IMPLEMENTATION OF HIGH PERFORMACE MULTIPLIERS BASED ON APPROXIMATE COMPRESSOR DESIGN

JIEMING MA*, KA LOK MAN*, Tomas KRILAVIČIUS**, SHENG-UEI GUAN*, Taikyeong JEONG***
* Xi'an Jiaotong-Liverpool University (XJTLU), China; ** Vytautas Magnus university and Baltic Institute of Advanced Technologies; *** Myongji University, Korea

***Abstract*:** Estimating arithmetic is a design paradigm for DSP hardware. By allowing structurally incomplete arithmetic circuits to occasionally perform imprecise calculations, higher performance can be achieved in many different electronic systems.
This paper presents a potential useful approach to implement tree multipliers by using estimating arithmetic. Experimental results show the applicability and effectiveness of the proposed approach.

***Keywords*:** computer arithmetic, estimating arithmetic, compressors, multipliers

1. Introduction

Real-world signals can be mathematically represented as analog signals. Since the basic radio communication and transceivers were invented at the beginning of the 20th century, a set of problems in analog communication systems has arisen. The primary disadvantage of Analog Signal Processing (ASP) is that any communication system has random unwanted variation. Armed with high reliability, accuracy, flexibility, and increased immunity-to-noise, Digital Signal Processing (DSP) has become one of the most attractive topics in semiconductor industry in the past 30 years. According to [1], the global market share of DSP processors and microcontrollers is more than 95% of the total volume of processors sold.
Multipliers are usually deemed as a critical component in digital signal processor design since a large number of multiplications are required in DSP applications [2]. It is not hard to understand that accurate multiplication operation is always complicated and might lead to huge delay. Contemporary high-performance DSP core usually takes more than one cycle to perform a fixed-point multiply-accumulate instruction. Also, it takes even more cycles to perform a simple division or floating-point instruction. However, so far we know, signals in multimedia system processing are usually tolerant of occasional errors. Estimating methods provide possible opportunities to achieve higher performance by using simplified approximate circuits.
Previous work has shown that computing performance of electronic systems can be improved by using estimating methods. Occasional errors in the intermediate calculations do not seriously affect the final computation results [3,4,7]. In this paper, we present an effective approach to implement tree multipliers by using estimating arithmetic.
The paper is organized as follows. The next section discusses related work. In Section 3, an overview of the general estimation methods is presented. Section 4 presents common approximate compressor design and its general performance. Section 5 presents our proposed multiplier design based on approximate compressors along with experimental results. Finally, concluding remarks are given in Section 6 and direction of future work is given.

2. Related work

Lu's adder [3] was the first prototype of estimating arithmetic hardware that could be used in a variety of ways. Compared to an exact adder, the 32×32 bit parallel approximate adder with 4-bit carry chain is faster at the cost of slight accuracy reduction. The simulation results show that close to 90% of the addition is correct when the input data are from real applications.
Several (4:2) counters: "saturating" and "reflecting" are introduced in Low Density Parity Check (LDPC) decoder design [4]. The decoders using saturating and reflecting counters in the check nodes can achieve even better decoding performance than the exact ones.
However, all the above-mentioned approximate circuits have not been applied to implement complex arithmetic hardware, e.g. tree multipliers, to achieve higher performance in computation.

3. Delay estimation methods

3.1. Logical effort delay model

Logical effort, which normalizes delay relative to the characteristic of unit inverters, is an easy-to-use method for CMOS circuit delay estimations. The major concepts to be considered while using logical effort are:

Parasitic delay (p): When a gate drives zero load, the delay of the gate is called parasitic delay. It is the ratio of diffusion capacitance to gate capacitance of a unit inverter.

Logical effort (g): It is defined as the ratio of the input capacitance of the gate to the input capacitance of an inverter that can drive the same output current.

Electrical effort (h): It is the quotient of the output capacitance divided by the input capacitance.

Branching effort (b): When a logic gate drives some inputs of next logic gates, some of the drive current is available along the path and some is off along the path. The branching can be expressed by Equation (1).

$$b = (C_{\text{on-path}} + C_{\text{off-path}}) / C_{\text{on-path}} \quad (1)$$

where $C_{\text{on-path}}$ is the capacitance of connections that leads the path "on" and $C_{\text{off-path}}$ is the capacitance of connections that leads the path "off". Equation (2) models the delay (**d**) through a single logic gate:

$$d = g \cdot h + p \quad (2)$$

In a multistage logic network (paths in a network are indexed by i, j, k …), the total path effort is:

$$F = G \cdot B \cdot H = \prod g_i \cdot b_i \cdot h_i \quad (3)$$

where G denotes the total path logical effort,
B denotes the total path branching effort,
H denotes the total path electrical effort.
Once the total path effort is determined, the ideal number of CMOS circuit stages (**N**) can be calculated with Equation (4).

$$N = round(\log_{\rho} F) \quad (4)$$

ρ is usually called the best stage effort. It is a constant that shows the number of CMOS stages required in a least-delay path. The value of ρ is based on the characteristic of unit inverter. In this paper, ρ is set at the nominal value of 4. The stage effort (**a**) can be calculated with Equation (5) and the total delay (**D**) is defined by Equation (6).

$$\alpha = F^{1/N} \quad (5)$$

$$D = N \cdot \alpha + \sum p \quad (6)$$

3.2. Simplified logical effort delay calculation

Before introducing the simplified way of delay calcutation, we first explain a new terminology known as *effort load*. The branching effort at one gate on the critical path can be rewritten using Equation (7) where $load_i$ is the effort load which represents the capacitance on node i and g_i denotes the logical effort of the subsequent node.

$$b_i = load_i / g_{i+1} \quad (7)$$

In an adder or multiplier design, the path electrical effort (**H**) is usually set at 1 since the module is connected to a copy of itself [5]. In this case, $b_N = load_0 / g_1$. Then Equation (3) can be rewritten as Equation (8).

$$\begin{aligned} F &= G \cdot B \cdot 1 = \prod g_i \cdot b_i \\ &= g_1 \cdot b_N \cdot \prod load_j \\ &= g_1 \cdot load_0 \cdot \prod load_j / g_1 \\ &= \prod load_k \end{aligned}$$

where $1 \le i \le N, 1 \le j \le N-1, 0 \le k \le N-1$ (8)

The delay of interconnection is roughly proportional to the length of wires. For a wire across M x-grid, the wire delay (D_{wire}) is usually approximated to M/20 of the unit inverter delay (D_{inv}) as shown in Equation (9).

$$D_{wire} = (M \cdot D_{inv}) / 20 \quad (9)$$

4. Compressor design

4.1. Approximate compressors

Approximate compressors using typical delay estimation methods as presented in Section 3 are the fundamental units in complex arithmetic hardware. Accurate modules are the circuits that can perform the full arithmetic operation while approximate modules are the ones performing rough calculations. In this paper, according to the accuracy, the approximate compressors can be divided into three categories:

High-accuracy compressors (HAC): 90% < accuracy ≤ 100%.

Medium-accuracy compressors (MAC): 70%< accuracy ≤ 90%.

Low-accuracy compressors (LAC): accuracy ≤ 70%.

Normally, a (n:m) compressor has n input bits and m output bits. In the conventional compressor design, the number of input and output bits is related as below:

$$m \ge \log_2(n+1), \text{ where } m, n \ge 0 \quad (10)$$

In order to obtain accurate results, the output bit should be equal to or greater than $\log_2(n+1)$. However, it is worth mentioning that increasing output bits will decrease the efficiency of multiplication since more calculation stages will be required in the partial product reductions. The output bit of an ideal accurate multiplier can be expressed as follows:

$$\begin{aligned} &\text{if } \log_2(n+1) \text{ is integer} \\ &\quad m = \log_2(n+1) \\ &\text{else } m = \text{round}(\log_2(n+1))+1 \end{aligned} \tag{11}$$

For (4:2) compressors using approximate modules, the following holds:

$$m < \log_2(n+1) \tag{12}$$

4.2. Design and evaluation of HAC1

A (4:2) compressor has four inputs and thus it has 2^4 different input values. When the total number of tests is equal to the total number of input values, there is only one case that the accuracy lies in the high-accuracy level, i.e. all inputs are set to logic 1.

For illustration purpose in this paper, below shows one possible High-Accuracy Compressor (HAC), which gives the sum of 1+1+1+1 as 2 (decimal). Table 1 shows its Karnaugh Map. In order to distinguish it from other HAC types, we call it HAC1 in this paper.

Table 1. Karnaugh Map for a (4:2) HAC1

CD \ AB	00	01	11	10
00	0	1	2	1
01	1	2	3	2
11	2	3	**2**	3
10	1	2	3	2

The function of the (4:2) HAC1 can be expressed as Equation (13):

$$SUM = \neg(\neg(A \oplus B \oplus C \oplus D)) \tag{13}$$

$$CARRY = \neg((\neg((A+B)\cdot(C+D)))\cdot(\neg(A\cdot B+C\cdot D)))$$

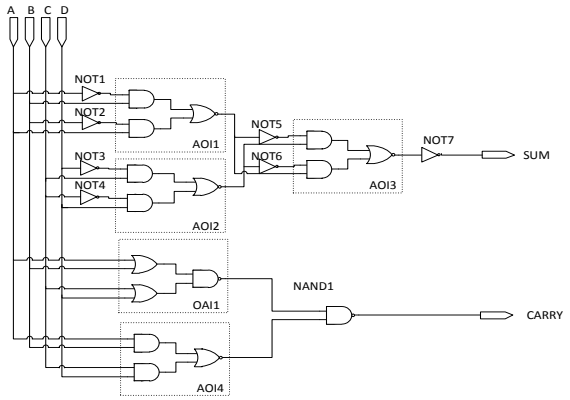

Fig. 1. Schematic view of a (4:2) HAC1

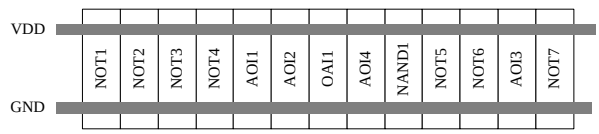

Fig. 2. Layout placement of the (4:2) HAC1 in Fig.1

The schematic, layout placement and Verilog code of the HAC1 are shown in Fig. 1, Fig. 2, and Fig. 3 respectively. Note that the approximate compressor in Fig. 1 has two XOR-delay stages. It is also not hard to see in Fig. 1 that the carryout number is less which leads to acceleration of the multiplication speed as fewer bits are connected to the subsequent columns of the compressor.

```
module com42_1_1 (sum,cout,a,b,c,d);
output sum,cout;
input a,b,c,d;
not (w1,a);
not (w2,b);
not (w3,d);
not (w4,c);
AOI22 A1(w5,w1,b,w2,a);
AOI22 A2(w6,w3,c,w4,d);
not (w7,w5);
not (w8,w6);
AOI22 A3(w9,w7,w6,w8,w5);
not (sum,w9);
OAI22 O1(w10,a,b,c,d);
AOI22 A4(w11,a,b,c,d);
nand (cout,w11,w10);
endmodule
```

```
module AOI22(out,a,b,c,d);
output out;
input a,b,c,d;
and (w1,a,b);
and (w2,c,d);
nor (out,w1,w2);
endmodule

module OAI22(out,a,b,c,d);
output out;
input a,b,c,d;
or (w1,a,b);
or (w2,c,d);
nand (out,w1,w2);
endmodule
```

Fig. 3. The Verilog HDL code for the (4:2) HAC1 in Fig.1

Table 2. Logical effort calculation for the (4:2) HAC1 in Fig.1

Cell	Fanout load	Effort load	Parasitic delay
NOT_1	AOI22+track	6/3+6/20=2.3	3/3=1
NOT_2	AOI22+track	6/3+4/20=2.2	3/3=1
NOT_3	AOI22+track	6/3+6/20=2.3	3/3=1
NOT_4	AOI22+track	6/3+4/20=2.2	3/3=1
AOI_1	AOI22+NOT+ track	(6+3)/3+19/20= 3.95	12/3=4
AOI_2	AOI22+NOT6+ track	(6+3)/3+15/20= 3.75	12/3=4
OAI_1	NAND2 +track	4/3+4/20=1.53	12/3=4
AOI_4	NAND2 +track	4/3+0/20=1.33	12/3=4
NAND_1 ※	AOI22+NOT+OA I22+AOI22+track	(6+3+6+6)/3+30/ 20=8.5	6/3=2
NOT_5	AOI22+track	6/3+2/20=2.1	3/3=1
NOT_6	AOI22+track	6/3+0/20=2	3/3=1
AOI_3	NOT +track	3/3+0/20=1	12/3=4
NOT_7 †	AOI22+NOT+OA I22+AOI22+track	(6+3+6+6)/3+20/ 20=8	3/3=1

※: assume output CARRY is connected to input B of the next HAC

† : assume output SUM is connected to input A of the next HAC

Logical effort method can be applied straightforwardly to all (4:2) HACs by incorporating the lateral fanouts directly into the branching efforts. Table 2 presents the logical effort calculation for the (4:2) HAC1 presented in Fig. 1. We assume the output SUM (see Fig. 1 for details) is always connected to input A as shown in Fig. 1, then the path effort of the critical path is:

$$F = \prod load_k = 2.3\times3.95\times2.1\times1\times8 = 152.63$$

The ideal logic stage is 3. However, the minimum logic state required in the circuit is 5 after bubble pushing (a

sort of removal of dead nodes). So the logic stage is still 5.

$$N = log_4 152.63 = 3.6 \approx 3$$

Then the stage effort is:

$$\alpha = 152.63^{1/5} = 2.7$$

The delay can be calculated by the following equation:

$$D = N \cdot \alpha + \sum p = 5 \times 2.7 + 1 + 4 + 1 + 4 + 1$$
$$= 24.7 \text{ delay units}$$
$$\approx 4.93 \text{ FO4(Fanout-of-4) inverter delays}$$

Area can be estimated by running experiments. Each of the module cells used has the same height of 10 y-grids, which is equal to 2.8 um. The width of each cell is specified with a fixed value (see Table 3 for details) and therefore it is possible to calculate the module width (W) through the summation over the basic cells' width along the layout placement.

There are 13 basic cells used in HAC1, including 7 NOTs, 1 NAND2, 4 AOI22s and 1 OAI22. The x-grid is poly-poly repeat and each grid is around 420 nm. Thus the total length of the module is:

$$W_{total} = 7 \cdot W_{NOT} + 1 \cdot W_{NAND2} + 4 \cdot W_{AOI22} + 1 \cdot W_{OAI22}$$
$$= 7 \times 2 + 1 \times 3 + 4 \times 5 + 1 \times 5 \text{ grids}$$
$$= 42 \text{ x-grids}$$
$$= 42 \times 420 \text{ nm}$$
$$= 17.64 \text{ um}$$

Then, the total area of HAC1 is:

$$AREA_{HAC1} = 17.64 \times 2.8 \text{ um}^2 = 49.4 \text{ um}^2$$

Table 3. Cell library (all values are for single-stick cells)

Cell	Equation	Width
NOT	z = !a	2
NAND2	z = !(a&b)	3
NOR2	z = !(a|b)	3
AOI22	z = !(a&b|c&d)	5
OAI22	z = !((a|b)&(c|d))	5
XNOR2	z = !(a⊕ b)	7

4.3. Other compressors

Similar to HAC1, HAC module/compressor can also be generally designed to produce the output result 0 (decimal) when all the inputs receive logic 1. A (4:2) HAC can efficiently reduce the bit load of each column since it cuts off one carry output from the module.

Unlike HACs, different (4:2) MACs and (4:2) LACs presented in this paper provide a possible way of speeding up the SUM logic, which can shorten SUM circuits from 2 XOR-delay stages to an XOR-NAND/NOR-delay stage since XOR – NAND/NOR is not a complete function circuit. All these mimic the SUM function as $A \oplus B \oplus C \oplus D$ along with some slight/occasional errors.

Table 4 presents the set of approximate results for HACs, MACs and LACs. As for MAC1, when the four inputs are set at "0000", the result will be decimal 1 rather than 0 as expected. Similarly, errors arise when inputs are "0011","1100" and "1111".

Table 4. A set of approximate results for (4:2) approximate compressors used in this paper

Module	Input (binary)				Approximate Result (decimal)	Accurate Result (decimal)
	A	B	C	D		
HAC1	1	1	1	1	2	4
HAC2	1	1	1	1	0	4
MAC1	0	0	0	0	1	0
	0	0	1	1	3	2
	1	1	0	0	3	2
	1	1	1	1	3	4
MAC2	0	0	0	0	1	0
	0	0	1	1	1	2
	1	1	0	0	1	2
	1	1	1	1	3	4
LAC1	0	0	0	1	0	1
	0	0	1	0	0	1
	1	1	0	1	2	2
	1	1	1	1	2	4
	1	1	1	0	2	3
LAC2	0	1	0	1	3	2
	0	1	1	0	3	2
	1	1	1	1	2	4
	1	0	0	1	3	2
	1	0	1	0	3	2
LAC3	0	1	0	0	0	1
	0	1	1	1	2	3
	1	1	1	1	2	4
	1	0	0	0	0	1
	1	0	1	1	2	3

4.4. Performance comparison

Table 5. The performance of various compressor families

*The width of transistors is 100 nanometers

Module	Error rate %	SUM delay (ps)	CARRY delay (ps)	MOS FET quantity	Area (um^2)
FA	0	224	98	42	29.4
HAC1	6.3	246.7	132	58	49.4
HAC2	6.3	255	250	70	60
MAC1	25	163.8	130	48	40
MAC2	25	160	134	40	40
LAC1	31	179	131	56	40
LAC2	31	179	131	56	40
LAC3	31	163.8	130	48	40

Table 5 shows the performance of all compressors presented in this paper. Though the approximate (4:2) compressors have occasional errors, their speed is faster or roughly equal to that of conventional full adders (FAs). Among the (4:2) compressors, MAC2 achieves the highest speed, the least CMOS quantity, and the smallest size since it has a simpler CARRY logic.

5 Design and analysis of approximate multipliers

We have already seen in Section 4, approximate compressors are high performance arithmetic circuits. It is not hard to see that compressors can be used to build multipliers. We apply tree topology to design approximate compressor based multiplier (namely (4:2) HAC1 based multiplier). Tree topology is used in the multiplier design since trees are very fast structures for summing up partial products [6, 7]. Details of the implementation of our approximate compressor based multiplier design may be subject to patent protection and such details are not given in this paper.
Nevertheless, Table 6 summarizes the general performance of our proposed approximate multiplier based on (4:2) HAC1 against the conventional Wallace tree through constrained random tests. Our proposed (4:2) HAC1 based approximate multiplier can be 30% faster while the accuracy decreases only by 12%.

Table 6. Performance comparison of tree multipliers

Tree multiplier	Speed ns	MOSFET quantity	Error rate %	Area um^2
Wallace multiplier	1792	39690	0	55566
(4:2) HAC1 based multiplier	1233.5	28212	12	49787

6. Conclusions

Approximate circuits provide an approach to accelerate computation by using simplified circuits. This paper has first reviewed a series of common estimation methods used by approximate circuits. Along with experimental results, the applicability and effectiveness of using approximate compressors for the implementation of multipliers has been shown. Our future work will focus on applying approximate compressors to implement large and reliable high-performance multipliers.

References

1. Liu, D., Embedded DSP Processor Design, Morgan Kaufmann Publishing, 2008. 808 p.
2. Dandapat, A., Ghosal, S., Sarkar, P., and Mukhopadhyay, D., A 1.2-ns16×16-Bit Binary Multiplier Using High Speed Compressors, International Journal of Electrical, Computer and Systems Engineering, 4(3), 2010, pp. 234-239.
3. Lu, Shih-Lien. Cover feature–Speeding Up Processing with Approximation Circuits, International Journal of Computers, 37(3), 2004, pp. 67–73.
4. Phillips, B.J., Kelly, D.R. and Ng, B.W., Estimating Adders for a Low Density Parity Check Decoder, in SPIE Proceedings of Advanced Signal Processing Algorithms, Architectures and Implementations XVI, San Diego, CA, USA, 2006.
5. Burgess, N., New Models of Prefix Adder Topologies, Journal of VLSI Signal Processing, 40(1), 2005. Pp. 125-141.
6. Flynn, M.J. and Oberman, S.S., Advanced Computer Arithmetic Design, Wiley, 2001. 344 p.
7. Kelly, D.R., Phillips, B.J. and Al-Sarawi, S., Approximate Signed Binary Integer Multipliers for Arithmetic Data Value Speculation, in Proceedings of Electronic Chips and Systems Design Initiative 2009 Conference, Sophia Antipolis, France, 2009. 8 p.