10

Low-Complexity and High-Speed Constant Multiplications for Digital Filters Using Carry-Save Arithmetic

Oscar Gustafsson and Lars Wanhammar
Linköping University
Sweden

1. Introduction

In many digital filter implementations the filter coefficients are known beforehand. Based on this fact, the problem of constant multiplications, replacing general multipliers with shifts and adders[1], has been an active research topic for a few decades. Much work has been done on finding algorithms and filter coefficients where the filter coefficients can be represented using few signed-power-of-two (SPT) terms (Lim, 1990; Yli-Kaakinen & Saramäki, 2007). Furthermore, there has been work on realizing constant multipliers using few adders (Dempster & Macleod, 1994; Gustafsson et al., 2006; Thong & Nicolici, 2009). Additionally, mainly motivated by transposed direct form FIR filters, as shown in Fig. 1, several algorithms have been proposed for utilizing redundancies when a single data is multiplied with several constant coefficients, known as multiple constant multiplication (Aksoy et al., 2010; Dempster & Macleod, 1995; Gustafsson, 2007; Hartley, 1996; Potkonjak et al., 1996; Voronenko & Püschel, 2007).
Most of this previous work has considered carry-propagation adders (CPAs), i.e., adders with two inputs and one output, as shown in Fig. 2. Even though there has been many different techniques proposed to accelerate the carry-propagation, these typically lead to an increased area and power consumption. For high-speed implementations, an alternative is to use carry-save adders (CSAs). These adders do not propagate the carry, but instead have two outputs, one for the sum and one for the carry. Furthermore, as no carries are propagated, the adder can use the carry-input as a third input. A carry-save adder is illustrated in Fig. 3.
The mapping between CPAs and CSAs is not consistent (Gustafsson, 2008). Hence, there is a need to solve the CSA constant multiplications using specialized algorithms. The inconsistency is illustrated in Fig. 4, where a multiple constant multiplication for the coefficients 3, 11,

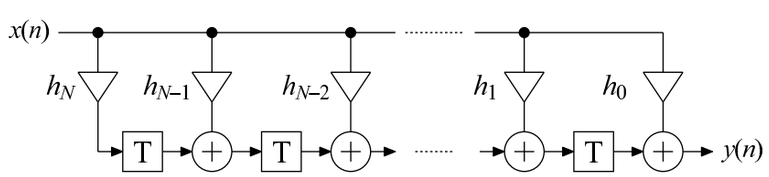

Fig. 1. Transposed direct form FIR filter.

[1] Adders refers to both adders and subtractors.

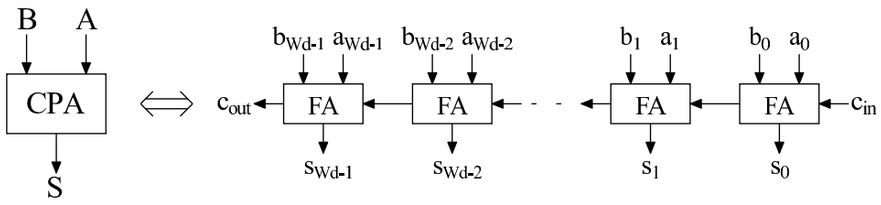

Fig. 2. Carry-propagation adder.

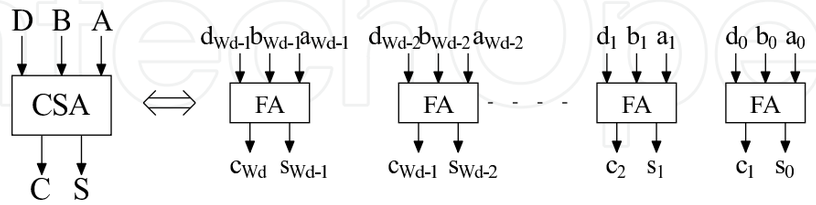

Fig. 3. Carry-save adder.

and 27 is shown. In Fig. 4, $<< n$ denotes a left-shift by n, i.e., a multiplication by 2^n. The CPA solution in Fig. 4(a) is optimal in terms of adders. However, when the three CPAs are mapped to CSAs, as shown in Fig. 4(b) it is clear that a CPA can result in zero, one, or two CSAs. The different cases are summarized in Table 1.

In this chapter we consider the realization of constant multiplications using CSAs. Primarily, we will consider the case where the input is in non-redundant format, typically two's complement, and the output is in carry-save format. In most application one would eventually convert the carry-save format back to non-redundant form using a CPA. However, it should be noted that it is possible to use CSAs throughout the application and that stability can be retained in wave digital filters (Kleine & Noll, 1987). As such we also consider single constant multiplication with carry-save input. In general, it is possible to use algorithms for CPAs as one CPA results in two CSAs when both inputs are in carry-save format, see Table 1. However, the number of cascaded adders does not follow directly, as the CSAs can be arranged in different structures. The work presented in this chapter originates from (Gustafsson et al., 2004; 2001; Gustafsson & Wanhammar, 2007). Related work has later on been presented in (Aksoy & Güneş, 2008; Hosangadi et al., 2006; Jaccottet et al., 2010).

2. Carry-Save Arithmetic

A carry-save adder as that in Fig. 3 can add three two's complement numbers and produce the result as two two's complement numbers, where the sum of the two outputs is the sum of the three inputs. The weights of the carry-bits are one higher than those of the sum-bits. This leads to two things: the least significant carry-bit is always zero and the MSB of the sum and

CPA input 1	CPA input 2	Number of CSAs
multiplier input	multiplier input	0
multiplier input	adder output	1
adder output	adder output	2

Table 1. Possible cases of mapping a CPA to CSA.

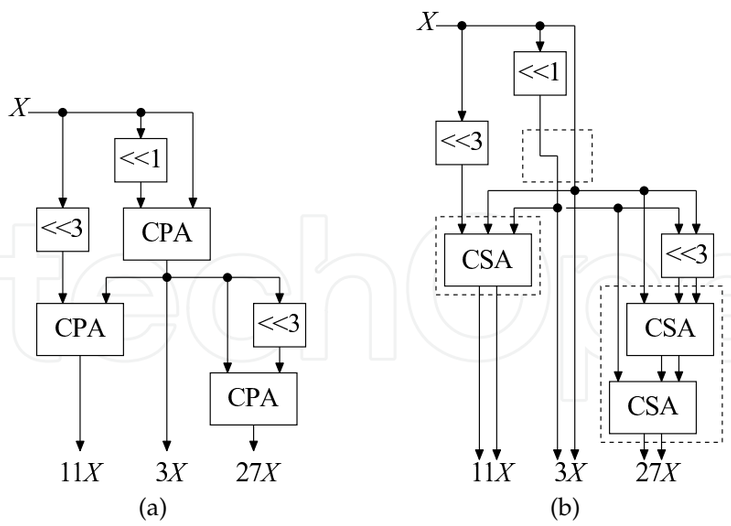

Fig. 4. Multiple constant multiplication for $\{3, 11, 27\}$. (a) Optimal CPA solution. (b) Mapped CSA solution.

carry have different weights. The latter causes problems when adding these in later stages as all two's complement vectors should have the same length for addition to work.

2.1 Subtraction in carry-save arithmetic

To subtract a two's complement number using CPAs, the standard way is to negate the number to be subtracted and add a one to the carry-input of the least significant full adder, indicated by c_{in} in Fig. 2. However, for a carry-save adder there is no such "free" input. Instead one can utilize the least significant carry-bit and set that to one in case of a subtraction. This clearly only works if one of the three inputs should be subtracted. For cases where two inputs should be subtracted it is often possible to change the sign of the output such that the initially positive term is now subtracted. This will be further illustrated in the example in Section 3.2.

2.2 Handling of sign-bits in carry-save arithmetic

Consider the addition of the three numbers 0, 0.5, and -0.5 in two's complement representation with a CSA as shown in Fig. 3. The inputs, $\{A, B, D\}$, and outputs, $\{C, S\}$ are

A	0.0	0.0_{10}
B	0.1	0.5_{10}
D	1.1	-0.5_{10}
C	01.	
S	1.0	

Now, as the result, 0, is within the valid range of the number representation used we can without any problems remove the leading carry bit and obtain the result in a carry save representation as $C = 1.0, S = 1.0$. Adding these gives the expected result $C + S = 0.0 = 0_{10}$, after removing the carry out of the carry propagation addition. However, now shift the results right one position to obtain $C = 1.10, S = 1.10$. If we add these vectors we get $C + S = 1.00 = -1_{10}$.

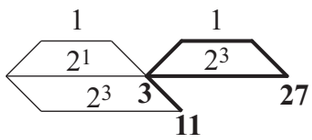

Fig. 5. Graph representation of the shift-and-add network in Fig. 4(b). The graph is directed from left to right.

Obviously, we can not straightforwardly shift the carry and the sum vector after a carry-save addition, despite the fact that they are both in two's complement representation and the shifted result for each vector separately is correct.
Shifting of carry-save data is crucial in the realization of CSA-based constant multiplication, and, hence, a shiftable representation is required.
In (Noll, 1991) this erroneous behavior was named *carry overflow*. A solution for a single CSA was proposed as replacing the most significant carry and sum bits with

$$c'_0 = c_{out} \quad (1)$$

$$s'_0 = s_0 \oplus c_0 \oplus c_{out} \quad (2)$$

where c'_0 and s'_0 are the corrected sign-bits. For the simple example above we would obtain the corrected vectors $C = 0.0, S = 0.0$ which clearly can be shifted arbitrarily and still resulting in a correct sum.
For the general case that we have two vectors C and S and want to truncate them to a given number of bits the sign-bits can be computed as

$$c'_i = c_i \oplus c_{i+1} \oplus s_{i+1} \quad (3)$$

$$s'_i = s_i \oplus c_{i+1} \oplus s_{i+1} \quad (4)$$

Hence, it is possible to add an arbitrary number of words using only one guard bit and obtain a valid two's complement representation with correct sign-bits that can be shifted arbitrarily, given that we know that the final result is within the given range.
An alternative technique is of course to sign-extend the sum-output. However, this would lead to an increasing wordlength compared to the corresponding non-redundant wordlength. Furthermore, that approach can not be used in recursive algorithms.

3. Carry-Save Arithmetic Constant Multipliers with Non-Redundant Input

It is in many cased practical to represent the shift-and-add networks as graphs, where the edges corresponds to shifts and the vertices corresponds to additions. Typically, the sign of the operation is represented on the edges. As an example, the network in Fig. 4(b) has a graph-representation as in Fig. 5, where the thin lines correspond to data in non-redundant format, while the bold lines corresponds to data in carry-save format. Each node has a value, called a *fundamental*, which is the ratio between the output of the adder and the input, i.e., the multiplier coefficient. The fundamentals are indicated with a bold font. The adder graphs are directed. However, for clarity, the arrows are neglected.
It is possible to define graphs corresponding to all possible interconnections of N adders. They have the following properties (Gustafsson & Wanhammar, 2007):

- Each edge can either be in non-redundant or in carry-save representation.

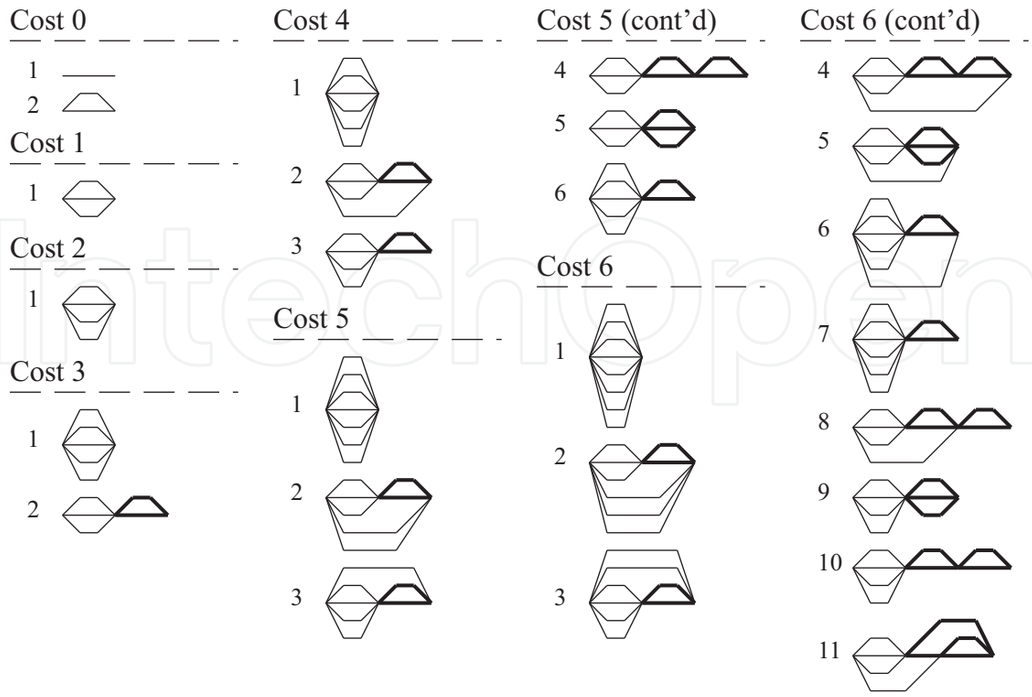

Fig. 6. Possible carry-save adder graphs with non-redundant input generating different coefficient sets for 0 to 6 carry-save adders. The graphs are directed from left to right.

- The cost of a vertex is the number of incoming edges corresponding to non-redundant words plus two times the number of incoming edges corresponding to carry-save words minus two.
- The output edge(s) of a vertex is in carry-save representation, except for the initial vertex.

All possible combinations of edge values for the given graphs can be searched and the minimum-adder solution can easily be found. In Fig. 6 all possible graphs generating different sets of coefficients using up to six carry-save adders is shown. As in Fig. 5 the thin lines represent data in a non-redundant format, while the bold lines represent carry-save format. Note that cost-0 graph 1 has a non-redundant output. The first graphs for each nonzero cost corresponds to the CSD multiplier. Hence, this case is always covered by the proposed approach.
It is worth noting that when four or more carry-save adders are required in a vertex it is possible to re-arrange the adders into, e.g., a Wallace tree (Wallace, 1964). This will reduce the adder depth of the multiplier.

3.1 Results

Exhaustive searches have been performed for multipliers containing up to six adders. This has been done by searching all different combinations of possible shifts and signs for all graphs up to six adders and saving the minimum number of adders in a table. The result is that all integer numbers between 1 and 2^k for wordlength k up to 19 can be obtained using six adders. The maximum number of adders required for a given wordlength is shown in Fig. 7 for both CSD multipliers and the proposed approach.

Number of adders	Graph number	Maximum nonzero digits	Minimum adder depth
0	1	1	0
	2	2	0
1	1	3	1
2	1	4	2
3	1	5	3
	2	6	3
4	1	6	3
	2	7	4
	3	8	4
5	1	7	4
	2	8	4
	3	9	5
	4	12	5
	5	9	4
	6	10	5
6	1	8	4
	2	9	4
	3	10	5
	4	13	6
	5	10	5
	6	11	6
	7	12	5
	8	14	6
	9	12	5
	10	16	6
	11	11	5

Table 2. Maximum number of nonzero digits and minimum adder depth for the CSA multiplier graphs in Fig. 6 with non-redundant input data.

The average number of adders required for a given wordlength is shown in Fig. 8. It is clear that savings only occurs when the coefficient wordlength is larger than nine bits. Figure 9 shows the average savings using the proposed approach. For 19 coefficient bits the savings are just over 10%.

The maximum number of nonzero digits and minimum depth for each graph is shown in Table 2. It can be seen that the graph 1 for each nonzero cost, the CSD multiplier graph, has the smallest adder depth, but also the lowest number of maximum nonzero digits. Furthermore, the graph with the highest number of maximum nonzero digits also is one of the multipliers with the largest depth. Based on the observations in Fig. 6 and Table 2 we can conclude that the maximum number of non-zero digits for $K > 0$ carry-save adders is

$$3 \cdot 2^{\frac{K-1}{2}} \tag{5}$$

for odd K and

$$2^{\frac{K+1}{2}} \tag{6}$$

for even K.

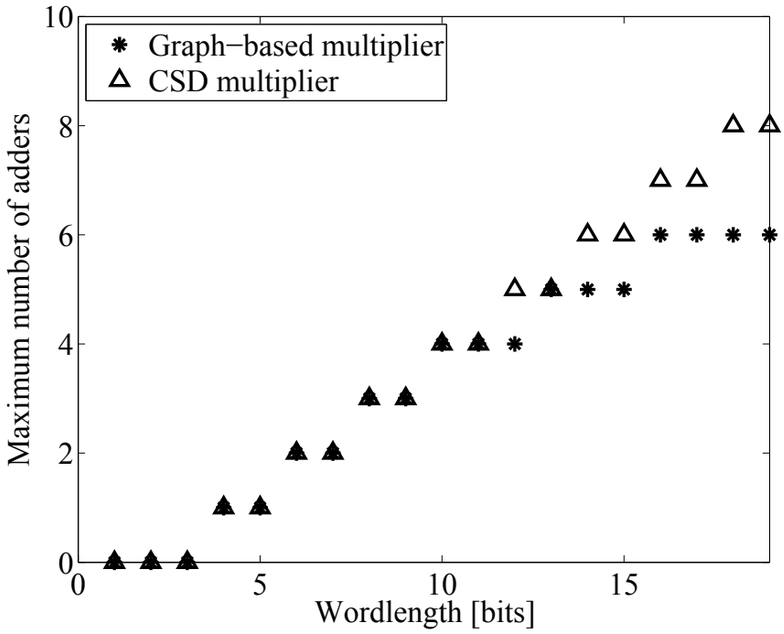

Fig. 7. Maximum number of CSAs as a function of coefficient wordlength for CSD multipliers and proposed optimal multipliers.

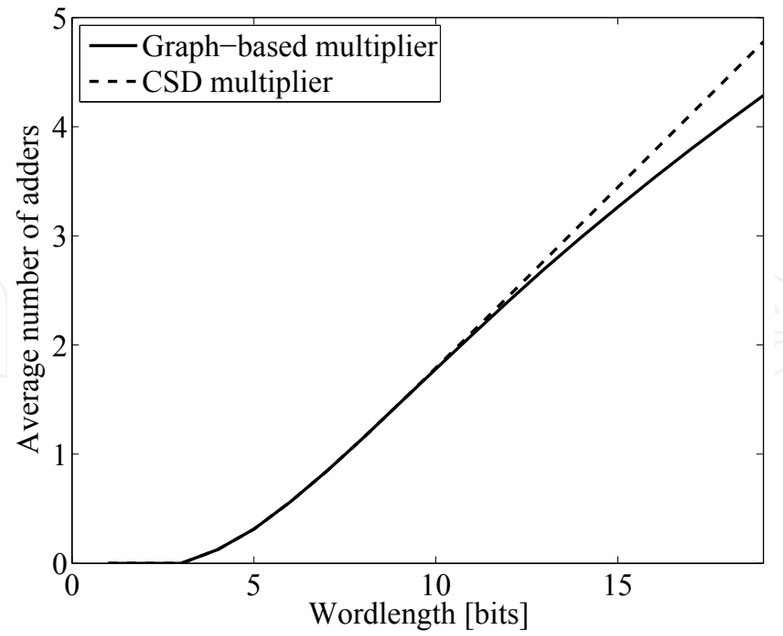

Fig. 8. Average number of CSAs as a function of coefficient wordlength for CSD multipliers and proposed optimal multipliers.

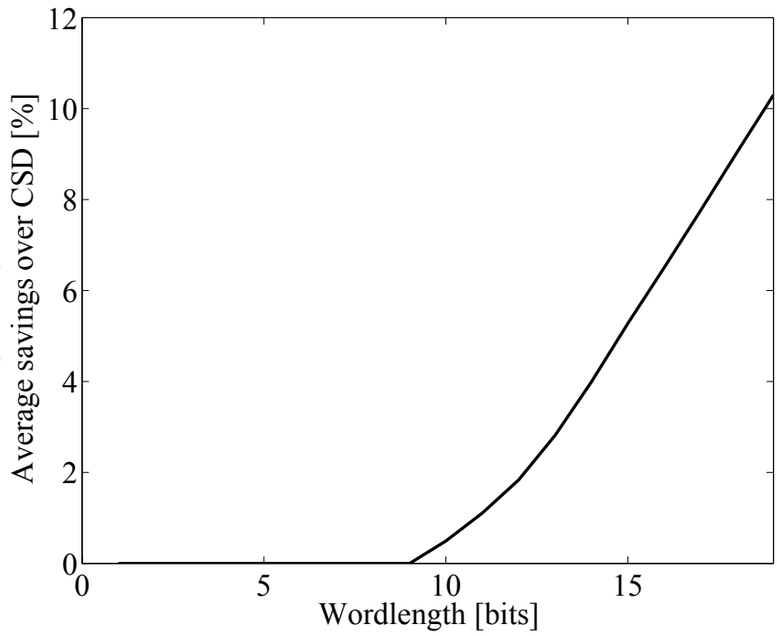

Fig. 9. Average percentage savings of CSAs for the proposed optimal multipliers over CSD multipliers as a function of coefficient wordlength.

3.2 Example

Consider the coefficient $693 = (10\bar{1}0\bar{1}0\bar{1}0101)_{\text{CSD}}$. To implement a multiplication with 693 using a CSD multiplier requires four CSAs. However, using graph-based multiplier 2 of cost 3 in Fig. 6 only three CSAs are required. The resulting graph and implementation is shown in Fig. 10, where $<< n$ denotes a left-shift of n bits.

For the example it can be noted that the first fundamental is -11 instead of 11 to avoid two negative input terms[2]. This is compensated for in the second adder stage.

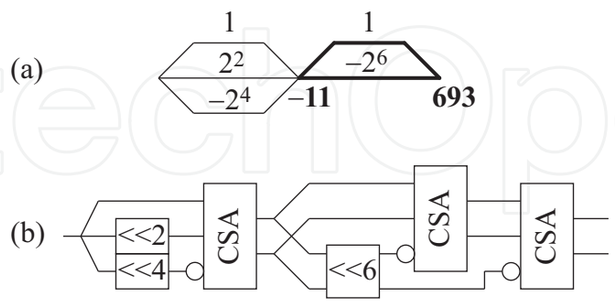

Fig. 10. Optimal CSA-based multiplication with 693: (a) graph representation directed form left to right and (b) structure.

[2] In this particular case it could also have been possible to use the representation $11 = 1 + 2 + 8$ to avoid subtractions.

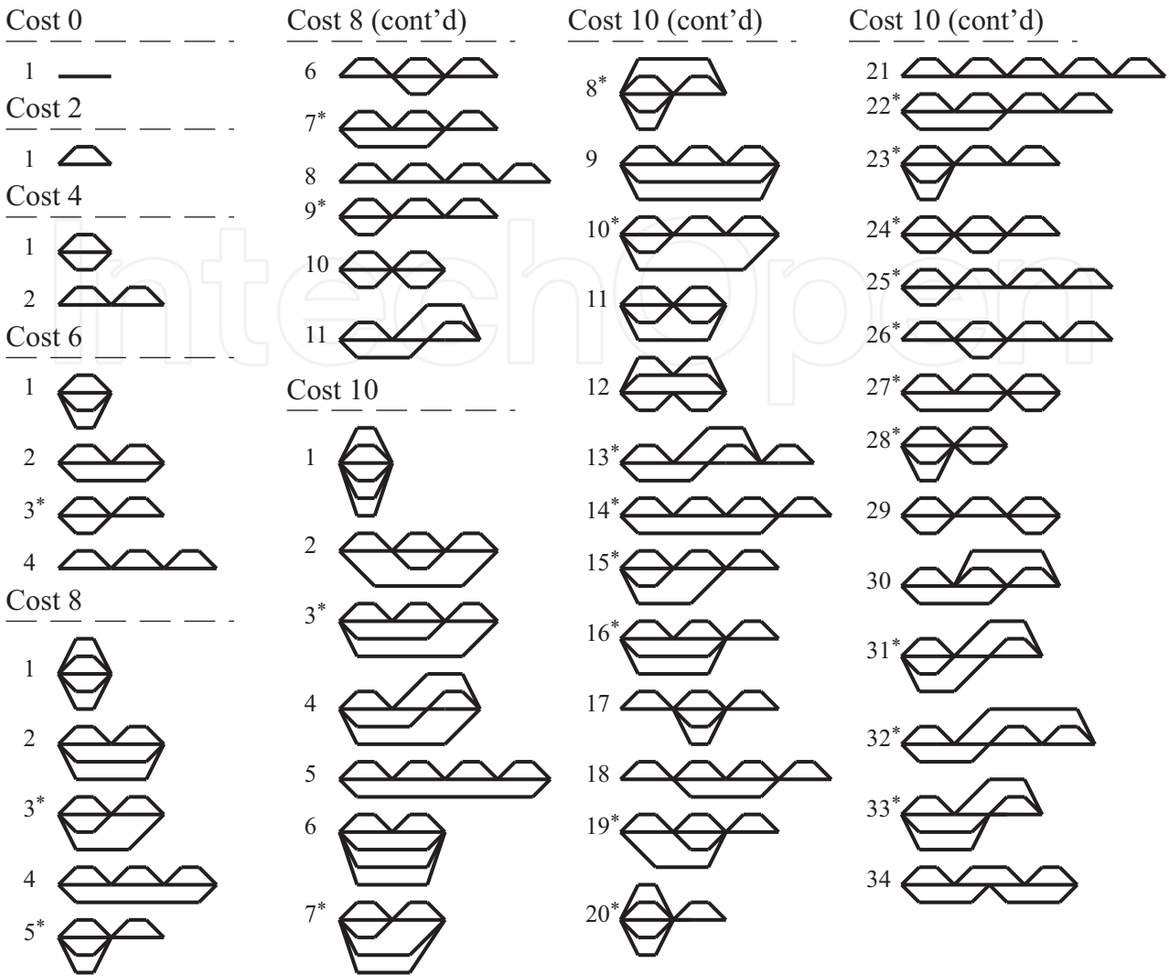

Fig. 11. Possible carry-save adder graphs with carry-save input generating different coefficient sets for 0 to 10 carry-save adders. The graphs are directed from left to right.

4. Carry-Save Arithmetic Constant Multipliers with Carry-Save Representation Input

When the input data is in carry-save representation it is possible to use the same graphs as in (Gustafsson et al., 2006). Now all words are in carry-save representation, and, hence, the number of carry-save adders is two times the number of incoming edges minus two. The possible graphs with up to ten adders are shown in Fig. 11.

4.1 Results

The possible savings in number of adders are similar to those in (Gustafsson et al., 2006) and the average number of adders is shown in Fig. 12. The average savings of the graph-based multipliers over CSD multipliers are shown in Fig. 13. Here, it can be seen that the average savings are about 25% for 19-bits coefficients. Also, the maximum number of CSAs required is reduced from 18 CSAs for a worst-case 19-bit CSD multiplier to 10 CSAs for a graph-based multiplier.

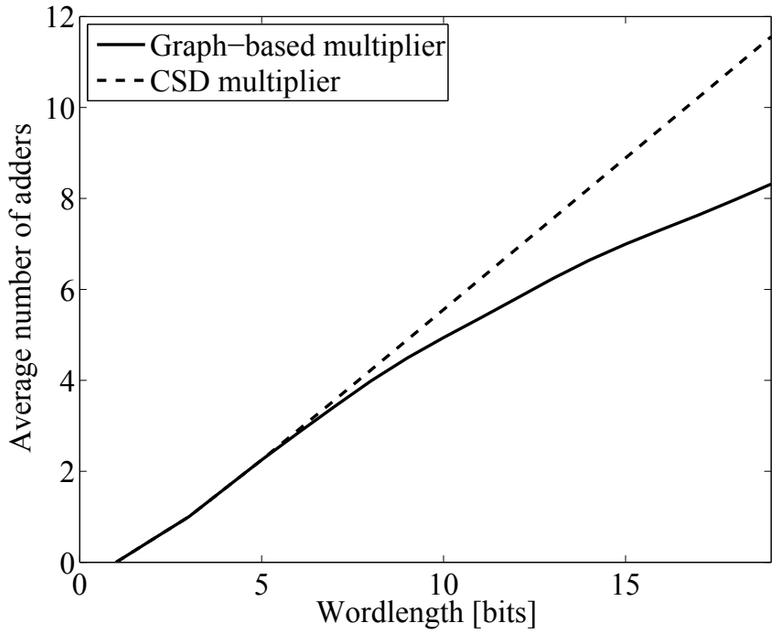

Fig. 12. Average number of CSAs as a function of coefficient wordlength for CSD multipliers and proposed optimal multipliers with carry-save input.

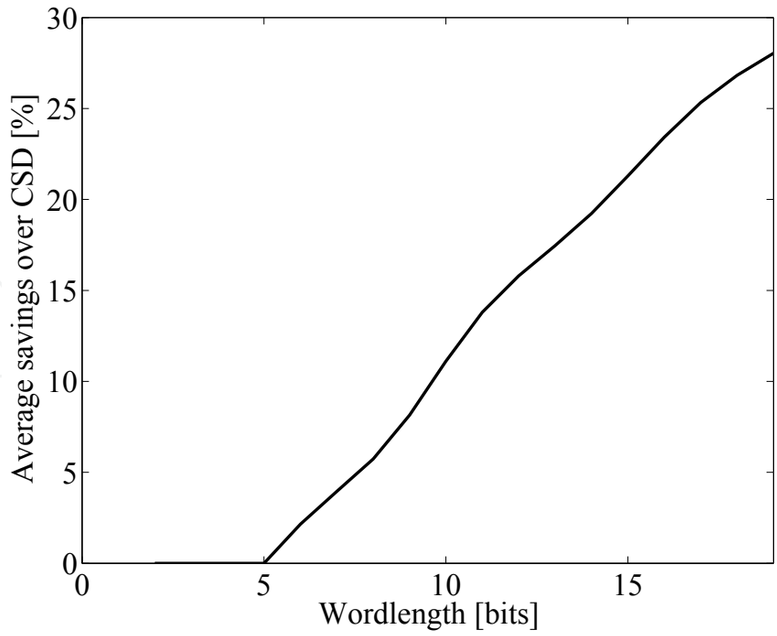

Fig. 13. Average percentage savings of CSAs for the proposed optimal multipliers over CSD multipliers as a function of coefficient wordlength with carry-save input.

The adder depth for the CSA-based graphs can not be easily computed based on results from the CPA-based graphs. The maximum number of nonzero digits and minimum depth for each graph is shown in Table 3. Using a similar reasoning as in (Gustafsson et al., 2006) we get that the maximum number of nonzero digits for a coefficient realized with K carry-save adders is (K is always even)

$$2^{K/2} \tag{7}$$

5. Multiple Constant Multiplication

For the case where several coefficients are multiplied with the same input a different approach can be used. Here, it is beneficial to be able to share partial results among the different coefficients to be able to reduce the total number of adders. It can be noted that the minimum number of adders per coefficient is simply one. Ideally, one would just need one extra adder for each unique[3] result. This is clearly the case for transposed direct form FIR filters, where the additions between the delay elements in Fig. 1, called *structural additions*, can be replaced by subtractions for negative coefficients. It may be beneficial to use CSA-based structural adders to obtain a high-speed implementation (Jain et al., 1991).

5.1 Proposed Algorithm

The proposed algorithm can be divided into an optimal part and a suboptimal part. The optimal part of the algorithm is described as:

1. The algorithm only considers positive odd fundamentals. Hence, negative fundamentals should be negated and even fundamentals should be divided by a suitable power of two to obtain an odd fundamental.
2. The fundamental one and fundamentals on the form $2^n \pm 1$ are removed as no CSAs are required to obtain these fundamentals. The remaining fundamentals form a set of unrealized fundamentals.
3. From the set of unrealized fundamentals add to the realized fundamental set all fundamentals, if any, that can be realized using one CSA, i.e., fundamentals on the form $2^m \pm 2^n \pm 1$, where $m > n > 1$.
4. Form all possible combinations of the fundamentals in the realized set times a power of two and a power of two, i.e., fundamentals on the form $2^m a \pm 1$ and $|a \pm 2^m|$, where a is an already realized fundamental. If any of these fundamentals are found in the unrealized set, move these to the realized set. If any fundamental has been realized and there are unrealized fundamentals remaining go to 4.

Each fundamental, added in steps 3 and 4, costs one adder. If all fundamentals are realized after this stage, the realization is known to be optimal in terms of adders. If not, at least two adders must be used to obtain one of the remaining fundamentals.
There are three different ways to obtain new fundamentals using two adders: fundamentals that requires two adders to be realized on its own, adding two powers of two to a power of two of an already realized fundamental, and a combination of two already realized fundamentals. As the two first ways realizes yet another fundamental, these two have preference over the combination of realized fundamentals. When two adders are required it is no longer certain that the solution is optimal. The possibly suboptimal part of the algorithm is described as:

[3] As shifts are free and sign often can be compensated for at some other part of the algorithm, all coefficients are normalized to be odd and positive.

Number of adders	Graph number	Maximum nonzero digits	Minimum adder depth
2	1	2	2
4	1	3	3
	2	4	4
6	1	4	4
	2	5	5
	3	6	5
	4	8	6
8	1	5	5
	2	6	6
	3	7	6
	4	9	7
	5	8	6
	6	12	7
	7	10	7
	8	16	8
	9	12	7
	10	9	6
	11	8	7
10	1	6	5
	2	13	8
	3	11	8
	4	9	8
	5	17	9
	6	7	7
	7	8	7
	8	9	7
	9	10	8
	10	13	8
	11	10	7
	12	8	6
	13	16	9
	14	18	9
	15	14	8
	16	12	8
	17	16	8
	18	20	9
	19	12	8
	20	10	7
	21	32	10
	22	20	9
	23	16	8
	24	18	8
	25	24	9
	26	24	8
	27	15	8
	28	12	7
	29	18	8
	30	12	8
	31	11	8
	32	14	9
	33	10	8
	34	13	9

Table 3. Maximum number of nonzero digits and minimum adder depth for the CSA multiplier graphs in Fig. 11 with carry-save input data.

5. From the set of unrealized fundamentals find all fundamentals that can be realized using two CSAs, i.e., fundamentals on the form $2^m \pm 2^n \pm 2^p \pm 1$, where $m > n > p > 1$. These fundamental can be derived from one and up to ten different fundamentals of cost-1. Find the cost-1 fundamental that is common to most unrealized fundamentals and add that fundamental to the realized set. Also move all fundamentals that can be realized from that cost-1 fundamental to the realized set. If there are more than one cost-1 fundamental that can realize the maximal number of fundamentals chose the minimum one. If there are unrealized fundamentals remaining and any fundamental was added go to 4.
6. If there are unrealized fundamentals remaining, form the set of all fundamentals that can be realized from one previously realized fundamental and two powers of two, i.e., on the form $|a \pm 2^m \pm 2^n|$ or $|2^m a \pm 2^n \pm 1|$. If any fundamental in the unrealized set is present in the generated set, move one of the fundamentals to the generated set. One intermediate fundamental is also generated, select the one (out of two) with the lowest magnitude to add to the set of realized fundamentals. If there are unrealized fundamentals remaining and any fundamental was added go to 4.
7. If there are unrealized fundamentals remaining, form a set of combinations of previously realized fundamentals times a power of two, i.e., on the form $|2^m a \pm b|$. If any fundamental in the unrealized set is present in the generated set, move one of the fundamentals to the generated set. If there are unrealized fundamentals remaining and any fundamental was added go to 4.
8. If there are unrealized fundamentals remaining, it is necessary to add a complete coefficient to the realized fundamental set. Complete coefficients with minimum number of adders can be generated using the work described in Section 3. Select the coefficient with the smallest sum of all its fundamentals (Dempster & Macleod, 1995). If there are there are unrealized fundamentals remaining go to 4.

5.2 Results

We compare our algorithm with the RAGn algorithm (Dempster & Macleod, 1995), where the resulting multiplier block is transformed to CSAs. Furthermore, we compare it to a modified version of the algorithm in (Pasko et al., 1999). In the original algorithm all subexpressions down to two bits were identified. As subexpressions with two bits are not useful when using CSAs, the algorithm is modified so that it only identifies subexpressions with at least three bits.

For sets of 25 coefficient with varying number of coefficient bits the average number of adders are shown in Fig. 14. For comparison the results using carry-propagation adders and the RAGn algorithm is included. Figure 14 shows that the proposed algorithm is better than both the modified algorithm from (Pasko et al., 1999) and design using CPAs. However, if only the actual number of adders is considered the CPA approach is better for nine coefficient bits and above. This is due to the greater flexibility in using intermediate fundamentals for CPAs.

The average number of adders for different sized coefficient sets with 12-bits coefficients is shown in Fig. 15. Again, the proposed algorithm is better compared to other algorithms. The multiplier block based on CPAs requires fewer adders for all sizes of the coefficient set with 12-bits coefficients.

It is clear that when CSAs are required the proposed algorithm is better than both the modified algorithm from (Pasko et al., 1999), which is based on subexpression sharing, and using the RAGn algorithm for CPAs. However, it is also clear that if only the number of adders, i.e., the

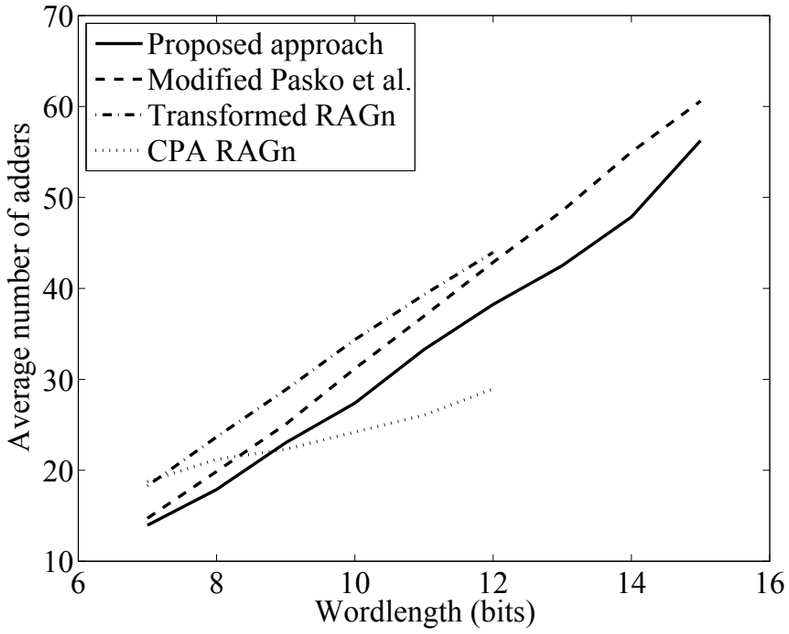

Fig. 14. Average number of adders for sets of 25 random coefficients.

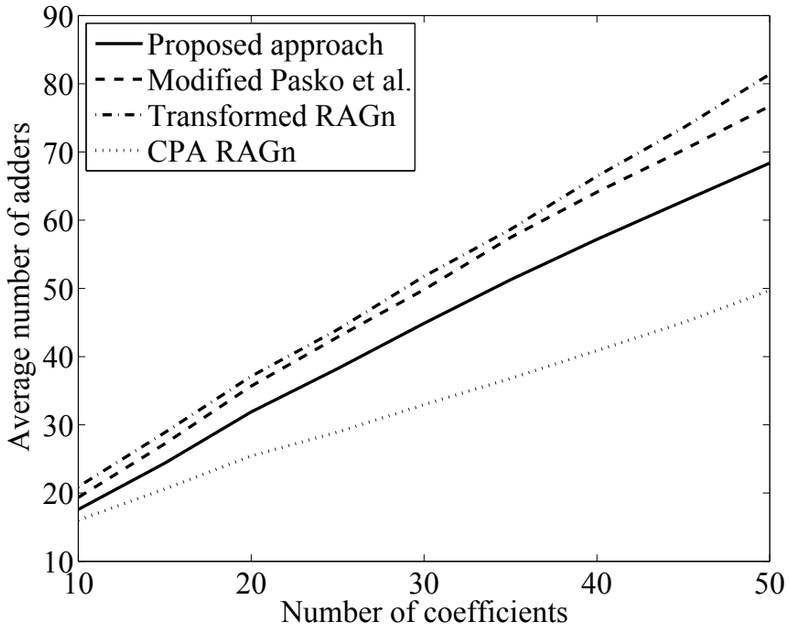

Fig. 15. Average number of adders for sets with 12-bits coefficients.

chip area, is of interest the RAGn algorithm with CPAs is the best choice. It should be noted that for the CSA multiplier block each coefficient requires a CPA to convert the carry-save representation to a non-redundant form, unless the redundant representation is used in later processing such as when carry-save structural adders are used.

6. Conclusions

Carry-save adders are useful to obtain high-speed implementation as carry-propagation can be avoided. However, when designing constant multipliers special care must be taken where the properties of the CSAs are considered. In this chapter we described the optimal design of single constant multipliers for coefficients with up to 19 bits wordlength. Both the cases with non-redundant representation as well as carry-save representation of the input was considered.

An algorithm for the multiple constant multiplication problem, suitable for transposed direct form FIR filters using carry-save representation of intermediate results but non-redundant input, was also presented.

For the non-redundant input cases, the results show that the number of CSAs is higher than the corresponding number of CPAs. Hence, from a complexity point of view, CPAs are advantageous. As such, the proposed techniques are useful when a high-speed realization is required.

7. References

Aksoy, L. & Güneş, E. O. (2008). Area optimization algorithms in high-speed digital FIR filter synthesis, *Proc. Symp. Integrated Circuits System Design*, pp. 64–69.

Aksoy, L., Güneş, E. O. & Flores, P. (2010). Search algorithms for the multiple constant multiplications problem: Exact and approximate, *Microprocessors and Microsystems* **34**(5): 151–162.

Dempster, A. G. & Macleod, M. D. (1994). Constant integer multiplication using minimum adders, *IEE Proc. Circuits Devices Systems*, Vol. 141, pp. 407–413.

Dempster, A. G. & Macleod, M. D. (1995). Use of minimum-adder multiplier blocks in FIR digital filters, *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing* **42**(9): 569–577.

Gustafsson, O. (2007). Lower bounds for constant multiplication problems, *IEEE Transactions on Circuits and Systems II: Express Briefs* **54**(11): 974–978.

Gustafsson, O. (2008). Comments on'A 70 MHz Multiplierless FIR Hilbert Transformer in 0.35 μm Standard CMOS Library', *IEICE Trans. Fundamentals* **91**(3): 899–900.

Gustafsson, O., Dempster, A. G., Johansson, K., Macleod, M. D. & Wanhammar, L. (2006). Simplified design of constant coefficient multipliers, *Circuits Systems Signal Processing* **25**(2): 225–251.

Gustafsson, O., Dempster, A. G. & Wanhammar, L. (2004). Multiplier blocks using carry-save adders, *Proc. IEEE Int. Symp. Circuits Systems*, Vol. 2, pp. 473–476.

Gustafsson, O., Ohlsson, H. & Wanhammar, L. (2001). Minimum-adder integer multipliers using carry-save adders, *Proc. IEEE Int. Symp. Circuits Systems*, pp. 709–712.

Gustafsson, O. & Wanhammar, L. (2007). Low-complexity constant multiplication using carry-save arithmetic for high-speed digital filters, *Proc. Int. Symp. Image and Signal Processing and Analysis*, pp. 212–217.

Hartley, R. I. (1996). Subexpression sharing in filters using canonic signed digit multipliers, *IEEE Trans. Circuits Systems II: Analog and Digital Signal Processing* **43**(10): 677–688.

Hosangadi, A., Fallah, F. & Kastner, R. (2006). Optimizing high speed arithmetic circuits using three-term extraction, *Proc. Conf. Design Automation Test in Europe*, pp. 1294–1299.

Jaccottet, D., Costa, E., Aksoy, L., Flores, P. & Monteiro, J. (2010). Design of low-complexity and high-speed digital finite impulse response filters, *Proc. IEEE/IFIP Int. Conf. VLSI System-on-Chip*, pp. 292–297.

Jain, R., Yang, P. & Yoshino, T. (1991). FIRGEN: A computer-aided design system for high performance FIR filter integrated circuits, *IEEE Trans. Signal Processing* **39**(7): 1655–1668.

Kleine, U. & Noll, T. (1987). On the forced response stability of wave digital filters using carry-save arithmetic, *AEU, Archiv für Elektronik und Übertragungstechnik* **41**(6): 321–324.

Lim, Y. C. (1990). Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude, *IEEE Trans. Circuits Systems* **37**(12): 1480–1486.

Noll, T. (1991). Carry-save architectures for high-speed digital signal processing, *J. VLSI Signal Processing* **3**(1): 121–140.

Pasko, R., Schaumont, P., Derudder, V., Vernalde, S. & Durackova, D. (1999). A new algorithm for elimination of common subexpressions, *IEEE Trans. Computer-Aided Design Integrated Circuits Systems* **18**(1): 58–68.

Potkonjak, M., Srivastava, M. B. & Chandrakasan, A. P. (1996). Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination, *IEEE Trans. Computer-Aided Design Integrated Circuits Systems* **15**(2): 151–165.

Thong, J. & Nicolici, N. (2009). Time-efficient single constant multiplication based on overlapping digit patterns, *IEEE Trans. VLSI Systems* **17**(9): 1353–1357.

Voronenko, Y. & Püschel, M. (2007). Multiplierless multiple constant multiplication, *ACM Trans. Algorithms* **3**.
URL: *http://doi.acm.org/10.1145/1240233.1240234*

Wallace, C. (1964). A suggestion for a fast multiplier, *IEEE Trans. Electronic Computers* (1): 14–17.

Yli-Kaakinen, J. & Saramäki, T. (2007). A systematic algorithm for the design of lattice wave digital filters with short-coefficient wordlength, *IEEE Trans. Circuits Systems I: Regular Papers* **54**(8): 1838–1851.

Digital Filters

Edited by Prof. Fausto Pedro GarcÃa MÃ¡rquez

ISBN 978-953-307-190-9
Hard cover, 290 pages
Publisher InTech
Published online 11, April, 2011
Published in print edition April, 2011

The new technology advances provide that a great number of system signals can be easily measured with a low cost. The main problem is that usually only a fraction of the signal is useful for different purposes, for example maintenance, DVD-recorders, computers, electric/electronic circuits, econometric, optimization, etc. Digital filters are the most versatile, practical and effective methods for extracting the information necessary from the signal. They can be dynamic, so they can be automatically or manually adjusted to the external and internal conditions. Presented in this book are the most advanced digital filters including different case studies and the most relevant literature.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Oscar Gustafsson and Lars Wanhammar (2011). Low-Complexity and High-Speed Constant Multiplications for Digital Filters Using Carry-Save Arithmetic, Digital Filters, Prof. Fausto Pedro GarcÃa MÃ¡rquez (Ed.), ISBN: 978-953-307-190-9, InTech, Available from: http://www.intechopen.com/books/digital-filters/low-complexity-and-high-speed-constant-multiplications-for-digital-filters-using-carry-save-arithmet

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen