Energy Efficient Deeply Fused Dot-Product Multiplication Architecture

Shmuel Wimer, *Member, IEEE*, and Israel Koren, *Fellow, IEEE*

***Abstract*—Dot-product is frequently used in many signal processing applications, and for this reason digital signal processors (DSPs) commonly include hardware units to speed up its calculation. In this paper, we propose a new circuit architecture for dot-product hardware units. This design fuses all the steps of the dot-product computation into a unit called a fused dot-product multiplier (FDPM), resulting in enhancements in performance, power and area vs. designs such as multiply-accumulate (MAC) units. We designed and implemented the FDPM in 28-nanometer and 65-nanometer technologies. Comparison of the fused unit to an implementation by an EDA design compiler showed that our architecture consumed 38% less power and 30% less area for the same clock cycle. Comparison to a previously proposed FDPM architecture showed a per-bit 1.9X - 4.7X energy reduction. Analysis of the dependence of the per-bit power consumption on vector length and word-width indicated a good match between theory and practice.**

***Index Terms*—Computer arithmetic, dot-product, multipliers.**

I. Introduction

DOT-products are commonly used in many signal processing applications such as FFT. Given two vectors with positive fixed-point operands $\mathbf{x}=(x_1,\ldots,x_m)$ and $\mathbf{y}=(y_1,\ldots,y_m)$, their *dot-product* $z=\mathbf{x}\cdot\mathbf{y}$ (also called the scalar *product*) is given by

$$z=\sum_{i=1}^{m} x_i y_i \,. \tag{1}$$

The basic $x_i y_i$ operation requires an n-bit multiplier where n is the number of bits in each operand. If a radix-4 Booth encoding is used, a carry-save array (CSA) summation of $n/2$ *partial products* (PPs) is required [1]. Fig. 1 depicts a dot diagram of the PPs in (a) with their *final sum and carry partial products* (FPPs) in (b), for which a final addition (not shown) produces the result $z=\mathbf{x}\cdot\mathbf{y}$. For high performance, the $n/2$ PPs can be added by using a *Wallace-tree* with 4:2 compressors. This can be followed by a prefix-tree adder (not shown) to add the $2n$-bit sum and carry FPPs.

The time complexity of computing the multiplication $x_i y_i$ is

$$T=\log_2 n \,. \tag{2}$$

The area A of the multiplier includes the area A_{Wallace} of the Wallace-tree and the area $A_{\text{prefix\_add}}$ of the prefix-tree adder,

$$A=A_{\text{Wallace}}+A_{\text{prefix\_add}} \,. \tag{3}$$

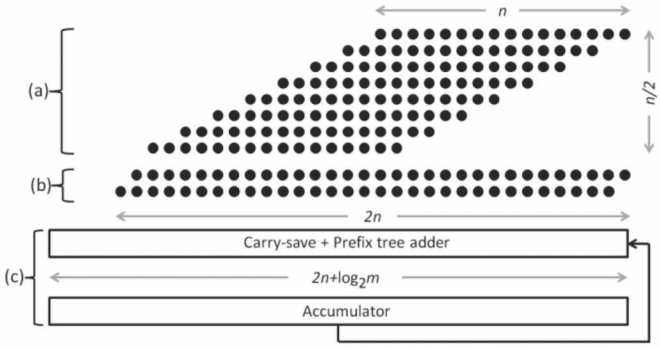


Fig. 1. A radix-4 multiplication dot diagram (a) and (b), within a MAC (c).

Radix-4 Booth encoding produces $n\times n/2=n^2/2$ dots at the first level of the Wallace-tree, as shown in Fig.1 (a). This is reduced by a factor of 2 from one level to the next with the use of 4:2 compressors. The area of the entire Wallace-tree is obtained by summing over all the tree levels yielding

$$A_{\text{Wallace}}=\sum_{l=0}^{\log_2 n-2}\frac{n^2/2}{2^l}<2\frac{n^2}{2}=n^2 \,. \tag{4}$$

The area of the prefix-tree adder which adds two $2n$-bit numbers can be estimated by summing the dots over all the levels of the adder tree as follows

$$A_{\text{prefix\_add}}=\sum_{l=0}^{\log_2 n}\frac{2n}{2^l}<4n \,. \tag{5}$$

Ignoring smaller orders of magnitude it follows from (2)-(5) that the area and the area-time product complexities for multiplication are, respectively,

$$A=n^2, \quad AT=n^2\log_2 n \,. \tag{6}$$

The final adder can be fused with the array of the PPs to form a *fused multiply-add* (FMA) [2]. This, however, does not change the area and time complexities.

To obtain the dot-product, multiplication must be performed m times whereas the summation can be built up incrementally with an accumulator for which a *multiply accumulate* (MAC) unit can be used by adding stage (c) in Fig. 1. Hence the area-time product complexity for a dot-product MAC implementation is

$$AT = mn^2 \log_2 n . \quad (7)$$

The MAC design described in [3] uses a CSA that has a delay of order n. Fig. 2(a), taken from [4], illustrates the 2-stage pipelined MAC proposed in [3], which requires $2m$ clock cycles to compute the dot-product in (1). It was modified and improved in [4] which, as shown in Fig. 2(b), takes the final addition out of the MAC loop, thus reducing the number of clock cycles required to compute (1) to $m+1$.

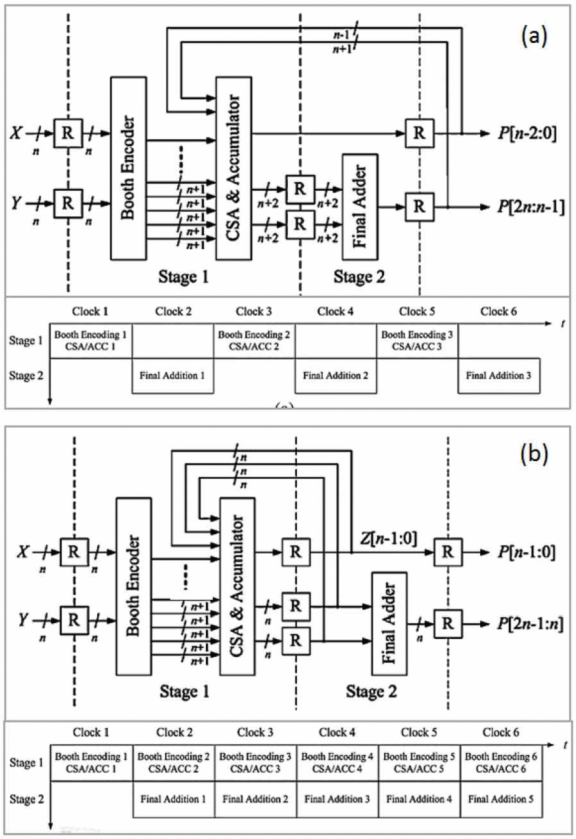


Fig. 2. MAC architecture: (a) proposed in [3], (b) modified in [4].

Fusion of the multiplication and addition steps into a single combinational circuit was proposed in [5] for a dot-product of two 2-element vectors. This type of operation is very common in FFT butterfly operations. We elaborate on this circuit since we will use it later for comparison with our implementation.

Fig. 3(a) shows a conventional parallel dot-product unit, and Fig. 3(b) shows a 32-bit floating-point *fused dot-product multiplier* (FDPM) unit. The fusion proposed in [5] results in a single combinational circuit comprising the two multipliers and the adder, and operating in a single clock cycle.

As shown in Fig. 3(c), the encircled circuits of the multipliers and adder in Fig. 3(b) were implemented as individual entities by the physical synthesis compiler. We propose a deeper fusion of the multipliers and the adder at the bit-level that yields a better power-delay product with considerably higher efficiency at high clock speeds. The authors of [5] presented the implementation in a 45-nanometer technology, reporting the power consumption and the propagation delay of the combinational logic. Their results are compared to our implementation in the experimental result section. Another fused dot-product hardware implementation of vectors comprising four elements is mentioned in [6], in the context of a reconfigurable architecture tailored for matrix operations. The implementation is similar to [5].

Since this paper deals with both circuit architecture and its physical implementation, a few words on the custom layout of the Wallace-tree are in order. The authors in [7] presented a notably efficient implementation, where they eliminated the natural dead area arising from inherent Wallace-tree expansion of the partial products from n-bit at the top of the tree to $2n$-bit at the bottom. As shown in Fig. 4(a), the authors split the parallelogram dot diagram of Fig. 1 around its center into lower bits on the right part and upper bits on the left part, and then applied a Wallace-tree summation to each part in opposite directions. Fig. 4(b) shows how the two opposite triangles fit nicely to form a rectangle, yielding an area-efficient layout.

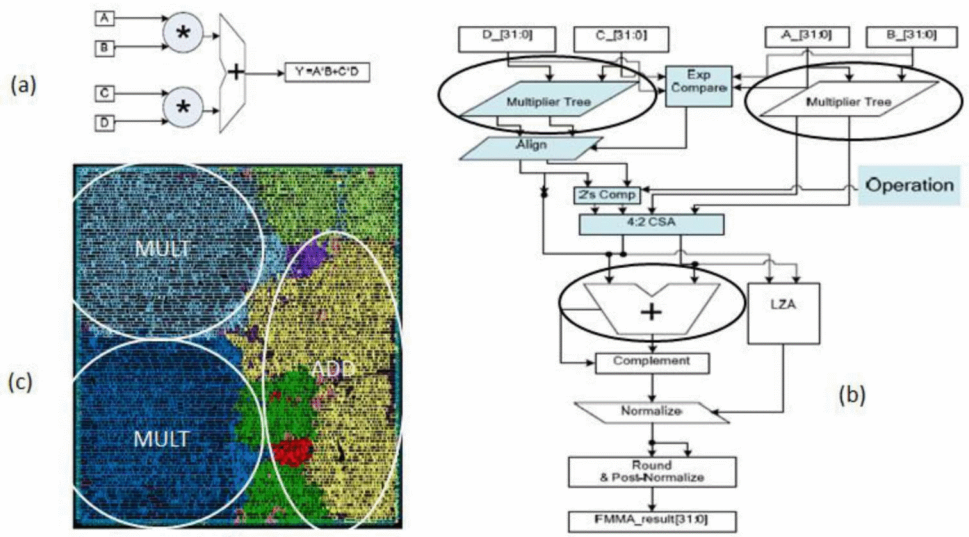


Fig. 3. FDPM [5]: (a) conventional two 2-tuple vectors parallel dot-product, (b) 32-bit floating-point FDPM unit, (c) layout generated by physical compiler.

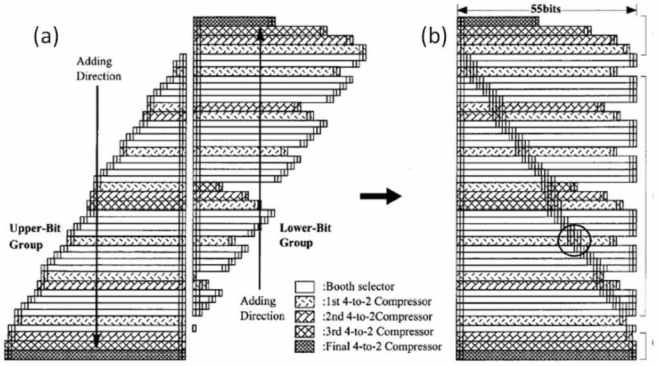


Fig. 4. Folding the Wallace-tree parallelogram into a rectangle [7].

The rest of the paper is organized as follows. Section II presents acceleration by deep fusion, whose power and energy are analyzed in Section III. Section IV presents the experimental results of various hardware implementations, and Section V concludes the discussion.

II. Accelerating the Dot-Product by Fusing the Wallace-Trees

In an effort to consume less power and area for the same speedup, we propose a combinational, modified Wallace-tree architecture. It speeds up the dot-product computation by a factor of m. The largest vectors and word sizes studied in this paper are $m=64$ and $n=64$, respectively. Since each dot-product operation requires two vectors, the largest case requires a $2\times 2^6\times 2^6 = 8$k bits cache-bus width. Such buses are feasible, and a larger cache-bus width of $2^{14} = 16$k bits was reported in [8]. The internal cache architecture must of course support the read of such number of bits in a single access.

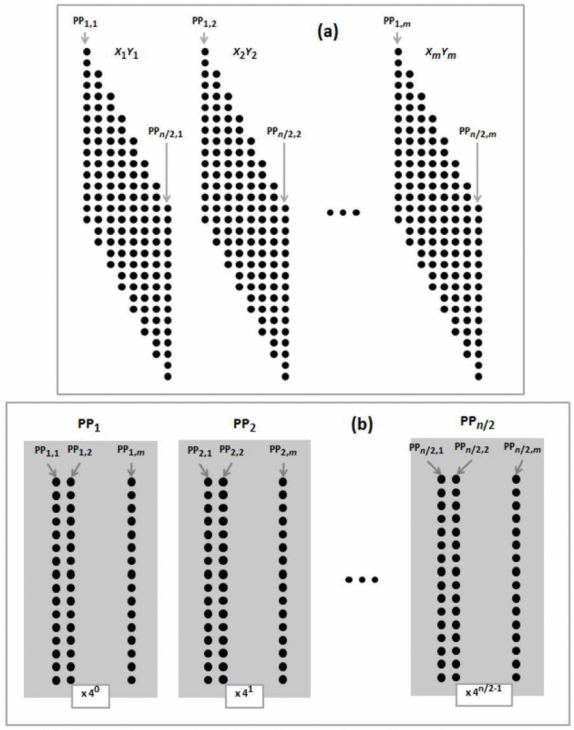


Fig. 5. (a) All the PPs involved in a dot-product of m n-bit operands, and (b) reorganization of dot-product summands and PPs.

Though our FDPM uses a Wallace-tree, a Dadda-tree can be used instead. The work in [9] concluded that the latter has about 10% lower delay and uses 5% less area. These advantages stem from the parallelogram shape of the partial product array. This is less relevant for our implementation which reorders the partial products in rectangles rather than parallelograms.

Fig. 5(a) illustrates the PPs of the m summands involved in (1), where the ith dot parallelogram corresponds to the product $x_i y_i$. These are reorganized in groups $\mathbf{PP}_i = \left\{\mathrm{PP}_{i,j}\right\}_{j=1}^{m}$, $1 \le i \le n/2$, as shown in Fig. 5(b). By using 4:2 compressors, an $(\log_2 m - 1)$-level Wallace-tree will reduce the $\mathbf{PP}_i$ of Fig. 5(b) to two $(n+\log_2 m)$-bit sum and carry FPPs of weight 4^{i-1}, as shown in Fig. 6. The weight of the sum and carry FPPs are shown on their left side, and FPP_i, $1 \le i \le n/2$ are aligned accordingly.

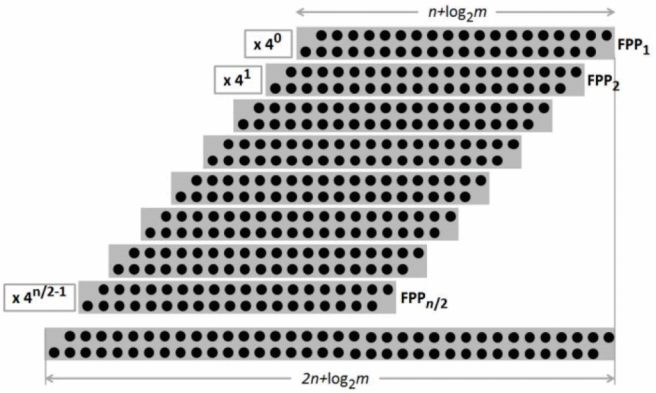


Fig. 6. Collecting the sum and carry FPPs of the **PP** groups of Fig. 7(b).

By using 4:2 compressors, another $(\log_2 n - 2)$-level Wallace-tree can sum the dot parallelogram in Fig. 6, resulting in the $(2n+\log_2 m)$-bit final sum and carry PPs shown at the bottom of Fig. 6. A prefix-tree adder can add them in $\log_2(2n+\log_2 m) \approx \log_2 n$ time complexity. The total time complexity of the FDPM architecture comprising Fig. 5(b) and Fig. 6 has the following components:

1. $\log_2 m$, the time to obtain the FPPs of the $\mathbf{PP}_i$ groups shown in Fig. 7(b) (whose corresponding weights are 4^{i-1}, $1 \le i \le n/2$).
2. $\log_2 n$, the time to sum them and obtain the final $(2n+\log_2 m)$-bit sum and carry PPs in Fig. 6.
3. $\log_2 n$, the time to add them to get the final result of the dot-product.

By adding the above components we obtain a $\log_2 m + \log_2 n$ time complexity. This is consistent with the total of $mn/2$ PPs involved in the dot-product, which would have required a $\log_2(mn/2) \approx \log_2(m) + \log_2 n$ computation time

complexity if a Wallace-tree had been applied directly to the PPs array in Fig. 5(a).

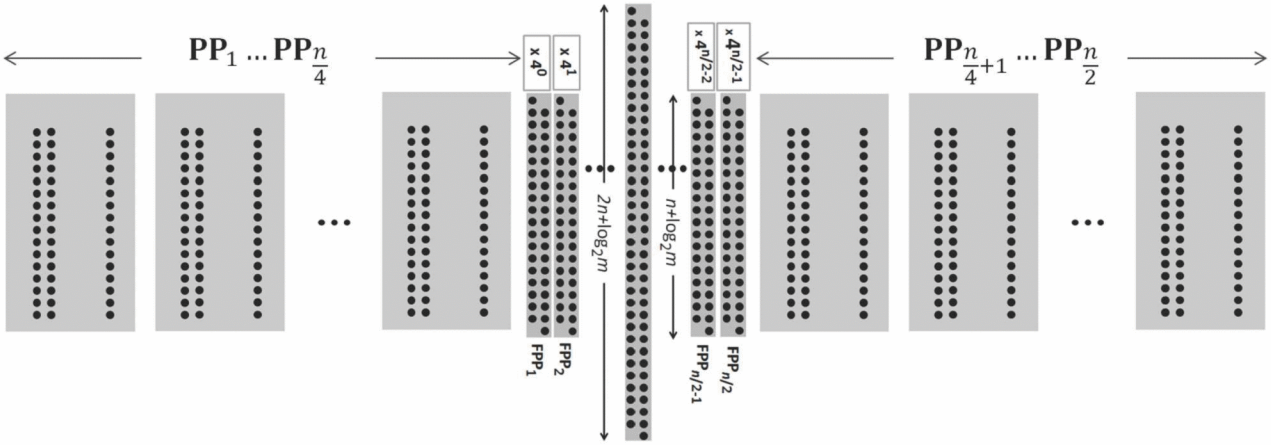


Fig. 7. Floorplan of the proposed FDPM architecture.

Compared to Fig. 5(a), the dot-product time complexity was not changed by the architecture in Fig. 5(b) and Fig. 6. However, there is a major difference in regard to their physical layout. Consider the straightforward Wallace-tree implementation of the dot diagram in Fig. 5(a). Its area (number of dots) is $n^2 m$, but the application of the Wallace-tree to each of the m dot parallelograms faces layout challenges due to an unutilized area. The squeeze of a parallelogram into a rectangle in the layout positions bits of the same weight in successive rows diagonally. Their summation in 4:2 compressors requires layout jumpers, using two metal layers and two vias each, which has a considerable physical layout and performance penalty.

The architecture proposed in Fig. 5(b), and its layout shown in Fig. 7, is more regular, and yields shorter interconnects and higher layout density. Summation of bits of the same weight as in Fig. 5(b) by 4:2 compressors uses a single metal layer bit-to-bit connection, thus avoiding metal jumpers and vias. This is made possible by the identical weights within each of the $\mathbf{PP}_1, \mathbf{PP}_2, \ldots, \mathbf{PP}_{n/2}$ groups in Fig. 5(b). Their corresponding Wallace-tree layout can be implemented in a rectangular area without the area deficiencies of Fig. 5(a). The proper weighting by 4^{i-1} of the various $\mathbf{PP}_i$, $1 \le i \le n/2$, takes place by positioning their FPPs in a single parallelogram as illustrated in Fig. 6. Though these are summed by another Wallace-tree that suffers from the routing deficiency problem of jumpers given the parallelogram squeeze, the number of dots involved is of the n^2 order compared to the $n^2 m$ order in Fig. 5(a). The fusion of the partial products is reminiscent of the bit-heap concept which has been used in [10] for FPGA arithmetic core generation.

Fig. 7 shows the floorplan of the proposed FDPM architecture. Half of the sum and carry FPPs are produced on the left side and the other half on the right side. Their accumulation occurs at the center. The interconnection wires must cross over $n/4$ Wallace-trees expanding m PPs each, where the size of the corresponding tree is also of order m. That brings the order of the interconnect length to mn, which requires the insertion of repeaters to avoid an excessive propagation delay.

III. Per-Bit Power and Energy Analysis

Speeding up computation by the FDPM comes at a cost. The large amount of hardware involved consumes power, which obviously increases with an increase in m and n. The really interesting measure is therefore how the *per-bit* power and energy depend on m and n. To see this, let p_{logic} be the power consumption charged to a logic element (bit) represented by a single dot in the dot diagrams. The power consumed by each of the Wallace-trees generating $\mathbf{PP}_1, \mathbf{PP}_2, \ldots, \mathbf{PP}_{n/2}$ in Fig. 5(b) is obtained by summing the dots over all its Wallace-trees levels, yielding

$$\sum_{i=1}^{\log_2 m - 1} \left(2^{-i+1} nm\right) p_{\text{logic}} \approx 2nm \cdot p_{\text{logic}} . \tag{8}$$

The factor $2^{-i+1} nm$ in (8) counts the number of dots (bits) involved at level i of a Wallace-tree, $1 \le i \le \log_2 m - 1$. Since there are $n/2$ such trees, the total power $P_{\text{logic}}^{\text{S1}}$ consumed by the Wallace-trees at the first stage is

$$P_{\text{logic}}^{\text{S1}} = n^2 m \cdot p_{\text{logic}} . \tag{9}$$

Similarly, the power consumption $P_{\text{logic}}^{\text{S2}}$ of the single Wallace-tree at the second stage located at the center of Fig. 7, summing $\mathbf{PP}_1, \mathbf{PP}_2, \ldots, \mathbf{PP}_{n/2}$ is

$$P_{\text{logic}}^{\text{S2}} = \sum_{i=1}^{\log_2 n - 2} \left[2^{-i+1} n\left(n + \log_2 m\right)\right] p_{\text{logic}} \approx 2n\left(n + \log_2 m\right) p_{\text{logic}} . \tag{10}$$

The power consumption $P_{\text{logic}}^{\text{S3}}$ of the prefix-tree adder at the third stage, summing the final $\left(2n + \log_2 m\right)$-bit sum and carry results, is

$$P_{\text{logic}}^{\text{S3}} = \sum_{i=1}^{\log_2\left(2n+\log_2 m\right)} 2^{-i+1} \left(2n + \log_2 m\right) p_{\text{logic}} \approx 2\left(2n + \log_2 m\right) p_{\text{logic}} . \tag{11}$$

$P_{\text{logic}}^{\text{S2}} / P_{\text{logic}}^{\text{S1}} \to 0$ and $P_{\text{logic}}^{\text{S3}} / P_{\text{logic}}^{\text{S1}} \to 0$ for sufficiently large m and n. For the purpose of the trend analysis we can ignore smaller order terms and conclude that

$$P_{\text{logic}} \approx n^2 m \cdot p_{\text{logic}} . \tag{12}$$

To get the per-bit power $P_{\text{logic}}^{\text{bit}}$, (12) is divided by $m \times n$ problem's size, yielding

$$P_{\text{logic}}^{\text{bit}} \approx n \cdot p_{\text{logic}} , \tag{13}$$

which is independent of m.

The logic only contributes partially to the power consumption. In today's VLSI technologies, interconnection (wires) consume comparable and even more power than the logic, so wires must be accounted for. Local interconnections within the trees can be charged to the logic power p_{logic}; hence we focus on the global interconnection. The floorplan in Fig. 7 shows that a wire connecting the sum and carry FPPs of each **PP** group must cross over other **PP** groups to accumulate in the second-level Wallace-tree positioned at the center. This yields a length of mn complexity, stemming from the $n/4$ Wallace-trees at the first level, whose width is of order m.

To avoid an excessive propagation delay, repeaters are inserted along these wires. One common practice is to insert repeaters at fixed distances, that are dictated solely by the process technology parameters. Denoting by p_{wire} the power consumed by a fixed-length interconnection driven by a repeater, the total power consumed by the global interconnections is

$$P_{\text{wire}} = mn\left(n + \log_2 n\right) \cdot p_{\text{wire}} \approx n^2 m \cdot p_{\text{wire}} . \tag{14}$$

The expression in (14) involves wires of length mn expanding the height $n + \log_2 m$ of a **PP** group in Fig. 7.

Similarly to the logic power calculation, to get the per-bit power $P_{\text{wire}}^{\text{bit}}$, (14) is divided by $m \times n$, yielding

$$P_{\text{wire}}^{\text{bit}} = n \cdot p_{\text{wire}} , \tag{15}$$

which is independent of m. It follows from (13) and (15) that the FDPM per-bit theoretical power consumption is independent of the vector length m and grows linearly with the word width n. This agrees with the experimental results presented in Section IV.

Fig. 8 shows the power portions consumed by the various stages of the FDPM architecture as measured for hardware implementation in a 65-nanometer technology. Unlike the custom 28-nanometer design studied in Section IV, this one was fully synthesized with a vendor RTL compiler, which may explain some minor deviations in the power portions from those nominally expected. The trend, however, is very clear. The portion of the first-stage Wallace-trees grows with increase in vector-length from $m = 4$ to $m = 64$, validating (12) and (14).

IV. Experimental Results

We denote by (n, m) an n-bit word and an m-entry vector FDPM. A $(16,16)$ architecture with the floorplan shown in Fig. 7 was designed such that all its building blocks, from 4:2 compressors, through Booth encoders and Wallace-trees stages, were custom built using a Verilog HDL code. The final adder to sum the final $2n + \log_2 m$ sum and carry words at the center of Fig. 7 was obtained using Cadence's design-ware, since there is no point in designing fast adders that are available commercially. We used a 28-nanometer TSMC technology low-leakage cell library. An implementation of the entire architecture with Cadence design-ware served as a reference for comparison. The Verilog HDL code described the dot-product operation in high-level, letting the Cadence tool pick the best library modules to deliver the target clock speed with minimum power.

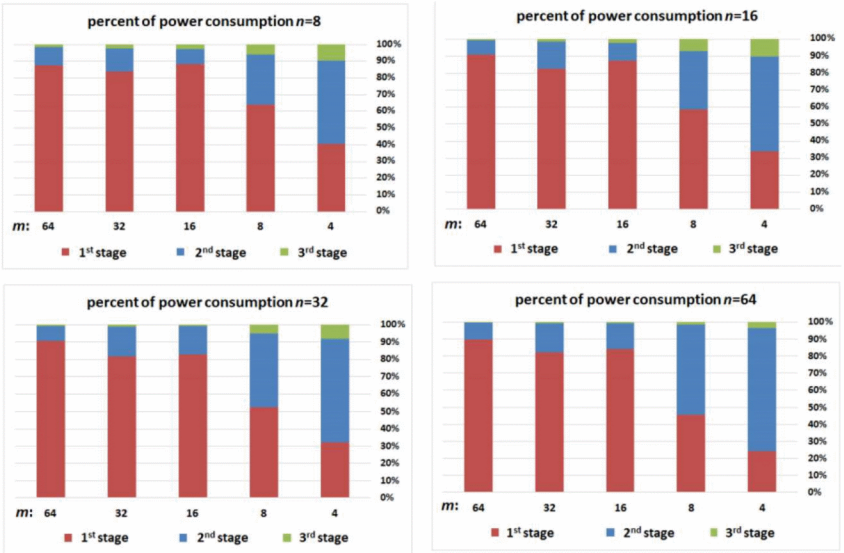


Fig. 8. Distribution of the power consumed by the FDPM architecture logic stages for various (m, n).

Three clock frequencies were tested: 200MHz, 500MHz and the maximum frequency that the proposed FDPM architecture in Fig. 7 could deliver, which was found to be 1.08GHz. Fig. 9(a) shows the power consumption comparison. The power estimation was done by simulating with the SpyGlass tool an extensive dot-product test bench comprising a wide variety of DSP applications made available by the CEVA Corporation. The most impressive improvement was observed at 1.08GHz, where our custom FDPM design consumes 26.03 mWatt, compared to a 41.83 mWatt for the Cadence design-ware implementation, which is 38% power reduction. Fig. 9(d) compares the energy per operation in pJoule. It shows that the energy per operation hardly changes between 200MHz and 500MHz, but jumps significantly for 1.08GHz. By definition the FDPM energy improvement over the reference design is identical to that of the power improvement.

The area and cell count comparison for the two designs is also of interest. Fig. 9(b) shows the area comparison. For a clock speed of 1.08GHz it shows a 30% improvement, slightly smaller than the power improvement, but still considerable. The cell count comparison is shown in Fig. 9(c), indicating that for a 1.08GHz frequency the design-ware required 28% more cells than our custom design.

Due to its large dimensions, the 1.08GHz design could not be sped up as is, and internal pipelining is a natural solution. To this end, its critical delay paths are analyzed below. Table 1 shows how the delay is accumulated from inputs to outputs. Such information is critical for obtaining a balanced timing allocation to the pipeline stages. The table shows the five stages involved in the combinational FDPM: input registers, the radix-4 Booth encoding, the first stage Wallace-trees producing $\mathbf{PP}_1,\ldots,\mathbf{PP}_8$ (see Fig. 7), the second stage Wallace-tree to sum them, and finally, a high-performance prefix-tree adder.

If the clock speed needs to double to 2.0GHz, the table shows that the pipeline registers should be inserted between the first and second PP generation stages, so that the accumulated delay would be evenly divided. While the throughput would be doubled, there would still be some power and area penalty due to the insertion of the pipeline registers.

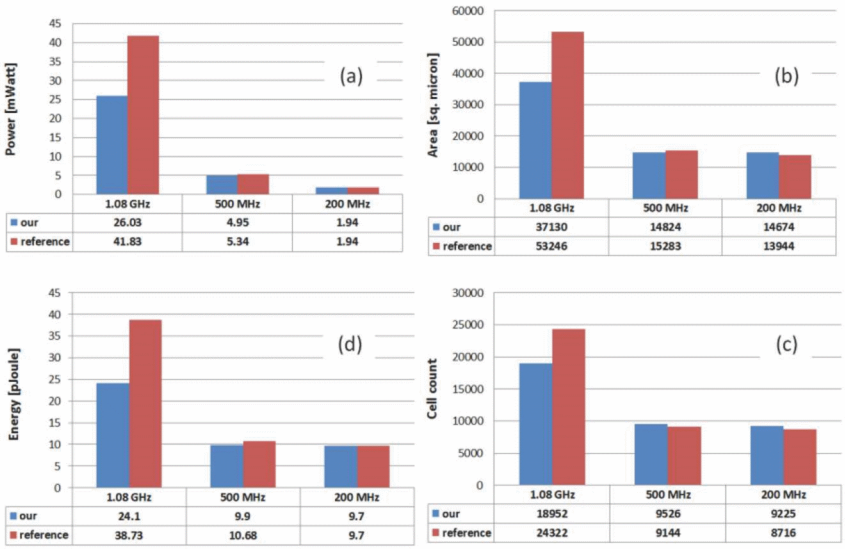
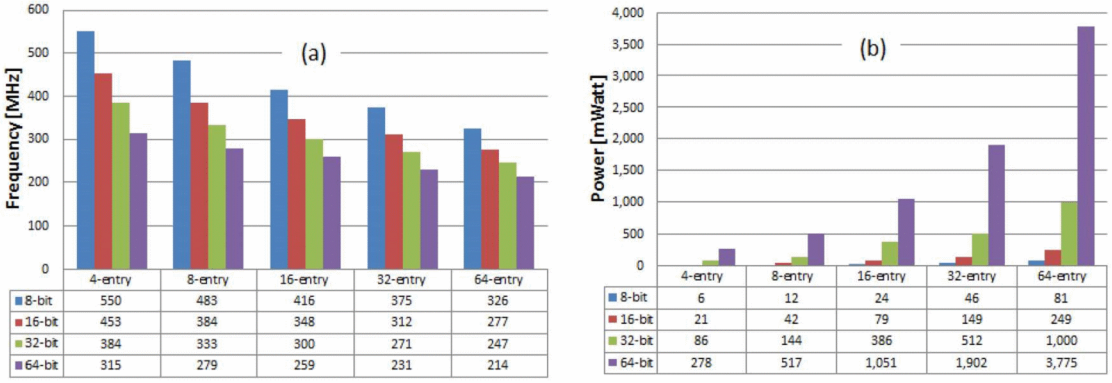


Fig. 9. Comparison of the 28nm FDPM custom and design-ware implementations, power (a), area (b), cell count (c) and energy (d).

TABLE I
TIMING ANALYSIS OF THE FDPM CRITICAL DELAY PATHS

Stage	Module	Delay (nSec)	stage delay (nSec)	Cumulative (nSec)
Clock → Q (input registers)		0.06	0.06	0.06
Radix-4 Booth encoding	input → single	0.05	0.12	0.11
	encoding	0.07		0.18
1st PP stage - 1st tree-level	4:2 compressor (in → fast carry)	0.04	0.09	0.22
	4:2 compressor (C_{in} → C_{out})	0.05		0.27
1st PP stage - 2nd tree-level	4:2 compressor (in → sum)	0.09	0.09	0.36
1st PP stage - 3rd tree-level	4:2 compressor (in → fast carry)	0.05	0.1	0.41
	4:2 compressor (C_{in} → sum)	0.05		0.46
2nd PP stage - 1st tree-level	4:2 compressor (in→ C_{out})	0.06	0.06	0.52
2nd PP stage - 2nd tree-level	4:2 compressor (in → fast carry)	0.04	0.1	0.56
	4:2 compressor (C_{in} → sum)	0.06		0.62
2nd PP stage - 3rd tree-level	4:2 compressor (in → fast carry)	0.03	0.08	0.65
	4:2 compressor (C_{in} → sum)	0.05		0.7
2nd PP stage - 4th tree-level	4:2 compressor (in → sum)	0.08	0.08	0.78
Final prefix tree adder		0.15	0.15	0.93



	4-entry	8-entry	16-entry	32-entry	64-entry
8-bit	550	483	416	375	326
16-bit	453	384	348	312	277
32-bit	384	333	300	271	247
64-bit	315	279	259	231	214

	4-entry	8-entry	16-entry	32-entry	64-entry
8-bit	6	12	24	46	81
16-bit	21	42	79	149	249
32-bit	86	144	386	512	1,000
64-bit	278	517	1,051	1,902	3,775

Fig. 10. Maximum frequency and the corresponding power of 65nm FDPM implementations.

To further explore the (n,m) design space for word sizes of $n=8,16,32,64$ and vector length $m=4,8,16,32,64$, 65-nanometer TSMC technology FDPM implementations were physically compiled and simulated. Fig. 10(a) shows the maximum frequency that could be achieved for each (n,m) combination. Note the linear decrease in clock speed with the doubling of the number of bits n and the doubling of the vector length m. This is not surprising, since as shown in Table I, the first and second PPs stages constitute the lion's share of the overall delay and these are implemented using Wallace-trees of depth $\log_2 n$ and $\log_2 m$, respectively.

The power is expected to grow faster than linearly with increased frequency, as indicated in Fig. 9 for the 28nm design. For the 65nm implementation we measured the power consumption using the Cadence design tool. The results are shown in Fig. 10(b). The figure shows that there is a nearly quadratic growth in power w.r.t. the bit count n and a nearly linear growth w.r.t. the vector length m, in line with (12) and (14).

We next compared the performance of our 28-nanometer 16×16 FDPM implementation with that of the 45-nanometer implementation in [5]. Since our dot-product is for integers, whereas [5] is for floating points, we only considered the mantissa logic. The different word widths, vector lengths and implementation technologies were also accounted for by normalizing the reported power and energy accordingly.

For fairness, the comparisons were per-bit power and energy consumption. The following normalizations were done. Firstly, our design involves 16-bit integers, whereas the reference had 32-bit floating-point operands with a 24-bit mantissa (IEEE Std-754). Secondly, we targeted 16-entry vectors, whereas the reference targeted $A\times B+C\times D$ operation; i.e., only 2-entry vectors. Thirdly, as shown in Fig. 3(c), the floating point dot-product, besides the multiplication and addition operations, also included alignment, normalization and exponent addition. Therefore for the reference design in [5] we only counted the circuits related to mantissa multiplication and addition, which consumed nearly 70% of the entire circuit. To eliminate the effect of different process technologies, the power of the reference design was decreased by a factor of $28/45=0.62$, which is roughly the scaling factor of the per-device capacitance [11] (Ch. 7.4). The per-bit power P_{bit} was calculated as

$$P_{\text{bit}}=\frac{\text{total power}\times\text{realtive area}}{\text{vector length}\times\text{word width}}\times\text{scaling ratio}. \qquad (16)$$

The total power reported in [5] was 7.2 mWatt. The calculated per-bit power is therefore $\left[(7.2\times0.7)/(24\times2)\right]\times0.62=65.3\mu\text{Watt}$. The reported latency was 2,721 pSec, corresponding to a 367 MHz clock frequency. As shown in Fig. 9, our 16×16 design consumed, for frequencies of 200, 500 and 1080 MHz, 1.94, 4.94 and 26.03 mWatt, respectively. Summarized in Table II, the respective per-bit power consumption of our design was 7.57, 19.3 and 101 μWatt, thus showing a considerable power improvement over [5].

TABLE II
POWER AND ENERGY COMPARISON OF FDPM IMPLEMENTATION

		Clock frequency [MHz]			
		200	367	500	1080
P_{bit} [μWatt]	our	7.57	---	19.3	101
	Ref [5]	---	65.3	---	---
E_{bit} [normalized]	our	0.212	---	0.217	0.525
	Ref [5]	---	1.0	---	---

TABLE III
PER-BIT POWER CONSUMPTION IN $\mu Watt$ (A), AND ITS DEPENDENCE ON VECTOR LENGTH m AND WORD-WIDTH n (B)

(a)

	$m=4$	$m=8$	$m=32$	$m=64$
$n=8$	26.9	32.0	38.4	39.4
$n=16$	56.5	62.0	69.6	71.1
$n=32$	118	124	133	134
$n=64$	233	239	260	265

(b)

	$m=4$		$m=8$		$m=32$		$m=64$	
$n=8$	1.0	1.0	1.19	1.0	1.20	1.0	1.02	1.0
$n=16$	1.0	2.10	1.09	1.93	1.12	1.81	1.02	1.80
$n=32$	1.0	2.08	1.05	2.00	1.07	1.91	1.01	1.88
$n=64$	1.0	1.97	1.03	1.93	1.09	1.95	1.02	1.98

To compensate for the different operation speeds, it is also important to compare the per-bit energy E_{bit} obtained by

$$E_{\text{bit}}=P_{\text{bit}}\times\frac{1}{\text{clock frequency}}. \qquad (17)$$

The energy comparison was normalized to per-bit energy of the reference design. Our implementation delivers normalized

per-bit energies of 0.212, 0.217 and 0.525 for 200, 500 and 1080 MHz, respectively. These represent respective improvements of $1.0/0.212 = 4.7\text{X}$, $1.0/0.217 = 4.6\text{X}$ and $1.0/0.525 = 1.9\text{X}$.

Our analysis of the per-bit power consumption, based on (13) and (15) shows that the FDPM is independent of the vector length m and grows linearly with the word width n. To validate this conclusion, Table III(a) presents the per-bit power consumption in µWatt of a 65-nanometer FDPM, designed for a 100MHz clock speed. Table III(b) shows the per-bit power growth of successive entries as a function of m in red and as a function of n in blue. The analysis showed that these values should theoretically be 1.0 and 2.0, respectively, corresponding to the independence of m and the doubling with n. The table shows that the real design behavior agrees nicely with the theoretical analysis.

V. Conclusion

We proposed a new circuit architecture, dubbed FDPM, for a hardware implementation of a dot-product by fusing all the computational steps. This architecture exhibits considerable improvements in power and energy efficiency compared to previous proposals and an EDA vendor design-ware. A wide range of bit lengths and vector lengths was implemented and analyzed. For a 28-nanometer technology we observed a limit of $1.08\,\text{GHz}$ for a $(16,16)$ FDPM but a straightforward pipelining could double the operating frequency. Future work could consider deriving the optimal pipeline depth to achieve acceleration with minimum per-bit power and energy.

Acknowledgment

This work was supported in part by the MAGNET Program of the Israel Ministry of Industry, HiPer consortium, and in part by the Israel Science Foundation under Grant Number 1678/13. The authors are thankful to CEVA corporation for providing simulation data, to Elad Margalit of CEVA for implementing the 28nm design, and to Ishai Alouche for implementing the 65nm design. The authors wish also to acknowledge the helpful comments and suggestions made by the anonymous reviewers.

References

[1] I. Koren, Computer arithmetic algorithms, A.K. Peters Press, 2002.

[2] B. Parhami, Computer arithmetic: algorithms and hardware designs, Oxford University Press, Inc., 2009.

[3] F. Elguibaly, "A fast parallel multiplier-accumulator using the modified Booth algorithm," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* vol. 47, no. 9, pp. 902-908, 2000.

[4] Y.-H. Seo and D.-W. Kim, "A new VLSI architecture of parallel multiplier–accumulator based on Radix-2 modified Booth algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 18, no. 2, pp. 201-208, 2010.

[5] H. H. Saleh and E. E. Swartzlander Jr, "A floating-point fused dot-product unit," in *IEEE International Conference on Computer Design*, 2008.

[6] F. Merchant, A. Maity, M. Mahadurkar, K. Vatwani, I. Munje, M. Krishna, S. Nalesh, N. Gopalan, S. Raha, S. K. Nandy and R. Narayan, "Micro-architectural Enhancements in Distributed Memory CGRAs for LU and QR Factorizations," in Proc. of *VLSI Design (VLSID), 2015 28th International Conference on*, 2015.

[7] N. Itoh, Y. Naemura, H. Makino, Y. Nakase, T. Yoshihara and Y. Horiba, "A 600-MHz 54× 54-bit multiplier with rectangular-styled Wallace tree," *IEEE Journal of Solid-State Circuits,* vol. 36, no. 2 , pp. 249-257, 2001.

[8] F. Hameed, L. Bauer and J. Henkel, "Simultaneously optimizing DRAM cache hit latency and miss rate via novel set mapping policies," in *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2013.

[9] W. J. Townsend, E. E. Swartzlander and J. A. Abraham, "A comparison of Dadda and Wallace multiplier delays," in Proc. of *Optical Science and Technology, SPIE's 48th Annual Meeting*, 2003.

[10] N. Brunie, F. De Dinechin, M. Istoan, G. Sergent, K. Illyes and B. Popa, "Arithmetic core generation using bit heaps," in Proc. of *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, 2013.

[11] N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed., Addion-Wesley, 2010.

[12] CEVA, 2014. [Online]. Available: http://www.ceva-dsp.com/CEVA-XC4000/.