Designing A Partitionable Multiplier

Hyuk-Jun Lee and Michael Flynn

Technical Report : CSL-TR-98-772

October 1998

This research has been supported by the Dept. of the Army under contract DABT63-96-C-0106

Designing A Partitionable Multiplier

by

Hyuk-Jun Lee and Michael Flynn

Technical Report : CSL-TR-98-772

October 1998

Computer Systems Laboratory

Stanford University

Gates Building 3A, Room 230

Stanford, California 94305

Email: hyukjunl@umunhum.stanford.edu

Web: http://umunhum.stanford.edu

Abstract

This report presents the design of a 64-bit integer multiplier core that can perform 32-bit or 16-bit parallel integer multiplications(PMUL) and 32-bit or 16-bit parallel integer multiplications followed by additions(PMADD). The proposed multiplier removes sign and constant bits from its core and projects them to the boundaries to minimize the complexity of base cells. It also adopts an array-of-arrays architecture with unequal array sizes by decoupling partial product generation from carry save addition. This makes it possible to achieve high speed for 64-bit multiplication. Two architectures, which are done in dual-rail domino, are tested for functionality in Verilog and simulated in HSPICE for TSMC 0.35μm process. The first architecture is capable of both PMUL and PMADD. The estimated delay is 4.9 ns(excluding a final adder) at 3.3V supply and 25c with an estimated area of 6.5 mm^2. The second architecture, only capable of PMUL, has an estimated delay of 4.5 ns. Its estimated area is 5.2 mm^2.

Key Words and Phrases: Multiplication, partitionable, MMX

Contents

List of Figures

List of Tables

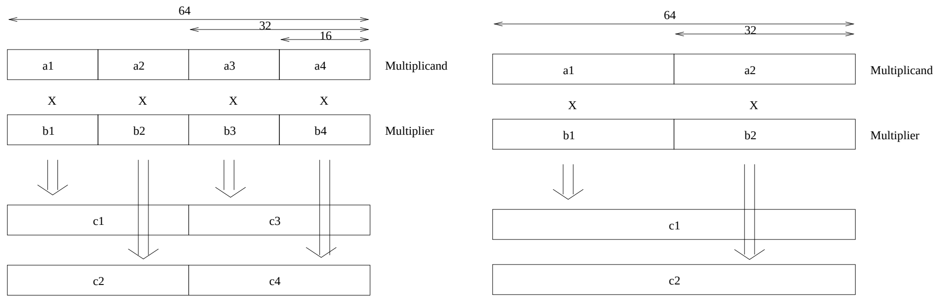


Figure 1: parallel multiplication(PMUL) (a) 16-bit (b) 32-bit

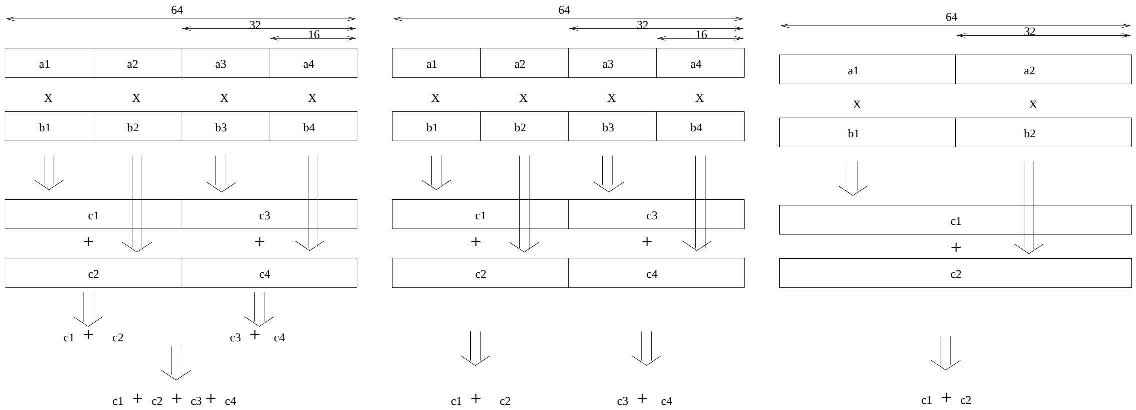


Figure 2: parallel multiplication and addition(PMADD) (a) 16-bit (b) 16-bit (c) 32-bit

1 Introduction

Current CPUs with MultiMedia eXtension(MMX) instructions exploit sub-word parallelism by employing programmable datapath. One example is four parallel 8-bit adders implemented with a 32-bit adder with programmable carry chains.

A multiplier is a basic arithmetic unit that could potentially use the sub-word parallelism. It is composed of three major blocks: an array of multiplexors to generate partial products, carry save adder(CSA) arrays or trees to sum partial products, and a final adder. An N-bit multiplier(no encoding) consists of roughly NxN multiplexors, (N-2)xN carry save adders, and a 2N-bit carry propagation adder(CPA). It is a sufficient number of elements to build at least M (N/M)-bit multipliers where N is divisible by M. For instance, a 64 bit multiplier, composed of 64×64 multiplexors, 62×64 CSAs, and a 128-bit CPA, can build 2 32-bit multipliers since they only require $2 \times (32 \times 32)$ multiplexors, $2 \times (30 \times 32)$ CSAs and 2 64-bit CPAs.

A set of MMX instructions involving multiplication can be catagorized into two groups. The first kind is parallel multiplication of 16-bit or 32-bit integers(PMUL) shown in figure 1. The second, figure 2, is parallel multiplication followed by addition of the multiplication results(PMADD). To achieve these operations, the datapath of a multiplier has to be partitioned into smaller size multipliers. We refer this as a partitionable multiplier in this report.

The design of a partitionable multiplier introduces new problems that were not come across in the design of an unpartitioned multiplier. First, the most important question is how we can minimize the complexity that is added to an unpartitioned multiplier. For instance, several operation mode, such as 64-bit, 32-bit, or 16-bit, complicates multiplexor design. It is because a multiplexor cell that generates a Booth encoded output $\{\pm 2M, \pm M, 0\}$ in 64-bit mode, now should be able to generate a constant$\{0 \text{ or } 1\}$, or a sign bit in another mode. The complexity is interpreted as more wires and gates per cell, which increases area and degrades performance. Second, how we can guarantee the correctness of operations is another crucial issue. The results of smaller multipliers located inside the core are likely to interfere with each other while they are propagated to the boundaries of a multiplier. It requires a mechanism that prevents the internal results from being corrupted. Third, researchers[2][3] have shown different trade-offs between architectures and circuit styles of a multiplier and have given the best architecture and circuit style based on constraints. However, it is not clear that a certain architecture and circuit style that best fits for an unpartitioned multiplier is the best for the partitionable multiplier due to the complexity and constraints set by aforementioned operations.

In section 2, background material on multiplication algorithms and multiplier architectures is presented. In section 3, various design issues including a partition method, an implementation of various operation mode, and related problems, are discussed. In section 4, two major techniques to solve the problems are discussed. In section 5, two proposed architectures based on these techniques are compared.

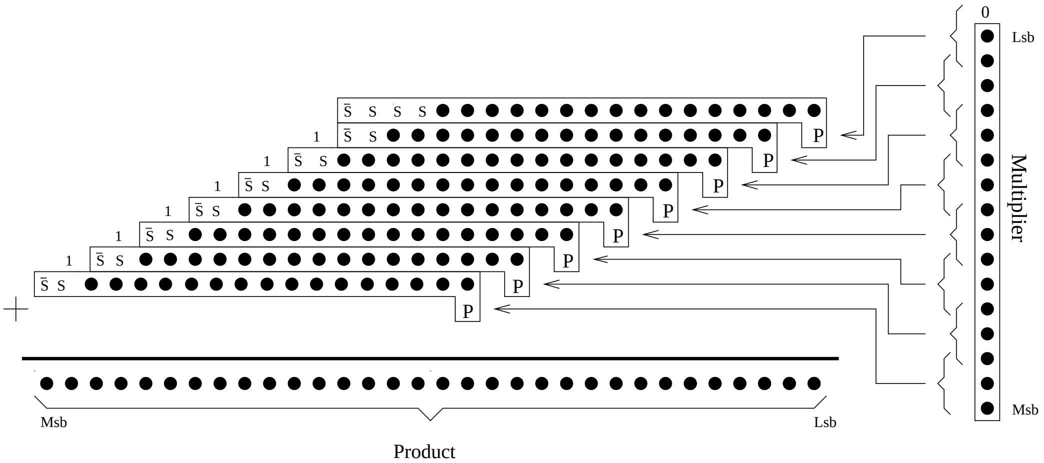
2 Background

2.1 Signed multiplication based on Booth's algorithm

A multiplication is obtained by addition of a certain number of partial products. A partial product is defined as a multiple of a multiplicand. Since the number of addition is directly interpreted as delay and silicon area, a small number of partial products to start with is crucial. Without any encoding method, N partial products need to be added for an N-bit multiplication. Modified Booth's algorithm[1], whose dot diagram is shown in figure 3, reduces the number of partial products to $\lceil \frac{N}{2} \rceil$ or $\lceil \frac{N+1}{2} \rceil$ by partitioning multipliers into overlapping groups of 3 bits. Each group is decoded in parallel to select a multiple of the multiplicand from the set $\{\pm 2M, \pm M, 0\}$. For an unsigned multiplication, an extra row is necessary to make the result positive. However, for a signed multiplication this step is not needed. We will focus on the signed multiplication throughout this report.

2.2 Tree vs. Array

While Booth encoding helps reduce a latency by reducing the number of additions, interconnection between additions can reduce the latency further by parallelizing the additions. Trees[6][7] and arrays are two different interconnection schemes. Trees of adders exploit



Partial Product Selection Table	
Multiplier Bits	Selection
000	+0
001	+M
010	+M
011	+2M
100	-2M
101	-M
110	-M
111	-0

Figure 3: 16 bit signed Booth 2

parallelism in adding partial products to reduce the critical path. These parallel additions, however, require more wiring tracks between two stages of additions in a tree. This slows down the circuit in two ways. First, more tracks and longer wires increase a cell area and slow down the speed by increasing capacitive loads due to longer wires. Second, the number of tracks make it hard to use fast circuit styles such as dual-rail domino[10] which requires two wiring tracks per output. This is a substantial setback considering that a dynamic adder is about twice as fast as the static version of the cell[3]. Arrays, on the contrary, have simple and regular layout at the cost of speed. Since additions are done in a serial fashion, smaller number of wiring tracks are needed and a dynamic logic fits well to build a faster cell. Hybrid architectures such as a tree of arrays and an array of arrays maintain simplicity and regularity inside a sub-array while they exploit parallelism by connecting the sub-arrays using trees or arrays.

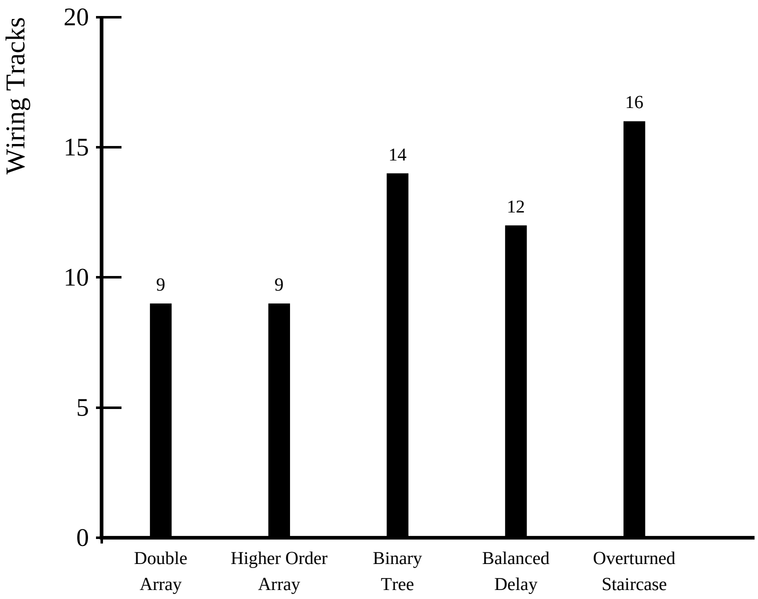


Figure 4: Minimum number of tracks per channel to use dual-rail circuits;double precision,Booth 2 encoding,MOSIS 0.8μ process

Recent works[2][3] compare the performance of trees, arrays and hybrids. The advantage of using logarithmic interconnection(trees) was diluted by its irregularity, a large area, and slow circuit styles. Figure 4, taken from Hesham's thesis[2], shows the number of tracks required for various architectures in dual-rail domino. The two leftmost bars represent array-of-arrays architectures and the three bars in the right represent trees. Tree architectures require 3-7 more tracks than the hybrids. For this reason, trees are forced to choose single-rail design such as static CMOS when the required number of tracks is not allowed. Figure 5 shows the delay of various architectures when the maximum number of tracks per channel is set to (a) 10 per channel and (b) 20 per channel. Higher order array(an array

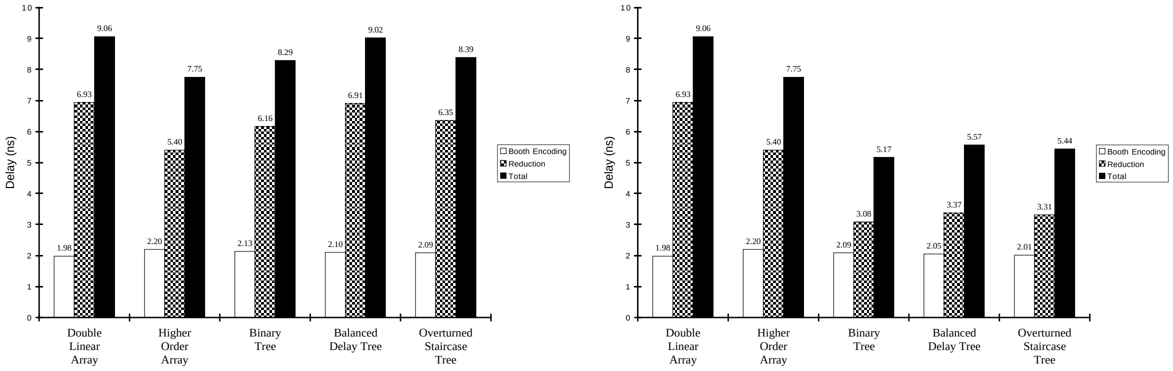


Figure 5: Delay for various topologies:double precision,Booth 2 encoding,MOSIS 0.8μ process (a) 10 tracks per channel (b) 20 tracks per channel

of arrays) outperforms trees in (a) since trees are forced to use static CMOS.Considering 80 to 120 $lambda$(10-15 tracks per channel) is reasonable bit pitch, tree architecture would be even worse for a partitionable multiplier since it requires more tracks than an unpartitioned multiplier. This research shows that we need two to five more tracks needed in a partitionable multiplier. Hence, in this report we focus on the hybrid architectures such as an array of arrays and a tree of arrays.

3 Design issues

3.1 Parallel multiplication(PMUL)

In many design, multiplicand and multiplier bits are running perpendicular to each other. Figure 6 shows a 64-bit multiplier broken into sub-blocks(smaller multipliers) for 16-bit and 32-bit mode. In 16-bit mode, multiplicand bits run vertically. They are labeled from bit 0 to 63 and broken into a1,a2,a3,a4. Multiplier bits, labeled 0 to 63 and broken into b1,b2,b3,b4, go through Booth encoders and run horizontally. Each sub-block in (a) represents a 16-bit multiplier. Groups of multilpicand and multiplier bits with same indices, i.e. a1 and b1, are crossed in shaded areas. This forces the sub-blocks producing multiplication results to be placed diagonally. Multiplexor outputs for non-shaded sub-blocks are masked out to zero. In 16-bit mode, the first partial product is $\{$48 zeros, M15,M14,M13,...,M2,M1$\}$. The 8th partial product is $\{$32 zeros, M31,M30,M29,...,M17,M16, 16 zeros$\}$. The results of multiplications are generated at the right and bottom sides of shaded sub-blocks. Then, the results are propagated to the right or the bottom of the 64-bit multiplier. For instance, the result of $a1 \times b1$ is propagated to the side that is marked as c1. The results of sub-blocks remain unchanged on their ways to the sides because they are added with zeros from masking.

One thing that is missing from figure 6 is all the sign and constant bits that were required in the signed Booth multiplication, figure 3. Figure 7 shows superimposed sign and constant bits on 16-bit and 32-bit mode. As it is pointed out, two adjacent results, i.e. $a1 \times b1$ and

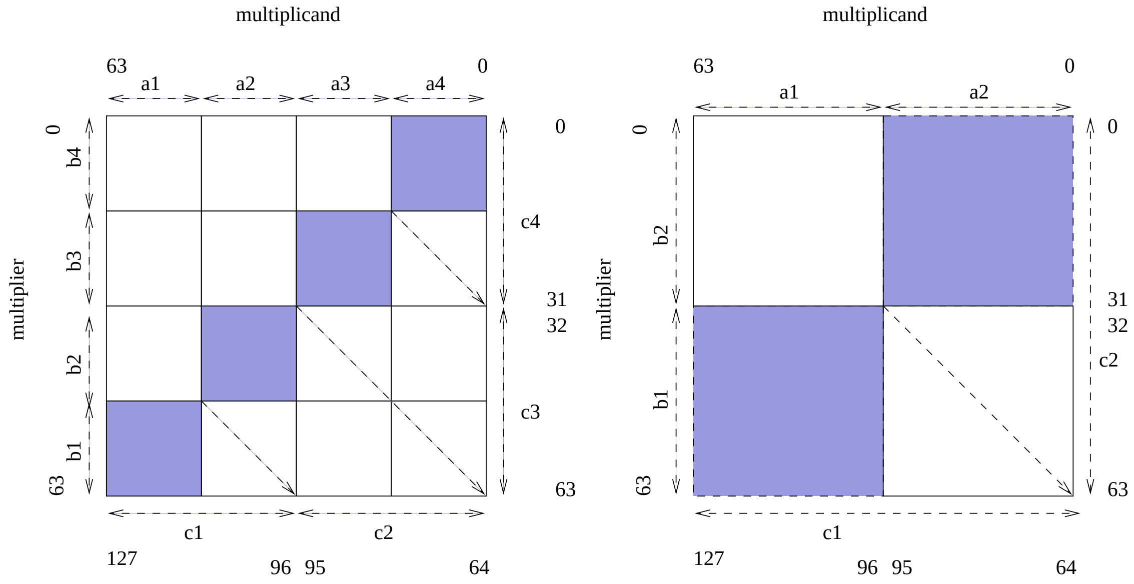


Figure 6: Parallel multiplication without sign and constant bits (a) 16-bit (b) 32-bit

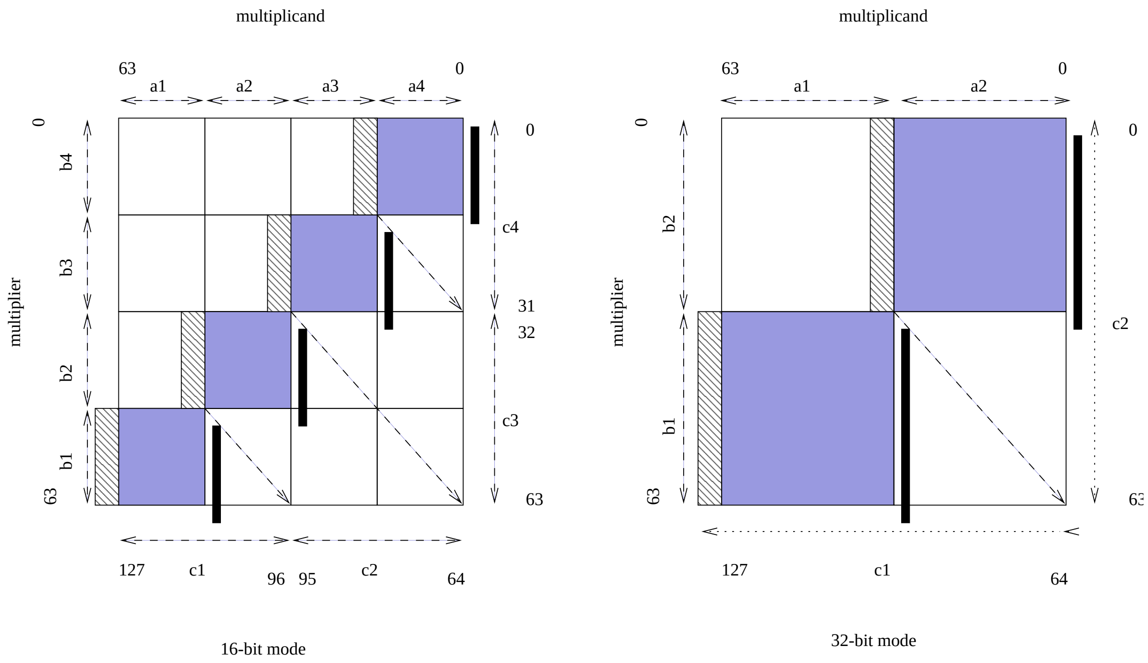


Figure 7: Parallel multiplication with sign and constant bits

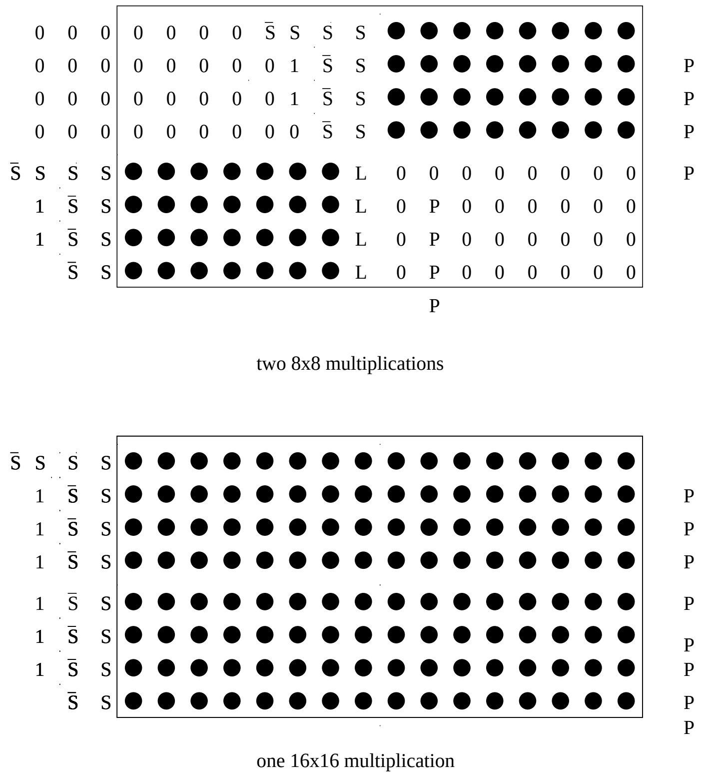


Figure 8: A dot diagram for a 16-bit partitionable multiplier

$a2 \times b2$, are now interfered in propagation because of carries from sign and constants bits.

Figure 8 shows a dot diagram for a 16-bit partitionable multiplier that is programmable as two 8-bit multipliers. In contrast to figure 3, all the partial products are lined up so that a dot finds the same weight dot in the next row by skipping two spaces to the right. A black dot represents a regular multiplexor. S,P, and L are defined as:

S : (Sign of Booth encoding for a current partial product) XOR (MSB of multiplicand)
P : (Sign of Booth encoding for a previous partial product)
L : a regular multiplexor with 2M set to 0

Overlapping two diagrams in figure 8 implies that multiplexors in the sign and constant bit positions become substantially complex. It is readily seen that the complexity of multiplexors get worse for a 64-bit multiplier. Minimization of this comlexity, therefore, is a critical issue.

3.2 Parallel multiplication followed by addition(PMADD)

While parallel multiplications(PMUL) use sub-blocks such that the results of smaller multipliers don't have same weight, parallel multiplications followed by additions(PMADD) use

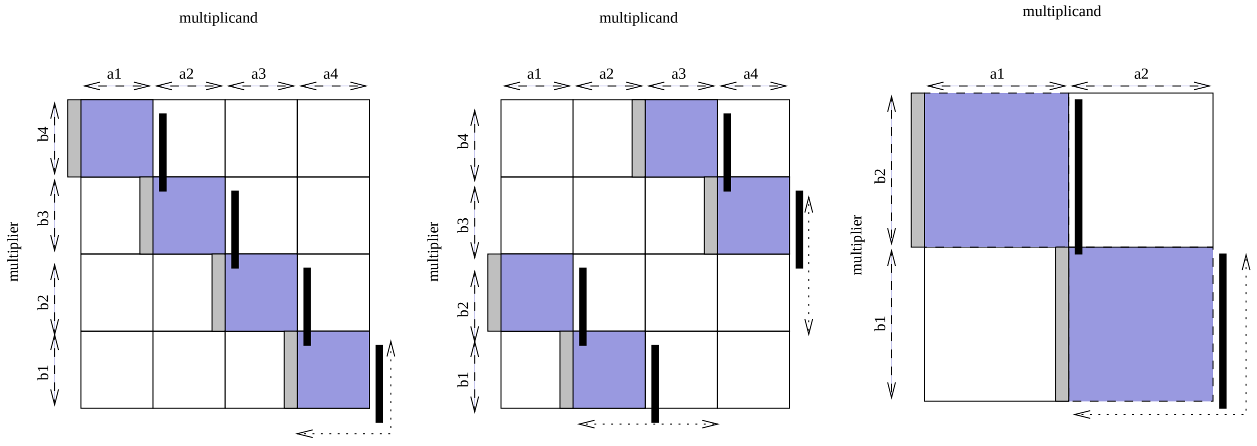


Figure 9: PMADD (a) 16-bit (b) two 16-bit (c) 32-bit

sub-blocks such that the results have same weight as shown in figure 9. (a) represents 16-bit parallel multiplications followed by serial additions of four results. This can be thought of as an inner vector product or a 4-tab finite impulse response(FIR) filter. Since all the sub-blocks have the same weight, the result of the first(leftmost) multiplier is added with that of the second and its result is again added with the result of next multipliers. (b) shows two separate PMADDs and (c) shows a 32-bit version. These operations are particularly interesting since all the multiplications and additions can be done in one multiplication cycle. This mode, however, requires additional attention to several overlapping bits shown in figure 9. The tail of P bits(black strips) is overlapped with multiplier sub-blocks(gray area). In (a), three ovelapping P bits need to be removed from the core and added at the end using additional carry save adders. In (b) and (c), two and one P bits need to be taken care of separately. These additional bit don't appear in the critical path since they can be added in parallel with other carry save additions in the core.

4 Key Methods

4.1 Scheme I: projecting sign and constant bits to the boundaries

Figure 10 shows a dot diagram (a) for 16-bit signed multiplication and two other equivalent ones, (b) and (c). The difference between (b) and (a) is that (b) extracts all the constants from each partial product and adds them back in the form of an extra row. The equivalence of (a) and (b) can be readily seen by adding 0101010101010110 and $\bar{S}S\bar{S}S\bar{S}S\bar{S}S\bar{S}S\bar{S}S\bar{S}S\bar{S}S$. In (c), all S and P's are also dropped from each partial product in (b) and added back at the bottom.

The equivalence provides a designer with the flexibility of positioning 1, S, and P bits. Figure 11 shows transformed version of figure 6 for 32-bit and 16-bit mode using equivalence (b). The shaded area is a constant vector, 1s and 0s. The configuration I projects all the 1s to the right or the bottom. The total number of partial products is 33 in which the

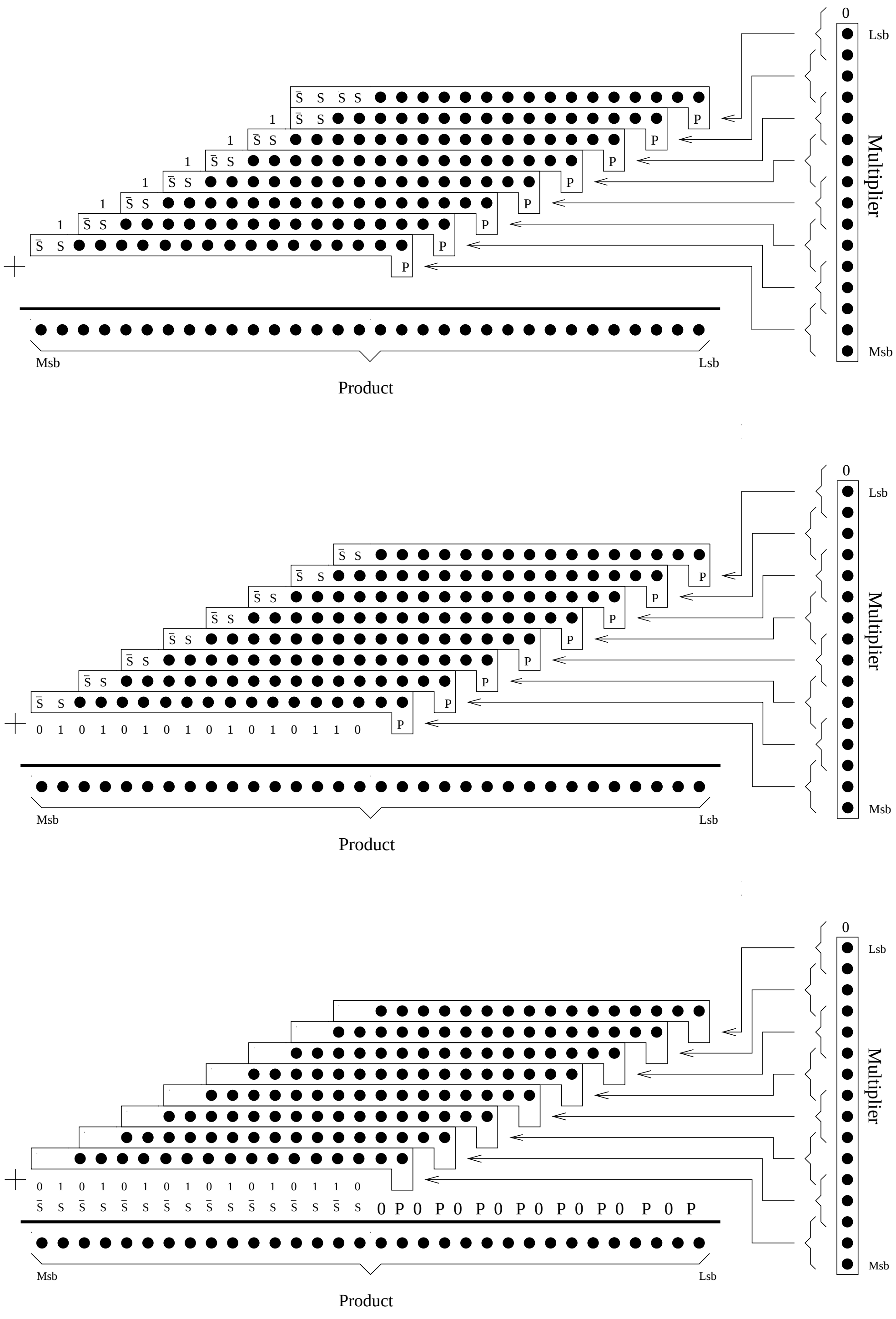


Figure 10: Three equivalent diagrams: (a),(b),(c)

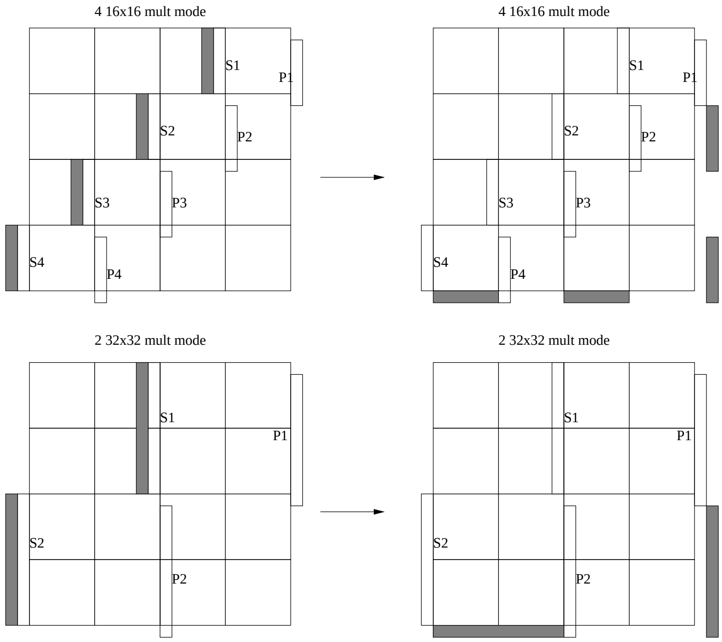


Figure 11: Configuration I: using equivalence (b)

first 32 partial products are generated from Booth 2 encoding and the final one is from dealing with the last P bit in 64-bit mode. The constant vector added at the bottom can be projected to the left side. However, this wouldn't reduce the height of partial products because of the last P bit. The advantage of this configuration is that first it makes the first partial product's sign bits same as others. That is, $\bar{S}SSSM_{15}M_{14}$... becomes $\bar{S}SM_{15}M_{14}$.... Second, constant vectors only depend on three different modes, neither multiplicand nor multiplier, and this requires only a few wires for mode selection bits, which leads to a small additional area.

Figure 12 shows another configuration using equivalence (c). In addition to the equivalence (c), it went through an additional step: projecting sign bits placed at the bottom to the left side. This additional step is necessary for two reasons. First, Booth encoding outputs, required to generate S and P, are available horizontally. Second, adding a vector of sign bits at the bottom would increase the number of partial products in a critical path to 34. The advantage of this configuration is that it makes multiplexors compact and interference between results proceeding to the right side can be avoided. However, interference between the results proceeding to the bottom still remains because in 16-bit mode $\bar{S}S$ in MSB positions, circled in figure 12, generate carries. Compared to configuration I, it takes more area to route Booth encoding signals, 64 wires each side, and multiplicand bits at the boundary.

In summary, scheme I reduces the complexity of a multiplexor and partially solves the interference problem.

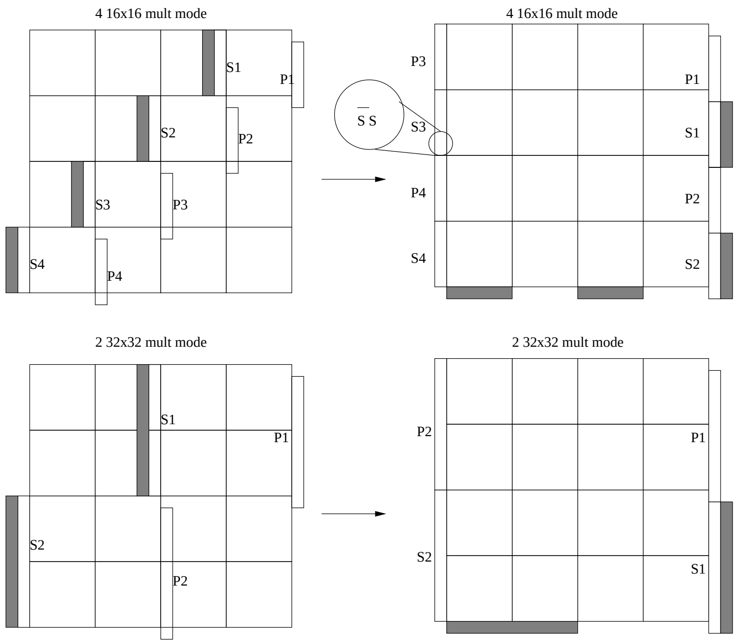


Figure 12: Configuration II: using equivalence (c)

4.2 Scheme II: decouple partial product generation from carry save addition

Interference between adjacent results can be avoided by dropping two MSBs located at the boundary of two multiplications. Figure 13 shows the boundary of two multiplications, $a3 \times b3$ and $a4 \times b4$. Those two MSBs are enclosed by (). Dotted wires between two partial products represent sums and carries. A sum goes directly down to the next level and a carry is shifted one to the left. We can picture each number or character associated with two incoming and outgoing wires as a (3,2) counter. The validity of this scheme can be shown by looking at the values of crosscut wires. In the worst case, they are

a:	00	10	10	11	11	11	11	...
b:	00	11	01	11	11	11	11	...
c:	01	00	11	01	01	01	01	...
d:	10	01	00	10	10	10	10	...

Since d has at most one 1 per (3,2) counter, the result is confined within 32 bits. The dropped two MSBs need to be added by a final CPA to get correct results. It is achieved by either adding 01 or 10 in two MSB positions of a segmented CPA since we dropped $\bar{S}S$.

The importance of this scheme is not only prohibiting interference between two adjacent results from corrupting the values in configuration I and II. But, it implies that a designer has far greater flexibility to choose an architecture among ones discussed in section 2.2. Without this scheme, an array-of-arrays architecture is forced to have equal array sizes,

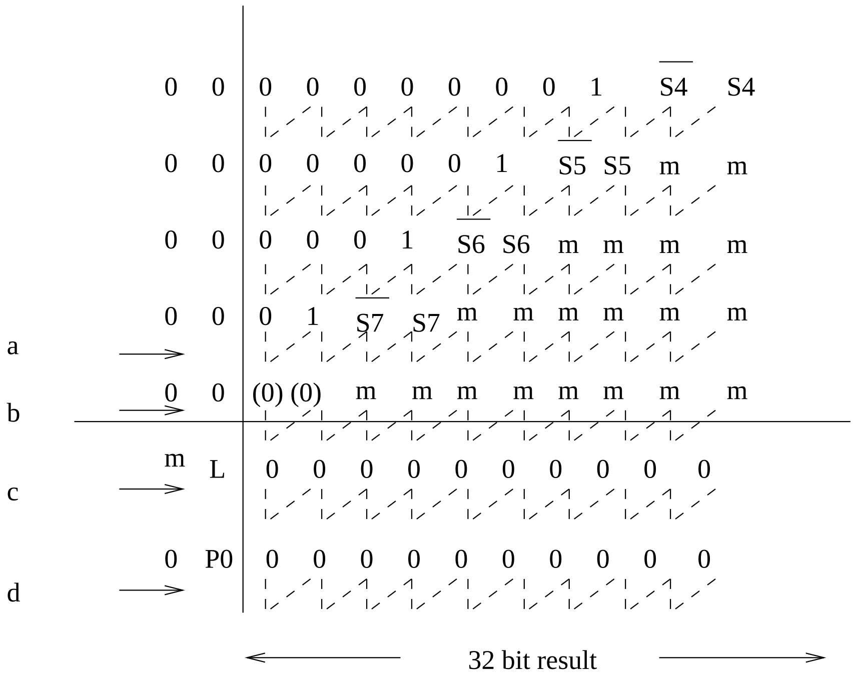


Figure 13: CSA connections at the boundary

i.e., 8 partial products in 16-bit mode, because otherwise a result of 16-bit multiplication is corrupted by adjacent one. Optimal sub-array sizes for an array-of-arrays should be increasing toward the end of an array to match the delay through all the sub-arrays. For instance, optimal sizes for 27 partial products(IEEE floating point standard) are 5-5-7-10. Our scheme allows different array sizes by decoupling partial product generation from carry save addition. This makes it possible to use an optimal architecture for 64-bit mode.

5 Proposed architectures

5.1 Array of arrays

5.1.1 Architecture I

This architecture is based on configuration I, shown in figure 11, and scheme II. A multiplexor, shown in figure 14, is implemented in dual-rail domino and laid out in $130\lambda \times 243\lambda$ area using 2 metal layers. Four extra signals are added to an unpartitioned multiplier. Table 1 shows these signals and how this cell is programmed to act as different cell types. A cell produces $\{\pm 2M, \pm M, 0\}$ if row_sel=1. Cell becomes L,P,and S when row_sel=0. x represents don't care condition and mcand[i] represents multiplicand bit at ith position

It chose an array-of-array structure, whose layout is shown in figure 15, and array sizes are chosen to be 7-7-8-11 from HSPICE simulation. The numbers are summed up to 33 since we have 32 partial products and one constant vector. Results of Array 1 and 2 are

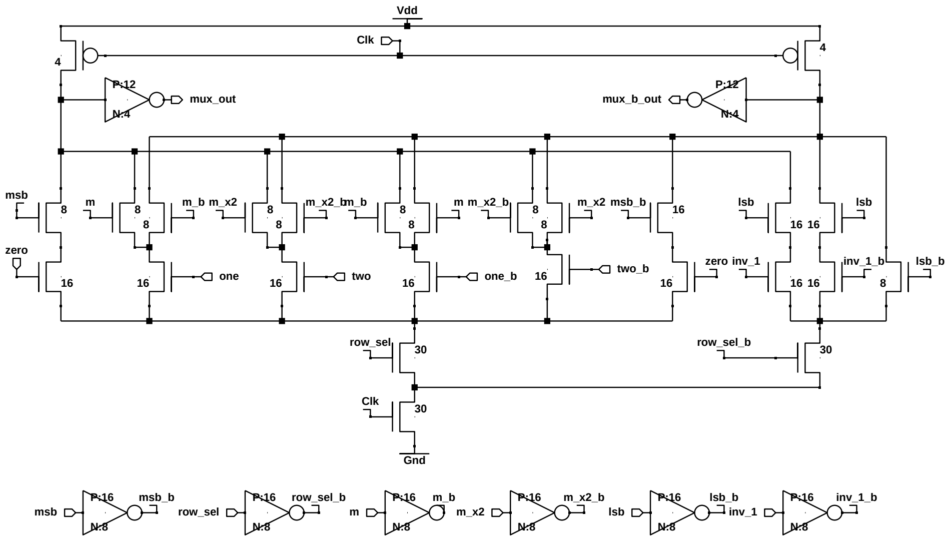


Figure 14: Multiplexor for Architecture I

	row_sel	msb	lsb	inv_1	m	m_x2
Regular Mux	1	0	x	x	mcand[i]	mcand[i-1]
S	1	0	x	x	mcand[i]	mcand[i-1]
$\bar{S}$	1	1	x	x	mcand[i]	mcand[i-1]
L	1	0	x	x	mcand[i]	0
P	0	x	1	valid	x	x
0(mask)	0	x	0	x	x	x

Table 1: Truth table for multiplexor inputs, Architecture I

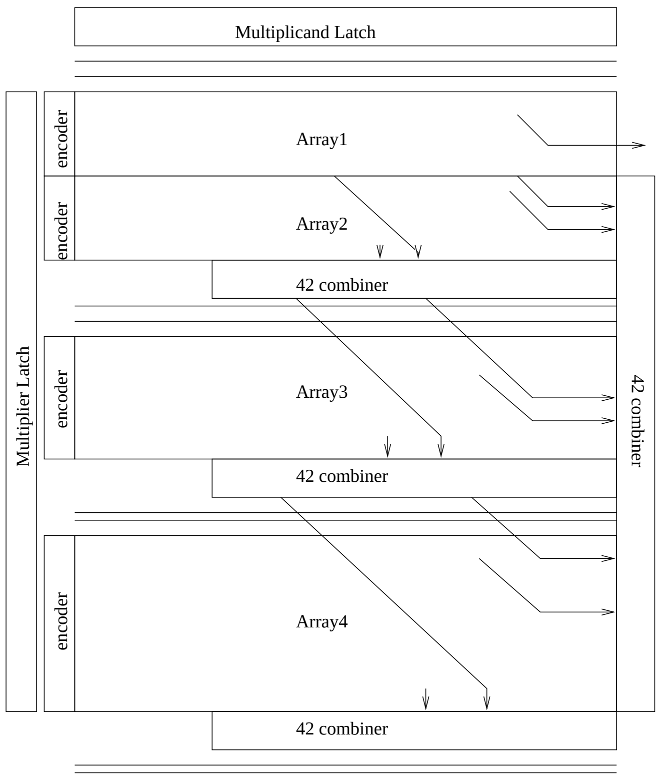


Figure 15: Layout for Architecture I

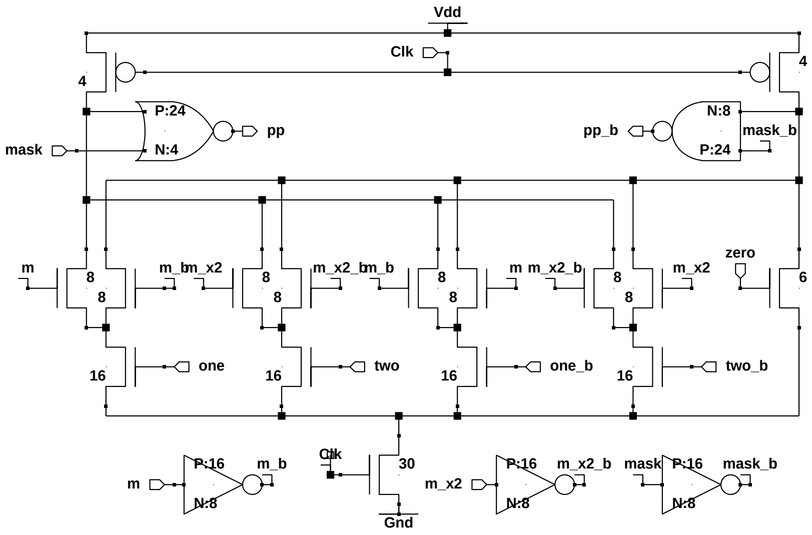


Figure 16: Multiplexor for Architecture II

added by (4,2) combiners. The result is again added with a result of array 3 and array 4 serially. Control signals are routed between arrays. The total number of carry save adders in critical path is 11.

Two versions were implemented. The first is only capable of PMUL which takes relatively small area for control wires between multiplexors. The second can perform both PMUL and PMADD which takes more area than the first for the control wires.

5.1.2 Architecture II

This architecture is based on configuration II, shown in figure 12, and scheme II. A multiplexor, shown in figure 16, is quite simple compared to architecture I and laid out in $116\lambda \times 189\lambda$ area. A regular multiplexor output of the cell is masked out to 0 by setting a mask bit.

Again, it chose an array-of-arrays structure. Figure 17 shows the layout. This architecture has an additional area allocated for sign bits and wire routing in the right and left side of a multiplier core. However, the total area is smaller than architecture I because it has a smaller core.

The functionality of this architecture is limited to PMUL. That is because sign and constant bits extracted from smaller multipliers in PMADD have same weights and increase

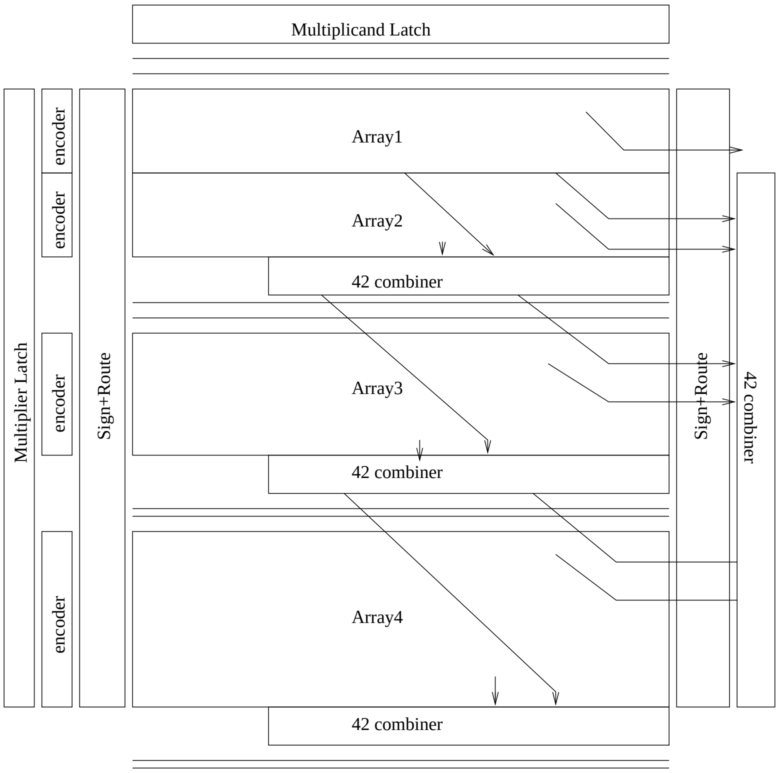


Figure 17: Layout for Architecture II

the number of carry save adders in the critical path.

5.2 Tree of arrays

This is another architecture that was considered in the beginning. The difference between a tree of arrays and an array of arrays is that sub-arrays are added in parallel in a tree of arrays. This parallel addition requires four more tracks per channel than the array of arrays in dual-rail design. At the cost of this additional tracks, a tree of arrays tends to have slightly smaller number of carry save adders in the critical path. In this design, however, it turns out that the size of sub-arrays is 8-8-8-9 that has 11 carry save adders in its critical path. This was the same number of CSAs found in the array-of-arrays architecture. For this reason, this architecture was not built.

5.3 Simulation results

A delay is measured by simulating a critical path. The critical path is broken into three segments: clock to a Booth multiplexor's output through a Booth encoder, a Booth multiplexor's output to the first array, and three stages of (4,2) combiners. We modeled a wire from estimated length of wires and unit length capacitance and resistance extracted for TSMC 0.35 μm process. All the fan-outs and wiring loads were taken into account. Table 2 shows the result. Only architecture I is capable of PMUL and PMADD. The simulation results for architecture I are based on the second version and given in the table. The results for first version, only for PMUL, is not in the table because its area and performance are worse than architecture II. The architecture II takes much less area than architecture I and perfroms relatively well compared to an unpartitioned multiplier but is not capable of PMADD.

		unpartitioned Multiplier	architecture I	architecture II
Area	Bit Pitch	102 λ	130 λ	116 λ
	Booth Mux	168 λ	243 λ	189 λ
	CSA	160 λ	160 λ	160 λ
	Total(excl. a final adder)	4.2 mm^2	6.5 mm^2	5.2 mm^2
Delay	Booth encoder	0.37 ns	0.37 ns	0.38 ns
	Booth Mux	0.21 ns	0.35 ns	0.24 ns
	Total(excl. a final adder)	4.16 ns	4.90 ns	4.49 ns

Table 2: Area and Delay

6 Summary

This report has shown the design of a partitionable multiplier. The main goal of this research was to increase the throughput for 32-bit and 16-bit multiplication by finding an optimal architecture. It is achieved by two schemes we explained in section 4. Two different architectures based on these schemes were simulated. The first is based on a little complex multiplexor cell with capability of both PMUL and PMADD and the second is based on a simple multiplexor cell with its capability limited to PMUL. The increased throughput for 32-bit and 16-bit PMUL of a partitionable multiplier(architecture II) over an unpartitioned multiplier were 1.85 and 3.7 while it takes 20 % additional area. The increased throughput for 16-bit PMADD(four integers) of a partitionable multiplier(architecture I) over an unpartitioned multiplier was 4.2 with 70 % increased area.

References

[1] O.L. McSorley, "High Speed Arithmetic in Binary Computers", *Proceedings of the IRE*, 49(1), pp. 67-91, Jan 1961.

[2] A. Hesham, "Area and Performance Optimized CMOS Multipliers", *Ph.D. Thesis, Stanford University*, Aug. 1997

[3] H. Dhanesha, K. Falakashahi and M. Horowitz, "Array-of-arrays Architecture for Parallel Floating Point Multiplication", *Proceedings. Sixteenth Conference on Advanced Research in VLSI*, pp. 150-157, March 1995.

[4] G. Bewick, "Fast Multiplication:Algorithms and Implementation",*Ph.D. Thesis, Stanford University*, 1994.

[5] M. Santoro, "Design and Clocking of VLSI Multipliers", *Ph.D. Thesis, Stanford University*, Oct. 1989

[6] D. Zuras and W. McAllister, "Balanced Delay Trees and Combinatorial Division in VLSI,"*IEEE J. Solid-State Circuits*, vol SC-21, No.5, pp. 814-819, Oct. 1986

[7] Z. Mou and F. Jutand, "A Class of Close to Optimum Adder Trees allowing Regular and Compact Layout", *IEEE Trans. Computers*, pp. 251-254, 1990.

[8] C. A.Mead and L. A. Conway, "Introduction to VLSI systems", Reading, MA, Addison Wesley, 1980.

[9] R. Krambeck, C. Lee and H-F. Lew, "High Speed Compact Circuits with CMOS", *IEEE Journal of Solid State*, pp. 614-618, June 1982.

[10] L. Heller, W. Griffin, J. Davis and N. Thomas, "Cascode Voltage Switch Logic: A differential CMOS Logic Family", *IEEE International Solid State Conference*, pp. 16-19, Feb. 1984.

[11] P. Song, "New circuit and structures for combinatorial multipliers", *Ph.D. Thesis Stanford University*, 1993.

[12] *http://www.stanford.edu/class/ee371*