

# CASCADED COEFFICIENT NUMBER SYSTEMS LEAD TO FIR FILTERS OF STRIKING COMPUTATIONAL EFFICIENCY

Jeffrey O. Coleman

Naval Research Laboratory  
 Washington DC  
 jeffc@alum.mit.edu

**Abstract** — Multiplierless FIR filters (or other fixed linear combiners) are built as add/subtract networks operating on bit-shifted input data. Classically, the computational structure required is determined by simply expressing the coefficients in canonical-signed-digit (CSD) form. In this paper, expressing coefficients in a higher-radix number system instead results in a computational structure for a partial solution, one that reduces a large linear-combination problem to a smaller one. A well-chosen sequence of such number systems then leads to a cascade of these problem-reducing networks that together solve the original problem with remarkable overall computational efficiency, especially for larger filters. An example FIR filter with a real chirp impulse response 3000 samples in length (a matched filter for a pulse-compression radar) was easily realized with  $-95$  dB rms approximation error using less than two add or subtract operations per coefficient. This is a reduction of approximately 60% relative to the usual CSD method.

## 1 INTRODUCTION

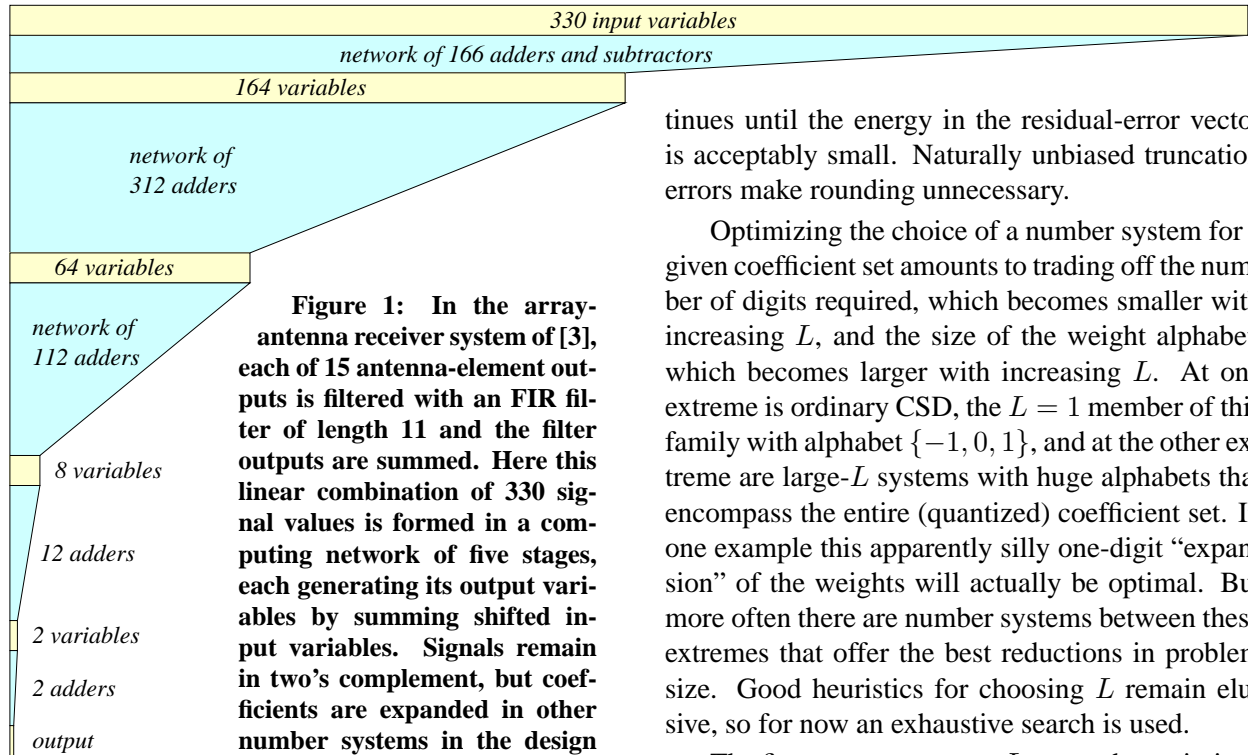
Fixed linear combiners in DSP systems, including FIR filters, can be straightforwardly configured as networks of adders and subtractors of two's complement input-data words subjected to sign-extending shifts of various numbers of bits. The design procedure is simple: Express coefficients in CSD, a radix-two number system with alphabet  $\{-1, 0, 1\}$ , to make the output a large linear combination of the same data set but now with each weight comprising a factor from this ternary alphabet and a power of (radix) two. These powers specify shifts, and the nonzero weights specify whether to add or subtract. It has long been known that using CSD in this way reduces the number of terms summed by approximately one-third relative to the number of terms summed when binary coefficient representations are used instead of CSD [1].

This work was supported by the Office of Naval Research.

This is easily generalized [2] to radically reduce the number of terms required by expressing coefficients in number systems having radices higher than the radix two of CSD. This reduction comes at the expense of having the weights on the terms summed drawn from a larger alphabet than CSD's simple  $\{-1, 0, 1\}$ . That expense is modest, because terms can be grouped by absolute value and the latter then factored out. Ultimately it's a three-line concept, given enough annotation:

$$\begin{aligned}
 y_n &= \sum_k \overbrace{h_k x(n-k)}^{\text{original linear combination}} \\
 &= \sum_k \sum_{\gamma} \overbrace{w_{k\gamma} r^{-\gamma} x(n-k)}^{\text{radix-}r \text{ expansion of } h_k} \\
 &\quad \uparrow \text{write as } |w_{k\gamma}| \text{sgn}(w_{k\gamma}) \text{ and} \\
 &\quad \text{group terms by } |w_{k\gamma}| \text{ value} \\
 &= \sum_{\substack{\text{a few } m \\ |w_{k\gamma}|=a_m}} \overbrace{a_m}^{\text{new linear combination}} \sum_{k,\gamma} \underbrace{\text{sgn}(w_{k\gamma})}_{\substack{\text{add,} \\ \text{subtract,} \\ \text{or ignore}}} \underbrace{r^{-\gamma} x(n-k)}_{\substack{\text{shift by} \\ \gamma \log_2(r) \text{ bits} \\ k\text{'th input}}}
 \end{aligned}$$

The variables combined in this new, smaller linear combination are created by shift/add/subtract computation networks operating on the original, larger set of inputs. The new linear combination has fewer input terms and combines them with simpler weights (small integers). This paper's central point is that *repeating this process using a sequence of number systems of decreasing alphabet size can reduce the problem in stages until trivial*. This approach can produce efficient structures, and the design computations required are practical even for large filters.



**Figure 1:** In the array-antenna receiver system of [3], each of 15 antenna-element outputs is filtered with an FIR filter of length 11 and the filter outputs are summed. Here this linear combination of 330 signal values is formed in a computing network of five stages, each generating its output variables by summing shifted input variables. Signals remain in two’s complement, but coefficients are expanded in other number systems in the design process (array example in text).

Figure 1 shows an example (to be discussed further) of such a multi-stage computational structure. Oddly, the design process is largely one of choosing, or even constructing, a sequence of number systems to best minimize the hardware resources ultimately required. Identifying the absolute best choice may be impossible, but it is easy to choose reasonable families of number systems based on general criteria and then to choose from those families actual number systems that are optimal for given coefficients. Here such “optimal” choices minimize the number of hardware adders and subtractors ultimately required. A metric more tuned to a particular design context to account properly for memory, interconnects, etc., could easily be substituted.

## 2 A DESIGN STRATEGY

Design examples in the next section use two new number-system families. The shift/add/subtract network for the first stage of each example is based on coefficient expansion in a radix  $2^L$  generalization [2] of CSD in which each shift is by  $L$  bits. As with CSD, these signed-digit systems represent positive and negative coefficients with equal ease and the richness of zero digits reduces the numbers of adders and subtractors. Parallel conversion of the coefficient vector begins at the MSB and con-

tinues until the energy in the residual-error vector is acceptably small. Naturally unbiased truncation errors make rounding unnecessary.

Optimizing the choice of a number system for a given coefficient set amounts to trading off the number of digits required, which becomes smaller with increasing  $L$ , and the size of the weight alphabet, which becomes larger with increasing  $L$ . At one extreme is ordinary CSD, the  $L = 1$  member of this family with alphabet  $\{-1, 0, 1\}$ , and at the other extreme are large- $L$  systems with huge alphabets that encompass the entire (quantized) coefficient set. In one example this apparently silly one-digit “expansion” of the weights will actually be optimal. But more often there are number systems between these extremes that offer the best reductions in problem size. Good heuristics for choosing  $L$  remain elusive, so for now an exhaustive search is used.

The first-stage parameter  $L$  cannot be optimized in isolation, however, because the alphabet elements actually appearing in the first-stage coefficient expansion become the weights expanded in designing the second stage, affecting then the optimal choice of number system to use there. This goes on, so that finally one must evaluate every sequence of number systems in some sequence universe and choose based on some metric. Here each candidate number system at stage  $M$  is evaluated recursively based on the total number of addition and subtraction operations required in stages  $M, M+1, \dots$  assuming that optimal number-system choices are made for stages  $M+1, M+2, \dots$ . The trivial optimal number system for the weight set  $\{1\}$  terminates the recursion. Amazingly, despite the brute-force (and memory-limited) recursive search, only the largest  $L$  tested for each example required more than a few seconds on a fairly ordinary laptop computer.

For the second and subsequent stages, the coefficients expanded are always positive integers, making the higher-radix CSD systems above unacceptable because when  $L > 1$ , they cannot represent some integers exactly with finitely many digits. (The original FIR-filter coefficients in stage one are typically infinite-precision reals.) So for the design here of the second and subsequent stages, simple number systems are constructed (next) that represent positive integers finitely and with CSD-like zero richness. They are as suboptimal here as they are expedient, but the results are impressive even so.

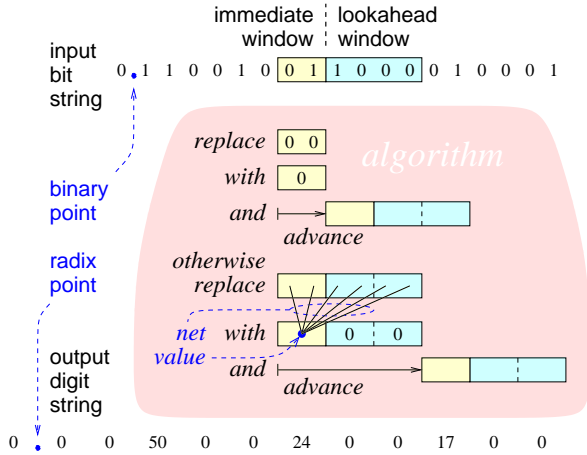


Figure 2: Two-digit-lookahead sliding-window conversion of an ordinary binary fraction to radix four.

## 2.1 Base $r$ with Lookahead

Conversion of binary integers to these new numbers systems, termed “base  $r$  with lookahead” or just  $base(r, L)$ , uses the sliding-window algorithm of Fig. 2. The bit string under conversion is at the top under the two windows whose widths parameterize the number system. If the width of the lookahead window is some multiple  $L > 0$  of the width of the immediate window, the output number system is said to have  $L$ -digit lookahead. Both window widths must be fixed, the immediate window cannot have zero width, and any nonzero lookahead  $L$  must be integral. The widths are otherwise arbitrary. Conversion begins with the left edge of the immediate window at the input binary point and  $L$  digits to the right of the output radix point.

If the lookahead window has zero width, the immediate window simply converts binary bit groups independently, in a memoryless way, so with widths of three or four, for example, it converts to octal or hexadecimal. The output radix  $r = 2^{\text{its width}}$ .

The lookahead window is fundamentally different. Whenever the contents of the immediate window are nonzero, the contents of the two windows are converted together. This forces every nonzero output digit to be followed by zero digits equal in number to the width of the lookahead window. This reduces the size of the eventual shift/add networks, but it also enlarges the alphabet of possible digit values in the output string.

As described this conversion yields only integers because the binary point is implicitly at the right of the lookahead window, effectively scaling the result by  $r^L$ . This is subsequently compensated in the software by offsetting the output radix point.

	chirp	array	LPF	$\mathcal{H}(\cdot)$	EQ
terms:	3000	330	100	14	14
$L = 1$	14863	1584	446	75	66
2	9823	1069	298	56	61
3	8281	828	256	58	64
4	5848	854	300	56	77
5	5832	878	355	53	67
6		775	290	51	75
7		901	359	60	86
8		1021	443	65	95
9		604	273	37	57
10		635	298	41	61
11				42	

Table 1: The total numbers of adders and subtractors for each example as a function of the radix  $2^L$  of the generalized CSD used for first-stage coefficients.

## 3 DESIGN EXPERIMENTS

In each of these five design examples, the first-stage coefficient expansion was terminated when the approximation error fell below  $-95$  dB rms.

**chirp:** The impulse response of this FIR filter of length 3000 is a Hamming-windowed cosine that chirps in  $100 \mu\text{s}$  over a 10 MHz range centered at  $3/4$  of a 40 MHz sample rate. (This is a completely made-up radar-receiver pulse-compression filter.)

**array:** This complex linear-phase vector FIR filter (see Fig. 1) synthesizes a wideband array pattern from a vector of complex-baseband  $(I + jQ)$  inputs.

**LPF** This nonlinear-phase complex lowpass FIR filter of length 50 is the example from [4] in which the stopband was minimized in the  $L_1$  norm.

**$\mathcal{H}(\cdot)$ :** This length-14 Hilbert filter is the imaginary part of the “complex image-suppression filter” in [5].

**EQ:** This length-14 nonlinear-phase real FIR equalization filter is from [5].

Operation counts are tallied in Table 1 versus the number system used for first-stage coefficients, with later stages optimized. The best solutions are boxed in Table 1 and detailed in Table 2. For complex filters **array** and **LPF** the totals are for computing the real or the imaginary output component but not both.

stage	inputs	system	ops	outputs
1	3000	csd(32)	4826	496
2	496	base(2,4)	976	16
3	16	base(2,3)	16	8
4	8	base(2,1)	12	2
5	2	binary	2	1
<b>chirp</b>	saved 60% over CSD		<b>5832</b>	total
1	330	csd(512)	166	164
2	164	base(2,6)	312	64
3	64	base(2,3)	112	8
4	8	base(2,1)	12	2
5	2	binary	2	1
<b>array</b>	saved 62% over CSD		<b>604</b>	total
1	100	csd(8)	198	28
2	28	base(2,3)	44	8
3	8	base(2,1)	12	2
4	2	binary	2	1
<b>LPF</b>	saved 43% over CSD		<b>256</b>	total
1	14	csd(512)	7	7
2	7	base(2,7)	1	14
3	14	base(2,3)	15	8
4	8	base(2,1)	12	2
5	2	binary	2	1
$\mathcal{H}(\cdot)$	saved 51% over CSD		<b>37</b>	total
1	14	csd(512)	0	14
2	14	base(2,7)	9	23
3	23	base(2,3)	34	8
4	8	base(2,1)	12	2
5	2	binary	2	1
<b>EQ</b>	saved 14% over CSD		<b>57</b>	total

**Table 2: The best solution found for each example.**

The tables contain interesting phenomenology. Local minima abound in Table 1, and vaguely periodic tendencies versus  $L$  suggest some relationship with the rms-error conversion-termination threshold. (This needs exploring.) In Table 2 the savings relative to CSD generally increases with an increasing number of inputs but with extra savings in the **array** and  $\mathcal{H}(\cdot)$  examples due to their linear-phase symmetry. The outcome for smaller filters seems less predictable. The **EQ** design is the most extreme. Its first stage requires no operations and fails to reduce the problem size! The first-stage coefficients were simply quantized, and all but one happened to be positive. Because the one  $-1$  coefficient can be pushed into the next stage and so requires no extra hardware, it was not counted as an operation.

## 4 CONCLUSIONS

The present work is preliminary, and easy further performance gains appear available through better choices of number-system families within which to optimize. Even so, for large FIR filters the design approach presented here already offers savings in computational structure on the same order as those gained through traditional approaches that identify and eliminate common subexpressions. This should be no surprise, because when examined carefully the present technique is seen to achieve its gains by factoring out common subexpressions (this will be elaborated further in subsequent papers).

The primary advantage then of the present multi-stage number-system-based method over traditional algorithms that optimize shift/add/subtract trees (e.g. [6]) is simplicity. This approach of this paper is easy to program—only an afternoon of matlab work was required—and runs so economically that very good solutions are easily found for large filters, such as the 3000-tap **chirp** example above, that would computationally overwhelm algorithms based on expression trees, pattern matching, etc. It may yet turn out that such algorithms are best used in a follow-on way to fine-tune solutions created using number systems. Time will tell.

## REFERENCES

- [1] H. L. Garner, “Number systems and arithmetic,” *Advanced Computers*, vol. 6, pp. 131–194, 1965.
- [2] J. O. Coleman, “Express Coefficients in 13-ary, Radix-4 CSD to Create Computationally Efficient Multiplierless FIR Filters,” in *Proc. European Conf. on Circuit Theory and Design*, Sept. 2001.
- [3] D. P. Scholnik and J. O. Coleman, “Formulating wideband array-pattern optimizations,” in *Proc. IEEE Int’l Symp. Phased Array Systems and Technology*, May 2000.
- [4] J. O. Coleman and D. P. Scholnik, “Design of nonlinear-phase FIR filters with second-order cone programming,” in *Proc. Midwest Symp. on Circuits and Systems*, Aug. 1999.
- [5] J. O. Coleman, J. J. Alter, and D. P. Scholnik, “FPGA architecture for Gigahertz-sampling wideband IF-to-baseband conversion,” in *Proc. Int’l Conf. on Signal Processing Applications and Technology (ICSPAT 2000)*, Oct. 2000.
- [6] A. Yurdakul and G. Dündar, “Multiplierless realization of linear DSP transforms by using common two-term expressions,” *J. VLSI Signal Processing* 22, pp. 163–172, 1999.