Area $\times$ Delay ($A \cdot T$) Efficient Multiplier Based on an Intermediate Hybrid Signed–Digit (HSD–1) Representation

Jeng-Jong J. Lue
Summit Systems, Inc.
22 Cortlandt St. 31st Fl
New York, NY 10007

Dhananjay S. Phatak
Electrical Engineering Department
State University of New York, Binghamton, NY 13902–6000
phatak@ee.binghamton.edu

Abstract

Intermediate Signed Digit (SD) representation can facilitate fast and compact VLSI implementations of partial product accumulation trees. It achieves a reduction ratio of 2:1 at every level and also leads to more regular layouts. Its disadvantage is that the number of bit lines that need to routed can be high. This can lead to a significant area overhead especially at smaller feature sizes where the wire/interconnect area and delay can be dominant.

A Hybrid Signed Digit (HSD) representation lets some of the digits be unsigned bits, thereby reducing the number of bit lines. By arbitrarily varying the positions of and distances between consecutive signed digits, this representation can trade off latency for area and offers a continuum of choices between the two's complement representation on the one hand and fully Signed Digit (FSD or simply SD) representation on the other.

In this paper, we illustrate an $A \cdot T$ (area $\times$ delay) efficient multiplier based on the HSD–1 representation which is one of the many possible HSD formats, wherein every alternate digit is signed and the rest are unsigned (ordinary) bits. It is seen that multipliers based on HSD–1 format require more transistors than those based on FSD format. However, they require fewer bit lines to be routed, which substantially reduces the interconnect area; thereby leading to a reduction in the total VLSI area and a lower $A \cdot T$ product. The design reaffirms that the interconnect area can be siginficant especially at small feature sizes.

1. Introduction

To speed up multiplication, Partial Products (PPs) are accumulated in parallel in some kind of a tree structure (such as a Wallace Tree [13]). Any carry propagation is deferred until there are only 2 partial sums left to be added, at which point a full carry propagation must occur. Traditionally, the accumulation tree has been based on (3,2) counters or full adders. This achieves a reduction ratio of 3:2 and requires "diagonal" propagation of the carries making the VLSI layout difficult. Other types of counters have been used as well [1, 7, 4]. Fundamentally, however, these generalized multi-input counters are still synthesized out of (3,2) counters.

Another approach to high speed partial product accumulation is to employ an intermediate Signed Digit (SD) representation [3, 5, 12]. In this representation, carry propagation is limited to 1 digit position (hence, addition can be thought to be "carry free" for all practical purposes). The SD representation achieves a reduction ratio of 2:1 and leads to more regular layouts, circumventing the need for diagonal signal propagation [5]. On the other hand, the number of bits required to represent an n digit partial sum is $2n$ because each binary Signed-Digit can assume any of the 3 values $\{-1, 0,1\}$, and requires two bits to represent it (note that any carry-save type scheme that postpones full carry propagation until the very end also needs 2 bits per position). Hence, opting to generate the intermediate partial sums in the SD format can lead to fast execution and a 2:1 reduction ratio; but number of bit-lines that need to be routed can be substantial. This can lead to a significant area overhead especially at smaller feature sizes where the wire/interconnect delay and area can be dominant.

A Hybrid Signed Digit (HSD) representation was introduced in [11]. It lets some of the digits remain unsigned bits, thereby reducing the total number of bit lines. The HSD representation is pictorially illustrated in Figure 1 (along with it's relationship with the GSD representation which was proposed in [10]). As seen in the Figure, in this representation, the carry propagation chains can be confined in between consecutive signed digits. In this format, the number and positions of signed digits can be arbitrary and hence can be selected to match desired application goals. By varying the positions of and distances between signed digits, this representation can trade off addition latency for area and offers a continuum of choices between the two's complement representation on the one hand and full Signed Digit (FSD or simply SD) representation on the other. Of particular interest is the HSD-1 format wherein, every alternate digit is signed and the remaining digits are unsigned

(ordinary) bits. The Least Significant Bit (LSB) is unsigned and the Most Significant Bit (MSB) has to be a signed digit (reasons for this and other details about HSD representations can be found in [11]). Note that the number of bits required to represent a partial sum in the HSD–1 format is only $\frac{3}{4}$th the corresponding number of bits required by the FSD format (since alternate positions in the HSD–1 format are unsigned bits). Hence, the routing area required for multipliers based on the HSD–1 format can be substantially smaller than FSD based designs. This was pointed out in [11], wherein, it was further argued that HSD–1 based multipliers might potentially have a smaller total area, as well as a lower AT product than their FSD based counterparts.

In this paper, we illustrate such an AT efficient design based on the HSD–1 format. The next section describes the number representations and the arithmetic algorithms involved; along with their effect on the overall architectural issues. Section 3 briefly outlines the implementation and measurements. Prototype 16×16 multipliers based on the HSD–1 and FSD formats were fully laid out in order to obtain accurate area estimates for higher wordlengths (32 and 64 bits). Delays were estimated from layouts and SPICE simulations. Based on this data we illustrate the AT comparisons in Section 3. Section 4 presents discussion and conclusions. The analysis bears out that a proper mix of HSD–1 and FSD formats can yield an improvement in the AT performance measure.

We would like to point out that this paper does not attempt to synthesize the fastest or the smallest multiplier, neither does it look at all possible classes of multipliers. It only compares the AT products of multipliers based on FSD and HSD-1 formats. Hence, comparisons with other recent 4:2 compressor based multipliers (such as [2, 9, 14], which do not use intermediate SD representation but employ efficient 4:2 compressors to achieve fast execution) is out of the scope of this paper.

2. Architectures

For n bit long operands, radix 4 Booth recoding reduces the number partial products to $\lceil n/2 \rceil$, (where $\lceil x \rceil$ = the smallest integer $\geq$ x). Let M be the multiplicand (being an input, it is assumed to be in two's complement format). Then each of the primary partial products (i.e., those generated as a result of Booth recoding) is a two's complement number. At the first level of the PP accumulation tree, two of these primary PPs are added to produce a signed-digit result. This can be done very efficiently by rewriting the addition $A + B$ as a subtraction:

$$A + B = (A - \overline{B} - 1) \text{ modulo } 2^n \quad (1)$$

where $\overline{B}$ is the one's complement of B which is obtained simply by inverting all the bits of B:

$$\overline{B} = 2^n - 1 - B \quad \text{, where } n \text{ is the word-length} \quad (2)$$

The -1 in equation (1) can be taken care of by forcing a carry (borrow)-in $c_0 = -1$ and this correction is deferred to the next levels of the PP accumulation tree. The modulo operation simply amounts to discarding the outgoing borrow. Since each of the bits of A and $\overline{B}$ can be 0 or 1, a bit–wise subtraction directly leads to a signed-digit output representing $(A - \overline{B})$, where each digit $y_i = a_i - \bar{b}_i$ is in the range $\{-1, 0, 1\}$. The bit–wise subtraction can be carried out simultaneously (i.e., in parallel) for all the digit positions because there is no need to propagate signals from one digit position to the next. Assuming the commonly used encoding $0 \leftrightarrow 00, 1 \leftrightarrow 01$ and $-1 \leftrightarrow 11$ to represent a signed-digit, the bit-wise subtraction can be achieved simply by one XOR and one NOR gate that operate in parallel. Because this operation can be executed so fast and with such little hardware it was simply "fused" with the partial product generator cells in [5] so that after this stage only $n/4$ partial products were generated (in FSD format). For $n = 64$, the PP tree to sum the 16 fully SD partial products is illustrated in Figure 2 (which is further explained a bit later below). Throughout the tree, extremely efficient (fast *and* compact) Redundant Binary Adder (RBA) cells presented in [5] are used to execute the additions. The critical path in their SD adder traverses 3 digit positions (i.e., 3 RBA cells). Each RBA cell accepts 2 SD operands and an incoming signed carry (for a total of 6 inputs bits) and generates a signed-digit sum output and a signed outgoing carry (i.e., a total of 4 output bits). Further details regarding the design of this cell can be found in [5].

Suppose Booth-recoding generates adjacent partial products αM and γM with $\alpha, \gamma \in \{0, \pm 1, \pm 2\}$, then the first level of the PP accumulation tree will have to perform $\alpha M + 4\gamma M = \alpha M + \beta M$. In general both α and β can assume positive or negative signs. The operations performed in all these cases are summarized in Table 1 below (on the next page). Note that if the signs of α and β are opposite, then a true subtraction is called for and in this case there is no need to "re-write" it. As seen in the table, true subtraction directly leads to the correct signed digit result which does not require the LSD correction, and does not generate the carry-out. Addition of the magnitudes, however leads to a SD result that needs the LSD correction and carry-out suppression. The LSD correction is "deferred" to the next level of PP accumulation tree and has significant consequences on the overall architecture as explained next.

At the second level, the two adjacent partial products to be added are shifted with respect to each other by 4 digit positions and are in full SD format. More important, each of the sums is generated as per Table 1, so that in the worst

Possible Cases	Operation (any of the two below)		Least Significant Digit (LSD) Correction	Weight of Carry Out
$+\|\alpha M\| + \|\beta M\|$	$+\|\alpha M\| - \overline{\|\beta M\|}$	$-\overline{\|\alpha M\|} + \|\beta M\|$	$-1 = -ulp$	-2^n
$+\|\alpha M\| - \|\beta M\|$	$+\|\alpha M\| - \overline{\overline{\|\beta M\|}}$	$-\overline{\|\alpha M\|} + \overline{\|\beta M\|}$	0	0
$-\|\alpha M\| + \|\beta M\|$	$-\overline{\overline{\|\alpha M\|}} + \|\beta M\|$	$+\overline{\|\alpha M\|} - \overline{\|\beta M\|}$	0	0
$-\|\alpha M\| - \|\beta M\|$	$+\overline{\|\alpha M\|} - \overline{\overline{\|\beta M\|}}$	$-\overline{\overline{\|\alpha M\|}} + \overline{\|\|\beta M\|}$	$1 = ulp$	2^n

Table 1 : Operation of the Subtracter at the First Level. ulp indicates adding one, i.e., a "unit in the least significant position"

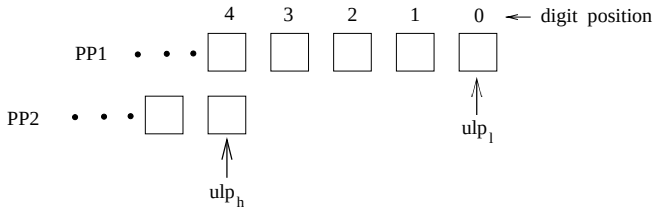


Illustration 1: Corrections required at the second level.

case, each of the two operands (being summed at the 2nd level) could need the LSD correction as shown in Illustration 1 below (on the next page). The LSD correction associated with the higher order partial product is denoted by ulp_h and the one required by lower order PP is denoted by ulp_l . It can be shown that in general, it is not possible to make both ulp_h and ulp_l corrections simultaneously at the second level (because it can lead to digit-value overflow at digit position 4 shown in Illustration 1). The natural question is which one should be corrected and which one should be deferred. If ulp_h is not corrected, it ultimately leads to a situation where one extra level needs to be added to the tree, just to add the ± 1 values in the correct positions. This is costly in terms of both execution delay as well as circuit area. Hence, ulp_h must be corrected and ulp_l must be deferred to the next level. This is shown in Figure 2 by the letter "c" (corrected) near the arrow originating at the higher significant operand and the letter "d"(deferred) near the arrow originating at the lower significant operand, in every pair that gets added. This way, at the very end the correction required is a ulp_l which can be accomplished by setting carry-in = (value of the uncorrected ulp_l). This can be easily done in the SD adder since each adder cell in a SD RBA accepts a carry-in which can assume any one of the three values $\{-1, 0, 1\}$.

Next we consider the HSD–1 (also abbreviated as HSD) PP accumulation tree(s). At the top level of the HSD tree, two PPs (which are generated by Booth recoding and are in two's complement format) are added to generate HSD–1 format output. The top cells used for this purpose are illustrated in Figure 3. From level 2 onwards the RBA cells shown in Figure 4 are utilized to add HSD format operands and generate an HSD format output. The figure shows an array of 4 cells along with the critical path which traverses 4 digit positions in this case. Alternate cells are for unsigned digit positions and accept two bits (labeled a_k and b_k in Figure 3), and a signed carry (bits u_k and v_k in the Figure),i.e., a total of 4 inputs and produce one output bit (denoted e_k) and outgoing signed carry, i.e., a total of 3 bits.

The adjacent cell adds 2 signed digits along with an incoming carry (a total of 6 inputs) and produces a signed digit output and a signed carry (i.e., a total of 4 outputs). Within every group of two adjacent cells, the higher significant cell handles signed digits while the lower significant cell handles the unsigned bits. This group is arrayed $n/2$ times to implement an adder of wordlength n. The transistor count and area required by this group turns out to be smaller than that required by a cascade of two SD RBA cells used in [5]. More important, for every 4 bits in the FSD design there are only 3 bits in the HSD design, so that interconnect area (required for routing the bit lines; for bus bendings when the operands are brought together at the next level, etc.) can be expected to be significantly less for the HSD design. Further details about these cells can be found in [11].

Note that the top level cells in the HSD tree shown in Figure 3 are far more complex than those used in [5]. Also, in a tree, the number of leaf nodes equals (1 + number of internal nodes). Thus, more than half of the adders are required at the top level. Hence, if pure HSD–1 format outputs are generated right from the first level of the tree, then the total number of transistors required is much larger than that for a tree based on full SD format, despite the fact that the adder cells required handle HSD–1 format inputs and outputs (shown in Figure 4) are smaller than adder cells that add two FSD numbers to generate an FSD output. So, at the top level of the PP adder tree, FSD format outputs should be generated and if possible from the second level onwards, HSD–1 format outputs should be produced; but, such a switch from FSD to HSD–1 format could lead to some extra delay and hardware.

Fortunately, however, the HSD format is flexible and it is possible to add a full SD and an HSD–1 format number to generate an HSD–1 format output in the same time it takes to add two HSD–1 format numbers. As a result, at the first level, some PPs can be generated in the FSD format and the rest in the HSD–1 format. These can then be combined appropriately at the subsequent levels. Consequently, there are various ways of "mixing" the FSD and HSD–1 formats as illustrated in Figure 5. Note that irrespective of the mix, the ulp_h always needs to be corrected and ulp_l needs to be deferred to the next level.

Unlike FSD, the addition of two's complement primary PPs into HSD–1 format output does not go though an intermediate subtraction. Hence, the operations carried out are like normal addition and are summarized in Table 2 below. Since no subtraction is performed in the HSD design, there is no "-1" in Table 2. The "$+2$" in the last case can not be corrected on the same level. Therefore, it is split as (ulp + ulp) and one of the "ulp "s is corrected at the first level, leaving a $+1$ to be corrected at latter levels. Note that unlike the FSD design, the first level HSD partial product adder can accept a carry-in (indicated by input $\overline{c_{i-1}}$ in Figure 3) which makes a correction possible right at the first level. Thus, when the first level outputs are in HSD–1 format, only one value is needed for the LSD correction at all subsequent levels of the tree in all cases: viz., $+1 = ulp$.

Possible Cases	Operation	LSD correction	Carry Out
$+\|\alpha M\| + \|\beta M\|$	$+\|\alpha M\| + \|\beta M\|$	0	0
$+\|\alpha M\| - \|\beta M\|$	$+\|\alpha M\| + \overline{\|\beta M\|}$	1	2^n
$-\|\alpha M\| + \|\beta M\|$	$+\overline{\|\alpha M\|} + \|\beta M\|$	1	2^n
$-\|\alpha M\| - \|\beta M\|$	$+\overline{\|\alpha M\|} + \overline{\|\beta M\|}$	2	2^{n+1}

Table 2: Operation of the HSD Adder at the First Addition Level

The sign of the LSD corrections is important when adding an SD PP and an HSD–1 PP together to generate HSD–1 format output. In fact, a consideration of the sign of the LSD correction (together with other constraints) dictates that the HSD number has to be the lower significant one and the SD number has to be the higher significant one whenever they are added together. The reason for this can be understood with the aid of Illustration 2 which shows a lower significant FSD PP being added to a higher significant HSD–1 PP to generate HSD–1 format output at the 2nd level of the tree (where the relative shift is 4 digits). Zeroes are padded in the last 4 digit positions of the HSD PP. Despite being added to zeroes, the last 4 "FFFF" digits of the FSD PP cannot be simply passed on as the output digits: they must be converted into HSHS (signed unsigned

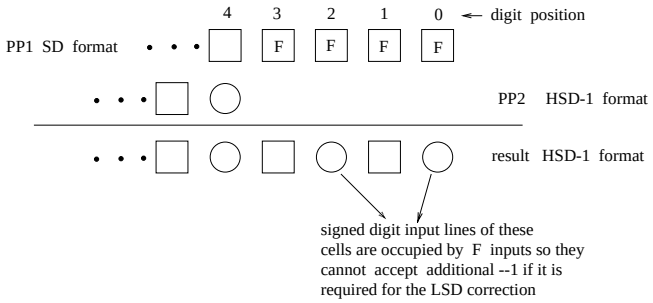


Illustration 2: HSD–1 format operand has to be the lower significant one when adding it to an FSD format operand.

signed unsigned) digits to conform to the HSD–1 format at the output. This can be achieved simply by extending the row of adder cells all the way to the LSD. In this case, note that the cells in position 0 and 2 will add a signed digit "F" (supplied by PP1) and an unsigned bit (which can be supplied by PP2 which is in HSD–1 format; or it can be a "padding" bit). In other words, these cells accept only 3 input operand bits. If a $+ulp$ correction is, required, the operation in the last 4 digits is ("FFFF" + "1111" + carry-in=1) and this can be handled by the adder cells being used. However, if a $-ulp$ correction is required, then the operation is ("FFFF" + "$-1-1-1-1$" + carry-in=-1) and this is not feasible since there is no way to input the additional -1 value in digit positions 0 and 2 because the only signed digit input lines in these cells are occupied by digits "F" from PP1.

We would like to point out that this restriction is not imposed by the specific VLSI cell design, rather, it is fundamentally dictated by the underlying HSD–1 format. In Illustration 2, if the positions of HSD and FSD PPs is switched, then both $\pm ulp$ corrections can be taken care of because the "signed digit" input lines are now free and they can be padded with any of the 3 values 0, 1 or -1. In summary, whenever HSD and FSD PPs are added, the FSD PP must occupy higher significant half. That's why organizations like "HFFF" or "HFHF" are not beneficial (and hence are not shown in Figure 5).

3. Implementation and Measurements

The main objective of this work is a "*relative*" comparison of the AT products of the FSD and HSD–1 designs. Hence, as long as the same circuit techniques, optimizations, etc. are uniformly applied to both designs, the *relative* comparison can yield meaningful results. The point being absolute raw nanoseconds and μm^2 values are less critical, rather it is the *ratio* of the AT products of the two designs that is more important.

The multipliers were synthesized from five main modules, viz., radix-4 Booth recoder, partial product generator (PPG), the redundant binary number adder (RBA),

	SD	HSD–1 (FFFH)	HSD–1 (FHFH)
Area	$1.44673104 \times 10^8 \ \lambda^2$	$1.19244980 \times 10^8 \ \lambda^2$ (82.4% of FSD)	$114706400 \times 10^8 \ \lambda^2$ (79.3% of FSD)
Transistor counts	33552	53186	51082
Delay Time	5.973 ns	6.895 ns	6.895 ns (115% of FSD)
$A \cdot T$ Product	$8.64132450192 \times 10^8 \ \lambda^2 \cdot ns$	$8.221941371 \times 10^8 \ \lambda^2 \cdot ns$	$7.90900628 \times 10^8 \ \lambda^2 \cdot ns$
% Improvement in AT		4.9%	8.5%

Table 3 : Area, delay and $A \cdot T$ Product estimates for 64×64 bit multipliers (excluding the final carry propagate addition) based on intermediate SD and HSD–1 format operands.

the sign correction circuits and the LSD correction circuits. Most of these building blocks were described in the previous section, along with the overall architectures of the multipliers (further details can be found in [6]). These modules were first verified with "CGATES", which is a public domain hierarchical gate-level logic-simulator [8]. The modules were then layed out, optimized as best as possible and verified via IRSIM and SPICE simulations (for these simulations, 0.5 micron technology files were used). Note that once the cells for these building blocks are finalized, they can be used in all the different architectures based on various FSD and/or HSD–1 mixes. Hence, there is no need to explicitly make a separate layout for every possible FSD/HSD–1 mix: accurate area estimates can be obtained from the cell dimensions and routing area information for one prototype layout. We layed out one FSD and two HSD–1 based multipliers ("FFFH" and "FHFH" schemes shown in Figure 5. The reasons for selecting these two architectures were briefly outlined toward the very end of Section 2. Further details can be found in [6]). Straightforward "V-tree" structure was adopted in all the designs.

After the double length multiplication output is generated in FSD or HSD–1 format, a conversion into two's complement format is required at the very end. This can be achieved by an adder-like circuit which essentially performs some sort of a carry/borrow propagation. This circuit is almost identical for both the FSD and HSD schemes and takes much smaller area (compared with the rest of the multiplier). It does increase the total execution delay (latency) by a noticeable amount, but the delay of this conversion is the *same* irrespective of whether the final result is in HSD–1 or FSD format, i.e., it increases the delays of either scheme by the same amount. Hence, excluding the converter actually gives a more pessimistic (worst case) result as far as the HSD scheme is concerned. This happens because the ratio of areas does not change much (with or without the final converter) but the ratio of delays of HSD and FSD based schemes is higher when the final converter is excluded.

16×16 multipliers based on intermediate FSD as well as two of the HSD–1 formats mentioned above were designed. They were first simulated using CGATES where some peculiar cases such as $0 \times 0, \quad 0 \times 1, \quad +MAX \times +MAX, \quad -MAX \times -MAX$, etc., were hand picked and simulated. In addition 10,000 randomly generated operand pairs were also simulated and verified. Then, the multipliers were layed out using Magic 6.4.5 and simulated using IRSIM and SPICE.

The wire bendings (required to bring the operands together to be added at the next level of the tree) accounts for about 50 percent of the area for each design. Since there is a 25 percent reduction in the number of wires in the HSD scheme compared to the FSD scheme, an approximate 25 percent reduction in area is expected. However, depending on how the 16×16 HSD multiplier is arranged, the area that is saved may be different. The 16×16 FSD multiplier has an area of $8.373018 \times 10^6 \ \lambda^2$ while the area of the HSD–1 multiplier is $7.778810 \times 10^6 \ \lambda^2$. There is only about seven percent improvement in the area. However, for higher wordlengths (for example, 64 bits) the interconnect area is even more dominant and at these wordlengths the area of the HSD–1 design is roughly 21% smaller than that of the FSD design.

The ultimate goal was to compare the AT products for 64×64 designs. The area for this worldlength can be accurately estimated from the 16×16 design because the same cells are used and the tree topology is the same. This makes it possible to precisely calculate the total width as well as height of the design, thus yielding the total area required. The area estimates of the 3 designs are indicated in Table 3 above.

To estimate the delay of the 64 bit designs (excluding the final conversion into two's complement format), the critical path was first identified along with it's "equivalent capacitance" for all the designs. This is done by counting the

number of transistors *associated* with the critical path. This includes transistors traversed by the critical path, as well as transistors whose inputs (i.e., gates) are connected to the critical path, weighed according to their sizing (so that a double sized transistor adds twice the load of a normal transistor to the critical path). This accounts (at least partially) for fanin and fanout loads. The critical path of the 64×64 FSD multiplier has 42 transistors "associated" with it, while the 64×64 HSD multipliers have 51. An approximate SPICE simulation for time delay estimation uses an inverter with the standard width for the P-type and N-type transistors (P-type 8λ wide and N-type 4λ wide in our designs) to drive a larger inverter. The total capacitance of these two inverters is made equal to the capacitance associated with the critical path. For example, for the FSD multiplier, the simulation used a regular inverter to drive a larger inverter that has a capacitance of 41 P-type transistors and 41 N-type transistors. The same was done for the HSD designs. The delays obtained are also shown in Table 3. This is not the best way of measuring the delay, but it was deemed acceptable for a *relative* comparison of the designs as long as *the same* methodology was uniformly applied to all of them.

Table 3 indicates that HSD–1 format multipliers have a lower AT product and corroborate the projections made in [11]. The more the number of PPs in FSD format at the top, the higher is the wire bending area. This is clearly seen in Table 3: the FHFH design requires lesser area than the FFFH design (both designs are shown in Figure 5). The critical path delay is the same for both because in the FFFH scheme, even though FSD additions happen faster, the result has to ultimately wait for the slightly slower HSD branch to complete it's part.

4. Conclusions and Discussion

As VLSI device feature sizes continue to shrink, the interconnect area becomes significant. Hence, to reduce the overall area, it is worthwhile to reduce the interconnect area (even though it might lead to an increase in the number of transistors). In this paper we showed that adopting the HSD–1 format instead of the FSD format for intermediate results in a multiplier can lead to a significant reduction in the interconnect area; which in turn leads to a reduction the total VLSI area. This reduction occurs despite the fact that the HSD–1 designs require more transistors than their FSD counterparts. The delay overhead of the HSD–1 based designs was shown to be moderate. The AT product of 64×64 multipliers based on the HSD–1 format was shown to be smaller than that of multipliers based on intermediate full sign digit format. More important, the HSD representation is seen to be highly flexible, allowing the designer a large number of choices to trade off area for delay and match the desired performance goals.

We used 3 metal layers in the layouts. Despite that, the "wire/bus bending" penalty could not be avoided. It is possible that by using more metal layers, wire bending area could perhaps be substantially reduced. Possible future work could address this issue.

References

[1] L. Dadda. Some Schemes for Parallel Multipliers. *Alta Freq.*, 34:349–356, 1965.

[2] Hanawa M., et. al. A 4.3ns 0.3 μm CMOS 54×54b Multiplier Using Precharged Pass-Transistor Logic. In *in ISSCC Digest of Technical Papers*, pages 364–365, 1996.

[3] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi. A high-Speed multiplier using a redundant binary adder tree. *IEEE Journal of Solid-State Circuits*, SC-22:28–34, Feb. 1987.

[4] I. Koren. *Computer Arithmetic Algorithms*. Brookside Court Publishers, Amherst, Massachusetts, 1998.

[5] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi. Design of high speed MOS multiplier and divider using redundant binary representation. *Proc. of the 8th Symposium on Computer Arithmetic*, pages 80–86, 1987.

[6] J.-J. J. Lue. Area $\times$ Delay ($A \cdot T$) Efficient Multipliers Based on an Intermediate Hybrid Signed–Digit Representation. Master's thesis, Electrical Engineering Dept. State University of New York, Binghamton, NY 13902–6000, 1997.

[7] M. Mehta, V. Parmar, and E. Swartzlander. High-speed Multiplier Design Using Multi-Input Counter and Compressor Circuits. In *Proc. of the 10th Symp. on Computer Arithmetic*, pages 43–50, 1991.

[8] R. M. Meyer. CGATES logic simulator. Canisius College, Buffalo, NY, http://www-cs.canisius.edu/~meyer/SOFTWARE/software.html.

[9] N. Ohkubo and Suzuki, M., et. al. A 4.4-ns CMOS 54×54-b Multiplier Using Pass-Transistor Multiplexor. *IEEE Journal of Solid-State Circuits*, 30(3):251–256, Mar. 1995.

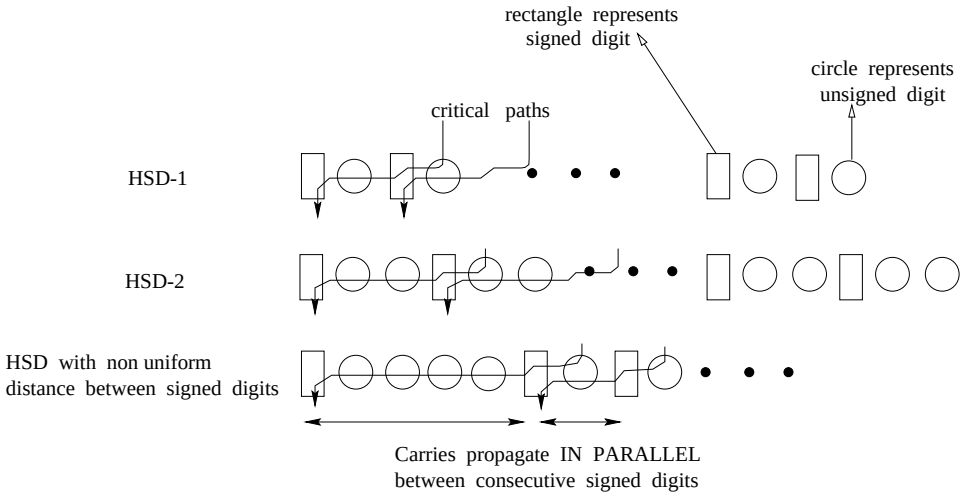
[10] B. Parhami. Generalized signed-digit number systems: a unifying framework for redundant number representations. *IEEE Transactions on Computers*, C-39:89–98, Jan. 1990.

[11] D. S. Phatak and I. Koren. Hybrid Signed–Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains. *IEEE Trans. on Computers, Special issue on Computer Arithmetic*, TC–43(8):880–891, Aug. 1994. (An unabridged version is available on the web via the URL http://www.ee.binghamton.edu/faculty/phatak).

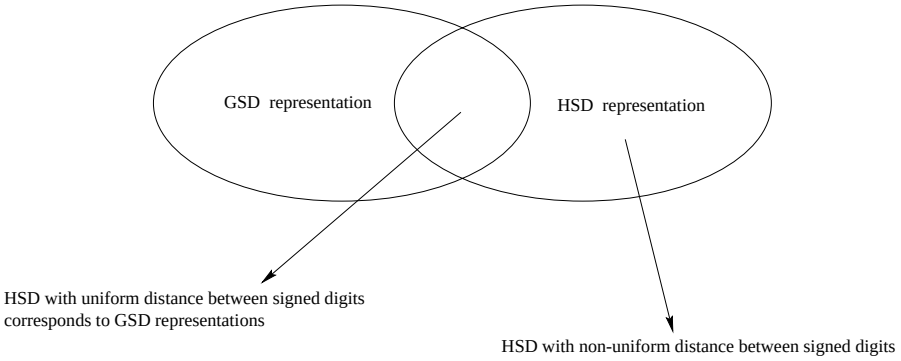
[12] N. Takagi, H. Yasuura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Transactions on Computers*, C-34:789–796, Sep. 1985.

[13] C. S. Wallace. A Suggestion for a Fast Multiplier. *IRE Transactions on Electronic Computers*, EC-13:14–17, 1964.

[14] R. Yu and G. Zyner. 167 MHz Radix-4 Floating Point Multiplier. In *Proc. of the 12th Symp. on Computer Arithmetic, Bath, England*, pages 149–154, 1995.



(a) Illustration of HSD formats



(b) Relationship between HSD and GSD number representations

Figure 1 : HSD formats and relationship between HSD [11] and GSD [10] number representations.
(a) Different HSD formats: HSD-k denotes a representation with k unsigned digits between every pair of consecutive signed digits. Last drawing illustrates a format with non uniform distance between signed digits.
(b) Relationship between HSD and GSD number representations.

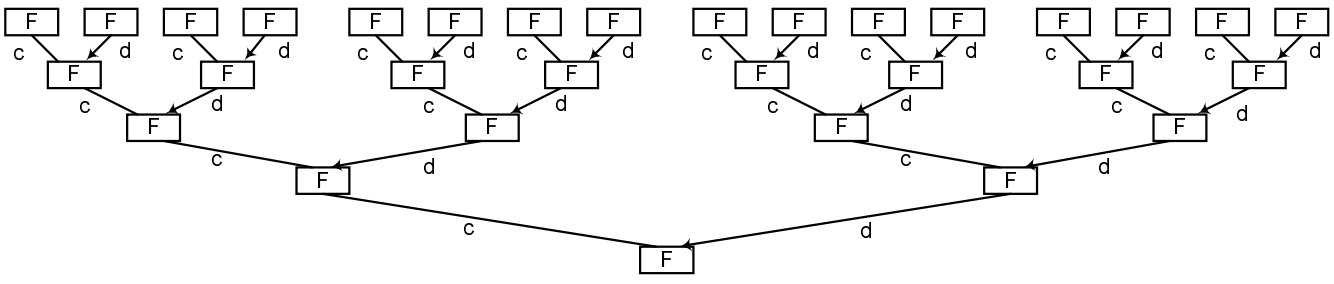


Figure 2 : Schematic of the FSD Partial Product accumulation tree for wordlength $n = 64$. Here, $n/4 = 16$ fully SD format PPs (indicated by "F" inside each box) are generated after Booth recoding and the first level addition performed by the "subtracter" as summarized in Table 1 in Section 2.

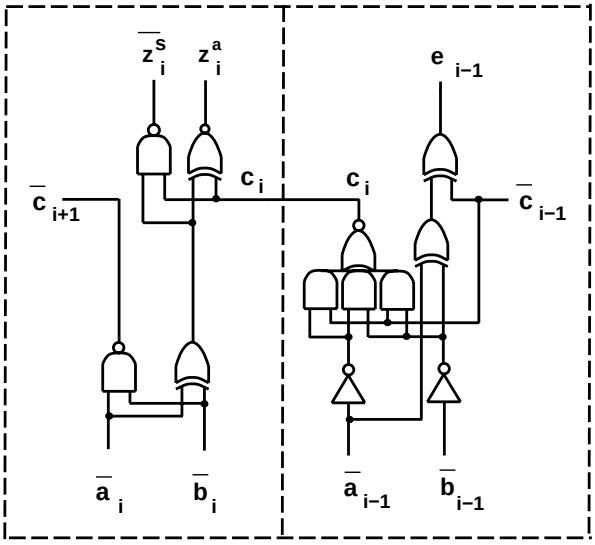


Figure 3 : Cells used to generate HSD–1 format output from the addition of 2 numbers in the Two's-Complement format. This pair of cells (left one at signed and the right one at unsigned digit position) is replicated $n/2$ times in an adder of wordlength n. These cells are used at the top level of the HSD PP accumulation tree to add Booth-recoded PPs in two's complement format to generate HSD-1 format output.

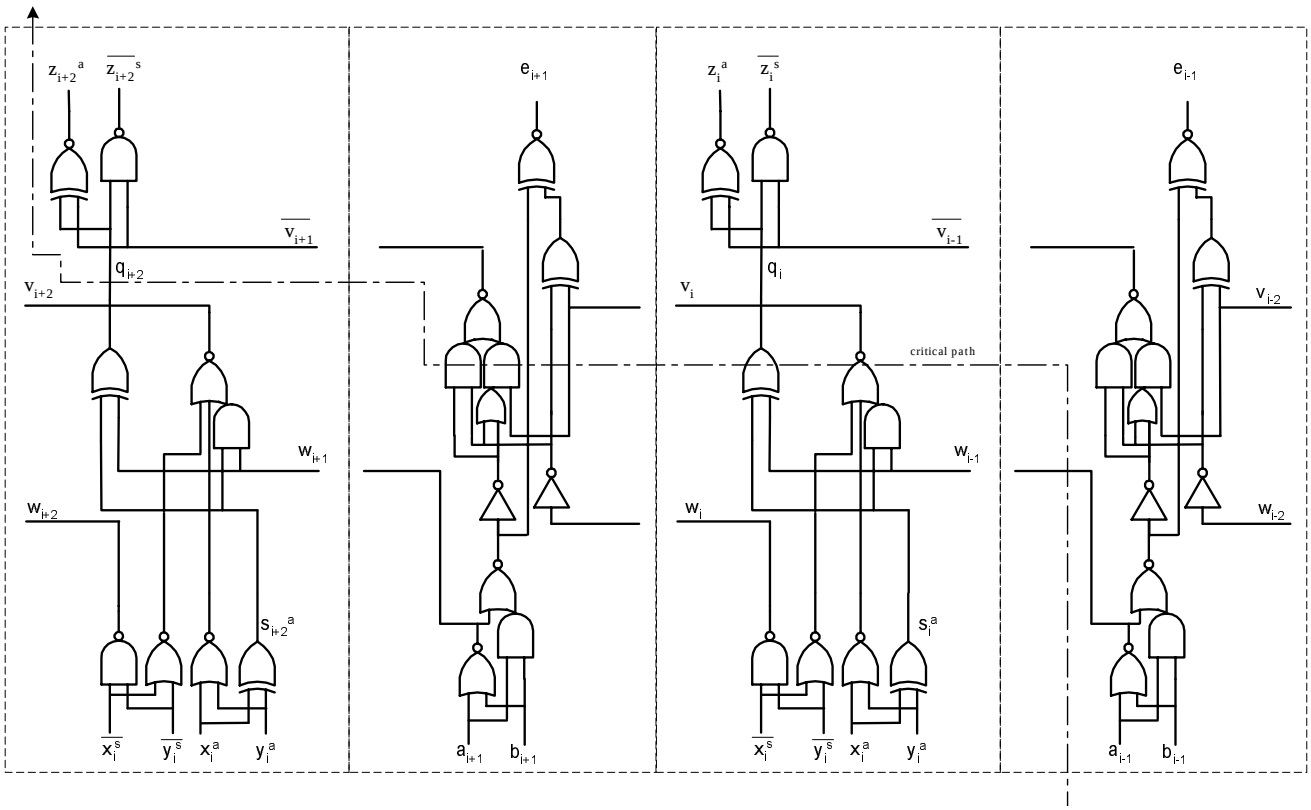


Figure 4 : RBA cells used to add two HSD-1 format operands to generate an HSD-1 format output. The figure shows an array of 4 cells along with the critical path (marked by the dashed line) which traverses 4 digit positions in this case.

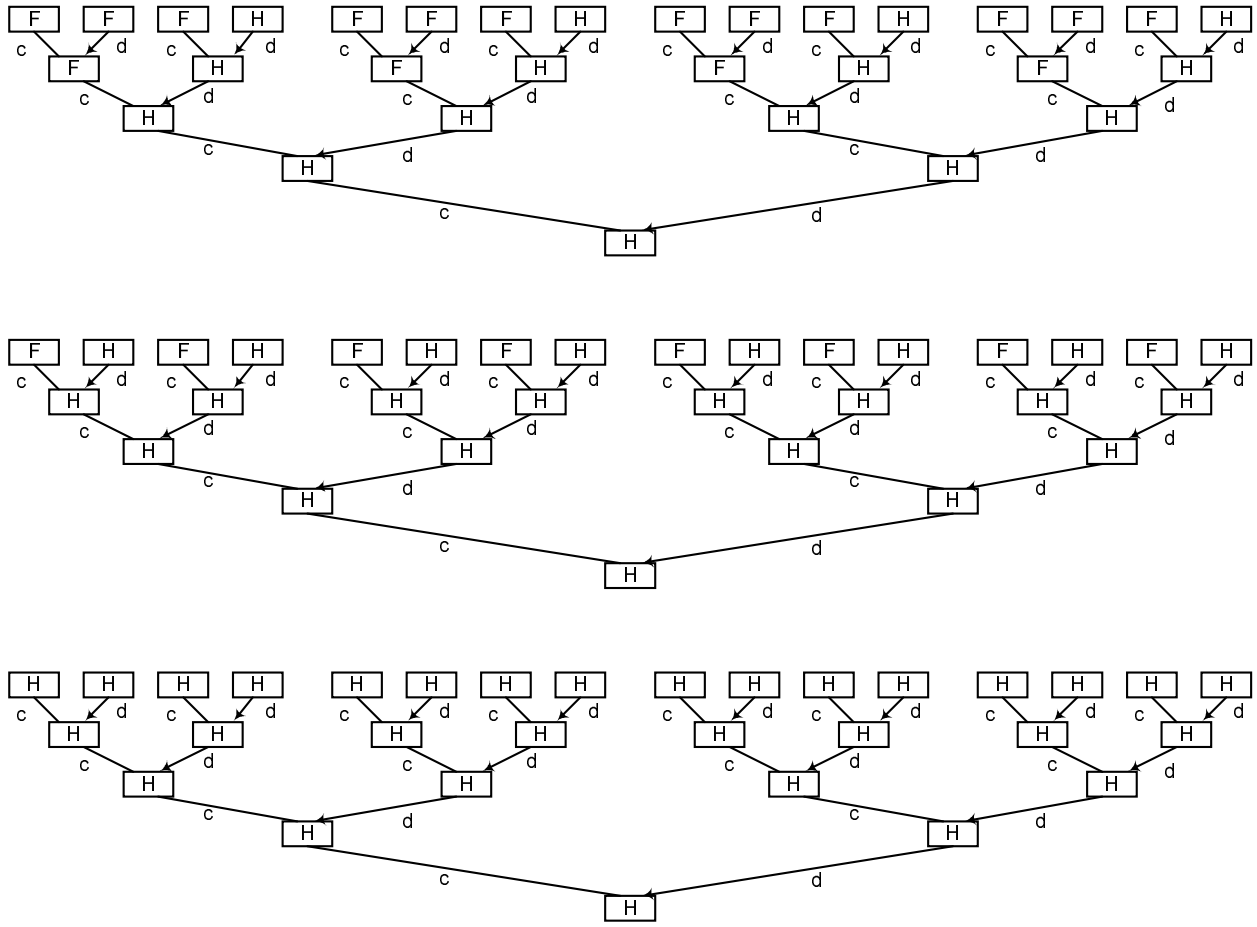


Figure 5 : Some of the many feasible architectures for partial product accumulation trees that use the HSD-1 format for wordlength $n = 64$. Each PP within the tree is indicated by a rectangular box. As in Figure 2, the letter "F" inside a box indicates a PP which is in the Fully Signed-Digit format, while the letter "H" indicates a PP that is in HSD-1 format.