IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Design of a 16-Bit Posit Multiplier With Power Efficiency

Monisha.R, Sivasakthi.V

Narasu's Sarathy Institute Of Technology
Salem

***Abstract*—Considering all the arithmetic operations known, multiplication is one of the most commonly used operation in various applications. Most notable applications are signal processing, graphics and scientific computation. Posit number system is widely used nowadays in replacement for IEEE floating point number system. In this paper, we propose the architecture of a 16 –bit modified posit multiplier, which results in less power consumption. In the posit multiplier, our work relates to the mantissa multiplier. The mantissa multiplier is designed for maximum possible bit- width, where the entire multiplier is divided into smaller ones. Only the necessary smaller multipliers are used during run time. Thus resulting in less power consumption.**

***Keywords*—Posit multiplier, mantissa multiplier, low power circuit.**

I. INTRODUCTION

Posit, a new datatype designed as a direct replacement for IEEE Standard 754 floating-point numbers (floats). Unlike earlier forms of universal number (unum) arithmetic, posits do not require interval arithmetic or variable size operands; like floats, they round if an answer is inexact. They provide compelling advantages over floats, including larger dynamic range, higher accuracy, better closure, bitwise identical results across systems, simpler hardware, and simpler exception handling. Posits never overflow to infinity or underflow to zero, and "Not-a-Number" (NaN) indicates an action instead of a bit pattern. A posit processing unit takes less circuitry than an IEEE float FPU In addition, its non-uniformed data distribution fits well with the data distribution of some applications, such as deep learning. The 8-bit or 16-bit posit formats are widely used in deep learning systems. The 32-it posit format is used in some scientific computation applications to take the place of the standard 64-bit floating-point format.

The general format for the posit number system is shown in fig 1. A posit number Posit(*nb*,*es*) is defined with the total bit-width *nb* and the exponent bit-width *es*. It has four components: sign (*s*),
regime (*rg*), exponent (*exp*), and mantissa (*frac*). The component bit-width is not constant. The regime bit-width varies for different values. The exponent and the mantissa will occupy the remaining bit positions and they will not be included in the format when the regime occupies all bit positions. The value of a number represented in posit format is:

$$value = (-1)s \times useedrg \times 2exp \times (1 + frac) \quad (1)$$

where $useed = 2^{2^{es}}$.

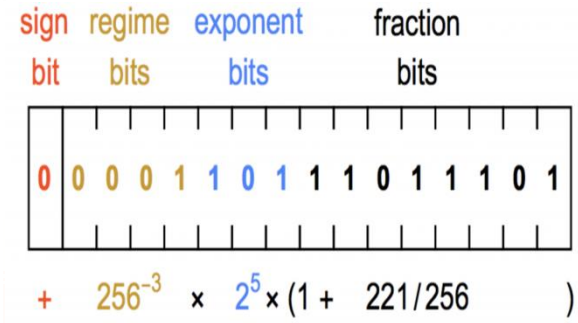

Fig 1. Posit number system

Unlike the conventional floating-point format, the bit-width of each component in posit format, as shown in Fig. 1, is dynamic (except the 1-bit sign). The sign and the regime always appear in the format. The remaining exponent and fraction part appears only when the sign and regime do not occupy all the bit positions. Therefore, the mantissa (including the implicit bit) bit-width can be from 1-bit to ($nb - es$)-bit, where *nb* is the total bit-width of posit format and *es* is the exponent bit-width. As the actual mantissa bit-width is not always the maximum value, the mantissa does not always require a ($nb - es$)-bit multiplier. When using ($nb - es$)-bit multiplier for small bit-width mantissa, power or energy is wasted.

In this paper, we establish the successful implementation of a 16 – bit posit multiplier, where the mantissa (fraction) multiplier is split into several small multipliers. Only the required multipliers are used at the run time. Thus we efficiently design a low power consuming posit multiplier architecture to improve power efficiency. This architecture can also be used in various multipliers other than posit multipliers to improve power efficiency of the corresponding component or device.

The upcoming sections are as follows: Section II describes the existing multiplier method. Then the proposed method is briefed in Section III. In the Section IV, we present the sample simulation results of the proposed multiplier. Lastly Section V concludes the brief.

II. EXISTING METHOD

Currently the working method of posit multiplier uses the modified booth's algorithm technique. This method is also known as bit-pair algorithm or radix-4 algorithm. It is possible to reduce the number of partial products by half in this method. the basic idea is that, instead of shifting and adding for every column of the multiplier term and multiplying by 1 or 0, we only take every second column and multiply by ±1,±2, or 0, to obtain the same

results.radix-4 booth encoder performs the process of encoding the multiplicand based on multiplier bits. It will compare 3 bits at a time with overlapping technique. Grouping starts from the LSB , and the first block only uses 2 bits from the multiplier and assumes a 0 for the third bit as shown in fig 2.

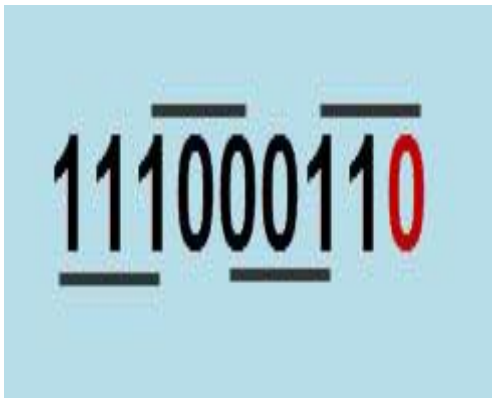

Fig 2. 3-bit pairing for Booth recoding

The functional operation of Radix-4 booth encoder is shown in the Table.1. It consists of eight different types of states and during these states we can obtain the outcomes, which are multiplication of multiplicand with 0,-1 and -2 consecutively.

Multiplier Bits Block			Recoded 1-bit pair		2 bit booth	
i+1	i	i-1	i+1	i	Multiplier Value	Partial Product
0	0	0	0	0	0	Mx0
0	0	1	0	1	1	Mx1
0	1	0	1	-1	1	Mx1
0	1	0	1	0	2	Mx2
1	0	0	-1	0	-2	Mx-2
1	0	1	-1	1	-1	Mx-1
1	1	0	0	-1	-1	Mx-1
1	1	0	0	0	0	Mx0

Table 1. Booth recoding table for radix-4 method

Radix-4 Booth algorithm is given below: (1) Extend the sign bit 1 position if necessary to ensure that n is even. (2) Append a 0 to the right of the LSB of the multiplier. (3) According to the value of each vector, each Partial Product will be 0, +y, –y, +2y or –2y. The negative values of y are made by taking the 2's complement. The negative values of y are made by taking the 2's complement and Carry-look-ahead (CLA) fast adders are used for addition in this paper. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing n-bit parallel multipliers, only n/2 partial products are generated. The advantage of this method is the halving of the number of partial products. This is important in circuit design as it relates to the propagation delay in the running of the circuit, and the complexity and power consumption of its implementation. The main disadvantage of the circuit is the complexity of the circuit to generate partial product bit in the booth encoding.

III. PROPOSED METHOD

In the proposed posit multiplier, the design differs from the existing method in the mantissa multiplier. The mantissa multiplier uses the method of radix-4 booth multiplier. It is a (nb - es) bit mantissa multiplier.

While doing multiplication, we do not always require a maximum bit-width mantissa multiplier. That is, there is no need to always use a (nb - es) mantissa multiplier. Generally, the unused bits in the multiplier and multiplicand will be assigned to zero normally. But those bits will be reverted to value one during recoding the multiplier when it is negative. Thus it results in an unwanted signal toggling. The unnecessary signal toggling must be avoided to reduce power consumption. Though we use the same radix-4 booth multiplication in the proposed system, the power efficiency can be achieved by splitting the multipliers into smaller ones and accessing or controlling it through a control signal. Such that this design also involves generating a control signal to enable the required smaller multiplier component when required during our run time. The design details of 16-bit multiplier is discussed in detail.

A. REDUCTION OF MULTIPLIER

Below given fig 3 shows the steps performed in the multiplier after the partial product generation. From this picture it is clear, that the number of bits used to generate an answer for 16-bit multiplication is very time, space and also memory consuming. Since we multiply all the 16 bit of the multiplier and multiplicand, an annoyingly long process takes place.

Thus in the proposed system we have decided to divide or split the multiplier digits.

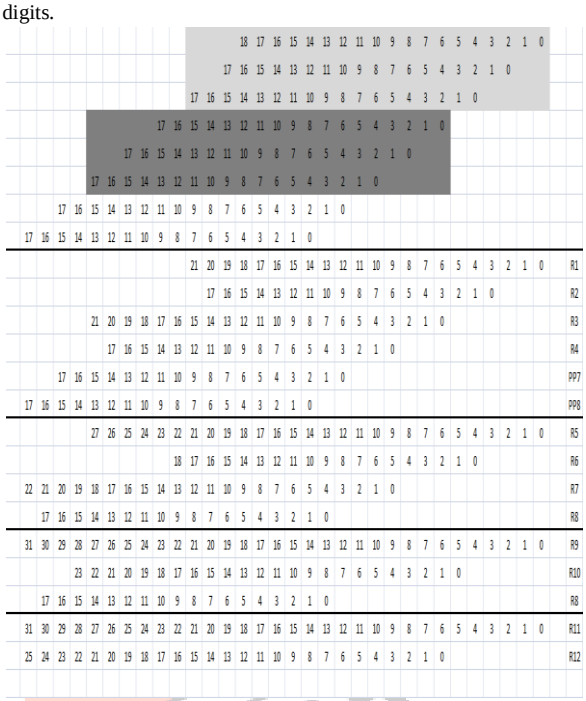

Fig 3. Partial product addition after multiplying two 16-bit numbers in the existing method.

In the proposed 15-bit mantissa multiplier. eight partial products are generated while using radix-4 booth algorithm. Each partial product is of 16 bits. In the proposed design, the 15-bit multiplier is divided into 4 groups: the MSB 3-bit forms one group, and the remaining 12-bit can be grouped into three 4-bit groups. Correspondingly, the eight partial products are divided into 4 groups, *PPH*_1, *PPH*_2, *PPH*_3 and *PPH*_4. If the multiplier is less than 3-bit, then only the two partial products in *PPH*_4 are generated while all others are set to zero. If the multiplier is more than 3-bit but less than 7-bit, then partial products in *PPH*_3 and *PPH*_4 are generated. If the multiplier is more than 7-bit but less than 11-bit, then partial products in PP*H*_2, *PPH*_3, and *PPH*_4 are generated. Finally, if the multiplier is more than 11-bit, then all the partial products are generated. Similarly, the 15-bit multiplicand is also divided into 4 groups. Each partial product is correspondingly divided into4 groups, *PPV*_1, *PPV*_2, *PPV*_3, and *PPV*_4. If the multiplicand is less than 3-bit, then only *PPV*_4 is generated while all others are set to zeros. If the multiplicand is more than 3-bit but less than 7-bit, then *PPV*_3 and *PPV*_4 are generated. If the multiplicand is more than 7-bit but less than

11-bit, then *PPV*_2, *PPV*_3, and *PPV*_4 are generated. Finally, if the multiplicand is more than 11-bit, then all bits in the partial product are generated.

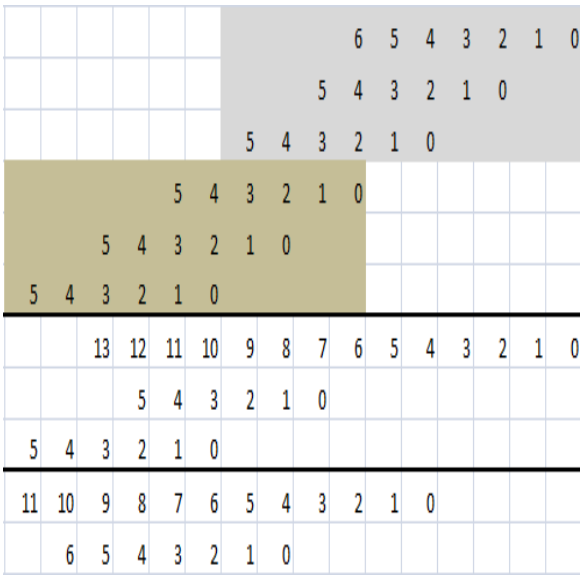

Fig 4. Reduced partial products bit in proposed system

Fig 4 shows an example when both the multiplier and multiplicand are less than 7-bit.

B. CONTROL SIGNAL

To design the proposed multiplier described in Section III-A, a control signal is required to select among the four horizontal regions and four vertical regions. During the component extraction of posit multiplier, the bit-width of the regime *shift_rg* is generated. This signal can be used to calculate the actual mantissa bit-width (including the implicit bit):

$$mant\_bit = nb - es - shift\_rg \quad (2)$$

By using the above equation, the control signal for both multiplicand and multiplier can be generated. Since four regions are used for multiplicand and multiplier, respectively, the control signal *ctl* for each will be 2-bit. When *ctl* is 00, only *PPH*_4 (or *PPV*_4)
is used. When *ctl* is 01, *PPH*_3 and *PPH*_4
(or *PPV*_3 and *PPV*_4) are used. When *ctl* is 10, three regions are used and all regions are used when *ctl* is 11. Regime is at least 2-bit so we care less about the cases of *shift_rg* = 0000 or *shift_rg* = 0001. When *shift_rg* = 1111, there is no space for exponent and mantissa in the format, however, there is still 1-bit implicit bit. The control signal can be generated as:

$$ctl[1] = shift\_rg[3] \text{ and } ctl[0] = shift\_rg[2] \quad (3)$$

In the proposed design, separate control signals for multiplicand and multiplier can be generated and used in the mantissa multiplier. Corresponding to 4 horizontal regions and 4 vertical regions, a total of 16 Booth-2 partial product generation (PPG) modules are implemented. One PPG is enabled only when both the corresponding horizontal control and vertical
control are enabled. In addition to these control signals, extra control to manage the partial product extension bits (*S* bits) is also required.

IV. SIMULATION RESULTS

The entire simulation and result obtaining using test vectors are done with the help of ModelSim software. We have used the ModelSim 6.3f version for its ease in finding errors.

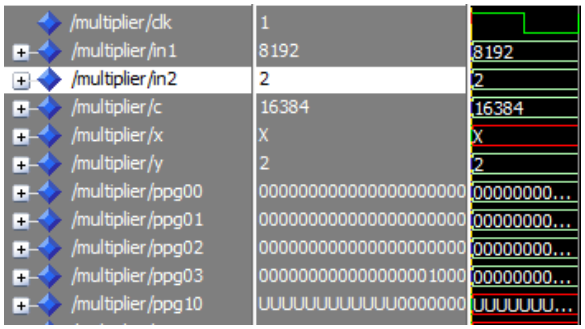

Fig 5. Sample output 1 after simulation

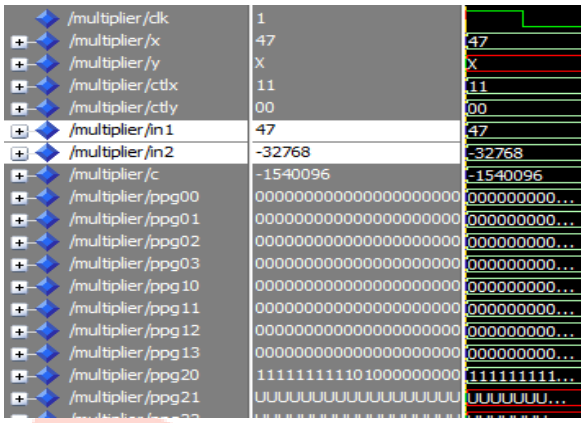

Fig 6. Sample output 2 after simulation

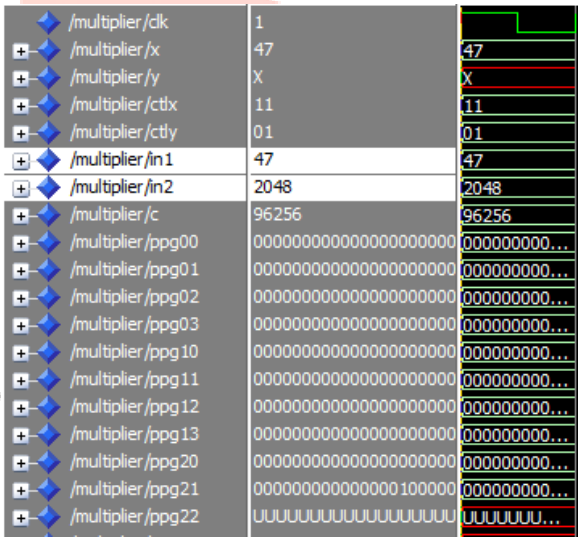

Fig 7. Sample output 3 after simulation

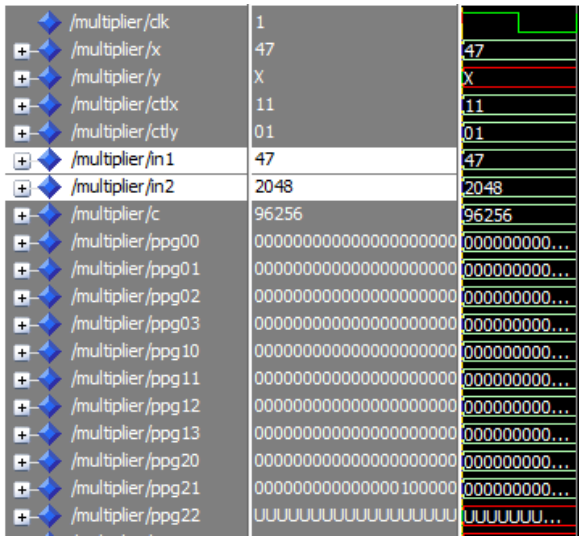

Fig 8. Sample output 4 after simulation

In the above presented figures 5 to 8, we have shown the simulation results after inputting test vectors. For the proposed design of Posit(16,1), we have achieved more than 20% power reduction. The proposed method divides the mantissa multiplier into small portions and only the required portions are enabled

at run-time. Other portions are disabled to avoid signal toggling. In addition, due to the unused portions are always set to zeros, the toggling in later stage addition and rounding is also avoided. All of these lead to the power reduction of the proposed designs. The proposed power reduction method can be effectively applied in those applications to achieve power efficient posit computation.

V. CONCLUSION

In this project, we have proposed the idea of 16-bit Posit multiplier architecture with power efficiency. Intrigued by the idea of dividing the mantissa multiplier in small portions, because of the fact that the whole mantissa multiplier is not used entirely all the time. At run-time, only the required portions are enabled to avoid unnecessary signal toggling to reduce the power consumption. The proposed method is evaluated and an average of 16% power reduction can be achieved. The proposed method is evaluated for 16-bit multiplier, whereas we can extend the work for 8-bit and 32-bit posit multipliers using the same technique. In the coming future, more power reduction techniques for multiplier architecture will be developed. The work need not be necessarily limited to multipliers alone. Future works can be extended for Posit adder or Posit multiply-accumulate unit.

VI. REFERENCES

[1] Hao Zhang and Seok-Bum Ko , "Design of Power Efficient Posit Multiplier," IEEE on Circuits and Systems—ii: express briefs, vol. 67, no. 5, May 2020.Transactions

[2] J. L. Gustafson and I. Yonemoto, "Beating floating point at its own game: Posit arithmetic," Supercomput. Front. Innovat. Int. J., vol. 4, no. 2, pp. 71–86, Jun. 2017.

[3] Sneha Singh and Prachi Singh, "Implementation of Radix-4 Booth Multiplier by VHDL," IJRREEE, vol 4, issue 1, pp 1-11, Jan 2017.

[4] Sukhmeet Kaur, Suman and Manpreet Singh Manna, "Implementation of Modified Booth Algorithm(radix-4) and its comparison with Booth Algorithm(radix-2)," Advances Electric and Electronic Engineering, ISSN 2231-1297, vol 3, no 6,pp 683-690, 2013.

[5] Bikash Chandra Sahoo, Sanjay Kumar Samant, "Design and Power Estimation of Booth Multiplier Using Different Adder Architectures," 2013.

[6] S.Shafiulla Basha, Syed, Jahangir Badashah, "Design and Implementation of Radix-4 Based High Speed Multiplier for ALU's using Minimal Partial Products,"IJAET, ISSN 2231-1963,vol 4, issue 1, pp 314-325.