Power Comparison of Low Bitwidth Multipliers

Ralf Hildebrandt
Fraunhofer IPMS, Dresden, Germany
Ralf-Hildebrandt@gmx.de

Abstract

Some common and new low bitwidth signed-digit, carry-save and carry-ribble multipliers are compared at point of view of power dissipation at gate level. Results of synthesis and power simulation with an analog simulator will be presented[1].

1. Introduction

Hardware multipliers accelerate the computation of many mathematical problems. The gain of speed over software multiplication can be used for higher throughput or longer power-down periods. Beside the need for speed and low area low power dissipation is desired.

In this paper some common and new parallel multipliers will be compared. The goal is minimizing the power dissipation. All comparisons are made at the algorithmic level, that is related to the real circuit at gate level. Not discussed will be place & route. Synthesis results and power simulations with an analog simulator (Spectre) will be presented. As an example a $16 \cdot 16 \Rightarrow 32$ bit Booth multiplier for signed and unsigned multiplication is taken. Booth encoding reduces the number of partial products and therefore area and power dissipation. [1]

The origin of dynamic power dissipation is charging and discharging of capacities in gates (switching of signals). Arrival times of signals can be balanced through structural modifications resulting in lesser dynamic power dissipation.

The most common multipliers are Carry-Save-Array and Wallace-Tree [2]. Irregularities of tree-structures are weakened in the Dadda-Tree [3], Overturned-Stairs-Tree [4] and Generalized Carry-Save-Adder [5]. A speedup of array-structures can be obtained with the Leapfrog-Array [6]. The Regular-Reduction-Tree [7] uses ribble adder.

Multipliers consist of 3 parts: Partial Product Generation PPG (with Booth encoding), Partial Product Reduction PPR and Carry Propagate Adder CPA.

2. Signed-Digit Multipliers

With Signed-Digit arithmetics a carry-free addition is possible. Carry-free means carry propagation only to the next digit position. In radix-2 SD-form a digit can have the values $(\bar{1}, 0, 1)$ with $\bar{1} = -1$.

[1]This work has been presented at the ICM2004 conference in Tunis. ©2005 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

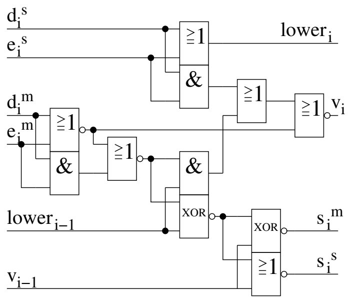

Fig. 1. SD-Adder (Takagi)

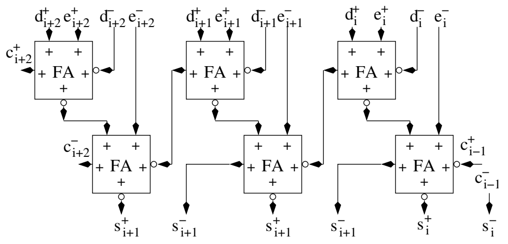
d^+_{i+2} e^+_{i+2} d^-_{i+2} e^-_{i+2}
d^+_{i+1} e^+_{i+1} d^-_{i+1} e^-_{i+1}
d^+_i e^+_i d^-_i e^-_i
c^+_{i+2}
c^-_{i+2}
c^+_{i-1}
c^-_{i-1}
FA
s^+_{i+1} s^-_{i+1} s^+_{i+1} s^-_{i+1} s^+_i s^-_i

Fig. 2. Borrow-Save-Adder [11], [12]

The m bit addition in SD-form can be computed in two steps. In the first step digits d_i and e_i $(i \in [0, m-1])$ are added to the intermediate sum p_i, that will be split into two parts u_i and c_i. These two parts will be added to sum s_i in the second step.

$$\begin{aligned} d_i + e_i &= p_i \\ p_i &= u_i + 2c_i \\ u_i + c_{i-1} &= s_i \end{aligned} \qquad (1)$$

The ranges of the variables from (1) are:

$$\begin{aligned} [\bar{1},1] + [\bar{1},1] &= [\bar{2},2] \\ [\bar{2},2] &= [\bar{1},1] + [\bar{1},1] \\ [\bar{1},1] + [\bar{1},1] &= [\bar{1},1] \end{aligned} \qquad (2)$$

Line 3 in (2) must be satisfied when splitting p_i.

A SD-adder working with this basic principle is very big, slow and power-consuming. The Takagi-Adder (TA) [9] jumps over p_i and is known as the smallest SD-adder [10] (see fig. 1). Another possibility is the BSA (see fig. 2), that is slightly bigger than the Takagi-Adder.

16 bit Booth-multiplication (signed and unsigned) results in 9 partial products per digit position, that can be reduced with a PPR with SD-arithmetics like shown in fig. 3. Reconversion from SD-form to 2th complement can be done with a classic subtractor and is similar to a conventional CPA.

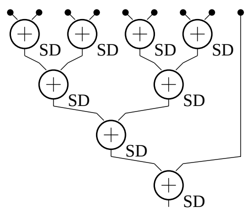

Fig. 3. Tree structure of a SD-PPR

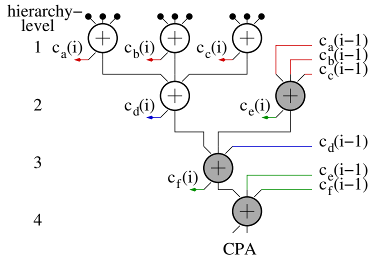

Fig. 4. Carry-Propagate-Tree CPT

With a SD-multiplier at each step 2 operands are reduced to one intermediate result. The necessary number of steps n_{SD} ($n_{SD} \in$ Z) for a multiplier like shown in fig. 3 is

$$1 \cdot 2^{n_{SD}} \geq k \tag{3}$$

Thereby k is the number of addends. Delay estimations of the TA and BSA result for both adders a delay of slightly more than 3 XOR-delays. Therefore the calculation time results in

$$t_{SD}/t_{XOR} = \lceil 3 \cdot \log_2 \lfloor k \rfloor \rceil \tag{4}$$

Thereby $y = \lceil x \rceil$ is the upper next integer number ($y \in$ Z).

For the addition of a digit slice in the PPR $k-1$ addition units are necessary (array- or tree-structure). The TA occupies slightly less gate area than 2 fulladders. BSA is slightly bigger.

3. Carry-Save multipliers

The best-known CS-multiplier with tree-structure is the Wallace-Tree [2]. A variant of this multiplier is shown in fig. 4. Sums and carrys have separated data paths in this realization, as long as possible. The structure is equivalent to a Dadda-Tree [3] at this number of addends. The final result is calculated by a CPA.

When reducing k addends, $p \geq \frac{k}{3}$ new sums and $c \geq \frac{k-1}{3}$ carrys are produced. So the next hierarchy level in the CPT has $s \geq \frac{k}{3} + \frac{k-1}{3}$ new addends. With simplification (5) the number of necessary reduction steps n_{CPT} satisfies (6).

$$\frac{2}{3}k \geq \frac{k}{3} + \frac{k-1}{3} \tag{5}$$

$$2 \cdot \left(\frac{3}{2}\right)^{n_{CPT}} \geq k \tag{6}$$

Factor 2 in (6) is necessary, because 2 final results (carry and sum) are left after all reduction steps. Each reduction step is carried out by a fulladder, that has a delay of 2 XOR-delays. Therefore the calculation time for the CPT is

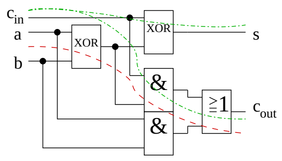

Fig. 5. Delay paths of fulladders

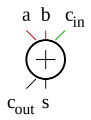

Fig. 6. Fulladder

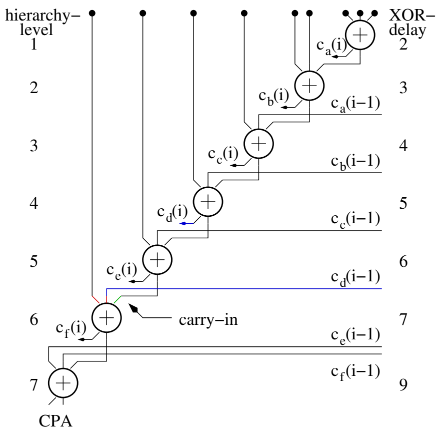

Fig. 7. Modified Leapfrog-CSA

$$t_{CPT}/t_{XOR} = \left\lceil 2\left(\log_{3/2}\lfloor k \rfloor - \log_{3/2}2\right)\right\rceil \tag{7}$$

The best-known array-structure (CSA) is more regular than the Wallace-Tree, but is significant slower for higher multiplier word widths. A speedup of the CSA can be achieved by using the asymmetric delays in standard fulladders: From the Carry-In to all outputs of a fulladder only one XOR-delay exists, but from the addend-inputs to the outputs there are 2 XOR-delays. (see fig. 5) In [6] the Leapfrog-Principle is derived from this. [1] describes some variants. In fig. 7 a new modification with delay estimation is shown. Not the sums, but the carrys jump in this modification. The idea behind is, that the sum-output of a fulladder is often slightly slower than the carry-out. Furthermore, when modeling a complete PPR, the original Leapfrog-PPR is in need of lesser optimizations on the left side than the new PPR, but after these modifications the arrival times of many signals are significant better balanced in the new PPR resulting in higher speed and lesser ribble.

$$\begin{aligned} t_{CSA}/t_{XOR} &= 2\,(\lfloor k \rfloor - 2) \qquad (k \geq 2) \\ t_{LFCSA}/t_{XOR} &= \lfloor k \rfloor \qquad\qquad (k \geq 4) \end{aligned} \tag{8}$$

CSA and LFCSA calculation times are shown in (8). Even in a Leapfrog-based PPR speed is a linear function of the number of addends. On the other hand a tree-structure is more irregular and harder to route. If one combines the two structures a new structure is derived (see fig. 8). For ($k \geq 5$) the calculation time of this new structure matches the following equation:

$$t_{CSTLFA}/t_{XOR} = 2 + 2 \cdot \lfloor k/3 \rfloor + (k \,\mathrm{MOD}\, 3) \tag{9}$$

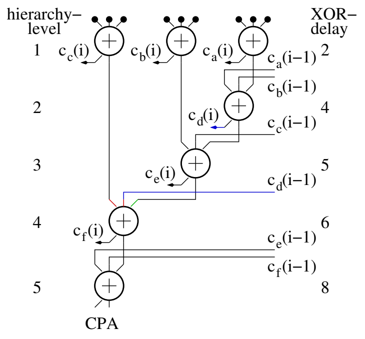

Fig. 8. CS-Tree-Leapfrog-Array (CSTLFA)

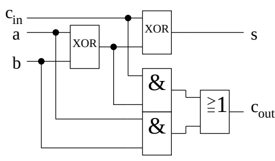

Fig. 9. Fulladder

All Carry-Save-Structures need (at least) $k-2$ fulladders for the addition of one bit slice in the PPR.

4. Carry-Ribble-Adder

Standard cells based on a CMOS technology are inverting. To realize e.g. an AND, one has to take a NAND and an inverter. If the desired logic function can be calculated with inverting cells, the inverters can be saved what may reduce area and delay.

There are some consequences for a Carry-Ribble-Adder, like partially described in [6]. At mathematical-logical view a fulladder is organized like shown in fig. 9. If one modifies the carry-path a little bit, one gets an adder with inverting carry-out (fig. 10). The gain by this modification is still small. The now inverted carry has to be taken in consideration at the following adder. When calculating the sum s_i one uses $d \oplus \overline{e} = \overline{d \oplus e}$. The inverted carry would invert the sum. Inverting the propagate $\overline{p_i} = \overline{a_i \oplus b_i}$ corrects this and results in a further reduction. Because carry-in c_{i-1} and propagate p_i are inverted, the carry-logic has to be modified (10).

$$\begin{aligned}\overline{c_i} &= \overline{g_i + c_{i-1}p_i} = \overline{g_i} \cdot \overline{(c_{i-1}p_i)} \\ &= \overline{g_i} \cdot (\overline{c_{i-1}} + \overline{p_i}) \\ c_i &= \overline{\overline{g_i} \cdot (\overline{c_{i-1}} + \overline{p_i})} \qquad (10)\end{aligned}$$

Equation (10) describes some further reductions: Generate g_i

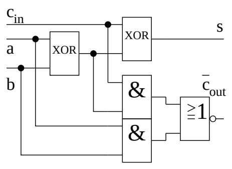

Fig. 10. Fulladder A

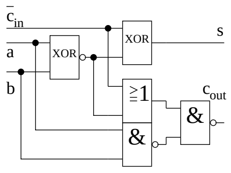

Fig. 11. Fulladder B

Table 1. Reviewed types of multipliers

Shortening	Name
CSA	CarrySaveArray (Synopsys DW)
NBW	WallaceTree (Synopsys DWF)
WALL	Booth/WallaceTree (Synopsys DWF)
BBSA	Booth/BorrowSaveTree ch. 2
BTA	Booth/TakagiAdderTree ch. 2
BCPT	Booth/CarryPropagateTree ch. 3
BCSA	Booth/CarrySaveArray ch. 3
BLFCSA	Booth/LeapfrogCarrySaveArray ch. 3
BLFCSAo	BLFCSA with original Leapfrog-PPR [6]
BCSTLFA	Booth/CarrySaveTreeLeapfrogArray ch. 3
BCRA	Booth/CarryRibbleArray ch. 4

and not-inverted carry-out can be realized with inverting gates. If one constructs a CRA alternating out of modified fulladder A and B, this CRA may be slightly smaller and faster than a traditional CRA - dependent of the used target-library.

If a fulladder can be efficiently realized in a target library by the connection of individual gates, the explicit description of the structure of the ribble-chain in a HDL may result in such a fCRA (fast CRA) automatically during synthesis. A fCRA can be used as a minimal CPA for a multiplier but also for the PPR. If one reorders the array of partial products to a trapezoidal shape one can construct a highly regular but minimal multiplier that has a high delay. Because of the trapezoidal shape only the last row of the array rules the speed and all signal delays are balanced well. The calculating time of this BCRA (Booth / Carry-Ribble-Array) multiplier, that combines PPR and CPA is $t_{BCRA}/t_{XOR} = 2 \cdot n$ if n is the bitwidth of the operands.

5. Comparison of the multipliers

5.1. Algorithmic comparison

SD-multipliers need the gate equivalent of around $2(k-1)$ fulladders, CS-multipliers need around $k-2$ fulladders per digit slice in the PPR, if k is the number of addends. A CS-tree-multiplier (7) is from an algorithmic point of view everytime faster than a SD-tree-multiplier (4).

The LFCSA is much faster than a traditional CSA (8). Especially at multiplication with narrow word width the LFCSA is only slightly slower than the CPT, but has a more regular structure than the CPT. The CSTLFA (9) is a compromise between tree and array.

5.2. Synthesis results

Table 1 shows a summary of the reviewed types of multipliers. The used CPA has a remarkable influence on the overall speed of a multiplier. The structure of a fast CPA is ruled by the delay profile of the PPR. In table 2 reviewed alternatives of the CPA are named. Not all alternatives are realized for all types of multipliers. In principle the consequences are visible well and can be carried over to the other multipliers.

In Table 3 synthesis results are presented. The target-library is a $0.5\,\mu m$ standard cell library. Listed are gate area, gate area in AND2-equivalences and critical path. Area and critical path are normalized on the Synopsys CSA.

Table 2. Variants of the CPA

Variant	Comment
CRA	CPA as CarryRibbleAdder
fCRA	CPA as ”fast CarryRibbleAdder” ch. 4
GEF	CPA as CLA by GEF [a] [13]
fCPA	CPA free for synthesis (a+b), fast

[a]Carry Lookahead Adder build by Generalized Earliest First (algorithm, that outputs an optimal fast adder dependent on a given delay profile)

Table 3. Gate area and critical path

multiplier	CPA	$\frac{A}{A_{AND2}}$	$\frac{A}{A_{\mathrm{CSA}}}$	$\frac{t_D}{t_{D_{\mathrm{CSA}}}}$
CSA	-	1297	1	1
NBW	-	1332	1,03	0,43
WALL	-	1256	0,97	0,46
BTA	fCPA	1697	1,31	0,58
BBSA	fCPA	1840	1,42	0,56
BCPT	fCPA	1295	1,00	0,39
BCPT	GEF	1235	0,95	0,43
BCPT	fCRA	1150	0,89	0,71
BCPT	CRA	1147	0,88	0,89
BCSA	fCPA	1262	0,97	0,53
BLFCSA	fCPA	1278	0,99	0,44
BLFCSA	GEF	1230	0,95	0,46
BLFCSA	fCRA	1133	0,87	0,72
BLFCSAo	fCPA	1273	0,98	0,43
BCSTLFA	fCPA	1306	1,01	0,42
BCSTLFA	fCRA	1170	0,90	0,75
BCRA	-	1130	0,87	0,84

5.3. Power simulation

All reviewed multipliers are benchmarked with an analog simulator (Spectre). Dynamic power consumption of digital circuits depends on the computed data. Because of this 3 benchmarks are made: 200 multiplications of a discrete wavelet transform (DWT_{200}; unsigned multiplication with big numbers), 100 multiplications (signed and unsigned) with pseudo-random data (PR_{100}) and 100 multiplications with 8 bit pseudo-random data sign-extended to 16 bits ($PR8_{100}$). In table 4 the dissipated energy, normalized on the Synopsys CSA is given.

5.4. Conclusion

The SD-multipliers are relatively big, slow and very power-consuming like expected. From the point of view of the synthesized netlist SD-multipliers are not suitable as a replacement for CS-multipliers. But routing of SD-trees is simpler than routing of a Wallace-tree because of the 2to1-reduction [14].

CS-tree structures have the highest speed because of short propagation paths. This results in smaller power dissipation compared to conventional array-structures. It was shown in [15] that this advantage is not lost after routing when using small word widths and standard cell libraries .

The LFCSAo is significant faster than a conventional CSA. The new modified LFCSA dissipates less power than the LFCSAo because of the better signal delay balancing in the PPR. The CSTLFA may be a good compromise between tree and array for multipliers with wider word width. At 16 bit Booth multiplication CPT and LFCSA have too close results, to offer a serious alternative with the CSTLFA.

Table 4. Energy $\frac{W}{W_{\mathrm{CSA}}}$

multiplier	CPA	DWT_{200}	PR_{100}	$PR8_{100}$
CSA	-	1	1	1
NBW	-	0,81	0,83	0,89
WALL	-	0,84	1,06	0,73
BTA	fCPA	1,01	1,37	1,27
BBSA	fCPA	1,17	1,58	1,29
BCPT	fCPA	0,73	1,07	0,74
BCPT	GEF	0,69	0,95	0,82
BCPT	fCRA	0,67	0,77	0,73
BCPT	CRA	0,66	0,97	0,71
BCSA	fCPA	0,87	1,04	0,89
BLFCSA	fCPA	0,66	0,91	0,60
BLFCSA	GEF	0,63	0,90	0,65
BLFCSA	fCRA	0,61	0,97	0,69
BLFCSAo	fCPA	0,76	1,11	0,88
BCSTLFA	fCPA	0,73	1,30	0,79
BCSTLFA	fCRA	0,61	1,06	0,66
BCRA	-	0,70	1,36	0,89

At benchmark PR_{100} non-Booth structures have higher efficiency than Booth structures. The Booth-encoder dissipates a significant amount of energy of multipliers with narrow word widths when computing pseudo-random data. Partial products will be eliminated seldom and signals switch often.

The CPA has a remarkable influence on the final speed. If the lower speed of the introduced fCPA is sufficient, a multiplier with this fCRA as CPA may dissipate less energy than one with a fast CPA because of the realization with very low area occupation.

References

[1] Zhijun Huang: *High-Level Optimization Techniques for Low-Power Multiplier Design*; http://arith.cs.ucla.edu/dissertations/dissertation_huang03.pdf

[2] C. S. Wallace: *A Suggestion for a fast multiplier*, IEEE Transactions on Electronic Computers, vol. 13, pp. 14-17, Feb 1964

[3] L. Dadda: *Some Schemes for parallel Adders*, Acta Frequenza, vol. 34, no. 5, pp. 349-356, May 1965

[4] Magnus Karlsson, Oscar Gustafsson, Jacob Wikner, Thomas Johansson, Weidong Li, Magnus Hörlin, Hans Ekberg: *Understanding Multiplier Design using ”Overturned-Stairs” Adder Trees*; http://www.es.isy.liu.se/publications/papers_and_reports/1998/CAA-report.pdf

[5] Magnus Karlsson: *A Generalized Carry-Save Adder Array for Digital Signal Processing*; http://www.es.isy.liu.se/norsig2000/publ/page287_id016.pdf

[6] S.S. Mahant-Shetti, P.T. Balsara, C. Lemonds: *High performance low power array multiplier using temporal tiling*; IEEE Transactions on very large scale integration (VLSI) systems, vol. 7, no. 1, p. 121-124, 03/1999

[7] Henrik Eriksson, Per Larsson-Edefors, William P. Marnane: *A regular parallel multiplier which utilizes multiplie carry-propagate adders* http://eee.ucc.ie/staff/marnanel/Files/papers/Eriksson_ISCAS2001.pdf

[8] A. Avizienis: *Signed digit number representation for fast parallel arithmetic*; IRE Trans. Electron. Comput., vol EC-10, pp. 389-400; 1961

[9] S. Kuntinobu, H. Edamasu, N. Takagi: *Design of high speed mos multiplier and divider using redundant binary representation*; 8th Symposium on Computer Arithmetic, New York, 1987

[10] A. Wassatsch, D. Timmermann: *Scalable Counter Architecture for a Pre-loadable 1GHz@0.6μm/5V Pre-scalar in TSPC*; http://www-md.e-technik.uni-rostock.de/veroeff/wassatsch_speeddiv_iscas2001.pdf

[11] Asger Munk Nielsen: *Number systems and Digit Serial Arithmetic*; http://citeseer.nj.nec.com/nielsen97number.html

[12] Ercegovac, Lang: *Digital Arithmetic: Fast Two-Operand Addition*; 2003 http://www.cs.ucla.edu/~milos/cs252a.htm

[13] Wen-Chang Yeh: *Arithmetic Module Design and its Application to FFT*; http://twins.ee.nctu.edu.tw/~chsung/dissertation_ts2.pdf

[14] Dhananjay S. Phatak, Steffen Kahle, Hansoo Kim, Jason Lue: *Hybrid Signed Digit Representation for Low Power Arithmetic Circuits*; http://www.csee.umbc.edu/~phatak/publications/hsd-lowpower.pdf

[15] Pascal C.H. Meier, Rob A. Rutenbar, L. Richard Carley: *Exploring Multiplier Architecture and Layout for Low Power*; http://www.ece.cmu.edu/~lowpower/cicc96.pdf