International Journal of Technology (2014) 1: 22-31
ISSN 2086-9614

IMPROVED FLOATING POINT MULTIPLIER DESIGN BASED ON CANONICAL SIGN DIGIT

P. Saha[1], P. Bhattacharyya[2], A. Dandapat[3*]

[1]*Department of Electronics and Communication Engineering, National Institute of Technology, Shillong, Meghalaya-793003, India*
[2]*Department of Electronics and Telecommunication Engineering, Bengal Engineering and Science University. Shibpur, Howrah-711103, India*
[3]*Department of Electronics and Communication Engineering, National Institute of Technology, Shillong, Meghalaya-793003, India*

(Received: March 2013 / Revised: August 2013 / Accepted: December 2013)

ABSTRACT
Improved floating point (FP) multiplier based on canonical signed digit code (CSDC) has been reported in this paper. Array structure was implemented through Hatamain's scheme of partial product generation along with Baugh-Wooley's (B.W) sign digit multiplication technique. Moreover, CSDC approaches were used for the addition of partial products in constant time without carry propagation and independent of operands. The functionality of these circuits was checked and performance parameters, such as propagation delay, dynamic switching power consumptions were calculated by spice spectre using 90nm CMOS technology. Implementation methodology ensures the stage reduction for floating point multiplier, hence substantial reduction in propagation delay compared with B.W.'s methodology, has been investigated. Implementation result offered propagation delay of the single precision floating point multiplier was only ~14.7ns propagation delay while the power consumption of the same was ~23.7mW. Almost ~40% improvement in speed from earlier reported FP multiplier, e.g. B.W implementation methodology, the best architecture reported so far, has been achieved.

Keywords: Baugh-Wooley (B.W) multiplier; CSD adder; CSD multiplier; High Speed

1. INTRODUCTION
Floating point (FP) multiplier plays a pivotal role in scientific computation for its dynamic range, high precision, and straightforward operating rules (Renxi, 2009). The increasing demand of FP multiplier for the high-precision computation gives a new dimension to the researchers to implement high performance (high speed and low power) processor designs (Zhou 2007; Zhao 2004; Wu 2005).

Generally, multiplication performs the computation P = X×Y. Where X, Y, P signifies multiplicand, multiplier, and product respectively. It is assumed that if both X, Y are 'N' bits, then the values of 'P' must be '2N' bits. A FP number consists of a mantissa (M) and an exponent (e), as shown in Equation (1) (Uya, 1984). There are several ways to represent the sign of the mantissa like as two's-complement representation, sign magnitude representation. The IEEE 754 single precision format is 32 bit wide and uses a 23 bit fraction, an eight bit exponent represented using excess 127, and one bit is used as a sign bit (Koren, 1993).

* Corresponding author's email: anup.dandapat@gmail.com, Tel. +91-3642501225, Fax: +91-3642501113
Permalink/DOI: http://dx.doi.org/10.14716/ijtech.v5i1.150

Thereby, the exponent is treated as a positive number from which a constant is subtracted to form the final exponent (Raafat, 2008). Thus, the mantissa is a normalized binary floating point number followed by the sign bit and does not stored together with the floating point number. The IEEE 754, a standard for floating point numbers (IEEE, 1996), dictates the format presented in Equation (2).

$$X = M\ 2^{e} \tag{1}$$

$$X = (-1)^{S}\ .1.M.2^{e-excess} \tag{2}$$

In general, high-speed FP multipliers, multiplication is carried out by generation of the partial products in parallel format, followed by the addition of these partial products to carry the final results (Hickmann, 2007; Hao, 2005). Addition has been carried out by parallel adders, which has been normalized and rounded to produce the final results. The exponent of the product is the summation of the exponents with proper bias adjustment and increment if the pre-normalized significant requires 1-bit shift for normalization (Quach, 2004). The sign of the product is simply the exclusive-OR (XOR) of the signs of the input operands (Uya, 1984).

In this paper, we report on a floating point multiplier based on CSDC where partial products are generated through the Baugh Wooley's (B.W) (Baugh, 1973) methodology. Generally, multiplication of two 'N' bit sign digit floating numbers generates N×N partial products, thereby N−1 adders are required to sum the partial products, if conventional two input N bit adders are used, since each adder reduces the number of partial products by one (Saha, 2009). Canonical sign digit multipliers have been shown to provide a very efficient method for constant fixed point multiplication by utilization of redundancy of sign digit code (Avizienis, 1961). CSD is a radix-2 signed digit representation for coefficient of the digit set {-1,0,1}. Thus CSD representation permits subtraction as well as addition of shifted data of the partial products, which is generated by the multiplication of two numbers (Takagi 1985). The features of the redundancy in this representation allow a coefficient implementation to be selected which requires a few numbers of adders/subtractors, thus yields a faster multiplier compared to the others (Saha, 2011).

The multiplier is fully optimized, so any configuration of input and output word-lengths could be elaborated. Transistor level implementation has been carried out for calculating the performance parameters like propagation delay, dynamic leakage power, and dynamic switching power consumption. Performance parameters of the implemented method was computed by spice spectre using 90nm standard CMOS technology and compared with the other design like conventional (con) (Wu, 2005) and B.W. (Baugh, 1973) FP multiplier. The calculated results revealed IEEE single precision (32×32) bit FP multiplier have propagation delay only ~14.7nS and consumes ~23.7mW dynamic switching power.

2. METHODOLOGY

2.1. Implementation Algorithm

Let us assume two operands X,Y are in the IEEE 754 format, thus FP multiplication (Renxi 2009) can be represented as:

$Result = P = (-1)^{X_S}(X_M \times 2^{X_e}) \times (-1)^{Y_S}(Y_M \times 2^{Y_e})$ involves the following steps, and implementation procedure is shown in fig. 1. Implementation stages have been given hereunder:

Step 1: If one/both operands are equal to zero, return the result as zero, otherwise:

Step 2: Compute the sign of the result Xs XOR Ys

Step 3: Compute the mantissa of the operands:

- *Multiply the mantissas: Xm × Ym*
- *Round the result to the allowed number of mantissa bits*

Step 4 : Compute the exponent of the result: Result exponent = biased exponent (X) + biased exponent (Y) - bias

Step 5: Normalize if needed, by shifting mantissa right, incrementing result exponent.

Step 6 : Check result exponent for overflow/underflow:

- *If larger than maximum exponent allowed return exponent overflow*
- *If smaller than minimum exponent allowed return exponent underflow*

Generally, in N bit floating point multiplication, N×N partial products are generated first and added them to obtain the product. The partial products may be added by using full adder or full adder with compressors. In the modified algorithm, partial products are added pair-wise by means of CSD adders (Saha, 2011). All intermediate results have been calculated in CSD format and the addition has been implemented through CSD addition. Finally, the product has been converted into binary number.

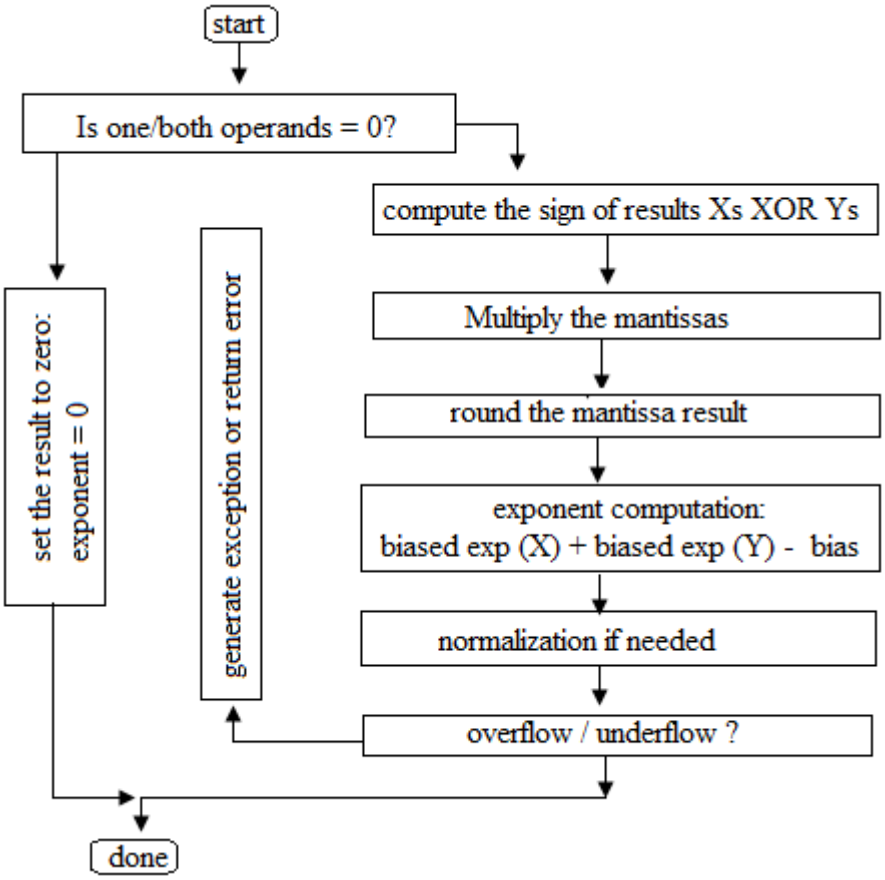

Figure 1 FP multiplication procedure

Improved multiplication algorithm

<Input>

X and Y: Multiplicand and multiplier respectively (Both are N Bits). Both are signed digit floating point numbers.

<Output>

Sum : the products of X and Y

Algorithm

Step 1 : Generate N×N bits partial products using Baugh-Wooley's method

Step 2 : Add the partial products using CSD adders. Perform the additions at each level in the tree in parallel

Step 3 : Convert Addition Results in 2's complement format

Flowchart diagram has been implemented through the algorithm and shown in Figure 2. Input data (multiplicand and multiplier) has been taken as 2's complement format. Transistor level implementation has been carried out for the same architecture.

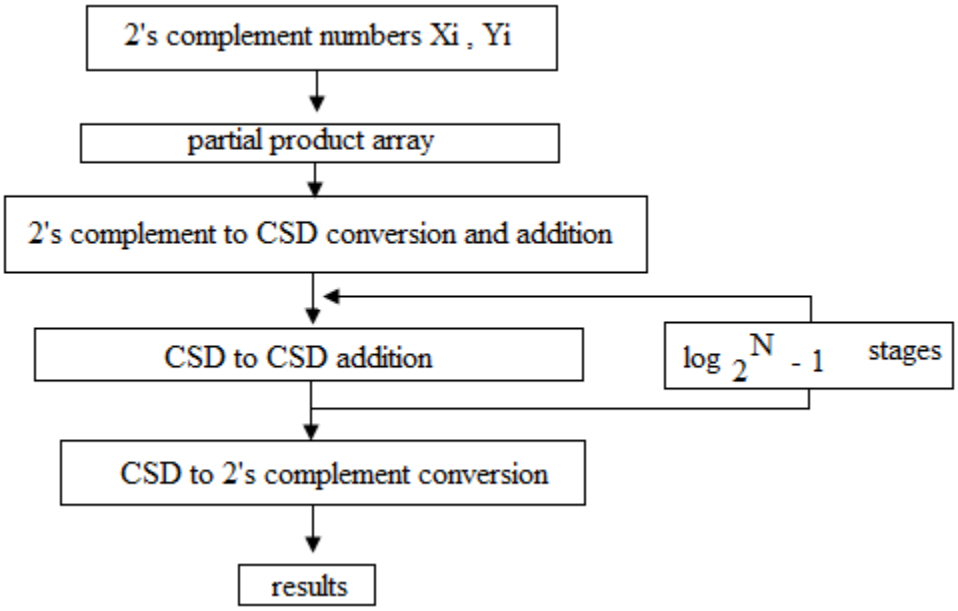

Figure 2 Flowchart diagram of improved multiplication algorithm using CSD

3. CIRCUIT MODULES

The advantages of CMOS transmission gate (TG) logic over conventional CMOS logic are well established. CMOS transmission gate are based on one PMOS and one NMOS, and they are connected in parallel, thereby ON resistance is smaller than even a single NMOS (Uyemura, 2001). The circuit modules required for computation decimal multiplication of two numbers are described in the following subsections, have been implemented using TG.

3.1. Partial Product Generation using Baugh-Wooley's Methodology

In case of signed or 2's complement floating point multiplication the multiplicand and multiplier can be represented as follows. According to the sign magnitude format (Wanhammar 1999), numbers can be represented as

$$X = \left(-x_0 + \sum_{i=1}^{n-1} x_i\, 2^{-i}\right) \text{ and } Y = \left(-y_0 + \sum_{j=1}^{n-1} y_j 2^{-j}\right) \tag{3}$$

Now, the multiplication of X & Y gives:

$$\text{P} = \text{XY} = \left(-x_0 + \sum_{i=1}^{n-1} x_i\, 2^{-i}\right)\left(-y_0 + \sum_{j=1}^{n-1} y_j 2^{-j}\right) \tag{4}$$

$$= x_0\, y_0 + \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} x_i\, y_j 2^{-i-j} - x_0 \sum_{j=1}^{n-1} y_j\, 2^{-j} - y_0 \sum_{i=1}^{n-1} x_i 2^{-i} \tag{5}$$

While generating P by combining the partial products, the sign of the partial products must be taken into consideration. Here $\left(-x_0 \sum_{j=1}^{n-1} y_j\, 2^{-j}\right)$ *and* $(-y_0 \sum_{i=1}^{n-1} x_i 2^{-i})$ are two partial product terms that represents negative quantities. Each of two negative terms can be rewritten as 2's complement form (Wanhammar, 1999). The partial products with negative terms are written at the end rows as shown in Figure 3. The partial product row in Figure 3 (5×5 floating point multiplication) containing the row {1 1 $\overline{x_0y_1}$ $\overline{x_0y_2}$ $\overline{x_0y_3}$ $\overline{x_0y_4}$ 1 1 1 1} is added with '1' to get the 2's complement form of $-x_0 \sum_{j=1}^{n-1} y_j 2^{-j}$. Similarly to get the 2's complement form of $-y_0 \sum_{i=1}^{n-1} x_i 2^{-i}$ the row of the partial product terms {1 1 $\overline{x_1y_0}$ $\overline{x_2y_0}$ $\overline{x_3y_0}$ $\overline{x_4y_0}$ 1 1 1 1} shown in Figure 3, is again added with '1'.

The B.W's algorithm (Baugh, 1973) is relatively straight forward way of doing signed digit floating multiplier. Figure 4 representing the algorithm for a 5-bit case, (for simplicity purpose a 5-bit case has been considered, higher number of bit array can be designed in similar manner) where the partial product bits have been recognized according to Hatamain's scheme (Hatamain 1986). The creation of the partial product array comprises of two steps.

I. The most significant bit (MSB) of the first N-1 partial and all bits of the last partial product rows except its MSB, are inverted.

II. '1' is added to the $(N+1)^{th}$ and $2N^{th}$ column.

The modified architecture has been shown in Figure 4. In this modified diagram, partial products have been added in three stages. Adders and different compressors are used to minimize the stage operations (Saha, 2009). Compressors and adders are used carefully, so that a minimum number of outputs would be generated. As an example, let us consider the column number five where five bits are added at the first time (Saha, 2009). These five bits could be added by using one full adder and a half adder but that will generate four (two from full adder and two from half adder) outputs, instead of this we have used one 5-3 (Saha, 2009) compressor that generates three outputs only which eventually decrease the number of bits for the next stage. The algorithm and architecture described here is only for five bit multiplier. Higher bit multipliers are developed in a similar manner.

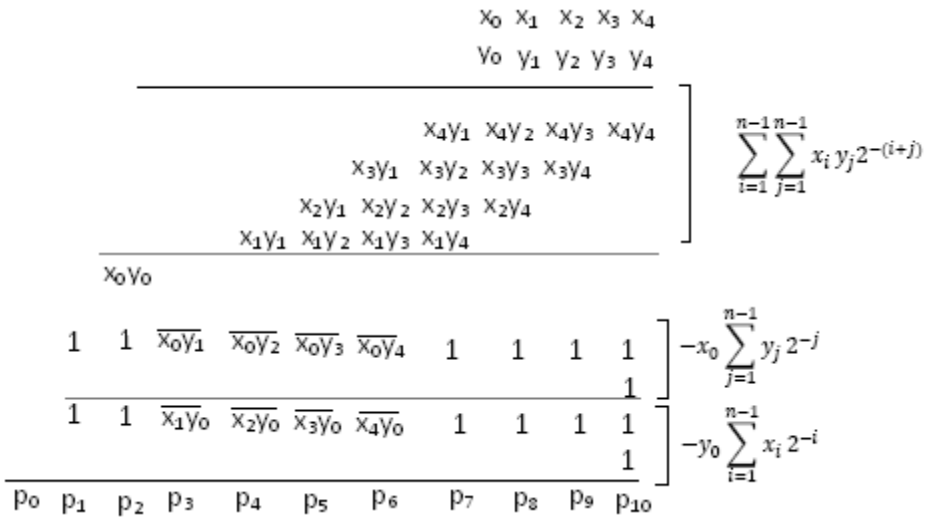

Figure 3 Conventional approach 2's complements floating point multiplication

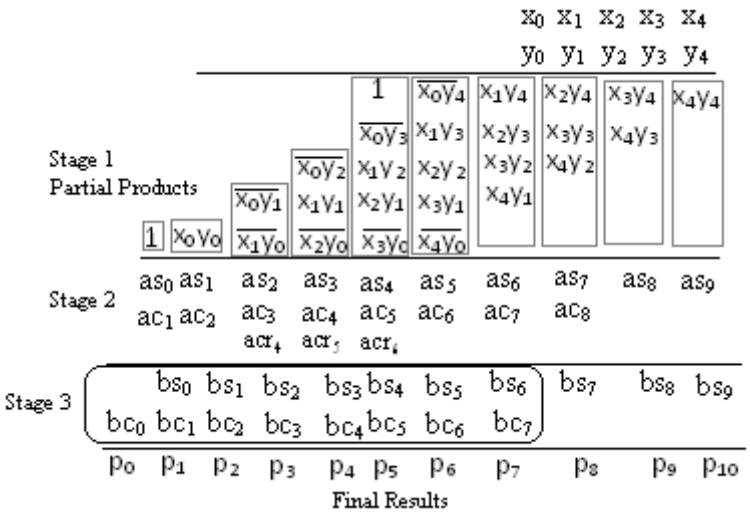

Figure 4 Implementation of B.W. multiplier using adders and compressors

3.2. Canonic Sign Digit (CSD)

The canonical signed digit (CSD) code is a sign digit code with minimal hamming weight, i.e it contains minimum numbers of non zero digits and no adjacent non zero digits (Avizienis 1961). It has a fixed radix 2 and digit set $[\bar{1}, 0, 1]$ where $\bar{1}$ denotes '-1'. An n digit CSD floating point number Y = $[y_0 \ldots\ldots y_{n-1}]_{SD2}$ where ($y_i \in \{\bar{1}, 0, 1\}$) has the value $\sum_{i=0}^{n-1} y_i \times 2^{-i}$ is similar to a binary floating point number except that y_i can be $\bar{1}$ (Das 1996). An n digit 2's complement binary floating point number $[y_0 \ldots\ldots\ldots y_{n-1}]_{\bar{2}}$ where ($y_i \in \{0, 1\}$) and n digit CSD floating point number $[\overline{y_0}\ y_1 \ldots\ldots y_{n-1}]_{CSD2}$ have the same value $-y_0 + \sum_{i=1}^{n-1} y_i 2^{-i}$ where $\overline{y_0}$ is $\bar{1}$ or 0 accordingly as y_0 is 1 or 0.

3.2.1. 2's complement to CSD conversion

Figure 5 represents the flow chart diagram of 2's complement to CSD number conversion. From the definition of CSD, mathematically it can be represented as $x = \sum_{i=0}^{n-1} x_i 2^i$ *where* $(x_i \in \bar{1}, 0, 1)$ (Avizienis, 1961). And two consecutive digits are non

zero (Lim, 1991). Thus, the mathematical expression becomes $x_i x_{i+1} = 0 \quad for\ 1 \leq i \leq n-1$.

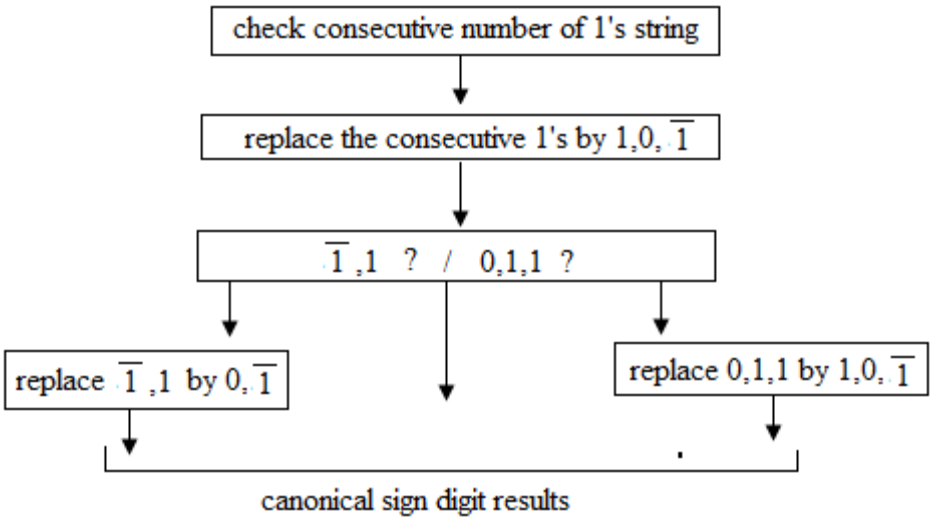

Figure 5 Flow chart diagram of 2's complement floating point number to CSD conversion

The conversion of 2's complement number to CSD code can be implemented through Table 1, where y is defining the 2's complement number and x is the corresponding CSD number representation.

Table 1 Conversion of two's complement number to CSD digit representation

y_{i+2}	y_{i+1}	y_i	x_{i+3}	x_{i+2}	x_{i+1}	x_i
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	1	0	$\bar{1}$
1	0	0	0	1	0	0
1	0	1	0	1	0	1
1	1	0	1	0	$\bar{1}$	0
1	1	1	1	0	0	$\bar{1}$

Circuit module has been carried out via Table 1, and shown in Figure 6. In this architecture y_i, y_{i+1}, y_{i+2} are the present state inputs, and the corresponding outputs are x_i, x_{i+1}, x_{i+2}, x_{i+3}. Si_x_i, Si_x_{i+1}, Si_x_{i+2},Si_x_{i+3} are representing the corresponding sign bits of the outputs.

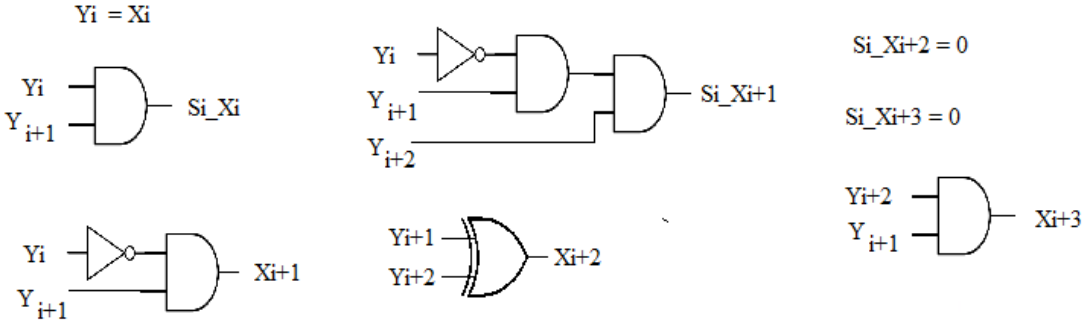

Figure 6 Hardware implementation of 2's complements floating point number to CSD conversion.

3.2.2. CSD addition

Carry propagation free CSD addition is performed in two steps.

I. Determining the intermediate carry { $C_i \epsilon (\bar{1}, 0, 1)$} and intermediate sum digits { $S_i \epsilon (\bar{1}, 0, 1)$, satisfying the condition $x_i + y_i = z_i + C_{i-1}$, where x_{i+1} and y_{i+1} are the augend and adding digits respectively.

II. Obtaining the sum digits {$Z_i \epsilon (\bar{1}, 0, 1)$} at each position by the addition of intermediate sum digits S_i and C_i from the next lower order positions.

Boolean expressions have been implemented through the above mentioned steps is shown in the following expressions (6-9) (Saha, 2011; Lim, 1991). Here z_i, C_{i-1} are representing the intermediate sum and intermediate carry. $Signx_i$ and $Signy_i$ are representing sign magnitude of x_i and y_i respectively. $SignC_{i-1}$ and $Signz_i$ are representing the sign magnitude of intermediate carry and intermediate sum respectively.

$$z_i = x_i \oplus y_i \tag{6}$$

$$C_{i-1} = \left((x_i\, y_i)\left(\overline{x_ı \oplus y_ı}\right) + (Signx_{i+1} + Signy_{i+1})(x_i \oplus y_i)\right) \oplus (Signx_i \oplus Signy_i) \tag{7}$$

$$Signz_i = z_i(\overline{Signx_{ı+1} + Signy_{ı+1}}) \tag{8}$$

$$SignC_{i-1} = \left(\overline{x_ı \oplus y_ı}\right)(Signx_i\, Signy_i) + (x_i \oplus y_i)(Signx_{i+1} + Signy_{i+1}) \tag{9}$$

4. RESULTS AND DISCUSSION

Transistor level simulation of the reported FP multiplier circuitry was performed through spice spectre simulator using 90nm CMOS technology with 1V power supply, operated at 25MHz. Dual threshold voltage(VT) (Uyemura, 2001) operating mode was considered for simulation to determine the performance parameters. In designing calculation of FP multiplier like (4×4), (8×8), (16×16) and (32×32) bits, all the individual modules such partial product generation using B.W's methodology, we focused our main concentration for reducing the propagation delay and dynamic switching power consumption.

For the comparison point of view, the ideas have been considered form the references and simulated, and performance parameters was computed using the same MOSFET technology file. Input data was taken in a regular fashion for experimental purpose. The delay and the power measured using the worst-case pattern and from the output where the delay is maximum, and shown in Figure 7. From Figure 7, it is also observed that the proposed design offered ~52%, ~42% improvement in propagation delay while corresponding reduction of power consumption are ~36%, ~26% for the (32×32) bit FP multiplication circuitry in comparison with conventional (Wu, 2005) and B.W (Baugh, 1973) based implementation respectively.

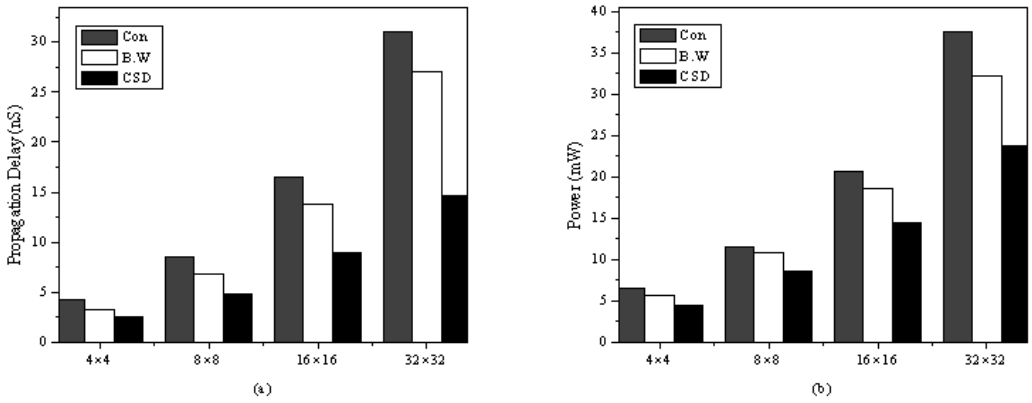

Figure 7 Performance parameters such as (a) propagation delay (ns); (b) dynamic switching power (mW) consumption of the different architectures, as a function of input number of bits, which have been implemented by Spice Spectre using 90nm CMOS technology file

5. CONCLUSION

In this paper, we report on a single precision high speed FP multiplier based on CSD, which is highly suitable for VLSI implementation. Array structure has been implemented through Hatamain's scheme of partial product generation along with Baugh-Wooley's (B.W) sign digit multiplication technique. The implementation methodology ensures stage reduction, leading to substantial reduction of the propagation delay and power. Transistor level simulation for FP multiplier circuit was performed through Cadence Spice Spectre simulator using 90nm CMOS technology. Implementation methodology offered ~52%, ~42% improvement in propagation delay while corresponding reduction of power consumption are ~36%, ~26% for the (32×32) bit FP multiplication circuitry in comparison with conventional and B.W implementation techniques respectively.

6. REFERENCES

Avizienis, A., 1961. Signed-digit Number Representations for Fast Parallel Arithmetic. *IRE Transaction on Electronics Computer*, Volume EC-10, pp. 389–400

Baugh, C.R., Wooley, B.A., 1973. A Two's Complement Parallel Array Multiplication Algorithm. *IEEE Transaction on Computer*, Volume C-22, pp. 1045–1047

Das, S.K., Pinotti, M.C., 1996. Fast VLSI Circuits for CSD Coding and GNAF coding. *Electronics Letters*, Volume 32, pp. 632–634

Hao, Z.-G., Zeng, X.-J., Li, G.-K., 2005. The CMOS Circuit Design of a High-Speed Floating-Point Multiplier. *Computer Engineering & Science*,Volume 21, pp.54–57

Hatamian, M., 1986. A 70-MHz 8-bit × 8-bit Parallel Pipelined Multiplier in 2.5-μm CMOS. *IEEE Journal on Solid-State Circuits*, Volume 21, pp. 505–513

Hickmann, B., Krioukov, A., Schulte, M., 2007. A Parallel IEEE P754 Decimal Floating-Point Multiplier. *Proceedings of IEEE 25th International conference on computer design*. Lake Tahoe, CA, 7-10th Oct, pp. 296–303

IEEE Standard 754 for Binary Floating-Point Arithmetic, 1996

Koren, I., 1993. *Computer Arithmetic Algorithms*, Prentice Hall

Lim, Y.C., Evans, J.B., Liu, B., 1991. Decomposition of Binary Integers into Signed Power-of-two Terms. *IEEE Transaction on Circuits and Systems*, Volume 38, pp. 667–672

Quach, N.T., Takagi,N., Flynn, M.J., 2004. Systematic IEEE Rounding Method for High-Speed Floating-Point Multipliers. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Volume 12, pp. 511–521

Raafat, R., Amira, M., Majeed, A., Samy R., 2008. A Decimal Fully Parallel and Pipelined Floating Point Multiplier. *Proceedings of IEEE 42nd Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 26-29th Oct, pp. 1800–1804

Renxi, G., Shangjun Z., Hainan Z., Xiaobi M.,Wenying G., Lingling X.,Yang, H., 2009. Hardware Implementation of a High Speed Floating Point Multiplier based on FPGA. *Proceedings of IEEE 4th International Conference on Computer Science & Education*, Nanning, 25-28th July, pp. 1902–1906

Saha, P., Banerjee, A., Dandapat, A., Bhattacharyya, P., 2011. ASIC Implementation of High Speed Processor for Calculating Discrete Fourier Transformation using Circular Convolution Technique. *International Journal of World Scientific and Engineering Academy and Society (WSEAS)*, Volume 10, pp. 278–288

Saha, P.K, Banerjee, A., Dandapat, A., 2009. High Speed Low Power Complex Multiplier Design using Parallel Adders and Subtractors. *International Journal on Electronic and Electrical Engineering, (IJEEE)*,Volume 07, pp. 38–46

Takagi, N., Yasuura, H., Yajima, S., 1985. High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree. *IEEE Transaction on Computer*, Volume C-34, pp. 789–796

Uya, M., Kaneko, K., Yasui, J., 1984. A CMOS Floating Point Multiplier. *IEEE Journal of Solid State Circuits*, Volume Sc-9, pp. 697–702

Uyemura, J.P., 2001. *CMOS Logic Circuit Design*, Kluwer Academic Publishers.

Wanhammar, L., 1999. *DSP Integrated Circuits*" PP- 479, Academic Press.

Wu, J., Ying, Z., 2005. Design of High Speed Floating-multiplier, *Journal of Circuits and Systems*, Volume 10, pp.6–11

Zhao, Z.-W., Chen, H., Han, Y.-Q., 2004. Design of High-performance 32-bit Floating-point Multipliers for ASIC. *Systems Engineering and Electronics*, Volume 26, pp.531–534

Zhou, D.-J., Sun, Feng., Yu, Z.-G., 2007. Design of a 32-bit High-Speed Floating-Point Multiplier. *Semiconductor Technology*, Volume 20, pp.871–874