On the Systematic Creation of Faithfully Rounded Commutative Truncated Booth Multipliers

Theo Drane, Samuel Coward
Graphics Numerical Hardware Group
Intel Corporation
Email: {theo.drane,samuel.coward}@intel.com

Mertcan Temel
Advanced Architecture Design Group
Intel Corporation
Email: mert.temel@intel.com

Joe Leslie-Hurd
Design Engineering Group
Intel Corporation
Email: joe.leslie-hurd@intel.com

***Abstract*—In many instances of fixed-point multiplication, a full precision result is not required. Instead it is sufficient to return a faithfully rounded result. Faithful rounding permits the machine representable number either immediately above or below the full precision result, if the latter is not exactly representable. Multipliers which take full advantage of this freedom can be implemented using less circuit area and consuming less power. The most common implementations internally truncate the partial product array. However, truncation applied to the most common of multiplier architectures, namely Booth architectures, results in non-commutative implementations. The industrial adoption of truncated multipliers is limited by the absence of formal verification of such implementations, since exhaustive simulation is typically infeasible. We present a commutative truncated Booth multiplier architecture and derive closed form necessary and sufficient conditions for faithful rounding. We also provide the bit-vectors giving rise to the worst-case error. We present a formal verification methodology based on ACL2 which scales up to 42 bit multipliers. We synthesize a range of commutative faithfully rounded multipliers and show that truncated booth implementations are up to 31% smaller than externally truncated multipliers.**

1. Introduction

Of the most common arithmetic circuits, multiplication consumes the greatest power and occupies the largest circuit area. As a result, binary multiplication has been the subject of significant academic and industrial research [1], [2], [3]. Amongst the most widely implemented multiplier architectures is the Booth Radix-4 multiplier [4]. In many applications, the requirement for exact multiplication can be dropped, and replaced with a faithful rounding requirement. A faithful rounding returns the machine representable number immediately above or below the infinitely precise result, unless the infinitely precise result can in fact be represented at the machine precision.

The additional freedom introduced by a faithful rounding, can be exploited, at the register transfer level (RTL), to improve multiplier power consumption and save circuit area. The standard approach to exploiting such freedom is to truncate the partial product array, as shown in Figure 1.

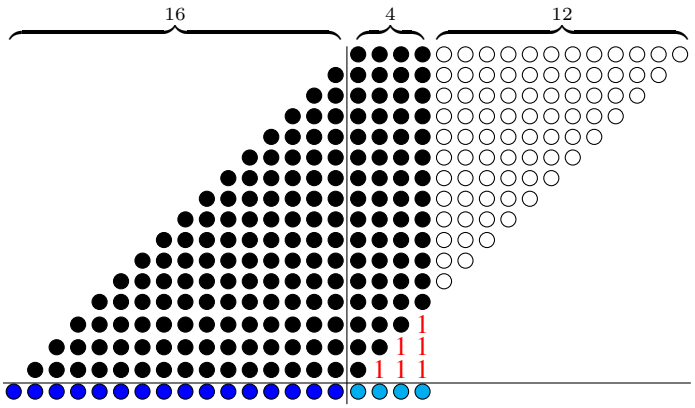

Figure 1. A 16-bit $a \times b$, implemented as a traditional partial product array, where each partial product bit is $a[i]\&b[j]$. From this array we truncate 12 columns and insert the compensation constant, 11 (red). Summing the truncated array (black) and discarding the light blue bits produces a faithfully rounded 16-bit multiplication result.

Unfortunately, the partial product array arising from a Booth Radix-4 encoding is not symmetric, as only one operand gets encoded. Applying truncation to a Booth Radix-4 multiplier, results in a non-commutative implementation [5], [6]. Compiler optimizations routinely implicitly assume mathematical properties of underlying hardware. Application level correctness may implicitly require monotonicity of a complex function say, but may not have been considered during the hardware design. But commutativity is a far greater pervasive assumption, to the extent that compiler engineers could not conceive that non-commutative multipliers could even be built. Preserving commutativity significantly reduces compiler complexity for this most fundamental of operations.

In this work, we first demonstrate how, for minimal hardware overhead, the commutativity property of a truncated Booth implementation can be recovered. We then analytically derive tight error bounds on Booth array truncation and describe necessary and sufficient conditions to implement a faithful rounding. Lastly, we describe a procedure to construct efficient hardware implementations. The result is a fully parameterizable faithfully rounded multiplier

design that exhibits better power and area than alternative approaches.

The paper is organized as follows. In Section 2 we discuss prior work on hardware efficient multiplier, with a particular focus on works that exploit error freedom. In Section 3 we derive compensation terms to recover commutativity and prove a set of mathematical bounds on truncation error. We then demonstrate how we can use this information to design an efficient faithfully rounded commutative Booth multiplier. In Section 4 we compare synthesis results for a range of different implementations and discuss our approach to verifying these multiplier designs.

The paper contains the following novel contributions:

- a precise description of the compensation hardware required to recover the commutativity property,
- proven mathematical bounds on the maximal error due to truncation of Booth partial product arrays,
- a procedure to construct faithfully rounded commutative truncated Booth multipliers with maximal truncation.

2. Background

2.1. Binary Multiplication

The most naive implementation of n-bit binary multiplication via primitive logic gates [4] consists of first forming a partial product (PP) array of n^2 PP bits, where each PP bit is the logical AND of two input bits. Next that PP array is reduced from n rows to two rows via compressor cells arranged in a reduction tree. Once reduced to a summation of two rows, a full parallel adder can be deployed, usually to produce a $2n$-bit result. Decades of research has led to efficient implementations of all stages of this approach. Array reduction was studied by Wallace and Dadda [1], [2] then more recently improved with timing driven compressor-tree construction [7], [8]. Parallel addition in ASIC design is now most commonly implemented via parallel prefix structures [3], [8]. In this work, we will be entirely focused upon the first step, the construction of the PP array, and rely on existing techniques for the efficient implementation of array reduction and parallel addition.

One of the most widely used PP array creation techniques, is to add a Booth encoding step, which groups bits together to reduce the number of rows in the PP array. As observed by Zimmerman [8], the overhead introduced by the additional encoding step only offers a net benefit at larger bitwidths, beyond 16-bits. The two primary variants are the Booth Radix-4 and Radix-8 encoding schemes. This paper modifies the Booth Radix-4 multiplication method, so we will describe it in detail. We define the following encoding function taking three single bit operands:

$$B(x, y, z) = -2x + y + z \tag{1}$$

For an n-bit signed multiplication, Booth Radix-4 encoding halves the PP array height:

$$a \times b = \sum_{i=0}^{n-1} 2^i a_i \times b \tag{2}$$

$$= \sum_{i=0}^{(n/2)-1} 4^i B(a_{2i+1}, a_{2i}, a_{2i-1}) \times b, \tag{3}$$

where $a_{-1} = 0$ and we assume an even n. Each PP row in (3) can be efficiently implemented using primitive logic gates. Minor modifications are required for odd n and unsigned multiplication [4].

2.2. Faithfully Rounded Binary Multiplication

The implementation of binary multiplication where the full result is not required is most commonly achieved via truncation schemes [9], [10], [11]. These schemes follow a similar structure to compute n-bit $a \times b$. First, truncate the partial product array, removing the k least significant columns, corresponding to an error value of Δ_k. To the remaining array add a compensation term, $f(a, b)$, to the k^{th} column and then perform standard array reduction. From this summation result a further $n - k$ columns can be truncated to recover a faithfully rounded multiplication. Early work in this domain, described as Constant Correction Truncated schemes (CCT), started from an AND array and considered constant $f(a, b)$ [12], [13]. An example of CCT is shown in Figure 1. Later work introduced Variable Correction Truncation, where $f(a, b)$ was considered to be a function of the inputs [14]. Other works considered linearization schemes [15] and approximate carry predictions [16]. Booth array truncation has similarly seen CCT techniques applied along with a range of statistical methods to approximate the expected truncation error [5], [6], [9]. These previous Booth truncation schemes have broken the commutativity property of multiplication, a property that, as we show in this work, can be recovered for minimal hardware overhead.

Faithfully rounded truncated multipliers are most commonly applied in Digital Signal Processing (DSP) but also can be found in floating point multipliers, where a lower accuracy can be permitted [17]. For transcendental function approximations, it is rarely necessary to compute full precision multiplication, since we already have the approximation error to factor in [18], [19].

3. Methodology

The cause of the non-commutativity of truncated Booth multiplier architectures stems from the asymmetry of how the multiplier and multiplicand are treated; namely that only one of the inputs is Booth encoded. Naturally to maintain the hardware benefits of Booth architectures and remain commutative, the solution is to Booth encode both inputs. For ease of exposition we will assume that both inputs are signed two's complement, have even bitwidth n. We focus on a Booth Radix-4 architecture. It is simple to extend

the analysis presented here to odd n and unsigned multiplication. Given these assumptions, *double* Booth encoding results in the following:

$$a \times b = \sum_{i,j=0}^{(n/2)-1} 4^{i+j} PP_{i,j} \quad (4)$$

$$PP_{i,j} = B(a_{2i+1}, a_{2i}, a_{2i-1}) \times B(b_{2j+1}, b_{2j}, b_{2j-1}), \quad (5)$$

where $a_{-1} = b_{-1} = 0$. The key observation to achieving commutative truncated Booth architectures is to truncate (4) directly. Ultimately we will target creating a faithfully rounded result returning the most significant n bits. By truncating k columns, a portion of the summation, Δ, will be deleted and the remainder, M, will be implemented as a partial-product array such that $a \times b = M + \Delta$.

$$M = \sum_{i,j=0, i+j \geq k/2}^{(n/2)-1} 4^{i+j} PP_{i,j}$$

$$\Delta = \sum_{i,j=0, i+j < k/2}^{(k/2)-1} 4^{i+j} PP_{i,j} \quad (6)$$

Note that the proposed architecture and analysis assumes $k < n$ such that Δ is strictly triangular in shape. (Hence the proposed architecture can be extended to return a faithfully rounded result with m bits as long as $m > n$.) The first challenge is how M is transformed into a binary array and the second is analyzing the error of Δ.

3.1. Commutative Truncated Booth Arrays

To begin reducing M to a binary PP array, we can undo the Booth encoding to one of the inputs:

$$M = \sum_{i=0}^{(n/2)-1} 4^i B(a_{2i+1}, a_{2i}, a_{2i-1}) \times bb_i \quad (7)$$

$$bb_i = (-2^{n-k+2i-1} b_{n-1} + b_{n-2:k-2i} + b_{k-2i-1}),$$

where $b_j = 0$ if $j < 0$ and $b_{n:m}$ denotes the bit slice of $b[n:m]$. Now the terms in the summation only differ from a standard Booth Radix-4 summation due to the presence of the b_{k-2i-1} term. Let us consider a standard Booth Radix-4 PP row row_i and contrast this with one of the terms in (7), row'_i. Such rows are of the form (for some c_i):

$$row_i = B(a_{2i+1}, a_{2i}, a_{2i-1})(-2^{n-2} c_{n-1} + c_{n-2:1})$$
$$row'_i = B(a_{2i+1}, a_{2i}, a_{2i-1})(-2^{n-2} c_{n-1} + c_{n-2:1} + c_0) \quad (8)$$

Now the bit level construction of row_i and row'_i can be found in Table 1 (note that for two's complement c and bit c_0, $-c - c_0 = \overline{c} + 1 - c_0 = \overline{c} + \overline{c_0}$).

Note that row_i and row'_i can both be expressed in the form integer pp plus bit s. Moreover truncated pp_i has identical values to truncated pp'_i, the key and only difference between a truncated Booth Radix-4 array and commutative

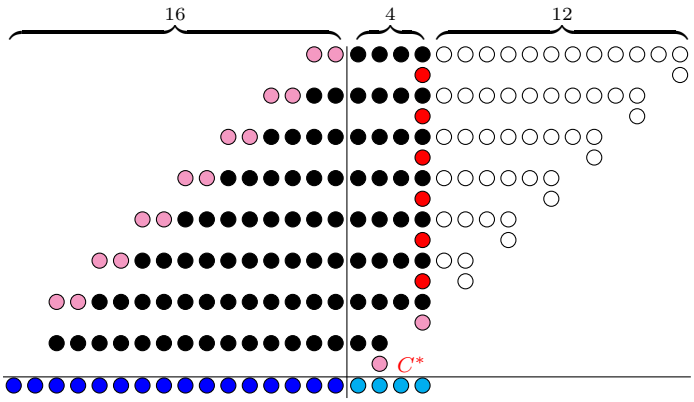

Figure 2. A 16-bit commutative truncated Booth multiplier, with 12 columns of truncation. The six red bits are the additional compensation bits s'_i. The pink bits represent the typical Booth sign bits s_i.

array are the s'_i bits. The Boolean expressions for s_i and s'_i, derivable from Table 1 are:

$$s_i = a_{2i+1} \& \overline{a_{2i} \& a_{2i-1}}$$
$$s'_i = a_{2i+1} \& \overline{a_{2i} \& a_{2i-1}} \& (c_0 \oplus a_{2i+1}) \quad (9)$$

We can now present the steps in constructing a commutative truncated Booth Radix-4 array:

1) Construct standard Booth Radix-4 array
2) Remove least significant k columns, k is even
3) In column k, for $i \in [0, \frac{k}{2} - 1]$, add additional bits

$$s'_i = a_{2i+1} \& \overline{a_{2i} \& a_{2i-1}} \& (b_{k-2i-1} \oplus a_{2i+1}).$$

The difference between a commutative and a non-commutative array is just the inclusion of these $k/2$ bits. Since these s' bits increase the array height of the least significant column, it is natural to ask whether this will affect the critical path.

Consider any array reduction that performs the summation of m addends of large length k. The greatest number of carries that the summation generates will occur when the addends are maximal, evaluating $m(2^k - 1) = (m-1)2^k + (2^k - m)$. For sufficiently large k, $2^k - m > 0$ and hence such a summation will generate at most $m - 1$ carries. This means the implementations are capable of dealing with $m-1$ carries and hence accepting $m - 1$ additional carry-in bits in its least significant column without significantly altering its delay characteristics.

The maximum array height in Figure 2 is $\approx n/2$, the number of additional carry ins such an array can handle is $< n/2$, there are $k/2$ additional s'_i bits which is by assumption on k, $< n/2$. Hence such commutative truncated Booth Radix-4 arrays are expected to exhibit minimal delay differences when compared to untruncated arrays.

It is important to note that while there are but $k/2$ s' bits additional bits required to make the truncated Booth Radix-4 array commutative and such inclusion is expected to have limited delay impact; these bits are in *no way* trivial. Omitting any one of them is likely to still produce

TABLE 1. PARTIAL PRODUCT CREATION FOR COMMUTATIVE TRUNCATED BOOTH RADIX-4 ARRAY

a_{2i+1}	a_{2i}	a_{2i-1}	$-2a_{2i+1}+a_{2i}+a_{2i-1}$	$row_i = pp_i + s_i$	$row'_i = pp'_i + s'_i$
0	0	0	0	0	0
0	0	1	1	$\{c_{n-1}, c_{n-1}...c_2, c_1\}+0$	$\{c_{n-1}, c_{n-1}...c_2, c_1\}+c_0$
0	1	0	1	$\{c_{n-1}, c_{n-1}...c_2, c_1\}+0$	$\{c_{n-1}, c_{n-1}...c_2, c_1\}+c_0$
0	1	1	2	$\{c_{n-1}, c_{n-2}...c_1, 0\}+0$	$\{c_{n-1}, c_{n-2}...c_1, c_0\}+c_0$
1	0	0	-2	$\{\overline{c_{n-1}}, \overline{c_{n-2}}...\overline{c_1}, 1\}+1$	$\{\overline{b_{n-1}}, \overline{c_{n-2}}...\overline{c_1}, \overline{c_0}\}+\overline{c_0}$
1	0	1	-1	$\{\overline{c_{n-1}}, \overline{c_{n-1}}...\overline{c_0}, \overline{c_0}\}+1$	$\{\overline{b_{n-1}}, \overline{c_{n-1}}...\overline{c_2}, \overline{c_1}\}+\overline{c_0}$
1	1	0	-1	$\{\overline{c_{n-1}}, \overline{c_{n-1}}...\overline{c_0}, \overline{c_0}\}+1$	$\{\overline{b_{n-1}}, \overline{c_{n-1}}...\overline{c_2}, \overline{c_1}\}+\overline{c_0}$
1	1	1	0	0	0

a faithfully rounded implementation, but it will not be commutative. These s' bits are not present in the original Booth Radix-4 array and any truncation or promotion of bits within the array will *not* result in a commutative array. Such compensation bits would need to be rederived for every Booth architecture variant. Moreover commutativity was achieved by truncating (4), not standard Booth summation formulae.

3.2. The Truncation Error

In order to create optimal faithfully rounded multipliers, the value range of Δ must be precisely known. This analysis will be facilitated by the following helper functions, representing hexadecimal summations:

$X_n =$	$Y_n =$	$Z_n =$	$W_n =$
$\overbrace{\texttt{222..222}}^{n}$	$\overbrace{\texttt{222..222}}^{n}$	$\overbrace{\texttt{000..000}}^{n}$	$\overbrace{\texttt{444..444}}^{n}$
	+444..444	+444..444	+444..444
+444..440	+444..440	+444..440	+444..440
+444..440	+444..440	+444..440	+444..440
+444..400	+444..400	+444..400	+444..400
+444..400	+444..400	+444..400	+444..400
...	...	...	...
+440..000	+440..000	+440..000	+440..000
+440..000	+440..000	+440..000	+440..000
+400..000	+400..000	+400..000	+400..000
+400..000	+400..000	+400..000	+400..000

These hexadecimal summations can be simplified. Consider reducing X_n:

$\overbrace{\texttt{222..222}}^{n}$	$= \frac{2}{15}\texttt{FFF..FFF}$	$= \frac{2}{15}\overbrace{\texttt{0FFF..FFF}}^{n+1}$
+444..440	$-\frac{8}{15}\texttt{111..110}$	$-\frac{8}{15^2}\texttt{0FFF..FF0}$
+444..440	$+\frac{8}{15}\texttt{111..110}$	$+\frac{8(n-1)}{15^2}\texttt{1000..000}$
+444..400	$+\frac{8}{15}\texttt{FFF..FF0}$	
+444..400	$+\frac{8}{15}\texttt{FFF..F00}$	
...	...	
+440..000	$+\frac{8}{15}\texttt{FF0..000}$	
+440..000		
+400..000	$+\frac{8}{15}\texttt{F00..000}$	
+400..000		

Therefore,

$$\begin{aligned} X_n &= \frac{2}{15}(16^n - 1) - \frac{8}{15^2}(16^n - 16) + \frac{8}{15}(n-1)16^n \\ &= \frac{2}{225}(16^n(60n - 49) + 49). \end{aligned}$$

Similarly the remaining helper functions can be reduced to:

$$\begin{aligned} Y_n &= \frac{2}{225}(16^n(60n - 19) + 19) \\ Z_n &= \frac{4}{225}(16^n(30n - 17) + 17) \\ W_n &= \frac{8}{225}(16^n(15n - 1) + 1) \end{aligned}$$

Lemma 1. *Extremal values of Δ occur when $a_i \neq a_{i-2}$ and $b_j \neq b_{j-2}$ for all $n > i, j > 1$.*

Proof. Consider the contribution of a_0 and a_2 in Δ:

$$\begin{aligned} \Delta =& (-2^{k-1}b_{k-1} + b_{k-2:0})a_0 \\ &+ 4(-2^{k-3}b_{k-3} + b_{k-4:0})a_2 + ... \\ =& -2^{k-1}(b_{k-1}a_0 + b_{k-3}a_2) \\ &+ 2^{k-2}(b_{k-2}a_0 + b_{k-4}a_2) + ... \\ &+ 4(b_2a_0 + b_0a_2) + ... \end{aligned}$$

For $a_0 = a_2 = 1$, extremal Δ occurs when $b_{k-1:2} = b_{k-3:0}$. If $a_0 = a_2 = 0$ made Δ extremal this would imply $b_{k-1:2} = b_{k-3:0}$. But if $a_0 = a_2$ implies $b_{k-1:2} = b_{k-3:0}$ and hence $b_2 = b_0$ then by a symmetric argument this would imply $a_{k-1:2} = a_{k-3:0}$. Up to input reordering, this implies the following extremal inputs:

TABLE 2. Δ WORST CASE INPUTS $a_i = a_{i-2}$ AND $b_j = b_{j-2}$

a	b	Δ
111...111	111...111	1
111...111	101...010	$\frac{2^k+2}{3}$
111...111	010...101	$-\frac{2^k-1}{3}$
101...010	101...010	$\frac{1}{18}((3k+16)2^k+16)$
101...010	010...101	$-k2^{k-2} - \frac{1}{3}(2^k-1)$
010...101	010...101	$\frac{1}{18}((3k-2)2^k+2)$
000...000	000...000	0

Alternatively if $a_0 \neq a_2$, swapping the value of a_0 with a_2 will produce the largest change to Δ if $b_{k-1:2} \neq b_{k-3:0}$. But if $a_0 \neq a_2$ implies $b_{k-1:2} \neq b_{k-3:0}$ and hence $b_2 \neq b_0$

then by a symmetric argument this would imply $a_{k-1:2} \neq a_{k-3:0}$.

As an example, consider $k/2$ is odd and $a = b =$ 10011001...100110:

$$\begin{aligned}\Delta &= \sum_{i,j=0,i+j<k/2}^{(k/2)-1} 4^{i+j} PP_{i,j} \\ &= -\sum_{i,j=0,i+j<k/2}^{(k/2)-1} (-4)^{i+j+1} \\ &= Z_{(k+2)/4} - 4W_{(k-2)/4} \\ &= \frac{2}{25}(2^k(5k+2)+2)\end{aligned}$$

Similarly, the helper functions can be used to simplify Δ for all the cases where $a_i \neq a_{i-2}$ and $b_j \neq b_{j-2}$ for all $n > i, j > 1$, four of these cases are:

TABLE 3. Δ WORST CASE INPUTS $a_i \neq a_{i-2}$ AND $b_j \neq b_{j-2}$

$k/2$	a	b	Δ
even	10011001...1001	01100110...0110	$4Y_{\frac{k}{4}} - X_{\frac{k}{4}}$
odd	10011001...10	10011001...10	$Z_{\frac{k+2}{4}} - 4W_{\frac{k-2}{4}}$
even	01100110...0110	01100110...0110	$Z_{\frac{k}{4}} - 4W_{\frac{k}{4}}$
odd	10011001...10	01100110...01	$-X_{\frac{k+2}{4}} + 4Y_{\frac{k-2}{4}}$

The remaining cases produce less extremal Δ values. These four cases can be combined and simplified using the helper function to conclude:

$$\begin{aligned}-\frac{1}{25}(2^k(10k+5(-1)^{k/2}-1)-5+(-1)^{k/2} \\ \leq \Delta \leq \\ \frac{1}{25}(2^k(10k-5(-1)^{k/2}-1)+5+(-1)^{k/2}\end{aligned} \quad (10)$$

Note that the leading coefficient of $k2^k$ in these bounds are $2/5$ versus $1/6$ and $1/4$ in Table 2 . Conclude that the extremal values of Δ occur when $a_i \neq a_{i-2}$ and $b_j \neq b_{j-2}$ for all $n > i, j > 1$. The worst case input vectors can be found in Table 3 and (10) contains tight bounds on Δ. $\square$

3.3. Faithfully Rounded Commutative Truncated Booth Arrays

The optimal truncation scheme will truncate the most number of columns while maintaining the faithful rounding condition. The truncated array M will have an additional constant C added to compensate for the loss of Δ. Once summed, the least significant bits (value D) will be truncated before returning the approximation.

$$\begin{aligned}&y = a \times b = M + \Delta \quad \text{Unrounded Result} \\ &y' = M + C - D \quad \text{Rounded Result} \\ &|y - y'| < 2^n \quad \text{Faithful Rounding Condition} \\ &\Rightarrow C - D - 2^n < \Delta < C - D + 2^n\end{aligned}$$

D can take any value between 0 and $2^n - 2^k$, hence necessary and sufficient condition for faithful rounding is:

$$\begin{aligned}C - 2^n &< \Delta < C + 2^k \\ \max(\Delta) - 2^k &< C < \min(\Delta) + 2^n\end{aligned}$$

Now C is a multiple of 2^k, hence:

$$\left\lfloor \frac{\max(\Delta)}{2^k} \right\rfloor \leq \left\lfloor \frac{\min(\Delta)}{2^k} \right\rfloor + 2^{n-k}$$

Substituting the extremal found in (10) and simplifying results in:

$$\left\lfloor \frac{2k-(-1)^{k/2}}{5} \right\rfloor + \left\lceil \frac{2k+(-1)^{k/2}}{5} \right\rceil < 2^{n-k} \quad (11)$$

Maximising k in (11) generates the optimal truncation value. The optimal truncation scheme values can now be presented:

$$k^* = \max_{\text{even } k} \left(k \leq 5 \times 2^{n-k-2}\right) \quad (12)$$

$$\frac{2k^* - 5 - (-1)^{\frac{k^*}{2}}}{5} \leq C^* \leq \frac{5 \times 2^{n-k^*} - 2k^* - (-1)^{\frac{k^*}{2}}}{5} \quad (13)$$

Example values of k^* and C^*:

n	k^*	$\min C^*$	$\max C^*$
8	4	1	14
16	12	4	11
24	20	7	7
32	26	10	53
53	46	18	109
64	58	23	41

We can now present the steps in constructing a commutative truncated Booth Radix-4 array for an n multiplication returning a faithful rounding of the n most significant bits:

1) Construct standard Booth Radix-4 array, Booth encoding a
2) Calculate $k^* = \max_{\text{even } k}(k \leq 5 \times 2^{n-k-2})$
3) Remove least significant k^* columns
4) In column k^*, for $i \in [0, \frac{k^*}{2} - 1]$, add additional bits
$$s'_i = a_{2i+1} \& \overline{a_{2i} \& a_{2i-1}} \& (b_{k^*-2i-1} \oplus a_{2i+1}).$$
5) In column k^*, include constant C^* which can be any value in the range:
$$\left[\frac{2k^* - 5 - (-1)^{\frac{k^*}{2}}}{5}, \frac{5 \times 2^{n-k^*} - 2k^* - (-1)^{\frac{k^*}{2}}}{5} \right]$$
6) Sum the array
7) Remove the least significant n columns

TABLE 4. SYNTHESIS RESULTS FOR THREE COMPETING ARCHITECTURES AT SEVERAL DELAY TARGETS, ACROSS FOUR DIFFERENT BITWIDTHS, n. THE PERCENTAGE IMPROVEMENTS ARE WITH RESPECT TO THE BASELINE IMPLEMENTATION. WE HIGHLIGHT THE BEST RESULT IN EACH ROW IN BOLD.

		Baseline		Truncated AND [9]		Commutative Truncated Booth	
n	Delay (ns)	Area (μm^2)	Power (μW)	Area (μm^2)	Power (μW)	Area (μm^2)	Power (μW)
16	0.175	74.0	450	**64.4** (*-13.0%*)	**319** (*-29.2%*)	69.3 (*- 6.4%*)	389 (*-13.7%*)
	0.2	56.1	346	**46.4** (*-17.4%*)	**252** (*-27.3%*)	48.3 (*-13.9%*)	286 (*-17.5%*)
	0.225	51.7	309	**40.9** (*-20.8%*)	**215** (*-30.4%*)	43.6 (*-15.8%*)	252 (*-18.5%*)
	0.25	47.5	283	**37.1** (*-21.9%*)	**198** (*-30.1%*)	39.7 (*-16.3%*)	226 (*-20.1%*)
24	0.225	127.7	815	108.6 (*-14.9%*)	622 (*-23.7%*)	**99.9** (*-21.8%*)	**600** (*-26.4%*)
	0.25	113.4	705	101.5 (*-10.4%*)	575 (*-18.4%*)	**88.7** (*-21.8%*)	**516** (*-26.8%*)
	0.275	108.0	653	93.5 (*-13.4%*)	502 (*-23.2%*)	**84.9** (*-21.3%*)	**498** (*-23.8%*)
	0.3	99.2	595	83.8 (*-15.6%*)	466 (*-21.8%*)	**77.3** (*-22.1%*)	**447** (*-25.0%*)
32	0.275	193.7	1225	161.3 (*-16.7%*)	897 (*-26.7%*)	**156.2** (*-19.4%*)	**914** (*-25.4%*)
	0.3	171.5	1097	156.3 (*- 8.8%*)	861 (*-21.5%*)	**148.0** (*-13.7%*)	**862** (*-21.4%*)
	0.325	164.3	1024	139.7 (*-15.0%*)	787 (*-23.1%*)	**134.2** (*-18.3%*)	**768** (*-25.0%*)
	0.35	152.9	945	137.2 (*-10.3%*)	772 (*-18.3%*)	**130.7** (*-14.5%*)	**748** (*-20.8%*)
64	0.3	823.0	5548	647.6 (*-21.3%*)	3808 (*-31.4%*)	**566.8** (*-31.1%*)	**3527** (*-36.4%*)
	0.325	745.5	4886	593.3 (*-20.4%*)	3403 (*-30.4%*)	**515.6** (*-30.8%*)	**3073** (*-37.1%*)
	0.35	709.2	4558	565.1 (*-20.3%*)	3165 (*-30.6%*)	**491.7** (*-30.7%*)	**2846** (*-37.6%*)
	0.375	638.6	4187	513.1 (*-19.7%*)	2936 (*-29.9%*)	**446.9** (*-30.0%*)	**2593** (*-38.1%*)

4. Results

We compare three implementations of faithfully rounded commutative n-bit multipliers returning an n-bit result. The baseline is a round to zero multiplier, implemented by computing the full precision n-bit multiplication, generating a $2n$-bit result, from which we return the n most significant bits. The second implementation is a truncated AND array, where we follow the approach in [9] to compute the maximum possible truncation and an efficient constant compensation term. We do not compare against alternative truncated Booth implementations since these designs do not retain commutativity.

4.1. Synthesis

We synthesize each design using a commercial logic synthesis tool, targeting a standard TSMC 5nm library. We present results for a range of bitwidths and at a number of delay targets for each parameterization. The combinational area and total power consumption results reported by the logic synthesis tool are shown in Table 4. We also present the percentage improvement in each metric with respect to the baseline. We synthesize multipliers at relevant bitwidths between 16 and 64 bits. At lower bitwidths the overhead of Booth encoding is detrimental [8] and at higher bitwidths multiplier decomposition techniques, such as Karatsuba [20], dominate [21].

In Table 4 we can see that the truncated Booth array is up to 31% smaller than the baseline implementation and consumes up to 38% less power. Furthermore, the truncated Booth array is up to 13% smaller than the truncated AND array and consumes up to 12% less power. For 16-bit multiplication, we see that the truncated AND array is both smaller and more power efficient across all delay targets. For 24-bit multiplication, the truncated Booth multiplier is superior. Prior work on exact multiplier implementations also observed an architectural crossover point around 16-bits [8], at which the overhead of Booth encoding is offset by the gains in array reduction. As we increase the bitwidth, the benefit offered by the truncated Booth array over the alternatives increases.

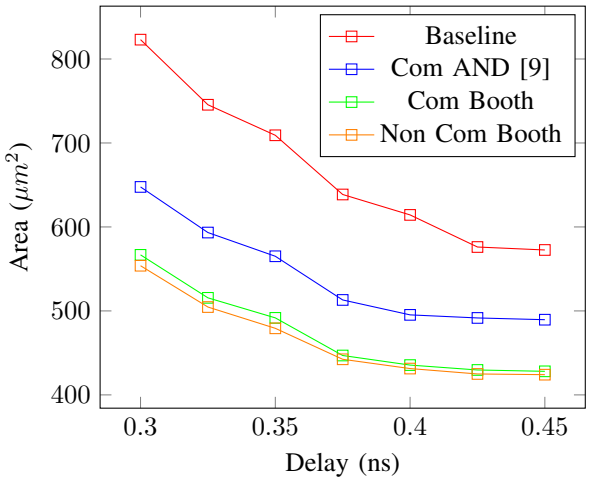

Figure 3. Area-delay profiles for the three competing commutative designs for n=64. We also plot the truncated Booth architecture without the compensation bits to recover commutativity.

In Figure 3 we present complete area-delay profiles for the competing 64-bit multiplier implementations. Across the complete delay spectrum the truncated Booth multiplier demonstrates roughly constant area reduction when compared against the truncated AND array.

To understand the penalty we must incur to recover commutativity, we also synthesize the 64-bit truncated Booth array without the additional compensation bits described in Section 3.1. As we can see in Figure 3, the area difference between the commutative and non-commutative truncated Booth implementations is minimal across the delay spectrum. At worst we pay a 2.5% area penalty.

TABLE 5. ACL2 RUNTIMES FOR PROVING THAT THE TRUNCATED BOOTH MULTIPLIER IMPLEMENTS A FAITHFUL ROUNDING AND IS COMMUTATIVE. THE DASH INDICATES A PROOF WHICH DID NOT CONVERGE.

n	Faithful (sec)	Commutative (sec)
4	3	0.6
6	4	0.6
16	7	0.7
24	44	0.7
32	63	0.9
36	117	1.0
42	835	1.1
64	–	2.0

4.2. Formal Verification

For multiplier implementations beyond 16-bit, formal verification becomes challenging as the number of inputs exceeds what can be simulated. Furthermore, the verification of custom multiplier implementations is particularly challenging due to the circuit complexity, leading to bespoke tools [22], [23] and methods [24], [25].

In this work, we deploy the S-C-Rewriting method [26], [27] built on the ACL2 theorem prover [28], supported by the Glucose SAT solver. We use the tool to prove that the output, `out`, of the Verilog code implementing a truncated Booth multiplier satisfies the following lemma:

```
mult = a[n-1:0]*b[n-1:0]
lsbs = mult[  n-1:0]
msbs = mult[2*n-1:n]

lemma (lsbs==0) ? out==msbs
                : 0<=out-msbs<=1
```

Where `a` and `b` are n-bit, and `mult` is $2n$-bit wide unsigned bit-vectors representing integer values. This lemma guarantees that `out` is a faithful rounding. We also use the same tool to prove commutativity of the truncated Booth multiplier.

We prove the two properties for a range of bitwidths n and present the proof runtimes in Table 5. Unfortunately, as the bitwidth n increases, the proof of faithful rounding runtimes grow exponentially, meaning we are unable to prove the correctness of the 64-bit truncated Booth multiplier.

5. Conclusion

This paper provides the first implementation of a commutative truncated Booth multiplier, that produces a faithfully rounded result. We first described how, for minimal circuit area overhead, we can recover commutativity, via the introduction of a small number of compensation bits. We then proved exact bounds on the maximal error due to Booth array truncation and used these bounds to calculate the maximum number of columns which can be truncated. Lastly, we described how the addition of a constant can compensate for the error induced by truncation. We synthesized a number of faithfully rounded multiplier implementations and were able to reduce circuit area by up to 13% and reduce power consumption by up to 12% when compared to the state of the art. Using an ACL2 based verification tool, we were able to prove the correctness of the commutative truncated Booth multipliers up to 42 bits.

Future work will look to generalize the results here to arbitrary Booth encoding radices, e.g. Booth Radix-8. A further generalization will consider different error thresholds, as opposed to the faithful rounding considered here. We will also address the limitations of our verification, exploring proof decomposition techniques. Lastly, an approach in [9] can be used to incorporate these multiplier components into larger hardware designs, for example a floating-point multiplier.

References

[1] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, 1964.

[2] L. Dadda, "Some schemes for parallel multipliers," *Alta frequenza*, vol. 34, pp. 349–356, 1965.

[3] R. Zimmermann and D. Q. Tran, "Optimized synthesis of sum-of-products," in *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, vol. 1, 2003.

[4] M. D. Ercegovac and T. Lang, *Digital arithmetic*. Elsevier, 2004.

[5] H. A. Huang, Y. C. Liao, and H. C. Chang, "A self-compensation fixed-width booth multiplier and Its 128-point FFT applications," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2006.

[6] Y. H. Chen, T. Y. Chang, and R. Y. Jou, "A statistical error-compensated Booth multipliers and its DCT applications," in *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, 2010.

[7] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Transactions on Computers*, vol. 45, no. 3, 1996.

[8] R. Zimmermann, "Datapath synthesis for standard-cell design," in *Proceedings - Symposium on Computer Arithmetic*, 2009.

[9] T. A. Drane, T. M. Rose, and G. A. Constantinides, "On the systematic creation of faithfully rounded truncated multipliers and arrays," *IEEE Transactions on Computers*, vol. 63, no. 10, 2014.

[10] K. J. Cho, S. M. Lee, S. H. Park, and J. G. Chung, "Error bound reduction for fixed-width modified booth multiplier," in *Conference Record - Asilomar Conference on Signals, Systems and Computers*, vol. 1, 2004.

[11] H. J. Ko and S. F. Hsiao, "Design and application of faithfully rounded and truncated multipliers with combined deletion, reduction, truncation, and rounding," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 5, 2011.

[12] M. J. Schulte and E. E. Swartzlander, "Truncated multiplication with correction constant [for DSP]," in *Proceedings of IEEE Workshop on VLSI Signal Processing VI, VLSISP 1993*, 1993.

[13] S. S. Kidambi and P. El-Guibaly, "Area-efficient multipliers for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 2, 1996.

[14] E. J. King and E. E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, vol. 2, 1998.

[15] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Designof fixed-width multipliers with linear compensation function," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 5, 2011.

[16] R. Michard, A. Tisserand, and N. Veyrat-Charvillon, "Carry prediction and selection for truncated multiplication," in *2006 IEEE Workshop on Signal Processing Systems Design and Implementation, SIPS*, 2006.

[17] K. E. Wires, M. J. Schulte, and J. E. Stine, "Variable-correction truncated floating point multipliers," in *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, vol. 2, 2000.

[18] B. Orloski, S. Coward, and T. Drane, "Automatic Generation of Complete Polynomial Interpolation Design Space for Hardware Architectures," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. Tokyo: Association for Computing Machinery, 2023, pp. 573–578.

[19] D. De Caro, E. Napoli, D. Esposito, G. Castellano, N. Petra, and A. G. Strollo, "Minimizing Coefficients Wordlength for Piecewise-Polynomial Hardware Function Evaluation With Exact or Faithful Rounding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 5, 2017.

[20] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Doklady Akademii Nauk*, vol. 145, no. 2, 1962, pp. 293–294.

[21] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, "IMpress: Large Integer Multiplication Expression Rewriting for FPGA HLS," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–10.

[22] A. Koelbl, R. Jacoby, H. Jain, and C. Pixley, "Solver technology for system-level to RTL equivalence checking," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2009.

[23] J. R. Burch, "Using BDDs to verify multipliers," in *Proceedings - Design Automation Conference*, 1991.

[24] R. Kaivola and N. Narasimhan, "Formal verification of the Pentium®4 floating-point multiplier," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2002.

[25] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining SAT and computer algebra," in *Proceedings of the 19th Conference on Formal Methods in Computer-Aided Design, FMCAD 2019*, 2019.

[26] M. Temel, A. Slobodova, and W. A. Hunt, "Automated and Scalable Verification of Integer Multipliers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12224 LNCS, 2020.

[27] M. Temel and W. A. Hunt, "Sound and Automated Verification of Real-World RTL Multipliers," in *Proceedings of the 21st Formal Methods in Computer-Aided Design, FMCAD 2021*, 2021.

[28] M. Kaufmann and J. S. Moore, "ACL2: An industrial strength version of Nqthm," in *COMPASS - Proceedings of the Annual Conference on Computer Assurance*, 1996.