International Journal of Emerging Engineering Research and Technology
Volume 3, Issue 7, July 2015, PP 160-171
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Efficient Partial Product Generation using Radix4 & Radix8 for Multi-Modulus Multiplication

Priya Sandhu[1], M. Anusha[2]

[1]*Department of ECE, MRCET, Hyderabad, India (PG Scholar)*
[2]*Department of ECE, MRCET, Hyderabad, India (Associate Professor)*

ABSTRACT

Now a day's multiplication and modulus takes crucial role so we are combining multiplication and modulus. A Novel multi-modulus multiplier with different widths of modulus operations. In this paper we have radix4 multi-modulus multiplier with 4bit, 32bit, 64bit and radix8 multi-modulus multiplier with 4bit, 32bit, 64bit.radix4 and radix8 multi-modulus multiplier using Residue multiplication is implemented by using xilix13.2.

Keywords: multi-modulus multiplier, radix4, radix8.

INTRODUCTION

Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation. The common multiplication method is "add and shift" algorithm. In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms. To achieve speed improvements Wallace Tree algorithm can be used to reduce the number of sequential adding stages. Further by combining both Modified Booth algorithm and Wallace Tree technique we can see advantage of both algorithms in one multiplier. However with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced speed, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing. On the other hand "serial-parallel" multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this lecture we introduce the multiplication algorithms and architecture and compare them in terms of speed, area, power and combination of these metrics. In computing, the modulo operation finds the remainder after division of one number by another (sometimes called modulus) .Given two positive numbers, a (the dividend) and n (the divisor), a modulo n (abbreviated as a mod n) is the remainder of the Euclidean division of a by n. For instance, the expression "5 mod 2" would evaluate to 1 because 5 divided by 2 leaves a quotient of 2 and a remainder of 1, while "9 mod 3" would evaluate to 0 because the division of 9 by 3 has a quotient of 3 and leaves a remainder of 0; there is nothing to subtract from 9 after multiplying 3 times 3. (Note that doing the division with a calculator will not show the result referred to here by this operation; the quotient will be expressed as a decimal fraction. Although typically performed with a and n both being integers, many computing systems allow other types of numeric operands. The range of numbers for an integer modulo of n is 0 to $n - 1$. (n mod 1 is always 0; n mod 0 is undefined, possibly resulting in a "Division by zero" error in computer programming languages) See modular arithmetic for an older and related convention applied in theory. When either a or n is negative, the naive definition breaks down and programming languages differ in how these values are defined.

Address for correspondence

priyasandhu25@gmail.com

RADIX4 MULTI MODULUS MULTIPLIER

The radix- booth encoded digit is formatted using three bits: a sign bit and one-hot encoded magnitude bits, and. The proposed multi-modulus radix- booth encoder using radix- booth encoder (be2) slices and one mux3block is shown in fig. 2. Booth encoding technique is applied to multi modulus multiplier modulo (2^n)-1, modulo 2^n and modulo (2^n) +1 operations as the basis. The Radix -4 multiplication

- The multiplier, Y in two's complement form can be written as in

$$Y = -Y_{n-1}\,2^{n-1} + \sum_{i} Y_i 2^i \quad ; 0 \le i \le n-2 \qquad (1)$$

- It can be written as

$$Y = (-2\,Y_{2i+1} + Y_{2i} + Y_{2i-1})\,2^{2i}; \qquad 0 \le i \le n-2 \qquad (2)$$

- Radix-4 Booth recoding encodes multiplier bits into [-2, 2].
- Radix-8 Booth recoding encodes multiplier bits into [-4, 4].

The radix-4 Booth encoded digit is formatted using three bits: a sign bit and one-hot encoded magnitude bits, m1i and m2i. The proposed multi-modulus radix- Booth encoder using radix-4 Booth Encoder (BE2) slices and one MUX3 block.

Proposed Multi Modulus Radix-4 Booth Encoder

The radix- Booth encoded digit is formatted using three bits: a sign bit and one-hot encoded magnitude bits. The proposed multi-modulus radix-4 Booth encoder using N/2 radix-4 Booth Encoder (BE2) slices and one MUX3 block. (N=4)The input corresponding to the modulus 2^n-1,2^n and 2^n+1 is selected when is "00," "01," or "10," respectively. Using the radix-4 Booth encoded multiplier form, the multi-modulus multiplication are

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{n/2-1} 2^{2i} d_i \cdot X \right|_m & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ \left| \sum_{i=0}^{n/2-1} 2^{2i} d_i \cdot X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases}$$

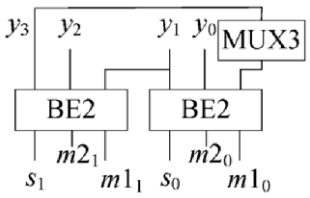


Figure1.

Proposed 3:1 Multiplexer (Mux3)

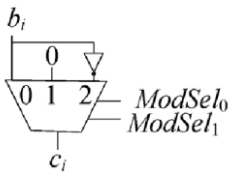


Figure2.

Modulo-Reduced Partial Products for Radix-4 Booth Encoding

In the following, the generation of the for the three moduli n/2 PPs, and is described. The standard circuit implementation of a bit slice of the radix- Booth Selector (BS2) generates a single bit of , i.e., ,

by selecting a bit of either the multiplicand or one-bit shifted multiplicand and conditionally inverting it. From Table I, the least significant bits of PPs for moduli 2^n-1,2^n and 2^n+1 are PPs, 0 and, PPs respectively. Hence, MUX3 of Fig.3.1(b) can be used at the output of the BS2 blocks in the least significant bit positions to select from PP, 0 or not PP. Furthermore, this input to the BS2 block is also selected using a MUX3. The proposed multi-modulus generation of the n/2 PPs using BS2 and MUX3 blocks is shown in Fig. 3.2 for n =4 . The number of BS2 and MUX3 blocks required in the PP generation is $n^2/2$and $n^4/2$, respectively.

Table1.

d_i	$\lvert 2^{2i} d_i \cdot X \rvert_{2^n-1}$	$\lvert 2^{2i} d_i \cdot X \rvert_{2^n} = PP_i + K_i$		$\lvert 2^{2i} d_i \cdot X \rvert_{2^n+1} = PP_i + K_i$	
	PP_i	PP_i	K_i	PP_i	K_i
+0	$\underbrace{0\cdots0}_{n}$	$\underbrace{0\cdots0}_{n}$	0	$\underbrace{0\cdots0}_{n-2i}\#\underbrace{1\cdots1}_{2i}$	$-2^{2i}+1$
+1	$\underbrace{x_{n-1-2i}\cdots x_0}_{n-2i}\#$ $\underbrace{x_{n-1}\cdots x_{n-2i}}_{2i}$	$\underbrace{x_{n-1-2i}\cdots x_0}_{n-2i}\#$ $\underbrace{0\cdots0}_{2i}$	0	$\underbrace{x_{n-1-2i}\cdots x_0}_{n-2i}\#$ $\underbrace{\overline{x}_{n-1}\cdots\overline{x}_{n-2i}}_{2i}$	$-2^{2i}+1$
+2	$\underbrace{x_{n-2-2i}\cdots x_0 x_{n-1}}_{n-2i}\#$ $\underbrace{x_{n-2}\cdots x_{n-1-2i}}_{2i}$	$\underbrace{x_{n-2-2i}\cdots x_0 0}_{n-2i}\#$ $\underbrace{0\cdots0}_{2i}$	0	$\underbrace{x_{n-2-2i}\cdots x_0\overline{x}_{n-1}}_{n-2i}\#$ $\underbrace{\overline{x}_{n-2}\cdots\overline{x}_{n-1-2i}}_{2i}$	$-2^{2i+1}+1$
−0	$\underbrace{1\cdots1}_{n}$	$\underbrace{1\cdots1}_{n-2i}\#\underbrace{0\cdots0}_{2i}$	2^{2i}	$\underbrace{1\cdots1}_{n-2i}\#\underbrace{0\cdots0}_{2i}$	$2^{2i}+1$
−1	$\underbrace{\overline{x}_{n-1-2i}\cdots\overline{x}_0}_{n-2i}\#$ $\underbrace{\overline{x}_{n-1}\cdots\overline{x}_{n-2i}}_{2i}$	$\underbrace{\overline{x}_{n-1-2i}\cdots\overline{x}_0}_{n-2i}\#$ $\underbrace{0\cdots0}_{2i}$	2^{2i}	$\underbrace{\overline{x}_{n-1-2i}\cdots\overline{x}_0}_{n-2i}\#$ $\underbrace{x_{n-1}\cdots x_{n-2i}}_{2i}$	$2^{2i}+1$
−2	$\underbrace{\overline{x}_{n-2-2i}\cdots\overline{x}_0\overline{x}_{n-1}}_{n-2i}\#$ $\underbrace{\overline{x}_{n-2}\cdots\overline{x}_{n-1-2i}}_{2i}$	$\underbrace{\overline{x}_{n-2-2i}\cdots\overline{x}_0 1}_{n-2i}\#$ $\underbrace{0\cdots0}_{2i}$	2^{2i}	$\underbrace{\overline{x}_{n-2-2i}\cdots\overline{x}_0 x_{n-1}}_{n-2i}\#$ $\underbrace{x_{n-2}\cdots x_{n-1-2i}}_{2i}$	$2^{2i+1}+1$

Proposed Multi-Modulus Partial Product Generation for Radix-4 Booth Encoding

The booth selector (BS2) module is used to generate partial products. The number of BS2 and MUX3 blocks required in the partial products generation is 8 and 4, respectively.(for n = 4)

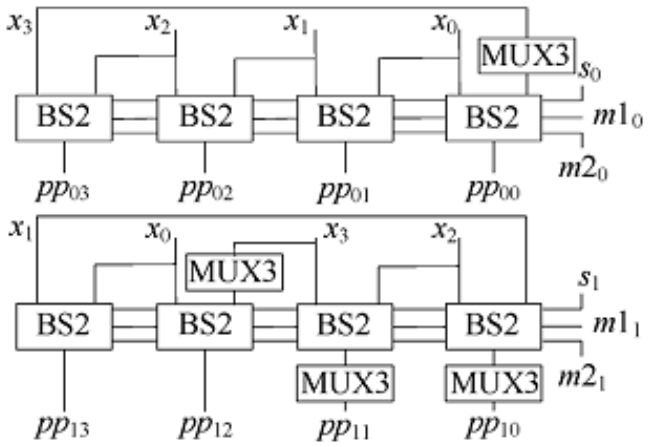


Figure3.

Bias for the Modulus 2^N for Radix-4 Booth Encoding

Table2.

d_i	K_i	$d_i=(-1)^{s_i}(m1_i+2(m2_i))$			$K_i=(2^{2i})a$
		s_i	$m2_i$	$m1_i$	a
+0	0	0	0	0	0
+1	0	0	0	1	0
+2	0	0	1	0	0
–0	2^{2i}	1	0	0	1
–1	2^{2i}	1	0	1	1
–2	2^{2i}	1	1	0	1

Bias for the Modulus 2^N for Radix-4 Booth Encoding

Table3.

d_i	K_i	$KD_i=K_i-KS_i$	s_i	$m2_i$	$m1_i$	a	b	c
+0	$-2^{2i}+1$	0	0	0	0	0	0	0
+1	$-2^{2i}+1$	0	0	0	1	0	0	0
+2	$-2^{2i+1}+1$	-2^{2i}	0	1	0	1	0	0
–0	$2^{2i}+1$	2^{2i+1}	1	0	0	0	1	0
–1	$2^{2i}+1$	2^{2i+1}	1	0	1	0	1	0
–2	$2^{2i+1}+1$	$2^{2i}+2^{2i+1}=$ $-2^{2i}+2^{2i+1}+2^{2i+1}$	1	1	0	1	1	1

Proposed Multi-Modulus Bias Generation for Radix-4 Booth Encoding

An efficient technique to include K_i with minimal hardware circuitry based on the properties of modulo 2^n and modulo 2^n+1 arithmetic is proposed. The generation of PPi using BS2 blocks, the bias K_i can be generated by decoding d_i.

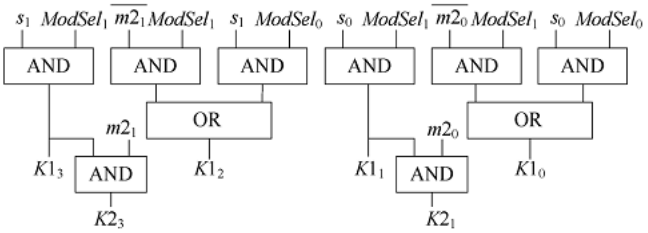


Figure4.

$$K1=\begin{cases}\sum_{i=0}^{n/2-1}2^{2i}\cdot 0 \\ +\sum_{i=0}^{n/2-1}2^{2i+1}\cdot 0 & \text{if } m=2^n-1\\ \sum_{i=0}^{n/2-1}2^{2i}\cdot s_i \\ +\sum_{i=0}^{n/2-1}2^{2i+1}\cdot 0 & \text{if } m=2^n\\ \sum_{i=0}^{n/2-1}2^{2i}\cdot \overline{m2_i} \\ +\sum_{i=0}^{n/2-1}2^{2i+1}\cdot s_i & \text{if } m=2^n+1\end{cases}$$

$$K2=\begin{cases}\sum_{i=0}^{n/2-1}2^{2i}\cdot 0 \\ +\sum_{i=0}^{n/2-1}2^{2i+1}\cdot 0 & \text{if } m=2^n-1\\ & \text{or } m=2^n\\ \sum_{i=0}^{n/2-1}2^{2i}\cdot 1 \\ +\sum_{i=0}^{n/2-1}2^{2i+1}(s_i\cdot m2_i) & \text{if } m=2^n+1\end{cases}$$

$$\left| \sum_{i=0}^{n/2-1} K_i \right|_m = K1 + K2$$

Proposed Multi-Modulus Partial Product Addition for Radix-4 Booth Encoding

A multi-modulus addition can be implemented using a MUX3 in the carry feedback path that selects from, 0 or. The multi-modulus addition of partial products in a CSA tree and a parallel-prefix two-operand adder is illustrated in Fig.5. The parallel- prefix adder is constructed from the pre-processing (PP), prefix and post-processing blocks and the implementation of these blocks is shown in Fig. 6 the number of MUX3 blocks needed for multi-modulus partial product addition.

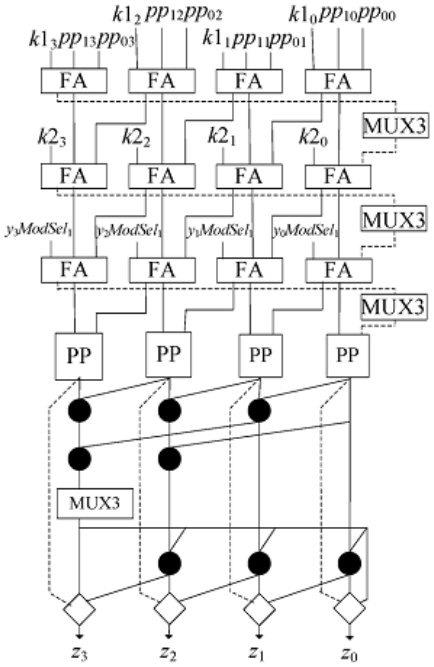


Figure.5

Implementation of Parallel-Prefix Adder Components

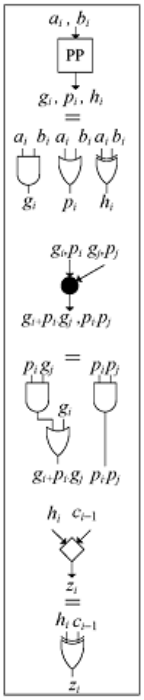


Figure6.

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{n/2-1} PP_i \right|_m & \text{if } m = 2^n - 1 \\ \left| \sum_{i=0}^{n/2-1} PP_i + \sum_{i=0}^{n/2-1} K_i \right|_m & \text{if } m = 2^n \\ \left| \sum_{i=0}^{n/2-1} PP_i + \sum_{i=0}^{n/2-1} K_i + Y \right|_m & \text{if } m = 2^n + 1 \end{cases}$$

RADIX8 MULTI MODULUS MULTIPLIER

Proposed Multi-Modulus Radix-8 Booth Encoder

The radix-8 booth encoded multiplier digit is formatted using five bits, a sign bit s_i and four one-hot encoded magnitude bits $m1_i$, $m2_i$, $m3_i$ and $m4_i$. [N/3]+1 partial product.

$$\sum_{i=0}^{n-1} 2^i y_i = \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i}(y_{3i-1} + y_{3i} + 2y_{3i+1} - 4y_{3i+2}) = \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i} d_i$$

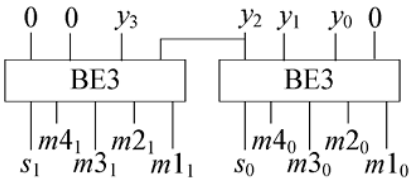


Figure7.

Modulo-Reduced Partial Products for Radix-8 Booth Encoding

Table4.

d_i	$\lvert 2^{3i} d_i \cdot X \rvert_{2^n-1}$	$\lvert 2^{3i} d_i \cdot X \rvert_{2^n} = PP_i + K_i$		$\lvert 2^{3i} d_i \cdot X \rvert_{2^n+1} = PP_i + K_i$	
	PP_i	PP_i	K_i	PP_i	K_i
+0	$\underbrace{0\cdots0}_{n}$	$\underbrace{0\cdots0}_{n}$	0	$\underbrace{0\cdots0}_{n-3i}\#\underbrace{1\cdots1}_{3i}$	$-2^{3i}+1$
+1	$\underbrace{x_{n-1-3i}\cdots x_0}_{n-3i}\#$ $\underbrace{x_{n-1}\cdots x_{n-3i}}_{3i}$	$\underbrace{x_{n-1-3i}\cdots x_0}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	0	$\underbrace{x_{n-1-3i}\cdots x_0}_{n-3i}\#$ $\underbrace{\overline{x}_{n-1}\cdots\overline{x}_{n-3i}}_{3i}$	$-2^{3i}+1$
+2	$\underbrace{x_{n-2-3i}\cdots x_0 x_{n-1}}_{n-3i}\#$ $\underbrace{x_{n-2}\cdots x_{n-1-3i}}_{3i}$	$\underbrace{x_{n-2-3i}\cdots x_0 0}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	0	$\underbrace{x_{n-2-3i}\cdots x_0\overline{x}_{n-1}}_{n-3i}\#$ $\underbrace{\overline{x}_{n-2}\cdots\overline{x}_{n-1-3i}}_{3i}$	$-2^{3i+1}+1$
+3	$CLS(\lvert 3X \rvert_{2^n-1}, 3i)$ $\underbrace{h_{n-1-3i}\cdots h_0}_{n-3i}\#$ $\underbrace{h_{n-1}\cdots h_{n-3i}}_{3i}$	$LS(\lvert 3X \rvert_{2^n}, 3i)$ $\underbrace{h_{n-1-3i}\cdots h_0}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	0	$CCLS(\lvert 3X \rvert_{2^n+1}, 3i)$ $\underbrace{h_{n-1-3i}\cdots h_0}_{n-3i}\#$ $\underbrace{\overline{h}_{n-1}\cdots\overline{h}_{n-3i}}_{3i}$	$-2^{3i}+1$ $-2^{3i+1}-2^{3i}+1$
+4	$\underbrace{x_{n-3-3i}\cdots x_0 x_{n-1} x_{n-2}}_{n-3i}\#$ $\underbrace{x_{n-3}\cdots x_{n-2-3i}}_{3i}$	$\underbrace{x_{n-3-3i}\cdots x_0 00}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	0	$\underbrace{x_{n-3-3i}\cdots x_0\overline{x}_{n-1}\overline{x}_{n-2}}_{n-3i}\#$ $\underbrace{\overline{x}_{n-3}\cdots\overline{x}_{n-2-3i}}_{3i}$	$-2^{3i+2}+1$
−0	$\underbrace{1\cdots1}_{n}$	$\underbrace{1\cdots1}_{n-3i}\#\underbrace{0\cdots0}_{3i}$	2^{3i}	$\underbrace{1\cdots1}_{n-3i}\#\underbrace{0\cdots0}_{3i}$	$2^{3i}+1$
−1	$\underbrace{\overline{x}_{n-1-3i}\cdots\overline{x}_0}_{n-3i}\#$ $\underbrace{\overline{x}_{n-1}\cdots\overline{x}_{n-3i}}_{3i}$	$\underbrace{\overline{x}_{n-1-3i}\cdots\overline{x}_0}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	2^{3i}	$\underbrace{\overline{x}_{n-1-3i}\cdots\overline{x}_0}_{n-3i}\#$ $\underbrace{x_{n-1}\cdots x_{n-3i}}_{3i}$	$2^{3i}+1$
−2	$\underbrace{\overline{x}_{n-2-3i}\cdots\overline{x}_0\overline{x}_{n-1}}_{n-3i}\#$ $\underbrace{\overline{x}_{n-2}\cdots\overline{x}_{n-1-3i}}_{3i}$	$\underbrace{\overline{x}_{n-2-3i}\cdots\overline{x}_0 1}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	2^{3i}	$\underbrace{\overline{x}_{n-2-3i}\cdots\overline{x}_0 x_{n-1}}_{n-3i}\#$ $\underbrace{x_{n-2}\cdots x_{n-1-3i}}_{3i}$	$2^{3i+1}+1$
−3	$CLS(\lvert\overline{3X}\rvert_{2^n-1}, 3i)$ $\underbrace{\overline{h}_{n-1-3i}\cdots\overline{h}_0}_{n-3i}\#$ $\underbrace{\overline{h}_{n-1}\cdots\overline{h}_{n-3i}}_{3i}$	$LS(\lvert\overline{3X}\rvert_{2^n}, 3i)$ $\underbrace{\overline{h}_{n-1-3i}\cdots\overline{h}_0}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	2^{3i}	$CCLS(\lvert\overline{3X}\rvert_{2^n+1}, 3i)$ $\underbrace{\overline{h}_{n-1-3i}\cdots\overline{h}_0}_{n-3i}\#$ $\underbrace{h_{n-1}\cdots h_{n-3i}}_{3i}$	$2^{3i}+1$ $2^{3i+1}+2^{3i}+1$
−4	$\underbrace{\overline{x}_{n-3-3i}\cdots\overline{x}_0\overline{x}_{n-1}\overline{x}_{n-2}}_{n-3i}\#$ $\underbrace{\overline{x}_{n-3}\cdots\overline{x}_{n-2-3i}}_{3i}$	$\underbrace{\overline{x}_{n-3-3i}\cdots\overline{x}_0 11}_{n-3i}\#$ $\underbrace{0\cdots0}_{3i}$	2^{3i}	$\underbrace{\overline{x}_{n-3-3i}\cdots\overline{x}_0 x_{n-1} x_{n-2}}_{n-3i}\#$ $\underbrace{x_{n-3}\cdots x_{n-2-3i}}_{3i}$	$2^{3i+2}+1$

Proposed Multi-Modulus Partial Product Generation for Radix-8 Booth Encoding

By the radix-8 Booth encoded multi-modulus multiplication is given by the booth selector (BS3) module is used to generate partial products. The number of BS3 and MUX3 blocks required in the partial products generation is 8 and 4, respectively (for n = 8) .The booth selector (BS3) module is used to generate partial product. The number of BS3 and MUX3 blocks required in the partial products generation is 8 and 4, respectively.(for n = 8)

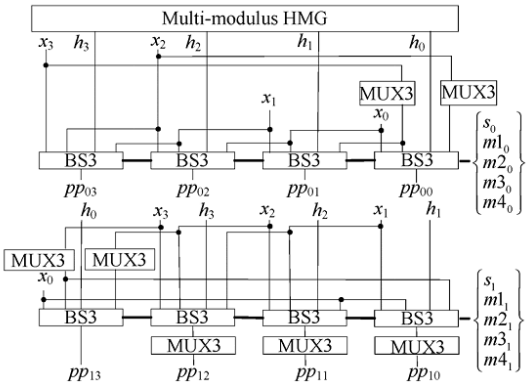


Figure8.

Proposed Multi-Modulus HMG

Application – specific adders known as HMG that compute only the sum of X and $2X_m$ to generate the hard multiple $+3X_m$. The HMG_s were designed by reformulating the carry equations of modulo 2^n-1 and 2^n+1 addition using the bit correlation between the addends and $2X_m$.The carry equation given below

$$(g_1^*, p_1^*) = \begin{cases} (x_0 \cdot (x_1 + x_{n-1}), \\ \quad x_0 + (x_1 \cdot x_{n-1})) & \text{if } m = 2^n - 1 \\ (x_0 \cdot (x_1 + \bar{x}_{n-1}), \\ \quad x_0 + (x_1 \cdot \bar{x}_{n-1})) & \text{if } m = 2^n + 1 \end{cases}$$

$$(g_i^*, p_i^*) = (x_{i-1} \cdot (x_i + x_{i-2}), x_{i-1} + (x_i \cdot x_{i-2}))$$

As modulo 2^n addition is equivalent to carry-ignore add the carry equation for modulo 2^n HMG becomes

$$c_i = (g_i^*, p_i^*) \bullet (g_{i-2}^*, p_{i-2}^*) \bullet \cdots \bullet (g_0^*, p_0^*) \bullet (0,0) \bullet \cdots \bullet$$

where (g_i^*, p_i^*) is defined as follows.

$$(g_0^*, p_0^*) = (0 \cdot (x_0 + 0), 0 + (x_0 \cdot 0)) = (0,0)$$
$$(g_1^*, p_1^*) = (x_0 \cdot (x_1 + 0), x_0 + (x_1 \cdot 0)) = (x_0 \cdot x_1, x$$
$$(g_i^*, p_i^*) = (x_{i-1} \cdot (x_i + x_{i-2}), x_{i-1} + (x_i \cdot x_{i-2}))$$

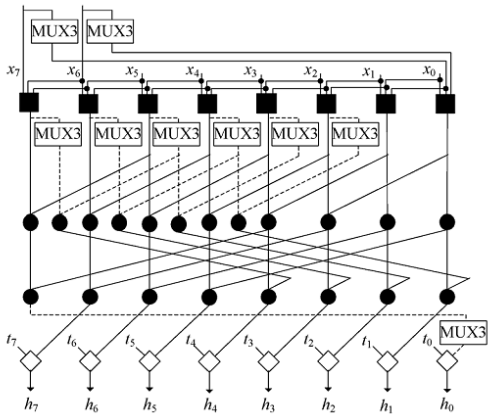


Figure9.

Bias for the Modulus 2^N for Radix-8 Booth Encoding

Table5.

d_i	K_i	$d_i=(-1)^{s_i}(m1_i+2(m2_i)+3(m3_i)+4(m4_i))$					$K_i=(2^{3i})a$
		s_i	$m4_i$	$m3_i$	$m2_i$	$m1_i$	a
+0	0	0	0	0	0	0	0
+1	0	0	0	0	0	1	0
+2	0	0	0	0	1	0	0
+3	0	0	0	1	0	0	0
+4	0	0	1	0	0	0	0
−0	2^{3i}	1	0	0	0	0	1
−1	2^{3i}	1	0	0	0	1	1
−2	2^{3i}	1	0	0	1	0	1
−3	2^{3i}	1	0	1	0	0	1
−4	2^{3i}	1	1	0	0	0	1

Bias for the modulus 2^n+1 for radix-8 booth encoding

Table6.

d_i	K_i	$KD_i=K_i-KS_i$	s_i	$m4_i$	$m3_i$	$m2_i$	$m1_i$	a	b	c	d	e
+0	$-2^{3i}+1$	0	0	0	0	0	0	0	0	0	0	0
+1	$-2^{3i}+1$	0	0	0	0	0	1	0	0	0	0	0
+2	$-2^{3i+1}+1$	-2^{3i}	0	0	0	1	0	1	0	0	0	0
+3	$-2^{3i+1}-2^{3i}+1$	-2^{3i+1}	0	0	1	0	0	0	1	0	0	0
+4	$-2^{3i+2}+1$	$-2^{3i}-2^{3i+1}$	0	1	0	0	0	1	1	0	0	0
−0	$2^{3i}+1$	2^{3i+1}	1	0	0	0	0	0	0	1	0	0
−1	$2^{3i}+1$	2^{3i+1}	1	0	0	0	1	0	0	1	0	0
−2	$2^{3i+1}+1$	$-2^{3i}+2^{3i+1}+2^{3i+1}$	1	0	0	1	0	1	0	1	1	0
−3	$2^{3i+1}+2^{3i}+1$	$-2^{3i+1}+2^{3i+1}+2^{3i+2}$	1	0	1	0	0	0	1	1	0	1
−4	$2^{3i+2}+1$	$-2^{3i}-2^{3i+1}+2^{3i+1}+2^{3i+1}+2^{3i+2}$	1	1	0	0	0	1	1	1	1	1

Proposed Multi-Modulus Bias Generation for Radix-8 Booth Encoding

The operators "+" and "." denote Boolean OR and AND, respectively when their operands are Boolean variables. For m = 2^n+1, the aggregate bias is composed of three n- bit words, K1,K2 and K3. For m= 2^n the aggregate bias is composed of one n- bit word K1. For m= 2^n-1, the aggregate bias equal zeros.

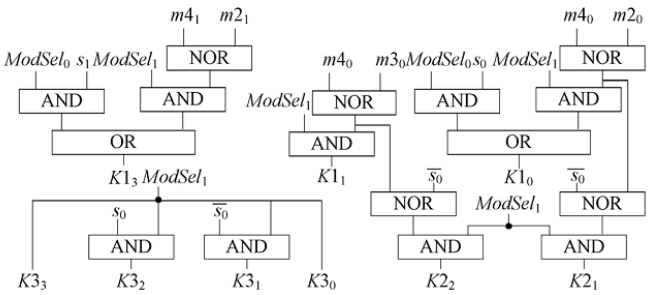


Figure10.

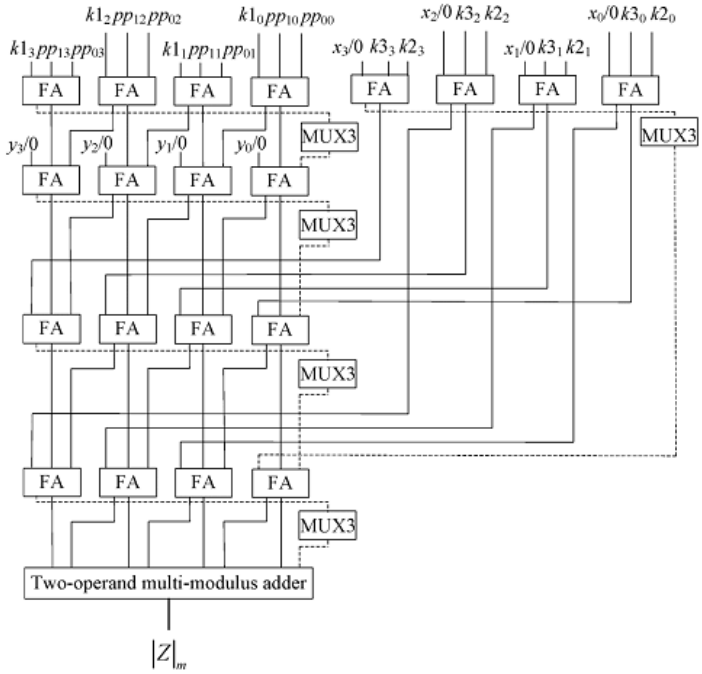
$$K1=\begin{cases}\sum_{i=0}^{\lfloor n/3\rfloor}(2^{3i}\cdot 0+2^{3i+1}\cdot 0)+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}\cdot 0 & \text{if } m=2^n-1\\ \sum_{i=0}^{\lfloor n/3\rfloor}(2^{3i}\cdot s_i+2^{3i+1}\cdot 0)+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}\cdot 0 & \text{if } m=2^n\\ \sum_{i=0}^{\lfloor n/3\rfloor}\left(2^{3i}(\overline{m2_i+m4_i})+2^{3i+1}(\overline{m3_i+m4_i})\right)+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}\cdot 0 & \text{if } m=2^n+1\end{cases}$$

$$K2=\begin{cases}\sum_{i=0}^{\lfloor n/3\rfloor}(2^{3i}\cdot 0+2^{3i+1}\cdot 0)+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}\cdot 0 & \text{if } m=2^n-1 \text{ or } m=2^n\\ \sum_{i=0}^{\lfloor n/3\rfloor}\left(2^{3i}\cdot 0+2^{3i+1}((m2_i+m4_i)\cdot s_i)\right)+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}((m3_i+m4_i)\cdot s_i) & \text{if } m=2^n+1\end{cases}$$

$$K3=\begin{cases}\sum_{i=0}^{\lfloor n/3\rfloor}(2^{3i}\cdot 0+2^{3i+1}\cdot 0)+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}\cdot 0 & \text{if } m=2^n-1 \text{ or } m=2^n\\ \sum_{i=1}^{\lfloor n/3\rfloor}2^{3i}\cdot 1+\sum_{i=0}^{\lfloor n/3\rfloor}2^{3i+1}\cdot\bar{s}_i+\sum_{i=0}^{\lfloor n/3\rfloor-1}2^{3i+2}\cdot s_i & \text{if } m=2^n+1\end{cases}$$

Proposed Multi-Modulus Partial Product Addition for Radix-8 Booth Encoding

A multi-modulus addition can be implemented using a MUX3 in the carry feedback path that selects from, 0 or. The multi-modulus addition of partial products in a CSA tree and a parallel-prefix two-operand adder. The parallel- prefix adder is constructed from the pre-processing (PP), prefix and post-processing blocks and the implementation of these blocks. Number of MUX3 blocks needed for multi-modulus partial product addition

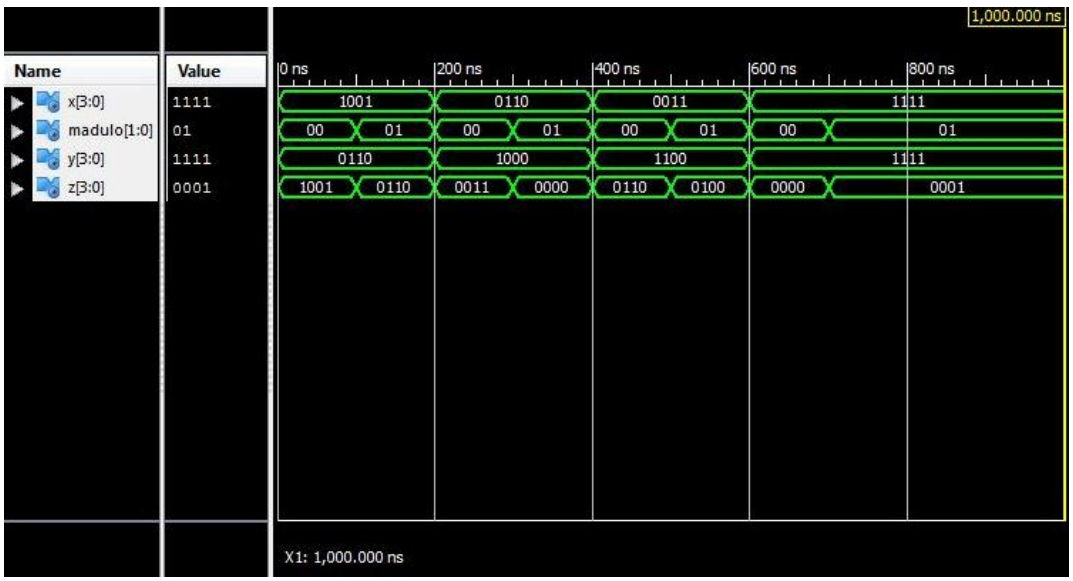


$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + K1 + K2 + K3 + 0 + 0 \right|_m & \text{if } m = 2^n - 1 \text{ or } m = 2^n \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + K1 + K2 + K3 + X + Y \right|_m & \text{if } m = 2^n + 1 \end{cases}$$

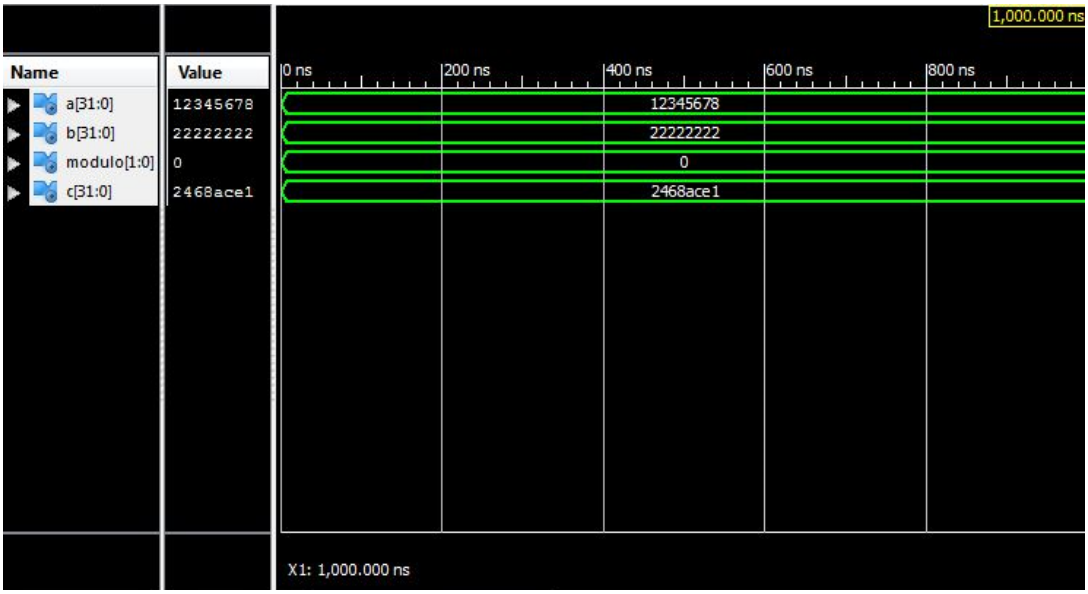
Figure11.

SIMULATION RESULTS

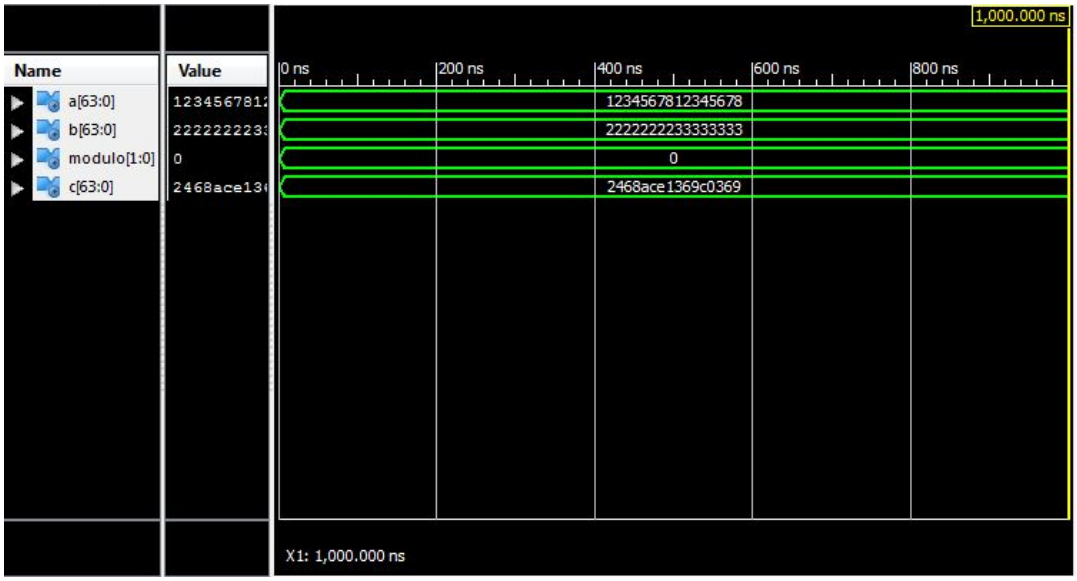
Radix4-4bit



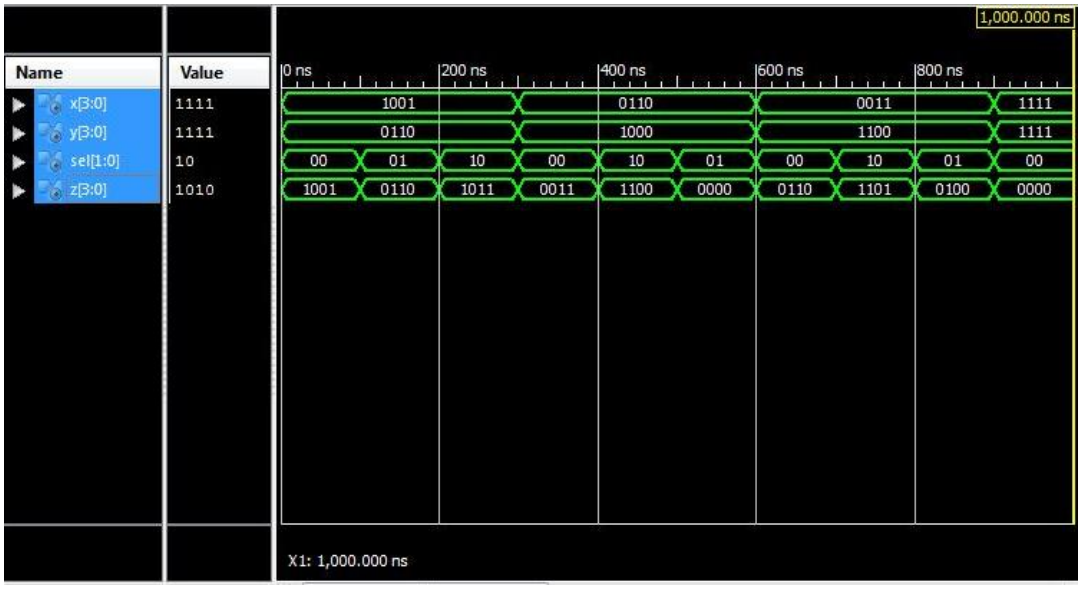
Radix4-32bit



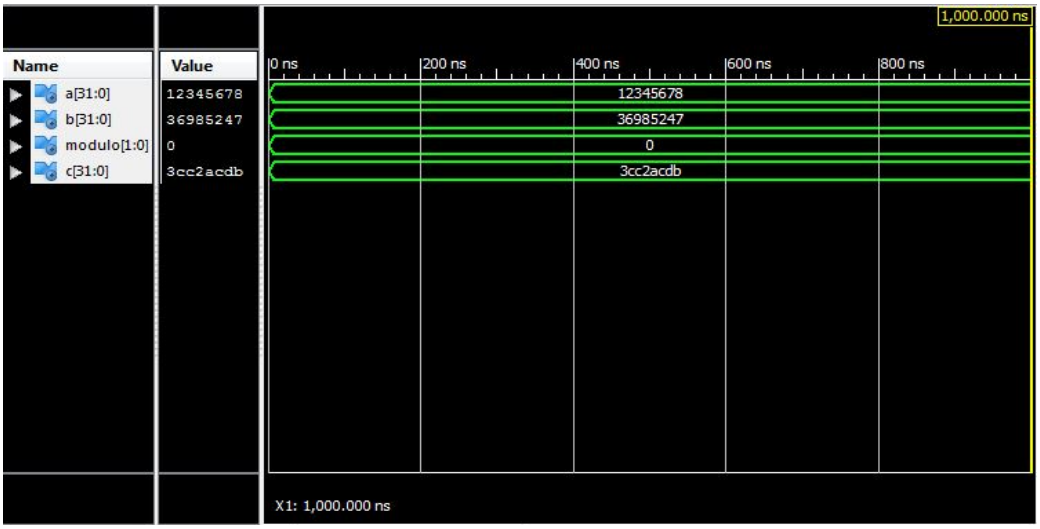
Radix4-64bit



Radix8-4bit



Radix8-32bit



Radix8-64bit

CONCLUSION

A new radix-4 and radix-8 Booth encoded multi-modulus multipliers that perform modulo multiplication for the three special moduli operations in this we are designed radix4 and radix8 multipliers then do modulus operation so radix 4 having 4,32,64 bit sizes and radix 8 having 4,32,64 bit sizes. These designs are implemented using Xilinx 13.2.

REFERENCES

[1] R. Conway and J. Nelson, "Improved RNS FIR filter architectures, "IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 51, no. 1, pp. 26–28,Jan. 2004

[2] A. S. Madhukumar and F. Chin, "Enhanced architecture for residue number system-based CDMA for high rate data transmission," IEEE Trans. Wireless Commun., vol. 3, no. 5, pp. 1363–1368, Sep. 2004.

[3] D. M. Schinianakis et al., "An RNS implementation of an elliptic curve point multiplier," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 56, no. 6, pp. 1202–1213, Jun. 2009.

[4] N. S. Szabo and R. I. Tanaka, Residue Arithmetic and its Application to Computer Technology. New York: McGraw-Hill, 1967.

[5] R. Zimmermann, "Efficient VLSI implementation of modulo addition and multiplication," in Proc. 14th IEEE Symp. Computer Arithmetic, Adelaide, Australia, Apr. 1999, pp. 158–167.

[6] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos,"High-speed parallel-prefix modulo adders," IEEETrans. Comput, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[7] S. J. Piestrak, "Design of squarers modulo A with low-level pipelining," IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 49, no. 1, pp. 31–41, Jan. 2002.

[8] H. T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-one modulo adder design," IEEE Trans. Comput., vol. 51, no. 12, pp.1389–1399, Dec. 2002.

[9] C. Efstathiou, H. T. Vergos, and D. Nikolos, "Modified Booth modulo multiplier," IEEE

AUTHORS' BIOGRAPHY

Priya Sandhu received B.Tech degree in Electronics and Communication Engineering from JIET SETG Jodhpur, Rajasthan, India, affiliated to Rajasthan Technical University, Kota, India in 2012. She is pursuing M.Tech in VLSI Design& Embedded Systems at Malla Reddy College of Engineering and Technology, Hyderabad, India.

Anusha Meneni received her B.Tech Degree in Electronics & Instrumentation Engineering from JNT University, Hyderabad, M.Tech in VLSI System Design from JNT University, Hyderabad. She is currently working as Associate Professor in the Department of ECE, Malla Reddy College of Engineering and Technology, Hyderabad, India