Proceedings of the 10th WSEAS International Conference on CIRCUITS, Vouliagmeni, Athens, Greece, July 10-12, 2006 (pp324-329)

Trade-Offs in Multiplier Block Algorithms for Low Power Digit-Serial FIR Filters

KENNY JOHANSSON, OSCAR GUSTAFSSON, and LARS WANHAMMAR
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping
SWEDEN
http://www.es.isy.liu.se/

Abstract: - In this paper trade-offs in digit-serial multiplier blocks are studied. Three different algorithms for realization of multiplier blocks are compared in terms of complexity and adder depth. Among the three algorithms is a new algorithm that reduces the number of shifts while the number of adders is on average the same. Hence, the total complexity is reduced for multiplier blocks implemented using digit-serial arithmetic, where shift operations have a hardware cost. An example implementation is used to compare the power consumption for five approaches: the three algorithms, using separate multipliers based on CSD representation, and an algorithm based on subexpression sharing. The design of low power multiplier blocks is shown to be a more complicated problem than to reduce the complexity. A main factor that needs to be considered is adder depth. Furthermore, digit-serial shifts will reduce glitch propagation.

Key-Words: - Multiple constant multiplication, Multiplier block, Multiplierless, Digit-serial arithmetic, FIR filter, Low power, Adder graph, Shift-and-add multiplication, Adder depth

1 Introduction

Multiplication with a constant-coefficient is commonly used in digital signal processing (DSP) circuits, such as digital filters. This type of multiplication can be efficiently implemented using shifts, adders, and subtractors. As the complexity is similar for adders and subtractors we will refer to both as adders, and the number of adders and subtractors as adder cost.

In some applications, e.g., the transposed direct form FIR filter as shown in Fig. 1, one input is multiplied with multiple coefficients [1],[2]. This is often referred to as the multiple-constant multiplication (MCM) problem, which can be realized using a multiplier block as illustrated by the dashed box in Fig. 1.

A simple method to realize multiplier blocks is to implement each multiplier separately, e.g. using the canonic signed-digit (CSD) representation [1],[3]. However, it is possible to utilize redundant partial results to reduce the number of adders required to realize multiple-constant multiplication [4]–[9].

Most existing work on MCM has focused on minimizing the number of adders, as the shift operations can be hardwired in a bit-parallel architecture. However, in bit- and digit-serial arithmetic the shift operations require flip-flops, and hence, they have to be considered as well. In [10] an algorithm that minimizes the number of shifts while keeping the adder cost low was proposed.

Most work on implementation of digit-serial FIR filters has focused on implementation in FPGAs and without using multiplier blocks [11]–[13]. However, in [14] the digit-size trade-off in implementation of

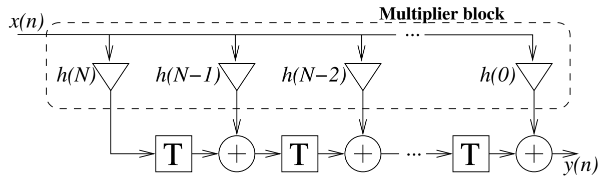

Figure 1. Transposed direct form *N*th-order FIR filter.

digit-serial transposed direct form FIR filters using multiplier blocks was studied. One of the best MCM algorithms in terms of number of adders, referred to as RAG-*n* [5], and the algorithm proposed in [10], referred to as RSAG-n, was used in the comparison.

The conclusion in [14] was that an algorithm that minimize the number of adders, while keeping the number of shifts low, would be preferable for most cases.

In this work we propose an algorithm that firstly aim to minimize the number of adders and secondly the number of shifts. We investigate how large savings that can be achieved compared with RAG-*n* and RSAG-*n*, respectively. The algorithms are compared in terms of complexity and adder depth. Furthermore, we provide an example implementation and compare the power consumption of the three algorithms with using CSD coefficients and using the algorithm in [7].

2 Digit-Serial Arithmetic

In digit-serial arithmetic, the words are divided into digits of d bits that are processed one digit at a time [15],[16]. The integer number d is usually denoted the

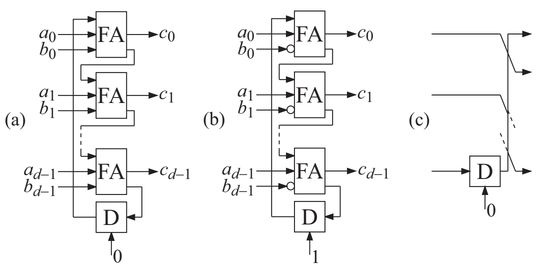

Figure 2. Digit-serial (a) adder, (b) subtractor, and (c) shift.

digit-size. This provides a trade-off between area, speed, and power consumption [16],[17]. For the special case where d equals the data wordlength we have bit-parallel processing and when d equals one we have bit-serial processing.

Digit-serial operators can be derived either by unfolding bit-serial operators [18] or by folding bit-parallel operators [19]. In Fig. 2, a digit-serial adder, subtractor, and shift operation is shown, respectively.

Considering the processing elements it is clear that the area of an FIR filter using digit-serial arithmetic will increase for larger digit-size. How speed and power consumption is affected is not obvious.

3 Proposed Algorithm

In [5] the n-dimensional Reduced Adder Graph (RAG-n) algorithm was introduced. This algorithm is known to be one of the best MCM algorithms in terms of number of adders. Based on this algorithm an n-dimensional Reduced Shift and Add Graph (RSAG-n) algorithm has been developed [10], that not only tries to minimize the adder cost, but also the number of shifts. However, this algorithm has an increased adder cost, which will be dominating for larger digit-sizes [14].

Here, an n-dimensional Reduced Add and Shift Graph (RASG-n) algorithm is proposed. The new algorithm is a hybrid of the RAG-n [5] and RSAG-n [10] algorithms. RASG-n work with odd coefficients, like RAG-n and only realizes one coefficient in each iteration, like RSAG-n. When it is possible to realize more than one coefficient RASG-n selects the one that require the lowest number of additional shifts. This makes it possible for RASG-n to minimize both the number of adders and shifts in an effective way.

These algorithms are graph based where edges corresponds to shifts and nodes to additions. Node values are referred to as fundamentals. Realized coefficients are removed from the coefficient set and added to an interconnection table that specifies how the value is obtained. The termination condition of the algorithm is that the coefficient set is empty. The steps in the RSAG-n algorithm are as follows:

1. Divide even coefficients by two until odd, and save the number of times each coefficient is divided. These shifts at the outputs can be considered to be free when other coefficients are synthesised.

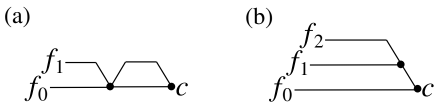

Figure 3. The coefficient c is obtained from (a) two existing fundamentals or (b) three existing fundamentals.

2. Remove zeros, ones, i.e., coefficients which corresponds to a power-of-two, and repeated coefficients from the coefficient set.
3. Compute the single-coefficient adder cost for each coefficient, which is done by using a look-up-table.
4. Compute a sum matrix based on power-of-two multiples of the fundamental values included in the interconnection table. At start this matrix is

$$\begin{array}{c|cccccc} & 1 & -1 & 2 & -2 & 4 & \dots \\ \hline 1 & 2 & 0 & 3 & -1 & 5 & \dots \\ -1 & 0 & -2 & 1 & -3 & 3 & \dots \\ 2 & 3 & 1 & 4 & 0 & 6 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{array}$$

and is then extended when new fundamentals are added. If any required coefficients are found in the matrix, compute the required number of shifts. Find the coefficients which require the lowest number of additional shifts, and select the smallest of those. Add this coefficient to the interconnection table and remove it from the coefficient set.
5. Repeat step 4 until no required coefficient is found in the sum matrix.
6. For each remaining coefficient, check if it can be obtained by the strategies illustrated in Fig. 3. For both cases two new adders are required. If any coefficients are found, select the smallest coefficient of those which require the lowest number of additional shifts. Add this coefficient and the extra fundamental to the interconnection table. Remove the coefficient from the coefficient set.
7. Repeat step 5 and 6 until no required coefficient is found.
8. Choose the smallest coefficient with lowest single-coefficient adder cost. Different sets of fundamentals that can be used to realize the coefficient are obtained from a look-up-table. For each set, remove fundamentals that are already included in the interconnection table and compute the required number of shifts. Find the sets which require the lowest number of additional shifts, and of those, select the set with smallest sum. Add this set and the coefficient to the interconnection table. Remove the coefficient from the coefficient set.
9. Repeat step 5, 6, 7, and 8 until the coefficient set is empty.

The basic ideas for the RAG-*n* [5], RSAG-*n* [10], and RASG-*n* algorithms are similar, but the resulting difference is significant. The main difference between the first two algorithms is that RAG-*n* chooses to realize coefficients by using extra fundamentals of minimum value, while RSAG-*n* chooses fundamentals that require a minimum number of shifts. The result of these two different strategies is that RAG-*n* is more likely to reuse fundamentals, due to the selection of smaller fundamental values and by that reduce the adder cost, while RSAG-*n* is more likely to reduce the number of shifts. As the proposed algorithm, RASG-*n*, is a hybrid of these strategies realizations with both few adders and few shifts are obtained.

It is worth noting that if all coefficients are realized before step 6 of the algorithm, the corresponding implementation has optimal adder cost [5].

4 Complexity

In this section the complexity, including adders and shifts, for the three algorithms are compared. Average results are shown for 100 random coefficient sets.

4.1 Coefficient Wordlength Effects

The different algorithms were used to design multiplier blocks with coefficient sets of varying wordlength. The setsize is fixed to 25 coefficients.

In Fig. 4 (a) the average number of additional adders for each coefficient using the RASG-*n* algorithm is shown. Coefficients that can be realized with no adders includes zeros, power-of-twos, and repeated coefficients. Most coefficients can be realized with only one additional adder. The number of adders is optimal for all coefficient sets of wordlengths up to 8 bits as shown in Fig. 4 (b). Corresponding statistics for the other two algorithms would look similar.

The average number of adders for the three algorithms are shown in Fig. 5 (a). It is clear that the number of adders is higher for RSAG-*n*. The average number of shifts is lower for RASG-*n* than for RAG-*n*, while RSAG-*n* has the lowest number of shifts as shown in Fig. 5 (b).

In Fig. 6 (a) a histogram for the required number of adders using 10 bits coefficients is shown. RASG-*n* and RAG-*n* only have a different number of adders in one out of the 100 cases. As can be seen in Fig. 6 (b) RASG-*n* have on average more than 11 shifts less than RAG-*n*. RSAG-*n* has the highest number of adders and the lowest number of shifts.

4.2 Coefficient Setsize Effects

With the coefficient wordlength fixed to 10 bits, the different algorithms were used to design multiplier blocks of varying setsize.

The average number of additional adders is shown in Fig. 7 (a) for the RASG-*n* algorithm. For a small

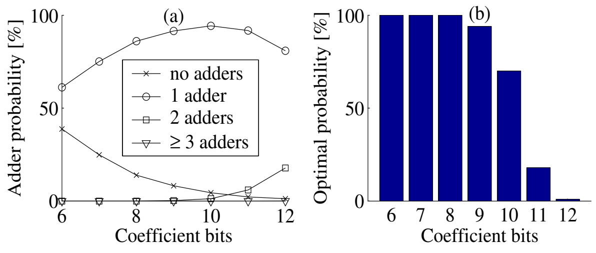

Figure 4. Statistics from realization of multiplier blocks using the RASG-*n* algorithm. (a) Average number of additional adders for each coefficient. (b) The probability of proven optimal adder cost.

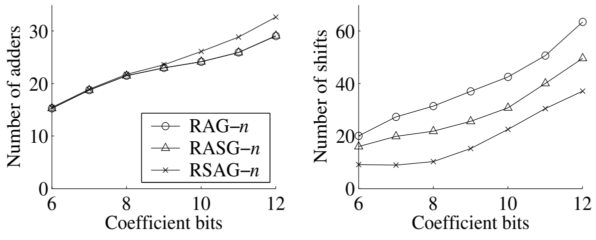

Figure 5. Average number of (a) adders and (b) shifts for sets of 25 coefficients.

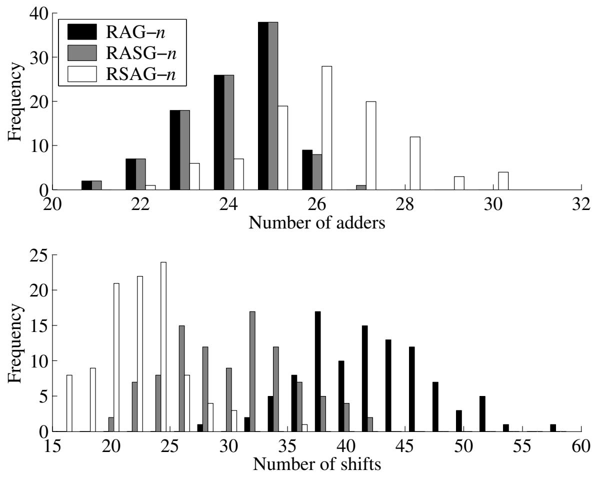

Figure 6. Frequency of the number of (a) adders and (b) shifts for the three different algorithms using 10 bits coefficients.

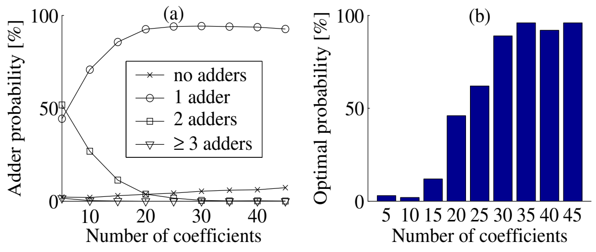

Figure 7. (a) Average number of additional adders for each coefficient and (b) probability of proven optimal adder cost for the RASG-*n* algorithm.

setsize many of the coefficients will require two additional adders, which result in a low probability of optimality as shown in Fig. 7 (b). For a large setsize most coefficients can be realized with only one additional

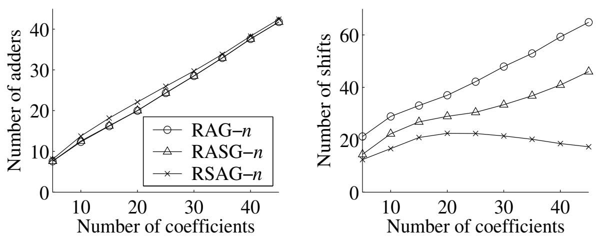

Figure 8. Average number of (a) adders and (b) shifts for 10 bits coefficients.

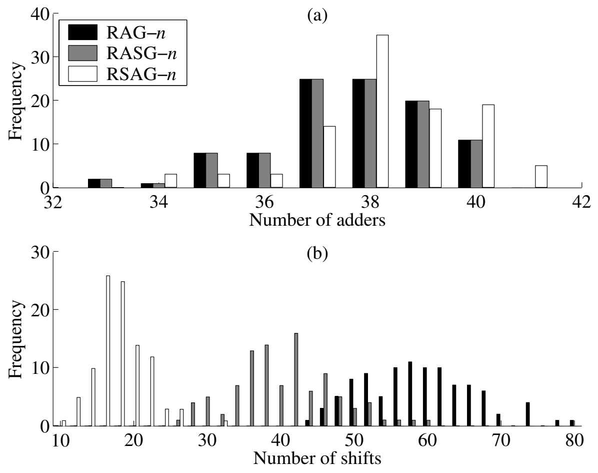

Figure 9. Frequency of the number of (a) adders and (b) shifts for the three different algorithms using sets of 40 coefficients.

adder, and the probability that the total number of adders is optimal is high.

In Fig. 8 (a) the average number of adders for the three algorithms are shown. Again, the number of adders for RAG-*n* and RASG-*n* are similar. All algorithms are likely to have an optimal number of adders for a large setsize, and the difference is naturally small for a small setsize. Hence, the difference between RSAG-*n* and the other two algorithms has a maximum, which occur for setsize 20.

The differences in number of shifts is increasing for larger setsize as shown in Fig. 8 (b). RSAG-*n* takes full advantage of the fact that coefficients are more likely to be obtained without additional shifts when more values are available, and of course has the lowest number of shifts. The average number of shifts is lower for RASG-*n* than for RAG-*n*.

In Fig. 9 (a) a histogram for the required number of adders using sets of 40 coefficients is shown. It can be seen that RASG-*n* and RAG-*n* have the same number of adders in all 100 cases. However, RASG-*n* have on average almost 18 shifts less than RAG-*n* as illustrated in Fig. 9 (b).

5 Adder Depth

In [20] and [21] methods to predict the number of transitions in multiplier blocks was introduced. These methods are based on the fact that high adder depth result in more transitions, and consequently higher power consumption.

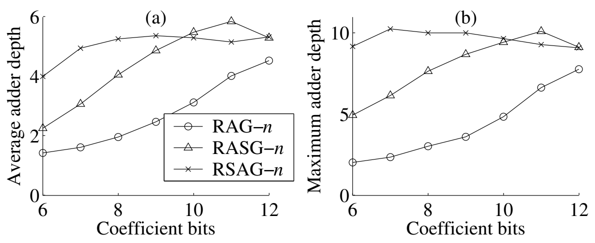

Figure 10. Adder depth for sets of 25 coefficients. (a) Average and (b) maximum adder depth.

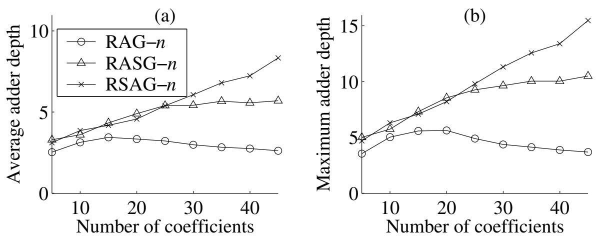

Figure 11. Adder depth for 10 bits coefficients. (a) Average and (b) maximum adder depth.

The characteristics for the three algorithms considering adder depth are shown in Figs. 10 and 11. The same coefficient sets as in Section 4 was used. It is clear that RAG-*n* has the lowest adder depth. Furthermore, the adder depth does not increase for larger coefficient sets for RAG-*n*.

6 Implementation Example

The power consumption is studied by the use of an example filter implemented by logic synthesis of VHDL code using a 0.35 µm CMOS standard cell library.

A 27th-order lowpass linear-phase FIR filter with passband edge 0.15π rad and stopband edge 0.4π rad is used for the evaluation. The maximum passband ripple is 0.01, while the stopband attenuation is 80 dB. The filter has symmetric coefficients {4, 18, 45, 73, 72, 6, –132, –286, –334, –139, 363, 1092, 1824, 2284}/8192. The filter is implemented using the transposed direct form structure shown in Fig. 1. Only the arithmetic parts are considered here.

The required number of adders and shifts for the three different algorithms is shown in Table 1. The RAG-*n* and RASG-*n* algorithms require 12 adders, which is optimal for this coefficient set. The RSAG-*n* algorithm requires the lowest number of shifts. Also included is an implementation using separate CSD multipliers and one based on the algorithm in [7]. The smallest area is obtained for RSAG-*n* for small digit-sizes, while for larger digit-sizes RASG-*n* is the best.

The maximum clock frequency and corresponding maximum sample frequency is shown in Fig. 12. Here it is seen that the CSD implementations have the highest sample frequency. This is because for CSD multipliers there are at least two shifts between each adder,

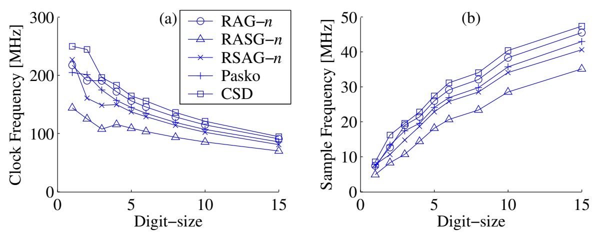

Figure 12. (a) Maximum clock frequency and (b) maximum sample frequency.

Table 1. Arithmetic complexity for the example filter.

Algorithm	**Adders**	**Shifts**		
		Internal	**External**	**Total**
RAG-*n* [5]	12	20	10	30
RASG-*n*	12	14	9	23
RSAG-*n* [10]	14	18	1	19
Pasko [7]	15	27	12	39
CSD [3]	28	78	20	98

and hence, the critical path is short. The slowest implementations are the ones based on RASG-*n*. This can be explained by that many adders are cascaded without any shifts in between for the RASG-*n* case.

The power consumption was obtained using NanoSim™ with 100 random input sample. As can be seen in Fig. 13 (a) the energy per sample for the shifts in the multiplier block is smallest for RSAG-*n* and largest for CSD. The energy per sample for the adders in the multiplier block is shown in Fig. 13 (b). RAG-*n* consumes less energy for any digit-size. By adding the energy for the adders and the shifts, the energy for the multiplier block is obtained, as shown in Fig. 13 (c). RSAG-*n* consumes the least energy for digit-sizes one and two and RAG-*n* for larger digit-sizes. Note that the energy consumption corresponding to shifts and adders dominates for small and large values of the digit-size, respectively. In Fig. 13 (d) the normalized energy per sample is shown. From this it can be seen that the optimal digit-size for RASG-*n* and RSAG-*n* is three, while for the other three algorithms it is six. The energy per sample consumed for the structural adders is shown in Fig. 13 (e), while the total energy for all arithmetic operations is shown in Fig. 13 (f). The power for the structural adders is only effected by the glitches from the multiplier block. It can be seen that the glitches are significantly higher for RASG-*n* and RSAG-*n*. For RSAG-*n* the reason is that the number of external shifts, which provides glitch reduction between the multiplier block and the structural adders, is small. For RASG-*n* the increased number of glitches due to high adder depth in the multiplier block is propagated to the structural adders.

A surprising result is that the energy consumed by the adders is larger for RASG-*n* than RSAG-*n*, although the number of adders is smaller. The reason for this will be discussed in the following.

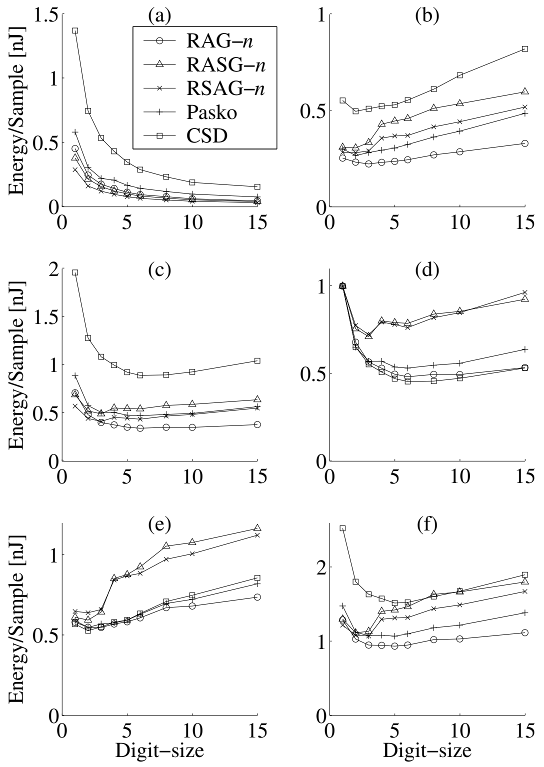

Figure 13. Consumed energy per sample for (a) shifts, (b) adders, (c) the total multiplier block, (d) normalized for the total multiplier block, (e) structural adders, and (f) all arithmetic parts.

In Fig. 14 the adder depth for each coefficient in the example filter using the three different algorithms is illustrated. It is clear that RASG-*n* has larger adder depth than RSAG-*n*, which explains the higher power consumption. RAG-*n* has the lowest adder depth.

The fact that adder depth is highly correlated with power consumption is established when the energy consumed in each adder is investigated. This is shown in Figs. 15 (a) and (b) for digit-size one and five, respectively. Note that the RSAG-*n* implementation includes two extra adders, hence, the total energy is larger than illustrated in Fig. 15.

7 Conclusions

In this paper trade-offs in digit-serial multiplier blocks was studied. Some conclusions regarding design guidelines for low power digit-serial multiplier blocks can be deduced. The actual complexity in terms of adder cost and number of shifts is not the main factor determining the power consumption. Instead the adder depth, as for parallel arithmetic, is a main contributor. Hence, an algorithm with low adder depth should be used. Furthermore, the shifts prevent glitch propagation through subsequent adders. For even coefficients the shifts can be placed either before or after the final additions. Hence, a heuristic for placing the shifts would be useful.

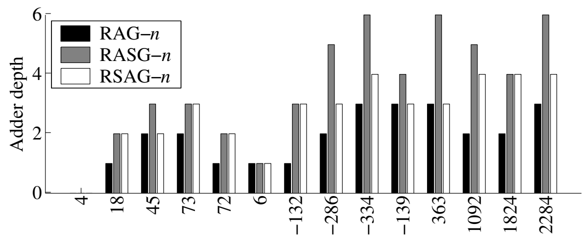

Figure 14. Adder depth for each coefficient in multiplier block realizations using three different algorithms.

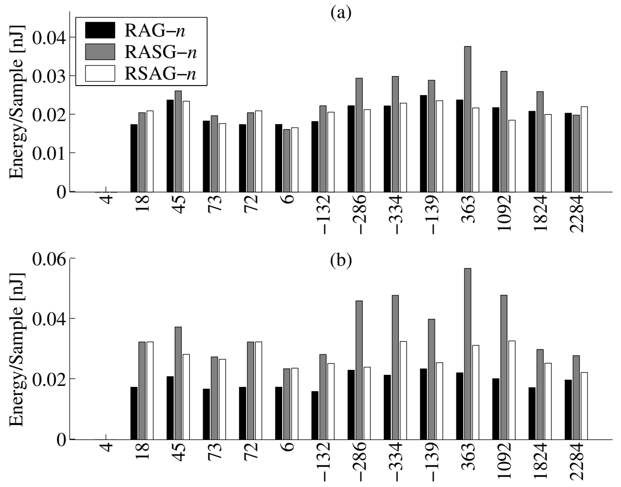

Figure 15. Energy per sample for the adders corresponding to each coefficient of the example filter. Digit-size (a) one and (b) five.

References:

[1] L. Wanhammar, *DSP Integrated Circuits*, Academic Press, 1999.

[2] L. Wanhammar and H. Johansson, *Digital Filters*, Linköping University, 2002.

[3] M. Vesterbacka, K. Palmkvist, and L. Wanhammar, Realization of serial/parallel multipliers with fixed coefficients, in *Proc. National Conf. Radio Science,* 1993, pp. 209–212.

[4] D. R. Bull and D. H. Horrocks, Primitive operator digital filters, *IEE Proc. G*, Vol. 138, No. 3, 1991, pp. 401–412.

[5] A. G. Dempster and M. D. Macleod, Use of minimum-adder multiplier blocks in FIR digital filters, *IEEE Trans. Circuits Syst. II*, Vol. 42, No. 9, 1995, pp. 569–577.

[6] R. I. Hartley, Subexpression sharing in filters using canonic signed digit multipliers, *IEEE Trans. Circuits Syst. II*, Vol. 43, 1996, pp. 677–688.

[7] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, A new algorithm for elimination of common subexpressions, *IEEE Trans. Computer-Aided Design Integrated Circuits*, Vol. 18, No. 1, 1999, pp. 58–68.

[8] O. Gustafsson, H. Ohlsson, and L. Wanhammar, Improved multiple constant multiplication using minimum spanning trees, in *Proc. Asilomar Conf. Signals, Syst., Comp.*, 2004, pp. 63–66.

[9] Y. Voronenko and M. Püschel, Multiplierless multiple constant multiplication, *ACM Trans. Algorithms*, 2006.

[10] K. Johansson, O. Gustafsson, A. G. Dempster, and L. Wanhammar, Algorithm to reduce the number of shifts and additions in multiplier blocks using serial arithmetic, in *Proc. IEEE Melecon*, 2004, pp. 197–200.

[11] S. He and M. Torkelson, FPGA implementation of FIR filters using pipelined bit-serial canonical signed digit multipliers, in *Proc. IEEE Custom Integrated Circuits Conf.*, 1994, pp. 81–84.

[12] J. Valls, M. M. Peiro, T. Sansaloni, and E. Boemo, Design and FPGA implementation of digit-serial FIR filters, in *Proc. IEEE Int. Conf. Electronics, Circuits, Syst.*, 1998, Vol. 2, pp. 191–194.

[13] H. Lee and G. E. Sobelman, FPGA-based FIR filters using digit-serial arithmetic, in *Proc. IEEE Int. ASIC Conf.*, 1997, pp. 225–228.

[14] K. Johansson, O. Gustafsson, and L. Wanhammar, Implementation of low-complexity FIR filters using serial arithmetic, in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, pp. 1449–1452.

[15] S. G. Smith and P. B. Denyer, *Serial-Data Computation*, Kluwer, 1988.

[16] R. I. Hartley and K. K. Parhi, *Digit-Serial Computation*, Kluwer, 1995.

[17] H. Suzuki, Y.-N. Chang, and K. K. Parhi, Performance tradeoffs in digit-serial DSP systems, in *Proc. Asilomar Conf. Signals, Syst., Computers*, 1998, Vol. 2, pp. 1225–1229.

[18] K. K. Parhi, A systematic approach for design of digit-serial signal processing architectures, *IEEE Trans. Circuits Syst.*, Vol. 38, No. 4, 1991, pp. 358–375.

[19] K. K. Parhi, C.-Y. Wang, and A. P. Brown, Synthesis of control circuits in folded pipelined DSP architectures, *IEEE J. Solid-State Circuits*, Vol. 27, 1992, pp. 29–43.

[20] S. S. Demirsoy, A. G. Dempster, and I. Kale, Transition analysis on FPGA for multiplier-block based FIR filter structures, in *Proc. IEEE Int. Conf. Elect. Circuits Syst.*, 2000, Vol. 2, pp. 862–865.

[21] S. S. Demirsoy, A. G. Dempster, and I. Kale, Power analysis of multiplier blocks, in *Proc. IEEE Int. Symp. Circuits Syst.*, 2002, Vol. 1, pp. 297–300.