2(the same argument is applied to all any three partial products) The final result generated by the array of adders is unchanged if we add carry from adding PP0 to PP1into the adder shown, or that adding in PP2. Each bit of the product is obtained by summing all partial product bits entering that column plus the carry into that column. Note that for an array multiplier, the computation of the sum bit from an adder cell may also be on the critical path. This leads us to the carry save structure.

Now the critical path is like "diagonal " as opposed to a "Manhattan" path, so reducing delay. At the final adder we do however have a row of carry signals still to be added the product. This can be done with final high speed adder.

Normalization: Normalization is required so that there is a '1' before the radix point. When radix point is moved one place to the left, exponent is incremented by '1' but when radix is moved to right by one place, exponent is decremented by '1'.

Rounding and Truncation: The process of reducing the output word length of multiplier so that it can be of a desirable length is known as word length reduction. Basically there are two types of word length reduction namely: right shifting and left shifting (truncation).

The reduction by a 24 bit right shift moves 24 bits from MS side to LS side with extension (sign). The reduction by a 24 bit left shift means we have to mask the output with '111111111111 000000000000'.

Underflow and Overflow: When result exponent is too small, underflow occurs. Whereas, if result's exponent is too large overflow occurs. The exponent size must be 8 bit only and can represent numbers from 1 to 255 otherwise it is not normalized result. Generally overflow occurs due to exponent addition, but that is compensated by subtracting the bias value. An underflow occurs by subtracting the bias to get intermediate exponent. An overflow flag is set to high when overflow occurs and the result comes as +/- infinity. An under flow flag is set to high when underflow occurs and the result come as +/- zero.

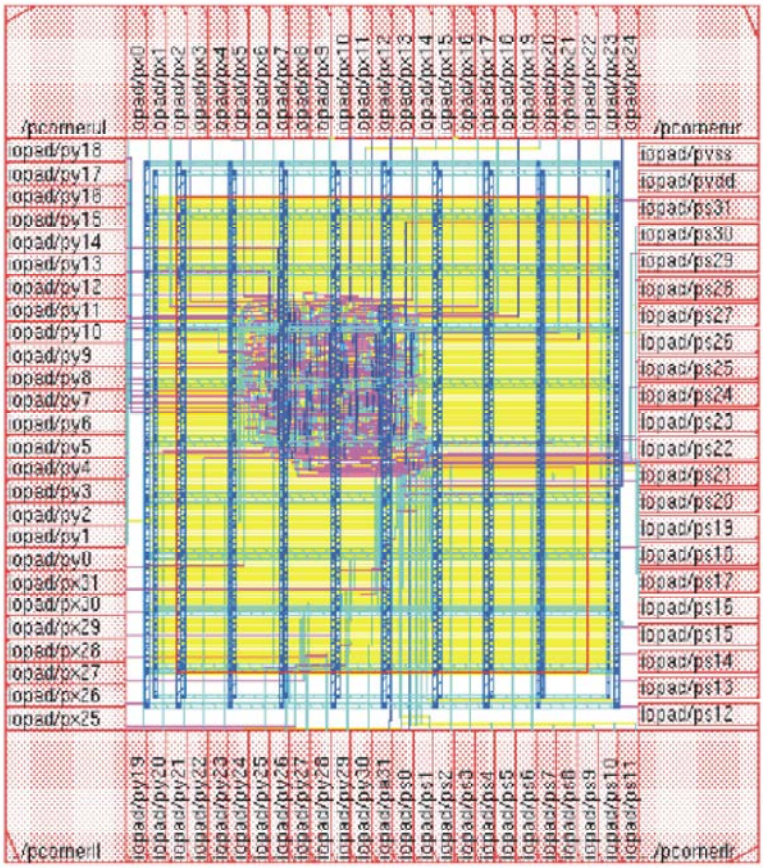
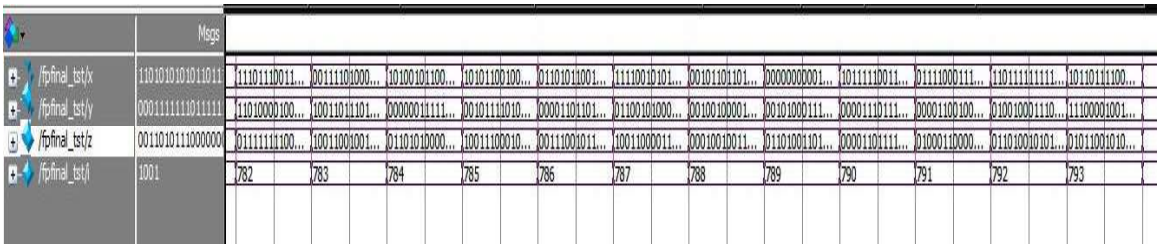


Fig. 5: Physical Design

Physical Design: The verilog module of floating point multiplication using carry save adder is synthesized in Cadence rc. Floor planning, placing and routing is done using Cadence Encounter as shown in Fig. 5.

RESULT AND DISCUSSION

The simulation is done in modelsim and the result is show below.



The table shown below shows the comparison between area and power in different technologies. As the technology is reducing there is significant effect on area and power.

Technology	Area	Power(nw)
180nm	61462	20137521.500
90nm	14318	1563143.988
45nm	4552	615803.059

REFERENCES

1. Crawley, D.G. and G.A.J. Amaratunga, 1988. A standard cell automated layout for a CMOS 50 MHz 8*8 bit pipelined parallel Dadda multiplier, Circuits and Systems, 1988., IEEE International Symposium on, 1: 977-980.
2. Doran, R.W., 1988. Variants of an improved carry look-ahead adder," Computers, IEEE Transactions on, 37(9): 1110-1113.
3. Efstathiou, C., Z. Owda and Y. Tsiatouhas, 2013. New High-Speed Multioutput Carry Look-Ahead Adders, Circuits and Systems II: Express Briefs, IEEE Transactions on, 60(10): 667-671.
4. Jeevan, B., S. Narender, C.V.K. Reddy and K. Sivani, March, 2013. A high speed binary floating point multiplier using Dadda algorithm," Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on, 22-23: 455-460.
5. Raghunath, R.K.J., H. Farrokh, N. Naganathan, M. Rambaud, K. Mondal, F. Masci and M. Hollopeter, 1997. A compact carry-save multiplier architecture and its applications, Circuits and Systems, 1997. Proceedings of the 40th Midwest Symposium on, 2(2): 794-797.
6. IEEE, 1985. Standard for Binary Floating-Point Arithmetic," ANSI/IEEE Std, 1: 754-1.