Tree and Array Multipliers
Ivor Page[1]

10.1 Tree Multipliers

In Figure 1 seven input operands are combined by a tree of CSAs. The final level of the tree is a carry-completion adder, probably a CLA-based design. In the tree multiplier k/b input operands are combined, where $b = 2$, 4, or 8 is the advantage gained by Booth recoding of the multiplier.

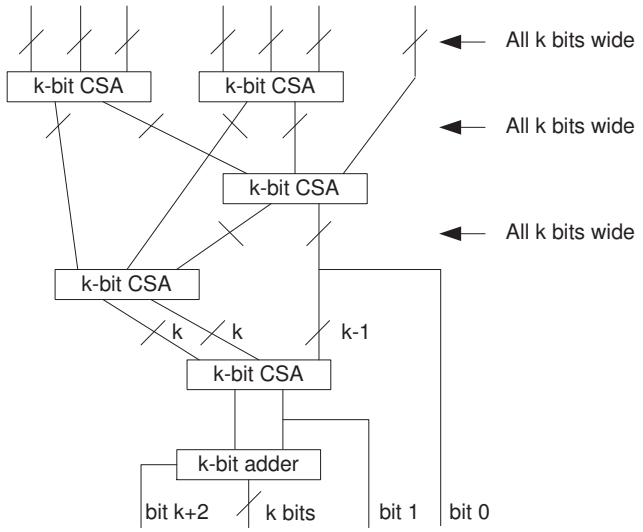

Figure 1: Tree Multiplier

Wallace or Dadda tree designs can be used, but the resulting layouts are irregular with varying signal path lengths and delays. An alternative design makes use of basic building blocks that comprise two CSAs to reduce 4-inputs to 2-outputs. These 4-to-2 blocks are used to form a simple binary tree that has a regular structure that is easy to layout. See Figure 2. The resulting binary tree has more CSA units than a Wallace or Dadda tree design, but its regular layout makes it attractive.

[1]University of Texas at Dallas

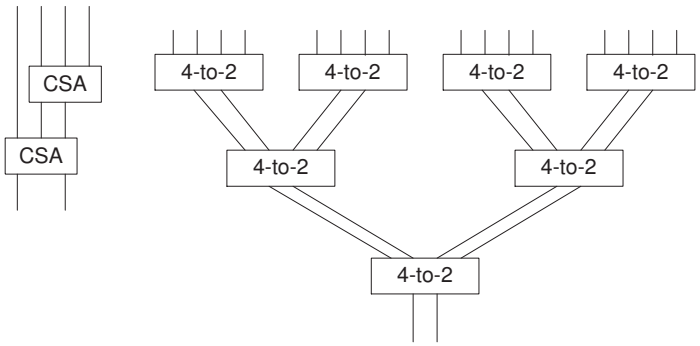

Figure 2: 4-to-2 Building Block and Associated Binary Tree Multiplier

The partial products to be added are progressively left shifted as illustrated in Figure 3:

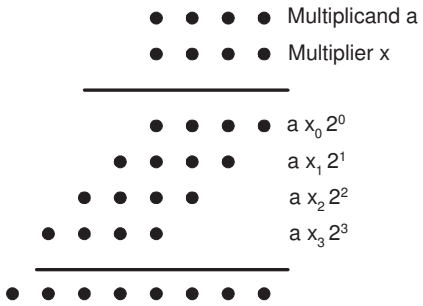

Figure 3: Arrangement of partial products

It would be wasteful to use CSAs of $k+3$ and more bits in a tree to add these partial products. Instead, each column of the array of partial products can be added with a separate tree of single-bit CSAs, or full-adders. Beginning in the 2^2 column and moving to the left, columns of 3, 4, $\cdots k$, $k-1, \cdots$, 4, 3, 2, 1 bits must be reduced to just two bits in each column.

As we saw in the last module, carry-in signals to each column add to these inputs. Wallace or Dadda trees can be used in these columns, with the carry signals propagating to the left.

The full-adders of these designs each reduce 3 input bits to two outputs, providing a log-depth multiplier, but an irregular layout with signal paths of varying lengths and varying delays.

Figure 4 shows a balanced-delay 11 input tree. Each output is valid after the same number of full-adder delays. This circuit must be replicated in each column in order to equalize the delay paths throughout the entire multiplier.

The number of inputs can easily be expanded to 18 by adding a column of seven full-adders to

the right of the diagram. These structures have good layouts, but only for specific numbers of inputs.

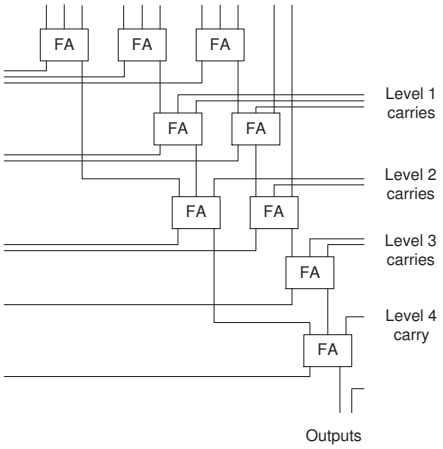

Figure 4: 11 Input Balanced Delay Counter

For comparison purposes, Figure 5 shows an 8-bit multiplier based on BSD redundant binary adders. In this layout the result is produced in the middle. As can be seen, approximately $3k/2$ connections pass vertically through the 3rd adder block. The $2k$ output connections and the k multiplicand inputs must also must be routed vertically in between the cells of the adder blocks.

The layout is normally arranged so that the result is produced at the bottom (see figure in the text). This increases the amount of pass-through connections. There are k^2 partial product digits to be added and the depth of the multiplier is $log\ k$. The area of the multiplier is $O(k^2 log\ k)$.

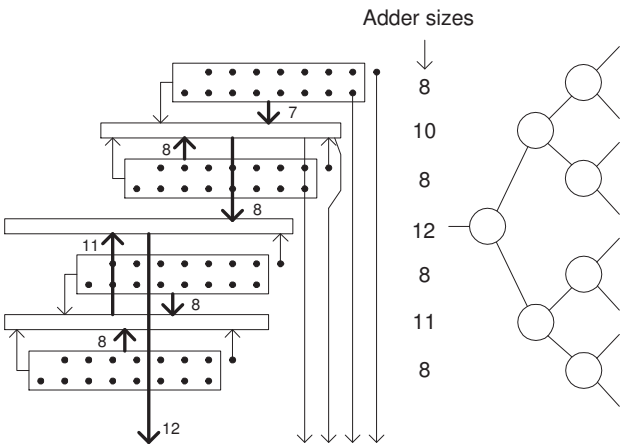

Figure 5: Multiplier Based on Redundant BSD numbers

Tagaki et.al.[2] gives the following table comparing the BSD multiplier with the standard array multiplier and one based on Wallace trees:

k	8	16	32	64	128
Array	29/528	61/2336	125/9792	253/40064	509/162048
Wallace	22/815	24/2939	30/9965	34/37423	40/142335
BSD	21/550	27/2213	31/8796	37/34388	41/135324

The elements of the table, are Depth/Gate Count. The design is based on a fan-in limit of 4. As can be seen, the differences between the BSD and Wallace tree versions are minimal. The only advantage that can be claimed for the BSD version is that the layout is more regular than in the Wallace Tree version. This may lead to better actual area and delay for the BSD version.

A layout similar to that of Figure 5 can be used for a multiplier based on the 4:2 units of Figure 2. Indeed, the 4:2 unit can be viewed as a GSD radix-2 adder for digit set $[0, 2]$ where the two outputs of each 4:2 unit have encoding: zero=(0,0), one=(0,1) or (1,0), two=(11). This observation makes the two designs very simple to compare. The area and delay of the units depends on the area and delay of the building blocks: BSD adder blocks vs. 4:2 reduction blocks. According to Takagi, a BSD adder block for k bits requires $13k$ or/nor gates. The technology used was emitter-coupled logic, which naturally provides two outputs, the positive output and its complement. If we are restricted to Nand gates, then several inverters would have to be added to this estimate. The k bit 4:2 adder block would require $18k$ 2-input gates. On this basis, it appears that the multiplier based on BSD blocks would have a slight area advantage, but the comparison is really too close to call.

[2]Naofumi Tagaki, Hiroto Yashuura, Shuzo Yajima, High Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree, IEEE Transactions on Computers, C-34,9 Sept. 1985

We therefore have at least three designs for multipliers that have log depth and comparable gate counts. The Wallace tree has the most complex and least efficient layout. All three designs can benefit from high radix Booth recoding, and pipeline registers can be inserted in all of the designs to speed throughput (without reducing delay time).

10.2 Tree Multipliers for Signed Numbers

Any form of Booth recoding, or the use of a redundant number system, will naturally take care of negative multiplier values. Negative multiplicands are also naturally catered for in a redundant system with symmetrical digit set (such as BSD). In any non-redundant tree (or array) multiplier it is still necessary to simulate sign extension of any negative partial products. The Baugh Wooley multiplier naturally takes care of both negative multipliers and multiplicands.

Here are some tables illustrating the development of the Baugh Wooley multiplier. The first is an array multiplier for unsigned values:

					a_4	a_3	a_2	a_1	a_0
			$\times$		x_4	x_3	x_2	x_1	x_0
					a_4x_0	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				a_4x_1	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			a_4x_2	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		a_4x_3	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	a_3x_4	a_2x_4	a_1x_4	a_0x_4				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

In the next example we make use of the fact that the value of a k bit 2's complement number is:

$$-x_{k-1}2^{k-1} + \sum_{i=0}^{k-2} x_i 2^i$$

If the sign bit of the multiplier, x_4, is 1, the multiplicand is subtracted in the final row.

					a_4	a_3	a_2	a_1	a_0
			$\times$		x_4	x_3	x_2	x_1	x_0
					$-a_4x_0$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$-a_4x_1$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$-a_4x_2$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$-a_4x_3$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	$-a_3x_4$	$-a_2x_4$	$-a_1x_4$	$-a_0x_4$				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

This table is impractical since it implies subtracting individual bits and adding others. The Baugh Wooley Multiplier uses only additions. Here is its table:

					a_4	a_3	a_2	a_1	a_0
			$\times$		x_4	x_3	x_2	x_1	x_0
					$a_4\overline{x_0}$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$a_4\overline{x_1}$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$a_4\overline{x_2}$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$a_4\overline{x_3}$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	$\overline{a_3}x_4$	$\overline{a_2}x_4$	$\overline{a_1}x_4$	$\overline{a_0}x_4$				
	$\overline{a_4}$				a_4				
1	$\overline{x_4}$				x_4				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

We now derive the Bough Wooley Multiplier equations. The product we intend to produce can be expressed as follows:

$$\begin{aligned} P &= A \times X = (p_{2k-1}p_{2k-2}\cdots p_1p_0) = -p_{2k-1}2^{2k-1} + \sum_{i=0}^{2k-2} p_i 2^i \\ &= \left(-a_{k-1}2^{k-1} + \sum_{i=0}^{k-2} a_i 2^i\right) \times \left(-x_{k-1}2^{k-1} + \sum_{j=0}^{k-2} x_j 2^j\right) \\ &= a_{k-1}x_{k-1}2^{2k-2} + \sum_{i=0}^{k-2}\sum_{j=0}^{k-2} a_i x_j 2^{i+j} \\ &\quad -2^{k-1}\sum_{i=0}^{k-2} a_{k-1}x_i 2^i - 2^{k-1}\sum_{i=0}^{k-2} a_i x_{k-1} 2^i \end{aligned} \tag{1}$$

Equation 1 must be manipulated so that there are no negative terms.

Here is the third term without its negative sign:

$$\begin{aligned} & 2^{k-1}\sum_{i=0}^{k-2} a_{k-1}x_i 2^i \\ = \; & 2^{k-1}\left(-0\times 2^k + 0\times 2^{k-1} + \sum_{i=0}^{k-2} a_{k-1}x_i 2^i\right) \end{aligned}$$

The inclusion of the two zeroes does not change the value.

The expression in parens can be thought of as a $k+1$ bit value with the sign bit in the 2^k position. We now negate that expression by complementing every bit and adding 1:

$$= \; -2^{k-1}\left(-2^k + 2^{k-1} + 1 + \sum_{i=0}^{k-2} \overline{a_{k-1}x_i} 2^i\right) \tag{2}$$

Equation 2 has zero value for $a_{k-1} = 0$ and for for $a_{k-1} = 1$ it has value:

$$-2^{k-1}\left(-2^k + 2^{k-1} + 1 + \sum_{i=0}^{k-2} \overline{x_i} 2^i\right)$$

Equation 2 can therefore be rewritten:

$$-2^{k-1}\left(-2^k + 2^{k-1} + \overline{a_{k-1}} 2^{k-1} + a_{k-1} + \sum_{i=0}^{k-2} a_{k-1}\overline{x_i} 2^i\right)$$

The fourth term of Equation 1 can similarly be rewritten.

The product becomes:

$$\begin{aligned} P &= a_{k-1}x_{k-1}2^{2k-2} + \sum_{i=0}^{k-2}\sum_{j=0}^{k-2} a_i x_j 2^{i+j} \\ &\quad +2^{k-1}\left(-2^k + 2^{k-1} + \overline{a_{k-1}} 2^{k-1} + a_{k-1} + \sum_{i=0}^{k-2} a_{k-1}\overline{x_i} 2^i\right) \\ &\quad +2^{k-1}\left(-2^k + 2^{k-1} + \overline{x_{k-1}} 2^{k-1} + x_{k-1} + \sum_{i=0}^{k-2} \overline{a_{k-1}} x_i 2^i\right) \end{aligned} \tag{3}$$

Equation 3 provides the basis for the Baugh Wooley multiplier.

Here again is the 5×5 Baugh Wooley multiplier table:

					a_4	a_3	a_2	a_1	a_0
			$\times$		x_4	x_3	x_2	x_1	x_0
					$a_4\overline{x_0}$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$a_4\overline{x_1}$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$a_4\overline{x_2}$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$a_4\overline{x_3}$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	$\overline{a_3}x_4$	$\overline{a_2}x_4$	$\overline{a_1}x_4$	$\overline{a_0}x_4$				
	$\overline{a_4}$				a_4				
1	$\overline{x_4}$				x_4				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

The first term of Equation 3 is $a_4x_4 2^8$.

The second term is

$$\sum_{i=0}^{k-2}\sum_{j=0}^{k-2} a_i x_j 2^{i+j}$$

It gives all the entries without complemented variables:

				a_3x_0	a_2x_0	a_1x_0	a_0x_0
			a_3x_1	a_2x_1	a_1x_1	a_0x_1	
		a_3x_2	a_2x_2	a_1x_2	a_0x_2		
	a_3x_3	a_2x_3	a_1x_3	a_0x_3			

The third term is:

$$2^{k-1}\left(-2^k + 2^{k-1} + \overline{a_{k-1}}2^{k-1} + a_{k-1} + \sum_{i=0}^{k-2} a_{k-1}\overline{x_i}2^i\right)$$

It gives $-2^9 + 2^8$ plus:

				$a_4\overline{x_0}$			
			$a_4\overline{x_1}$				
		$a_4\overline{x_2}$					
	$a_4\overline{x_3}$						
$\overline{a_4}$				a_4			

And the fourth term is:

$$2^{k-1}\left(-2^k + 2^{k-1} + \overline{x_{k-1}}2^{k-1} + x_{k-1} + \sum_{i=0}^{k-2} \overline{a_{k-1}}x_i2^i\right)$$

It gives $-2^9 + 2^8$ plus:

	$\overline{a_3}x_4$	$\overline{a_2}x_4$	$\overline{a_1}x_4$	$\overline{a_0}x_4$			
$\overline{x_4}$				x_4			

The two terms $-2^9 + 2^8$ sum to form $-2^{10} + 2^9$. The negative term is outside the range of the product.

The Baugh-Wooley multiplier deals with both negative multiplicands and negative multipliers without sign extension. The two additional levels add to the delay, although it is possible to remove one of these levels. A CLA adder can used to sum the two values emerging at the bottom of the array.

The table can be implemented with Wallace trees in each column.

Alternatively pipeline registers can be inserted between the rows of the table, enabling higher throughput, but not reducing delay.