A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters

Uwe Meyer-Baese
The Department of ECE
FAMU-FSU College of Eng.
Tallahassee, FL USA
Email: umb@eng.fsu.edu

Jiajia Chen, Chip Hong Chang
Nanyang Technological University
School of EE Engineering
Blk S2,Nanyang Avenue, Singapore
Email:
{chen0183,echchang}@ntu.edu.sg

Andrew G. Dempster
The School of Surveying and
Spatial Information Systems
University of New South Wales
Sydney 2052, Australia
Email:a.dempster@unsw.edu.au

***Abstract*—The paper starts with an overview of distributed arithmetic (DA) and n-dimensional reduced adder graph (RAG-n) multiplierless filter design methods. Since DA designs are table-based and RAG-n designs are adder-based, FPGA synthesis design data are used for a realistic comparison. Benchmark FIR filters [1-4] of length 11 to 63 are compiled. For a wide set of realistic design examples, it will be shown that pipelined RAG-n designs achieve on average a gain of 71% in area, equivalent performance in speed, and a 56% improvement in cost compared with DA-based designs.**

I. Introduction

Field-programmable gate arrays (FPGAs) are on the verge of revolutionizing digital signal processing. Many front-end digital signal processing (DSP) algorithms, such as FFTs, multi channel filterbanks, or wavelets, to name just a few, previously built with ASICs or programmable digital signal processors, are now often replaced by FPGAs.

II. Multiplier Block Coding and the RAG-n Algorithm

There are only a few applications (e.g., adaptive filters) where a general programmable filter architecture is required. In many applications, the filters are linear time-invariant (LTI) systems, and the coefficients do not change over time. In this case, the hardware effort can essentially be reduced by exploiting the constant coefficient multiplier coding and adder (trees) used to implement the FIR filter multiplier block (see Fig. 1). The coefficients of the filter are f={3,0,-25,0,150,256,150,0,-25,0,3}.

In several DSP systems it is found that multipliers share the same input. These multipliers can be combined in a multiplier block. The transposed FIR filter shown in Figure 1 is a typical example for a multiplier block. Dempster and Macleod [5] have introduced a systematic algorithm, which produce s an n-Dimensional Reduced Adder Graph (RAG-n) of a block multiplier. In general, however, finding the optimal RAG-n is an NP-hard problem. RAG-n determines in the first steps an optimal coding; for the suboptimal part, heuristics are applied. The full 10-step RAG-n algorithm is summarized in [5]. To illustrate the RAG-n algorithm, consider the coefficients defining the F5 halfband FIR filter of Goodman and Carey [1]. For the halfband F5 filter in direct form, using a CSD code, 9 adders are required. If the transposed filter and RAG-n algorithms are used, the number of adders is reduced from 9 to 3. Fig. 1 shows the resulting reduced-adder graph for the design of the F5 filter.

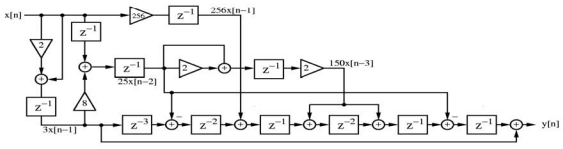

Fig. 1. Pipelined RAG-n of F5 [1].

A. RAG-95s

Let us first briefly clarify the algorithms and not-so-obvious steps of the original RAG-95 [5]: In *Step 1* all coefficients are reduced to positive odd fundamentals, since this maximizes the number of partial sums, and the negative signs of the coefficients are implemented in the output adder taps of the filter. This works fine except in the unlikely case when all coefficients are negative. Then a sign complement operation has to be added to the filter output. In *Step 5*, all sums of two extended fundamentals are considered. It may happen that a final division is also required, i.e. $g=(2^n f_1 +/- 2^v f_2)/2^w$. Note that multiplication or division by power-of-two can be implemented by hardwired left and right shifts, respectively, without incurring additional hardware resources. For instance, the coefficient set {7,105,53} in MAG coding required 1, 2, and 3 adders, respectively. In RAG-n the set is synthesized as 7=8-1;105=7*15; 53=(105+1)/2, requiring only 3 adders in total but also a divide/right shift operation. In *Step 7* an adder cost-2 coefficient is added, and the algorithm selects a non-output fundamental (NOF) with the smallest values. This is motivated by the fact that an additional small NOF will generate more additional coefficient than a larger NOF at no cost. For example, assume that the coefficient 45 needs to be

1-4244-0387-1/06/$20.00 ©2006 IEEE

added and that an NOF value has to be decided upon. The NOF LUT lists all possible NOF values as 3, 5, 9, or 15. It can now be argued that if 3 is selected, more coefficients are generated than if any other NOF is used, since 3, 6, 12, 24, 48,… can be generated without additional effort from NOF 3. Other NOF values, such as 15, can generate 15, 30, 45,… etc., so the choices towards producing other coefficients at no cost are significantly restricted.

B. *RAG-2005 Modifications*

Compared with the original algorithms only minor improvements had been made over the years [4]:

- The MAG LUT table used has been extended to 14-bit using the concept of multiplicative, additive, and leapfrog graphs (see [8,9]). All 32-MAG adder cost-4 graphs are now considered when computing the minimum NOF sum. Within 14 bits, only 2 coefficients (i.e. 14709, 15573) are of cost-5. As long as these coefficients are not used, the computed minimum NOF sum list will be optimally based on RAG-95.

- In *Step 7* all cost-2 adder graphs are now considered. There are three such graphs, i.e., a single fundamental followed by an adder cost-2 factor, a sum of two fundamentals, and an adder cost-1 factor, or a sum of 3 fundamentals.

- The last improvement is based on the adder cost-2 selection that sometimes produced suboptimal results in the RAG-95 algorithm where multiple adder cost-2 coefficients have to be implemented. For instance, for the coefficient set {13, 59, 479} the minimum NOFs values used by RAG-95 are {3, 5, 7} because 13=4*3+1; 59=64-5; 479=59*8+7, giving a solution with 6 adders. If the NOF {15} is chosen instead, then all the coefficients (13=15-2; 59=15*4-1; 15*32-1) benefit, and RAG-05 requires only 4 adders, a 30% improvement. Therefore, instead of selecting the smallest NOF for the smallest adder cost-2 coefficient, a search for the best NOF is done over all adder cost-2 coefficients.

C. *RAG-2005 Benchmarks*

Although the RAG-n algorithm has been in use for quite some time, a large set of reliable benchmark data that can be verified and reproduced has not been produced. In a recent paper by Wang and Roy [6], for instance, 60% of the comparison RAG-n data were declared "unknown." A benchmark should cover filters used in practical applications that are widely published or can easily be computed. Less useful is a generation of "random" number filter coefficients that (a) cannot be verified by a third party, and (b) are of no practical relevance. The problem with the RAG-n benchmark data is that the heuristic part may give different results depending on the exact software implementation or the NOF table in use. In addition, since some filters are rather long, a benchmark that lists the whole RAG-n is not practical in most cases. It is therefore suggested that a benchmark based on the following equivalence transformation be used:

Theorem 1: Let S_1 be a coefficient set that can be synthesized by RAG-n with a set of F_1 output fundamentals and N_1 non-output fundamentals, (i.e., internal auxiliary coefficients). A congruent RAG-n is synthesized if a coefficient set S_2 is used that contains both output and non output fundamentals from set S_1.

The proof of Theorem 1 is based on the minimum number of adders of both sets. A corollary of Theorem 1 is that graphs can now be classified as (guaranteed) optimal and heuristic graphs. An optimal graph has no more than 1 NOF, while a heuristic graph has more than 1 NOF. It is only required to provide a list of the NOFs that describe a unique OF graph. If this set of NOFs is added to the coefficient set, all OFs are synthesized via the optimal part of the algorithm that can easily be programmed. This program will in fact be available for downloading from the author webpage as soon as possible. Some example benchmarks are given in Table I, where L is the filter length. Note that the number of CSD adders given has already taken advantage of coefficient symmetry, i.e. $f(k)=f(L-k)$. Common sub-expression (CSE) data are used from [6]. It can be seen that the examples from Samueli and Lim & Parker all produce optimal RAG-n results. Notice, particularly for long filters, the improvement of RAG-n compared to CSD and CSE adders. Filters F7 and F9 are from the Goodman & Carey [1] set of halfband filters and were improved using RAG-2005. The benchmark data from Samueli [2] and Lim & Parker [3] work very well for RAG-n since the filters are lowpass and therefore taper smoothly to zero at both sides, improving the likelihood of an adder cost-1 OF, which increases the likelihood of optimality.

TABLE I. REQUIRED NUMBER OF ADDERS FOR CSD, CSE, AND RAG ALGORITHMS FOR LOWPASS FILTERS

Filter	L	CSD	CSE	# OF	# NOF	RAG 95	RAG 05	NOF
F5	11	6	-	3	0	3	3	-
F6	11	9	-	4	1	5	5	3
F7	11	7	-	3	1	5	**4**	23
F8	15	10	-	5	2	7	7	11, 17
F9	19	14	-	5	4	11	**9**	13, 1261
S1	25	11	6	6	0	6	6	-
S2	60	57	29	26	0	26	26	-
L1	121	145	57	51	1	52	52	49
L2	63	49	23	22	0	22	22	-
L3	36	16	5	5	0	5	5	-

A more challenging benchmark is the filter required in the Rader DFT algorithm

$$X[g^k \bmod L] = x[0] + \sum_{n=0}^{L-2} x[g^n \bmod L] W_N^{g^{n+k \bmod (L-1)}} \quad (1)$$

where $W_N=\cos(2\pi kn/L)-j\sin(2\pi kn/L)$ and L is prime. Equation (1) converts a DFT computation into a filter operation. This filter does not have many small coefficients as the lowpass filter and is therefore more difficult for RAG-

n to be synthesized. The required $2L$-2 filter coefficients (without Rader permutation) are coded in the C programming language as: `f[k] = round((1 << B) * cos(2*M_PI*k/L));f[k+L-1]=-round((1 << B)*sin(2*M_PI*k/L));` For instance, with L=7 and B=14 the coefficients are: f={10215, -3646, -14761, -14761, -3646, 10215, -12810, -15973, -7109, 7109, 15973, 12810}. The set has 6 OFs, and 7 NOFs, or a total of 13 adders that are required for its implementation using RAG-n. Table II shows the data of Rader DFT filters [4] with 8 to 14 bits and 7 to 127 taps that use NOFs (see Theorem 1).

TABLE II. REQUIRED NUMBER OF ADDERS FOR CSD AND RAG ALGORITHMS FOR RADER DFT FILTERS

L	*B*	*CSD*	*# OF*	*# NOF*	*RAG 95*	*RAG05*	*NOF*
7	8	24	6	1	7	7	3
7	10	32	6	4	10	10	5,7,11,13
7	12	42	6	5	12	**11**	7, 15, 71, 103, 1841
7	14	52	6	7	13	13	7,11,31,59,101, 177,319
17	10	94	15	2	17	17	17,21
17	12	118	16	5	22	**21**	3,17,23,121,551
17	14	138	16	7	28	**23**	3,35,103,415,1153,1249,8051
31	12	206	29	3	33	**32**	3, 5, 123
31	14	244	30	8	42	**38**	3,9,133,797,877,975,1179,3235
61	12	402	60	1	61	61	5
61	14	496	60	6	71	**66**	5,39,51,205,265,3211
127	14	1060	125	1	126	126	5

III. DISTRIBUTED ARITHMETIC

Distributed Arithmetic (DA) is a different approach used to compute the sum-of-product compared with a standard MAC approach [4]. In the standard MAC approach, in each operation a coefficient and a data word are multiplied. In DA, however, one coefficient is multiplied with a single bit from *all* data words in one operation, i.e.,

$$y = \sum_{n=0}^{L-1} c[n]x[n] = \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{L-1} f(c[n], x_b[n]), \qquad (2)$$

where $x_b[n]$ describes the bit b of data word n, and the function $f(c[n],x_b[n])$ is usually implemented via a look-up table (LUT). One limitation of the techniques seems to be the LUT requirement that grows exponentially with the number of inputs, i.e., the number of taps. This limitation can be overcome by using table partitioning, i.e., an LUT table with $2N$ inputs can be split up into 2 LUTs with N inputs and an addition adder at the output. Both the LUT and adder tree in the case of more than 2 parallel LUT can be pipelined fully, providing maximum speed in an FPGA. The second limitation is that for L coefficients; L clock cycles are required for the computation of the SOP output. This limitation, however, can be overcome by implementing a LUT for each bit b of the input data word. These two modifications allow DA-based filters to run at high speed, and FPGA vendors in general prefer distributed arithmetic-(DA)-based filter generators since these designs are characterized by: (a) fully pipelined architecture, (b) a short compilation time, (c) good resource estimation, and (d) area independent results from the coefficient adder cost. DA-based filters do not require any coefficient optimization or the computation of a RAG-n graph that may be time consuming when the coefficient set is large. DA-based code generation, including all VHDL code and testbenches, is done in a few seconds using the vendors FIR compilers [7].

IV. FPGA IMPLEMENTATION RESULTS

With the NRE cost of cell-based ASICs reaching $4M for the 60nm process, FPGAs have become the dominating implementation vehicle for most DSP designs. The fully pipelined DA filters were generated using Altera's FIR filter compiler [7]. The circuits were then synthesized from their VHDL descriptions and optimized for speed using Quartus synthesis tools from Altera. The core element of an FPGA is a logic element (LE) that contains an LUT and a register. The LE can be configured as a 2^4×1 table (as used for DA) or in the arithmetic mode as two 2^3×1 tables to implement a full adder cell. The device used in this paper is an EP2C35F672C6ES, a popular device from the Cyclone II DSP development board. The device has 33216 LEs—much more than is actually needed for the largest filter design to ensure that the limitation in resources will not have a negative effect on the filter performance.

A. *Pipelining the RAG-n*

Due to the logic delay in the RAG-n running through several adders, the resulting register performance of the design is not very high even for a small graph. A single register placed at the output of the RAG already provides a speed improvement of 50% when compared with the non-pipelined design (210.22 MHz versus 148.39 MHz for F5). For the fully pipelined design for F5, one needs to build: x3<=2*x+x; x25<=x3*8 + x*z^{-1}; x75<=x25*2+x25, i.e., one extra pipeline register is used in building x25, and a maximum delay of 3 pipeline stages is needed. Fig. 1 shows the resulting F5 fully pipelined graph for F5 using MatLab/Simulink blocks. Notice that additional registers are introduced to implement the pipeline retiming, i.e., the multiplier outputs are aligned according to their pipeline stages. In this halfband filter design the pipeline retiming synthesis results reveal that the design now runs at 323 MHz, which is about the maximum performance that can be achieved with this device family.

B. *Variation of the Input Bitwidth*

When it comes to scaling of the design by data input bitwidth, the RAG-n approach area measured in LEs increases only linearly. The register performance decreases due to the overall longer adder delays of the design. In the DA design not only are twice as many LEs needed to implement the LUT, but additional adders are also required for the adder tree to achieve a high throughput. Overall it can be concluded from Table III that the fully pipelined RAG-n provides a cost gain between 78%-130% when the input bitwidth changes from 8 to 32 bits with about the same register performance as the DA filters.

TABLE III. COST VERSUS INPUT BITWIDTH

	RAG-n			Parallel DA			
Bin	**LEs**	F_{max} (MHz)	$\frac{LEs}{F_{max}}$	**LEs**	F_{max} MHz	$\frac{LEs}{F_{max}}$	**Gain%**
8	216	323	0.67	396	332	1.19	78.35
16	343	301	1.14	751	298	2.52	120.92
24	471	254	1.85	1086	261	4.15	124.41
32	599	227	2.63	1479	241	6.12	132.40

C. *FPGA Benchmark Data*

In order to create a reliable set of data, a larger set of filters has been designed using VHDL for fully pipelined RAG-n and using the FIR core compiler from Altera that implements a full parallel DA filter [7]. Table IV shows the results for 3 halfband filters from Goodman and Carey, two of the Samueli filters, and two of the Lim & Parker filters. It can be seen that the RAG-n filters on average enjoy size reductions of 71%, the registered performance of the DA filter is 8% better, and the overall cost expressed as LEs/Fmax is on average 56% better for RAG-n based designs when a fully pipeline approach is used. It can also be seen from Table IV that without pipelining (pipe=0) the DA-based approach gives better results. With 6% increase in area, the cost for RAG-n pipelining is quite reasonable.

V. CONCLUSION

This paper shows that RAG-n is a viable approach to the FPGA implementation of filters when the multiplier block is implemented with a fully pipelined reduced adder graph. Full pipelining increases the area by 6 %, the speed by 111% over the non-pipeline implementation, and the overall cost expressed as LEs/Fmax is improved by 93%. Compared with the DA-based designs this paper has demonstrated the substantial reduction in size, sustained registered performance resulting in overall better cost matrices.

Further studies will be directed towards a combination of DA and RAG-n methods, since RAG-n works best when many small coefficients are available, while DA offers greater advantages if there are many large coefficients that require many adders in the MAG coding.

TABLE IV. SIZE, SPEED, AND COST COMPARISON

Filter	**RAG-n: pipe 0/max**			**Parallel DA**		
	LEs	F_{max} (MHz)	$\frac{LEs}{F_{max}}$	**LEs**	F_{max} (MHz)	$\frac{LEs}{F_{max}}$
F5	196	148.4	1.32			
	216	323.3	0.67	396	332.3	1.19
F8	326	135.8	2.40			
	360	323.4	1.11	570	340.7	1.67
F9	461	97.2	4.74			
	534	304.0	1.76	717	326.2	2.20
S1	460	130.6	3.52			
	492	296.6	1.66	985	356.5	2.76
L3	651	205.3	3.17			
	671	310.3	2.16	1406	321.3	4.38
S2	1672	129.9	12.86			
	1745	252.9	6.90	2834	289.0	9.81
L2	1446	134.9	10.72			
	1531	265.5	5.77	2590	282.4	9.17
Mean0	745	140.3	5.53			
Mean	793	296.6	2.86	1357	321.2	4.45
Gain %	-6	111	93	71	-8	56

REFERENCES

[1] D.J. Goodman, M.J. Carey, "Nine Digital Filters for Decimation and Interpolation," IEEE Transactions on ASSP, pp. 121-126, April 1977.

[2] H. Samueli, "An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Powers-of-Two Coefficientss of," IEEE Transactions on C&S, vol. 36, pp. 1044–1047, July 1989.

[3] Y. Lim and S. Parker, "Discrete Coefficient FIR Digital Filter Design Based Upon an LMS Criteria," IEEE Transactions on C&S, vol. 36, pp. 723–739, October 1983.

[4] U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, 2rd ed., Heidelberg: Springer, 2004, pp.251–257.

[5] A. Dempster, M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," IEEE Transactions on C&S II, vol. 42, pp. 569-577, September 1995.

[6] Y. Wang, K. Roy, "CSDC: A New Complexity Reduction Technique for Multiplierless Implementation of Digital FIR Filters," IEEE Transactions on C&S I, vol. 52, pp. 1845-1852, September 2005.

[7] Altera Corp., "FIR Compiler: MegaCore Function User Guide" Ver. 3.1.0, June 2004.

[8] Dempster, A.G. and M.D. Macleod, "Constant integer multiplication using minimum adders", IEE Proceedings - Circuits, Devices & Systems, 141(5):407-413, October 1994.

[9] O. Gustafsson, A. Dempster and L Wanhammer, "Extended Results for Minimum-Adder Constant Integer Multipliers", Proc ISCAS 2002, Phoenix, pp. 73-76, May 2002.