Vamsi Krishna Pedarla, P.Nagaraju / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp. 349-353

# Modified Covalent Redundant Binary Booth Encoding For Fast 64*64 Multiplier Design

**Vamsi Krishna Pedarla (M.Tech),**
**P.Nagaraju (M.Tech)**
Dept .Of Electronics and Communication Engineering
Nova College of Engineering & Technology
Jangareddy Gudem-534447

## Abstract

In this paper introducing a novel technique so called redundant binary booth algorithm.The redundant binary in design of high speed digital multiplier is beneficial due to high modularity and carry free addition. Generally,in high radix modified booth encoding algorithm the partial products are reduced in multiplication process. But it yields complexity in producing in generation of hard multiples.Therefore booth encoding scheme along with redundant binary scheme solves this problem by using booth encoding,RB partial product generator,RB partial product accumulator,RB to NB converter stage.In this paper implemented in VHDL.

## INTRODUCTION

**T**he digital multiplier is a ubiquitous arithmetic unit in microprocessors, digital signal processors, and emerging media processors. It is also a kernel operator in application- specific data path of video and audio codecs, digital filters, computer graphics, and embedded systems. Compared with many other arithmetic operations, multiplication is time-consuming and power hungry. The critical paths dominated by digital multipliers often impose a speed limit on the entire design. Hence, VLSI design of high-speed multipliers, with low energy dissipation, is still a popular research. Redundant binary (RB) representation is one of the signed digit representations first introduced by Avizienis in 1961 for fast parallel arithmetic. This new arithmetic was applied for fast multiplication by Takagi et al and implemented in VLSI by Edamatsu In conventional RB multiplier design, a modified Booth encoding algorithm in NB regime is employed to reduce the number of partial products, and then pairs of NB partial products are encoded to form RB partial products. In this process, an additional constant binary vector is introduced to compensate for the aggregate errors resulting from both the RB and Booth encodings. This correction vector incurs hardware overhead in the RB summing tree and, to a certain extent, offsets the regularity of the layout and increases switching activities.

RB multipliers for power-of-two operand length.

## Redundant Binary representation:

A redundant binary representation (RBR) is a numeral system that uses more bits than needed to represent a single binary digit so that most numbers have several representations. RBR is unlike usual binary numeral systems, including two's complement, which use a single bit for each digit. Many of RBR's properties differ from those of regular binary representation systems. Most importantly, RBR allows addition without using a typical carry. When compared to non-redundant representation, RBR makes bitwise logical operation slower, but Arithmetic operation are faster when large bit width are used. Usually, every bit has a sign that is not necessarily the same as the sign of the number represented. When digits have signs, the RBR is also a signed-digit representation.

RBR is a place-value notation system. In RBR, digits are pairs of bits, that is, for every place; RBR uses a pair of bits. The value represented by an RBR digit can be found using a translation table. This table indicates the mathematical value of each possible pair of bits. As in conventional binary representation, the integer value of a given representation is a weighted sum of the values of the digits. The weight starts at 1 for the rightmost position and goes up by a factor of 2 for each next position. Usually, RBR allows negative values. There is no single sign bit that tells if a RBR represented number is positive or negative. Most integers have several possible representations in an RBR.

An integer value can be converted back from RBR using the following formula, where *n* is the number of digit and $d_k$ is the interpreted value of the *k*-th digit, where *k* starts at 0 at the rightmost position:

$$\sum_{k=0}^{n-1} d_k 2^k$$

The conversion from RBR to two's complement can be done in O(log(n)) using prefix adder where *n* is the number of digit.

## RB Booth Encoding (RBBE):

Hard multiples could be obtained from the differences of two simple power-of-two multiples. In radix-16 RB Booth encoding, the multiplier bits are $y_{4i+3}y_{4i+2}y_{4i+1}$ $y_{4i}y_{4i-1}$ and each of the original hard multiples selected by $\pm 3M, \pm 6M$ and $\pm 7M$ are replaced by $\pm(4X - 1X), \pm(8X - 2X)$ and $\pm(8X - 1X)$, respectively. The partial products generated in this way conform to the format of the RB coding. The only exception is that, hard multiples selected by

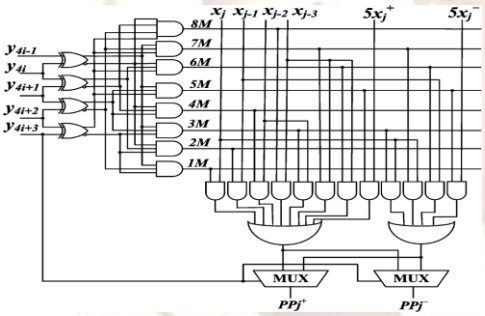



**Figure 1.1: Radix-16 RBBE and its partial product generator.**

$\pm 5M$ Cannot be readily generated in this manner, a simple carry-free RB adder is used to add 4X and X. The advantage of this method is the correction vector due to the two's complement arithmetic and the RB coding has been completely eliminated. Comparing with NBBE, the ease of generating the hard multiples by RBBE, to a certain extent has been offset by its complex circuitry.

High-radix RBBE requires high fan-in gates in the partial product generator circuit. Since the circuit for each digit of the RB partial product will be duplicated in a large number, the overhead of high fan-in gates is more prominent in long operand length multipliers. Besides, as

only one Booth encoded digit is consumed for one RB partial product, half of the binary bits representing an RB partial product generated from a simple power-of-two multiple in the RBBE are filled with "0"s, which is rather inefficient.[6]

| Multiplier Bits | Multiple | | Multiplier Bits | Multiple | |
|---|---|---|---|---|---|
| | $+M$ | $-M$ | | $+M$ | $-M$ |
| 0 0 0 0 (0) | 0 | 0 | 1 0 0 0 (0) | 0 | $8M$ |
| 0 0 0 0 (1) | $M$ | 0 | 1 0 0 0 (1) | $M$ | $8M$ |
| 0 0 0 1 (0) | $M$ | 0 | 1 0 0 1 (0) | $M$ | $8M$ |
| 0 0 0 1 (1) | $2M$ | 0 | 1 0 0 1 (1) | $2M$ | $8M$ |
| 0 0 1 0 (0) | $2M$ | 0 | 1 0 1 0 (0) | $2M$ | $8M$ |
| 0 0 1 0 (1) | $4M$ | $M$ | 1 0 1 0 (1) | 0 | $5M$ * |
| 0 0 1 1 (0) | $4M$ | $M$ | 1 0 1 1 (0) | 0 | $5M$ * |
| 0 0 1 1 (1) | $4M$ | 0 | 1 0 1 1 (1) | 0 | $4M$ |
| 0 1 0 0 (0) | $4M$ | 0 | 1 1 0 0 (0) | 0 | $4M$ |
| 0 1 0 0 (1) | $5M$ * | 0 | 1 1 0 0 (1) | $M$ | $4M$ |
| 0 1 0 1 (0) | $5M$ * | 0 | 1 1 0 1 (0) | $M$ | $4M$ |
| 0 1 0 1 (1) | $8M$ | $2M$ | 1 1 0 1 (1) | 0 | $2M$ |
| 0 1 1 0 (0) | $8M$ | $2M$ | 1 1 1 0 (0) | 0 | $2M$ |
| 0 1 1 0 (1) | $8M$ | $M$ | 1 1 1 0 (1) | 0 | $M$ |
| 0 1 1 1 (0) | $8M$ | $M$ | 1 1 1 1 (0) | 0 | $M$ |
| 0 1 1 1 (1) | $8M$ | 0 | 1 1 1 1 (1) | 0 | 0 |

* Hard multiples.

**Table 1.1: Radix-16 redundant binary booth encoding (RBBE-4)**

## 2. COVALENT REDUNDANT BINARY BOOTH ENCODING (CRBBE) ALGORITHM

Covalent redundant binary booth encoding (CRBBE) algorithm is used to simplify the generation of hard multiples and reduce the number of RB partial products without introducing any form of correction vector.

### 2.1 CRBBE-4 Algorithm:

CRBBE-4 is composed of two adjacent radix-4 Booth encoders. Its gate-level implementation is represented, where the sign and magnitude of the radix-4 Booth encoded digit $d_i$ are represented with three binary bits, $\mathbf{sgn_i}$ , $\mathbf{m_i^{(2)}}$, $\mathbf{m_i^{(1)}}$ , and as follows:

$$d_i = (-1)^{sgn(i)}(m_i^{(2)} + m_i^{(1)})$$

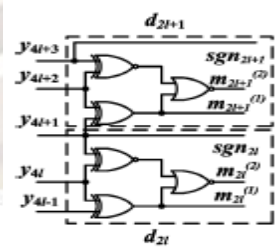
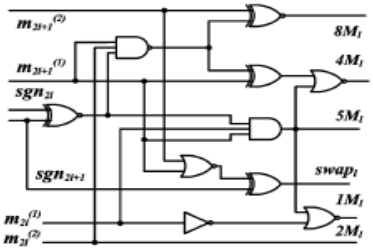



**Figure 2.1: Two adjacent radix-4 Booth encoder**

The above figure shows the '**l**' th slice of a radix-16 CRBBE-4 circuit for the generation of the control signals $c_lM_l$. The indexes '**i**' and '**l**' are related by i=2l. The lower encoder takes three consecutive bits $y_{2i+1}y_{2i}y_{2i-1} = y_{4l+1}y_{4l}y_{4l-1}$ from the multiplier to generate the magnitude bits $m_{2l}^{(2)}$ and $m_{2l}^{(1)}$ of $d_i$ Its sign bit $sgn_i = y_{4l+1}$. The upper encoder takes the binary bits $y_{2i+3}y_{2i+2}y_{2i+1}$ =

$y_{4l+3}y_{4l+2}y_{4l+1}$ and generates the magnitude bit $m_{2l+1}^{(2)}$ and $m_{2l+1}^{(1)}$ of $d_{i+1}$. Its sign bit $sgn_{i+1} = y_{4l+3}$. All of these output signals are mapped to the polarization circuit. The control signals $c_lM_l$, generated are used to select the RB partial products correspond to the multiples $c_lX$. [7]




**.Figure 2.1: Polarization circuit**

The polarization circuit performs the mapping ($d_{i+1}$, $d_i$) ( $p_l^+$, $p_l^-$) .

The control signals $1M_l$, $2M_l$, $4M_l$ and $8M_l$ are computed as follows:

$$1M_l = m_{2l}^{(1)} \cdot \overline{5M_l}$$
$$2M_l = m_{2l}^{(2)}$$
$$4M_l = \overline{\left(m_{2l+1}^{(1)} \cdot m_{2l}^{(2)} \cdot (sgn_{2l+1} \odot sgn_{2l}) \odot m_{2l+1}^{(1)}\right)} \cdot \overline{5M_l}$$
$$8M_l = \overline{m_{2l+1}^{(1)} \cdot m_{2l}^{(2)} \cdot (sgn_{2l+1} \odot sgn_{2l}) \odot m_{2l+1}^{(2)}}.$$

The 5M multiple is generated as

$$5M_l = (sgn_{2l+1} \odot sgn_{2l}) \cdot m_{2l+1}^{(1)} \cdot m_{2l}^{(1)}.$$

The control flag, *swap* is used to exchange $p_l^+$ and $p_l^-$ in the partial product generator to negate the selected RB partial product. When $d_{i+1}$ is 0, the sign bit of $d_{i+1}$ is complemented before it is used as an active high swap flag to
the RBPPG. Otherwise, the original sign of $d_{i+1}$ is used as the swap flag. Therefore, the *swap* signal can be generated by:

$$swap_l = \overline{\left(m_{2l+1}^{(1)} + m_{2l+1}^{(2)}\right)} \oplus sgn_{2l+1}$$

## 3. DESIGN OF CRBBE-4-BASED RB MULTIPLIER

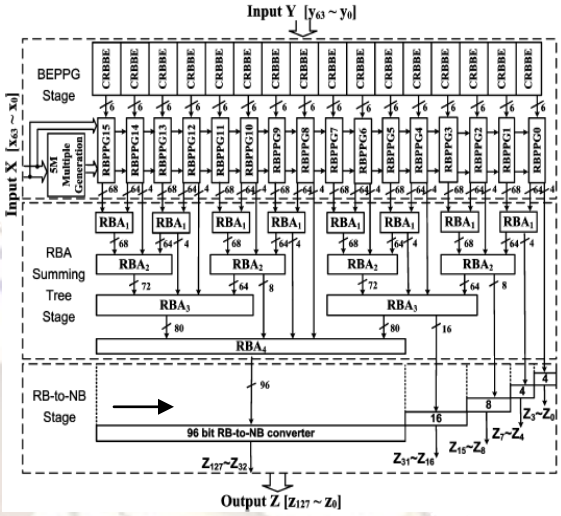



**Figure 3.1: Block diagram of 64*64 RB multiplier**

The block diagram of 64*64 consists of 3 stages:

(1)Booth encoder and partial product generator stage (BEPPG stage)

(2)Redundant binary adder summing tree stage (RBA summing stage)

(3)Redundant binary to NB conversion stage (RB-to-NB stage)

### Booth encoder and partial product generator stage (BEPPG stage):

Booth encoder and partial product generator affect the efficiency of the partial product generation. The number of partial products that can be saved by this stage impacts the cost, performance, and power consumption of the RB summing tree and the multiplier as a whole. In the first stage, 16 CRBBE-4 slices are used to generate the control signals from the multiplier. The hard multiple 5X is generated. The multiplicand bits are shifted and selected into 16 rows of RB partial products in 16 slices of RBPPG.

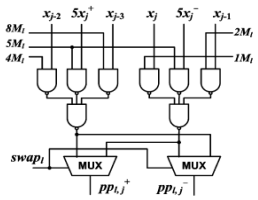



**Figure 3.2: RB partial product generator (RBPPG) of CRBBE-4.**

## Redundant binary adder summing tree stage (RBA summing stage):

In the second stage, a 4-stage RBA summing tree is used to sum 16 RB partial products. Each RBA block contains 64 RB full adder (RBFA) cells and a varying number of RB half adder (RBHA) cells depending on where it is located. The RBA block in the **i**-th level, designated $RBA_i$ ($i$=1 to 4) contains $\mathbf{2^{i+1}}$ RBHA cells in its most significant digit positions. Due to the positive-negative-complement coding, the second binary bit $\mathbf{pp_{i,j}^-}$ of the RB partial product generated from CRBBE-4 and RBPPG circuit should be inverted before it is input to the RBA. A preprocessing circuit is needed for each RB digit to avoid the inconsistent representations of "0" prior to the RBA summing tree stage. An important benefit of the coding format adopted in this design is that these preprocessing circuits can be completely eliminated due to its symmetry.

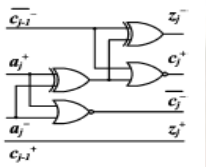


**Figure 3.3: RB Half adder**

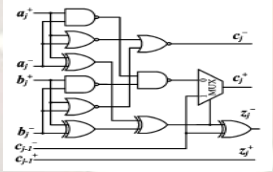


**Figure 3.4: RB Full adder**

## Redundant binary to NB conversion stage (RB-to-NB stage):

An RB-to-NB converter converts the final accumulation result to NB representation. Due to the unequal delay profile of the final RB result bits, the conversion can be carried out in uneven groups of consecutive digits according to their arrival time. Groups of 4, 4, 8, 16 and 96 digits from the least significant digit position are evaluated concurrently. The first three groups of 4, 4, and 8 digits can be independently converted with ripple-carry adders to reduce the circuit complexity. The carry generation of the next group of 16 digits can be evaluated with a carry-look ahead adder as they do not depend on the final summation results in the RBA tree stage.

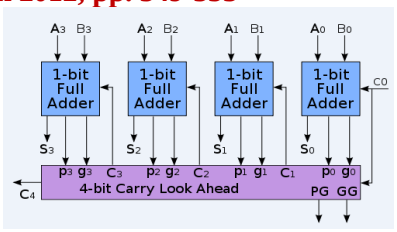



**Figure 3.5: 4-Bit carry look ahead adder**

The conversion speed of the RB-to-NB stage depends solely on the conversion time of the most significant 96-digit group. This group is converted with a hybrid carry-look ahead or carry-select adder since it is widely known as one of the most efficient structures for fast parallel adder design.[8]

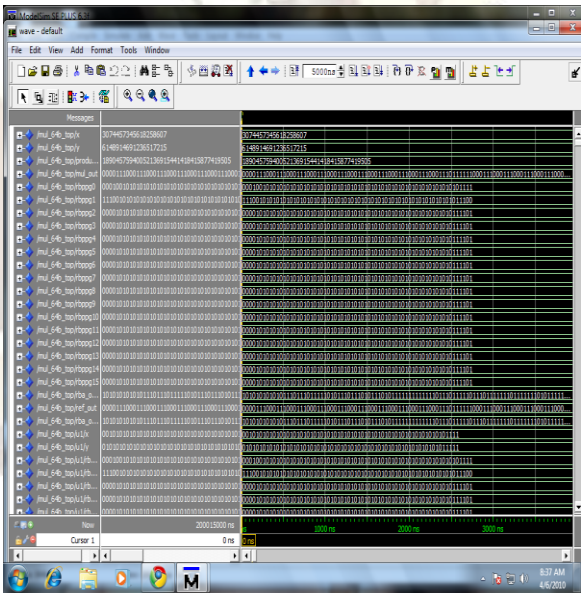
## RESULT:

Simulation is performed successfully. By taking the input values of the multiplicand and multiplier, the product is obtained without any error.

Here X= 307445734561825860
Y= 6148914691236517215
PRODUCT=1890457594005213691544141841587749505
877419505

## SIMULATION REPORTS:

### FINAL MODULE:



## CONCLUSION

Hence, a high-speed and energy-efficient RB multiplier is designed based on new covalent RB Booth encoding algorithm. The idea is to polarize two adjacent Booth-encoded digits into a differential pair to restore the effective RB partial

product reduction rate without the NB-to-RB conversion overhead. This method fully exploits the characteristics of the positive–negative complement coding of RB number to directly generate an RB partial product from two adjacent Booth-encoded digits. Consequently, it shares the same advantages of RB Booth encoder for the ease of generating hard multiples and avoidance of error compensation vector, the two problems that are confronted by RB multiplier with normal binary Booth encoding.

## 4. REFERENCES

[1] C. Shi, W.Wang, L. Zhou, L. Gao, P. Liu, and Q. Yao, "32B RISC/DSP media processor: MediaDSP3201," SPIE Embedded Processors for Multimedia and Communications II, vol. 5683, pp. 43–52, Mar. 2005.

[2] B. Parhami, Computer Arithmetic Algorithms and Hardware Designs. New York: Oxford Univ. Press, 2000.

[3] G. W. Bewick, "Fast multiplication: Algorithms and implementation," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1994.

[4] O. L. MacSorley, "High-speed arithmetic in binary computers," IRE Proc., vol. 49, pp. 67–91, Jan. 1961.

[5] P.Kornerup, "Digit-set conversions: Generalizations and applications," IEEE Trans. Comput., vol. 43, no. 5, pp. 622–629, May 1994.

[6] J.-Y. Kang and J.-L. Gaudiot, "A simple high-speed multiplier design," IEEE Trans. Comput., vol. 55, no. 10, pp. 1253–1258, Oct. 2006.

[7] Y. He, C. H. Chang, J. Gu, and H. A. H. Fahmy, "A novel covalent redundant binary Booth encoder," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'2005), Kobe, Japan, May 2005, vol. 1, pp. 69–72.

[8] V. Kantabutra, "A recursive carry-lookahead/carry-select hybrid adder," IEEE Trans. Comput., vol. 42, no. 12, pp. 1495–1499, Dec. 1993.