Efficient Approximate Wordlength Optimization

Paul D. Fiore, Member, IEEE

Abstract—In this paper, the problem of finding good wordlength combinations for fixed-point digital signal processing flowgraphs is addressed. By formulating and solving an approximate optimization problem, an estimated Pareto-optimal curve for attainable cost/quality combinations is rapidly calculated. This curve and the associated wordlength combinations are useful in several situations and can serve as starting points for real design searches. Examples that illustrate these concepts are given.

Index Terms—Wordlength optimization, Pareto optimization, DSP implementation.

1 INTRODUCTION

There is continuing interest in implementing digital signal processing (DSP) algorithms in very large-scale integration (VLSI) devices and field-programmable gate arrays (FPGAs) [1], [2], [3], [4], [5], [6], [7], with applications ranging from compression and encryption for cellular phones, wireless networks, digital multimedia devices, filtering, sophisticated hearing aids, template matching [8], computer vision [9], [10], space systems [11], military applications such as digital receivers [12], and efficient emitter detection and location [13]. While it is true that the amount of logic available in these devices is constantly increasing, it is still of interest to minimize the amount of logic required to implement an algorithm. This is particularly true when low power dissipation is important.

A straightforward way to reduce hardware is to round or truncate data at various points throughout an algorithm. Many operations such as multiplication or accumulation can greatly expand the wordlength of the data. If these operations are cascaded without some form of wordlength reduction, excessive hardware growth results.

Wordlength reduction introduces noise into the data stream, so the designer must balance the need for an efficient implementation with output quality. To perform this tradeoff, the effect of rounding is modeled as the injection of additive uncorrelated white uniform noise [14], [15].

In this paper, we address the problem of finding good wordlength combinations for directed acyclic single-output fixed-point DSP flowgraphs using an approach initially developed in [9]. Let the DSP flowgraph be represented by

\[ y = f(z), \]

where \( y \) is the output of the flowgraph, \( z \) is the vector of inputs to the flowgraph, and \( f() \) is the given DSP function to be implemented by the flowgraph.

Each arc \( i = 1, \ldots, N \) of the flowgraph will have a wordlength of \( b_i \) bits associated with it. We desire to find positive integer vectors \( b = [b_1 \ldots b_N]^T \) of wordlengths that have both a good value of a given cost function \( C(b) \) and a given performance function \( V(b) \). To simplify the optimization process, the positive integer constraint will be relaxed, allowing the \( b_i \)'s to be real numbers. Then, a curve in the \((C(b), V(b))\) plane that is close to optimal will be traced.

Even though this curve, an approximate Pareto-optimal bound, is derived with the integrality constraint relaxed, it is still useful in that it provides information on the range of feasible trade-offs for the flowgraph in a rapid manner. Other nonconstructive bounds in signal processing, such as the Cramer-Rao bound, have proven extremely useful in understanding the limits of attainable performance. It is hoped that the bounding approach described here will similarly provide insight into achievable performance, before laborious simulations are undertaken. As a practical matter, designs obtained by rounding the elements of \( b \) on the bounding curve will usually be close in performance to the true optimal bound. These can provide good starting points for search procedures for real designs.

The problem of good wordlength design has of course received research attention. Fiore [9] utilized a Markov Chain Monte Carlo approach and simulated annealing to search for good wordlength combinations. Other searching approaches with different strategies are given by Kum and Sung [5], [16], Han and Evans [1], Lee and Villasenor [17], and Cantin [18]. Constantinides et al. [19] formulate and solve for optimal wordlengths using a mixed-integer linear programming formulation. Various authors, including Herve et al. [2], Chan and Tsui [20], and Fiore and Lee [21], consider constrained approaches, where either cost or accuracy is constrained, and the other metric is optimized.

A modern class of architectures, known as “reconfigurable” or “adaptive” computers [8], [11], [22], [23], [24], [25], [26], allows the design to change dynamically, as a function of the data. The current research on these devices is focused on the architectural trade-offs involved in terms of the number of gates, context switching speed, update methodology, and combinations of FPGA and general-purpose computing resources. These new architectures bring the possibility of adapting the design with no predesigned library at all. A usable control law for the wordlengths \( b \) as a function of the data is needed in this situation. The iteration described in this paper could form the basis of one such control law.
Conventional usage of FPGAs for reconfigurable computing focuses on loading different hardware contexts at different points in time. Generally, it is assumed that the contexts are entirely predesigned, with the appropriate design being loaded under the control of some master processor. A theoretical lower bounding curve can be useful in this situation, because it can help identify useful library elements.

For simplicity, the cost function $C(b)$ is assumed to be the area of the design, measured appropriately to the given technology. For example, in a custom VLSI device, gates are perhaps the appropriate measure of area, while for FPGAs, configurable logic blocks are appropriate. Various implementation issues such as bit-serial processing and pipelining are not being considered. This removes execution time from the optimization problem. Of course, the results derived here can be extended to the more general case.

The remainder of this paper is organized as follows: In Section 2, the mathematical form of the performance function $V(b)$ is determined. In Section 3, the set of optimal designs that define the theoretical bound is detailed. In Section 4, an iterative minimization approach for finding the bound is presented, along with a detailed discussion of the convergence issues. These concepts are illustrated with design examples in Section 5. Finally, Section 6 offers a summary and conclusions.

2 DETERMINATION OF THE PERFORMANCE FUNCTION

For simplicity, the performance function $V(b)$ will be taken to be the noise mean-square error (MSE) at the output of the flowgraph due to internal wordlength reduction. We focus only on output noise MSE, because this is the primary concern of the algorithm designer. Note that other performance metrics such as power dissipation have been under intense scrutiny, [27], [28], [29]. Power dissipation is not addressed in this paper; it is merely treated as another constraint that has been relaxed to derive the theoretical bound.

In order to determine the rounding noise MSE $V(b)$, classical results from numerical analysis [30] are used, and Fig. 1 illustrates the approach. Let the vector $x$ represent both the primary inputs $z$ to the algorithm and the additional “inputs” created by the injection of additive noise; the flowgraph output is now represented as $y = f(x)$. The change in the output $y$ as a function of a change in the inputs $x$ is approximately calculated by expanding $f(x)$ in a Taylor series and keeping only the first-order terms [3], [9], [31]:

$$
\Delta y = \sum_{i=1}^{N} \frac{\partial f(x)}{\partial x_i} \Delta x_i.
$$

This shows that the partial derivative of a flowgraph arc acts as a “transfer function,” which determines the contribution of noise to the output of the calculation. Since the partial derivatives vary depending on the values of the primary inputs, the transfer function changes, depending on the particular operating point.

![Fig. 1. Numerical analysis approach. (a) Flowgraph $y = f(x)$ with internal arc $i$, numerical value $x_i$, and its wordlength-reduced version $<x_i>$ shown. The remaining data paths from the inputs $z$ to $y$ create a transfer function from arc $i$ to $y$ (shown as a dashed line). (b) Approximate model for numerical sensitivity analysis. Uncorrelated uniform noise $e_i$ is added to $x_i$, followed by multiplication by the transfer function $f_i$.](image)

A question now arises about how best to combine results of the partial derivatives over the range of possible inputs $z$. As shown in the Appendix, for the linearized model, the optimal combination of the partial derivatives is to take the root-mean-square (RMS) combination of the partial derivatives for a representative set of inputs $z$. That is, for each injection point, obtain the transfer function from the injection point to the output, while maintaining all the auxiliary paths from the other primary inputs, and then combine these via RMS.

By the above discussion, the gain coefficients are

$$
\sigma_i = \text{RMS} \left( \frac{\partial f(x)}{\partial x_i} \right).
$$

Elementary probability gives the output noise MSE as

$$
\text{MSE}_y = \sum_{i=1}^{N} \sigma_i^2 \sigma_i^2,
$$

where $\sigma_i^2$ is the variance of zero-mean rounding noise $e_i$ at an input $x_i$. If arc $i$ has the maximum absolute value $M_i$, the arc is first rounded up to the next power of two to prevent overflows:

$$
M_i = 2^\left(\text{log}_2(M_i)\right).
$$

Now, rounding arc $i$ to $b_i$ bits introduces uniform noise with approximate range $[-M_i 2^{-b_i}, M_i 2^{-b_i}]$. The variance of this noise is [14]

$$
\sigma_i^2 = \frac{1}{12} M_i^2 2^{-2b_i},
$$

(note that a more accurate model is given in [15]). We therefore have

$$
V(b) = \sum_{i=1}^{N} a_i 2^{-2b_i},
$$

where $a_i$ is a constant that depends on $x_i$ and the way it is rounded. By injecting additive noise at the inputs $z$, we can define the partial derivative $\partial f(x)/\partial x_i$ with respect to the injection point $z_i$ and take the RMS of these partial derivative estimates. This is the effective number of bits (ENOB) $a_i$.
with

\[ a_i \triangleq s_i^2 \hat{M}_i^2 / 12. \]  

(7)

Thus, \( a_i \) determines the effect that wordlength reduction at node \( i \) will have on algorithm performance; the \( a_i \)'s are called the “noise injection scale factors” and are collected into a vector \( \mathbf{a} \triangleq [a_1, \ldots, a_N]^T \). The form of \( V(b) \) in (6) shall be used to find optimal designs, although there are a few points to consider:

- The ranges \( M_i \) and the RMS slopes \( s_i \) must be determined in order to calculate \( a_i \). In simple cases, these values can be hand calculated. However, for complicated flowgraphs, this becomes tedious. One therefore may need to resort to search-based methods [12], profiling [7], [16], automatic differentiation [31], [32], [33], or range propagation [34], [35] to approximately determine these values.
- The assumption of uncorrelatedness of the rounding noise inputs in (6) is a coarse approximation. Signal correlation during numerical simulations can cause the actual MSE to be either above or below \( V(b) \).
- The approach described above relies on the gradients of the flowgraph function. Functions with points where the slope is infinite will thus present a problem for this method. In this situation, two approaches may be taken. The first is to simply bound the independent variables away from the singular point. The second approach is to use an upper bound on the wordlength that may be known a priori.
- If truncation rather than rounding is used, the variance (5) is unaltered, but there can be a bias introduced in the calculation; [36] treats this case. A different wordlength reduction scheme given in [37] can be used to achieve zero bias and the variance given in (5), without the additional hardware cost of rounding.

3 Pareto-Optimal Designs

In this section, wordlength combinations that attempt to simultaneously minimize both \( C(b) \) and \( V(b) \) metrics are investigated. Consider a known cost function \( C(b) \) and a known MSE function \( V(b) \). For any wordlength combination \( b \), one could calculate and plot \( C(b) \) versus \( V(b) \). The combinations of interest are those on the boundary of the set of data points closest to the coordinate axes.

These combinations of interest are known as “Pareto-optimal” points [38]. A point is Pareto-optimal if no other point has both a lower cost and a lower MSE. This curve provides designer flexibility in choosing designs, when one is not quite sure how much cost can be tolerated or how much output noise can be tolerated. Given the lower bounding curve, if one finally is also given a hard cost constraint, then the optimal design (i.e., the minimum MSE subject to not exceeding the cost constraint) is easily determined. Another situation where all the lower bounding points are needed is when one is populating a design library.

If the integer constraint on the elements of \( b \) is relaxed, a continuous lower bounding curve results (as long as \( C(b) \) and \( V(b) \) are continuous). The lower bounding curve on the Pareto-optimal solutions can be found by considering convex combinations of the \( C(b) \) and \( V(b) \):

\[ l(b; \alpha) \triangleq C(b) + \alpha V(b), \quad \alpha \geq 0. \]  

(8)

If this is minimized with respect to \( b \)

\[ b^* = \arg \min_b l(b; \alpha), \]

(9)

then \( b^* \) is a Pareto-optimal solution. To see this, assume that the solution \( b^* \) was not Pareto optimal. Then, by definition, there exists a \( b \) such that both \( C(b) < C(b^*) \) and \( V(b) < V(b^*) \) are true. Multiplying the second inequality by \( \alpha \) and adding to the first inequality gives \( l(b; \alpha) < l(b^*; \alpha) \), which contradicts (9).

Thus, (9) may be used to derive a lower bounding curve on the Pareto-optimal solutions. We first relax the integrality constraint on \( b \). Then, \( \alpha \) is initialized to a large number, \( b^* \) is found via (9), and \( C(b^*) \) is plotted against \( V(b^*) \). Next, \( \alpha \) is decreased slightly, and the procedure is repeated. Each minimization is initiated by the previous value of \( b^* \) to speed up the optimization.

The MSE \( V(b^*) \) using the model (6) at the relaxed optimum \( b^* \) can be related statistically to the MSE \( V(b) \) for integer \( b \), derived by rounding the elements of \( b^* \). Let \( b = b^* + r \), where elements \( r_i \) are independent uniform random variables such that \( r_i \in [-1/2, 1/2] \). Under this model, it is not difficult to show that the expected MSE at the rounded wordlengths is given by \( E[V(b)] = 1.08202 \cdot V(b^*) \), which shows that on the average, the relaxed MSE will remain close to the MSE at rounded integer wordlengths.

Blanket statements for the change in cost in rounding relaxed solutions \( b^* \) are more difficult to make, because the form of the cost function \( C(b) \) is much less constrained than that of the MSE function. However, in the special case of a quadratic cost function \( C(b) = b^T C b + d^T b \) (which is treated in more detail in Section 4.3), it is straightforward to show that \( E[C(b)] = C(b^*) + \text{tr}(C)/12 \). Thus, the expected relative change in cost is small if the wordlengths \( b \) are large enough.

A typical approach to performing the optimization in (9) is to employ Newton’s method [38], giving an iteration

\[ b^{k+1} = b^k - \left( \nabla^2 l(b^k; \alpha) \right)^{-1} \nabla l(b^k; \alpha). \]  

(10)

To use Newton’s method, one must construct and evaluate the Hessian of the weighted sum of the cost function and the variance function during the iterations. Also, the Hessian must be inverted (or an equivalent linear system solved) at each iteration, requiring an \( O(N^3) \) process. Another issue is that the Hessian might not even exist, might not be positive definite over all \( b \), or might be inconvenient to evaluate.

In contrast, the iteration given in the next section does not require the Hessian at all, only gradient information, which could easily be generated numerically if required.
4 Minimization Algorithm and Convergence Results

For a fixed α, the desired minimization is

$$\mathbf{b}^* = \arg \min_{\mathbf{b}} C(\mathbf{b}) + \alpha V(\mathbf{b}),$$  \hspace{1cm} (11)

where the integer constraints on \( \mathbf{b} \) are relaxed. Taking gradients and setting them to be equal to zero gives

$$\nabla C(\mathbf{b}) + \alpha \nabla V(\mathbf{b}) = 0.$$  \hspace{1cm} (12)

Using (6) (the definition of \( V(\mathbf{b}) \)) and rearranging (12) gives

$$a_i 2^{-2b_i} (-2 \ln 2) = -\frac{1}{\alpha} \nabla C(\mathbf{b})_i, \quad i = 1, \ldots, N.$$  \hspace{1cm} (13)

Taking logarithms and rearranging gives

$$b_i = -\frac{1}{2} \log_2 \frac{\nabla C(\mathbf{b})_i}{2a_i \ln 2}, \quad i = 1, \ldots, N.$$  \hspace{1cm} (14)

This is a set of nonlinear equations for the optimal \( \mathbf{b} \). To actually solve them, define an iteration [9]

$$b_i^{(k+1)} = -\frac{1}{2} \log_2 \frac{\nabla C(\mathbf{b})^{(k)}_i}{2a_i \ln 2}, \quad i = 1, \ldots, N,$$  \hspace{1cm} (15)

where \( k \) is the iteration index. These equations make the assumption that the elements of \( \nabla C(\mathbf{b}) \) are positive. It is easiest to start the iterations with a large \( \alpha \), which then decreases; the iterations should be stopped when the wordlengths become very short (i.e., a few bits).

As a practical matter, most operations have a maximum number of output bits, determined by the number of input bits (i.e., a multiplier with \( n \) input bits can produce at most \( n \) + 1 output bits). During the iteration, the number of bits at the output of functional blocks should not be allowed to grow to more than what the operation can produce.

Whenever a nonlinear iterative algorithm is proposed, one must be concerned with convergence issues. The following sections discuss the convergence of iteration (15) in detail. As will be shown, the main object of study in convergence analysis is the Jacobian matrix of the iteration. Section 4.1 defines the Jacobian matrix, gives its key property, and presents an expression for the Jacobian matrix of the iteration (15). Section 4.2 gives an easily calculable metric based on the Jacobian matrix that can be monitored to assure that the iteration is converging. Finally, Section 4.3 specializes this metric to the important case of quadratic cost functions, giving a very simple sufficient condition for convergence.

4.1 Convergence Rate

The convergence of an iteration of the form \( \mathbf{w}^{(k+1)} = \mathbf{h}(\mathbf{w}^{(k)}) \) to the solution \( \mathbf{w} = \mathbf{h}^*(\mathbf{w}) \) is governed by the Jacobian matrix \( \mathbf{J} = \partial \mathbf{h}/\partial \mathbf{w}^T \) [39]. The iteration will converge if the largest magnitude eigenvalue \( |\lambda| \) of \( \mathbf{J} \) is less than one around the equilibrium point \( \mathbf{w}^* \). It is easily shown that the Jacobian matrix of the iteration (15) is

$$J_{ij} = \frac{\partial h_i(\mathbf{b})}{\partial b_j} = -\frac{1}{(2 \ln 2)} \frac{\partial \nabla C(\mathbf{b})_i}{\partial b_j} \frac{1}{\nabla C(\mathbf{b})_i},$$  \hspace{1cm} (16)

for \( i, j = 1, \ldots, N \).

A few comments are in order:

- For most flowgraphs, \( \mathbf{J} \) is a sparse matrix. This is because two arcs \( i \) and \( j \) can only appear together in a term of \( C(\mathbf{b}) \) if the two arcs are connected to a common functional block. For \( i \) and \( j \) such that the two arcs are not connected to a common functional block, \( \nabla C(\mathbf{b})_i \) does not depend on \( j \), so that \( \partial \nabla C(\mathbf{b})_i/\partial b_j = 0 \), and hence, \( J_{ij} = 0 \).
- It is interesting to note that \( \mathbf{J} \) does not depend on either the noise injection scale factors \( a_i \) or the particular value of \( \alpha \).
- For cost functions \( C(\mathbf{b}) \) that are separable in that they can be written as a sum of terms, each of which only depends on one variable, (16) gives that \( \mathbf{J} \) is diagonal. This really means that we can independently solve for each component of \( \mathbf{b} \) in (14). For the special case of a linear cost function \( C(\mathbf{b}) = \sum_{i=1}^{N} c_i b_i \), one can actually get a closed-form expression for the optimal \( \mathbf{b} \) from (15):

$$b_i = -\frac{1}{2} \log_2 \frac{c_i}{2a_i \ln 2}, \quad i = 1, \ldots, N,$$  \hspace{1cm} (17)

where \( a_i \) is given by (7), and \( \alpha \) is the relative weighting used in (11). A similar result was first derived in [21], where the \( \alpha \) played the role of a Lagrange multiplier in the constrained optimization problem: minimize \( C(\mathbf{b}) \) subject to \( V(\mathbf{b}) = A \). This was also rederived in [20].

4.2 Eigenvalue Upper Bound

If it is desired to monitor the iterations when solving (11) using the iteration (15), one could calculate \( \mathbf{J} \) using (16) and then find the largest eigenvalue magnitude at every step. Since this might be rather costly, a more efficient alternative is to make use of the inequality [40]

$$|\lambda| \leq \|\mathbf{J}\|_2^2 \leq \|\mathbf{J}\|_1 \|\mathbf{J}\|_\infty,$$  \hspace{1cm} (18)

where \( \|\mathbf{J}\|_p \) is the induced \( p \)-norm of a matrix \( \mathbf{J} \). In particular,

$$\|\mathbf{J}\|_1 = \max_j \sum_{i=1}^{N} |J_{i,j}| \text{ and } \|\mathbf{J}\|_\infty = \max_i \sum_{j=1}^{N} |J_{i,j}|,$$  \hspace{1cm} (19)

and \( \|\mathbf{J}\|_2 \) is equal to the largest singular value of \( \mathbf{J} \). For a sufficient convergence condition, simply calculate the product \( \|\mathbf{J}\|_1 \|\mathbf{J}\|_\infty \). If the product is less than one, then the iteration (15) is stable. If the product is greater than one, it may indicate that the iteration is unstable. In this case, a more careful calculation of \( |\lambda| \) is required. For sparse \( \mathbf{J} \), the calculation of (19) will be particularly efficient, requiring only absolute values and additions.

4.3 Quadratic Cost Function

Cost functions that are weighted sums of products of pairs of wordlengths are termed quadratic cost functions and written as \( C(\mathbf{b}) = \mathbf{b}^T \mathbf{C} \mathbf{b} + \mathbf{d}^T \mathbf{b} \). This type of cost function arises quite often in DSP system implementation, by virtue of the fact that most operations have two inputs. For example, two-input parallel multiplier structures tend to dominate the resource requirements of filters, filter banks, and fast-Fourier-transform engines. A reasonable approximation to the area complexity of a fixed-point arithmetic \( b_x \times b_y \)-bit multiplier is \( \gamma b_x b_y \), where \( \gamma \) is a scale factor (for example, a
PEZARIS ARRAY MULTIPLIER. Of course, more asymptotically efficient structures are known, but these generally are not implemented in FPGA hardware, due to the irregular layout and routing. Later, in Section 5, the area of a Pezaris array with output truncated to $b_i$ bits shall be taken to be $\min(b_i b_j b_k b_l)$, which can be understood by noting that a small $b_i$ has the effect of shaving columns off the least significant bit end of the array. This cost function is locally quadratic, so results in this section apply to this piecewise quadratic cost model. Note that this cost function has many regions where an element of the gradient is equal to zero.

A reasonable assumption is that all elements $C_{ij}$ of the matrix $C$ are greater than or equal to zero. For this case, as is shown in [9], an upper bound on the maximum eigenvalue for the iteration for quadratic cost functions is given by

$$|\lambda_i| \leq \frac{1}{(2 \ln 2) \min(b)} \left( \max_j \sum_{i=1}^{N} \left( \frac{C_{ij}}{\sum_{k=1}^{N} C_{ik}} \right) \right)^{1/2}. \quad (20)$$

While complicated looking, this is in fact much less costly to evaluate than (18) since the expression under the square root in (20) may be precalculated.

A more convenient but looser bound is also derived in [9] as

$$|\lambda_i| \leq \frac{\sqrt{N}}{(2 \ln 2) \min(b)}, \quad (21)$$

and we thus obtain a sufficient condition for stability for a quadratic cost function:

$$\min(b) \geq \frac{\sqrt{N}}{2 \ln 2}. \quad (22)$$

This sufficient condition holds for any flowgraph with a quadratic cost function. For example, with $N = 64$, all $b_i \geq 6$ is a sufficient condition for the iteration to be converging.

5 USAGE EXAMPLES

Next, three examples illustrate aspects of the preceding algorithm. The first example is a direct application of the algorithm to a small nonlinear function. The second example is a set of larger flowgraphs used to illustrate the efficiency of the algorithm and the dependence on representative inputs. The third example is a small flowgraph with feedback; to utilize the algorithm in this case, the feedback loop must be disconnected (causing an increase in the modeling error).

5.1 Polynomial Evaluation

In this section, the algorithm in Section 4 is applied to the problem of polynomial evaluation via Horner’s rule, i.e., $y = \cdots \left((c_n z + c_{n-1}) z + \cdots + c_0\right).$ Fig. 2 shows a flowgraph of the proposed calculation. Note that we are not studying the approximation error due to inexact coefficients in this model.

We will treat the example of approximating one cycle of the sine function. Specifically, a minimum-MSE fifth-degree polynomial approximation to $y = \sin(2\pi z)$ over the interval $z \in [0, 1]$ has coefficients given by $c_0 = 0.0137,$

$$1.000 \begin{array}{l} 18.432 \end{array} \begin{array}{l} 28.313 \end{array}$$

$$55.250 \begin{array}{l} 0.334 \end{array} \begin{array}{l} 37.991 \end{array}$$

$$138.062 \begin{array}{l} 0.334 \end{array} \begin{array}{l} 607.853 \end{array}$$

$$1.000 \begin{array}{l} 34.005 \end{array} \begin{array}{l} 96.350 \end{array}$$

$$82.812 \begin{array}{l} 0.378 \end{array} \begin{array}{l} 195.286 \end{array}$$

$$99.446 \begin{array}{l} 0.378 \end{array} \begin{array}{l} 195.286 \end{array}$$

$$1.000 \begin{array}{l} 11.242 \end{array} \begin{array}{l} 10.531 \end{array}$$

$$M = \begin{array}{l} 22.151 \end{array} \begin{array}{l} 0.447 \end{array} \begin{array}{l} 17.079 \end{array}$$

$$11.174 \begin{array}{l} 0.447 \end{array} \begin{array}{l} 4.270 \end{array}$$

$$1.000 \begin{array}{l} 4.865 \end{array} \begin{array}{l} 1.972 \end{array}$$

$$6.824 \begin{array}{l} 0.577 \end{array} \begin{array}{l} 1.778 \end{array}$$

$$5.786 \begin{array}{l} 0.577 \end{array} \begin{array}{l} 1.778 \end{array}$$

$$1.000 \begin{array}{l} 2.952 \end{array} \begin{array}{l} 0.726 \end{array}$$

$$1.014 \begin{array}{l} 1.000 \end{array} \begin{array}{l} 0.333 \end{array}$$

$$1.004 \begin{array}{l} 1.000 \end{array} \begin{array}{l} 0.333 \end{array}$$

1. Reference [9] proved the eigenvalue bounds for $C(b) = b^T C b$. When the linear term $d^T b$ with $d_i \geq 0$ is added to the cost, the key inequalities [9, eq. (3.31) and eq. (3.34)] still hold.
As a first demonstration of the utility of the new iteration, Fig. 3 compares the iteration (15) to a steepest descent approach. \( C_{11} = 10^{-6} \) was chosen, the steepest descent step size was set to \( C_{22} = 3 \), and both the steepest descent and the new iteration were initialized at \( b_i = 32 \) bits. With the steepest descent method, increasing the step size \( C_{22} \) led to unstable iterations. In the figure, (8) versus iteration number is plotted for both methods. The figure shows that the new iteration converged very rapidly compared to the steepest descent method.

Next, the iteration (15) was performed for different values of \( C_{11} \), and the resulting lower bounding cost/MSE curve is shown in Fig. 4. At each \( C_{11} \), the iteration (15) was initialized at the \( b \) calculated for the previous \( C_{11} \). To generate the figure, only two iterations of (15) per \( C_{11} \) were used. Note that nearly identical results were obtained when only one iteration per \( C_{11} \) was used.

The lower bound for this example was particularly tight; by rounding the noninteger wordlengths of the lower bound designs, the “+” symbols shown in Fig. 4 were obtained. Deriving optimal integer-constrained wordlength designs is an exponential-time problem, whether integer programming techniques are used, or perhaps naive random guessing techniques. In fact, also plotted in Fig. 4 are the results of 1,000 randomly guessed integer wordlength designs (denoted by the “×” symbols). Clearly, many guesses would be required to obtain a reasonable approximation to the lower bound. In contrast, rounding the lower bound curve wordlength is extremely fast and can give very good results.

Also plotted as “×” in Fig. 4 are the results of numerical simulations using the rounded wordlengths. The results closely match the Pareto-optimal curve derived using the relaxed noninteger wordlengths. Fig. 5 shows elements of the resulting noninteger wordlengths \( b \) as the algorithm was performed. Due to the form of the cost function (23), the adder output wordlengths remained fixed for this example. Fig. 6 shows the maximum absolute eigenvalue of \( J \), as well as the upper bounds given by (18), (20), and (21). Note that for this example, the sufficient condition given by (22) is \( \min(b) \geq 2.79 \); all \( b_i \) during the iterations in this example were greater than five; thus, convergence was guaranteed. Fig. 7 shows the resulting approximate sinewave functions obtained using the rounded lower bound designs corresponding to two implementations.

Newton’s method was also tried for this problem, by initializing the Newton iterations at the wordlength values found in Fig. 4; a slightly noticeable improvement was visible only toward the right-hand edge of the plot.

### 5.2 Fixed-Weight Beamformer

In this example, the optimized wordlength design of a fixed weight beamformer [41] for a uniform line array of...
half-wavelength-spaced elements is performed. The beamformer is tuned to replicate a narrow-band signal $s(t)$ emanating from an angle $\theta_1$ from the array broadside. Due to the propagation delay, the signal is phase-shifted at every array element, and the signal at element $k$ is $x_k(t) = s(t) \exp(-jk \sin \theta)$. The beamforming operation to be performed is $y(t) = \sum_{k=0}^{K-1} x_k(t) \exp(jk \sin \theta)$, which can be regarded as the inner product of a complex input vector with complex weights.

Here, a complex multiplication will be implemented using four real multipliers and two real adders, and a complex addition will require two real adders. Using this approach, the flowgraph contains $12K - 2$ arcs. The performance function $V(b)$ used will be the sum of MSEs of the real and imaginary output components.

$K$ half-wavelength-spaced elements is performed. The beamformer is tuned to replicate a narrow-band signal $s(t)$ emanating from an angle $\theta_1$ from the array broadside. Due to the propagation delay, the signal is phase-shifted at every array element, and the signal at element $k$ is $x_k(t) = s(t) \exp(-jk \sin \theta)$. The beamforming operation to be performed is $y(t) = \sum_{k=0}^{K-1} x_k(t) \exp(jk \sin \theta)$, which can be regarded as the inner product of a complex input vector with complex weights.

Here, a complex multiplication will be implemented using four real multipliers and two real adders, and a complex addition will require two real adders. Using this approach, the flowgraph contains $12K - 2$ arcs. The performance function $V(b)$ used will be the sum of MSEs of the real and imaginary output components.

As $\theta$ is varied, the correlation among the inputs $x_k$ varies, so we expect different cost/performance trade-offs and, hence, different wordlengths to result. Fig. 8 shows this variation. The coefficients were fixed at 16 bits of precision, so that only input correlation would be changing from angle to angle.

Table 1 shows the execution time as the number of sensors $K$ is varied from 5 to 30 (as derived from Matlab “tic/toc” commands, on a 3-GHz Intel Xeon running Windows XP). All gradients were calculated numerically; the number of function evaluations increases linearly with $K$, but since the complexity of evaluating the cost and MSE expressions also increases linearly with $K$, quadratic behavior results for this example as shown.

### 5.3 Infinite Impulse Response (IIR) Filter

Next, the behavior of the approach in Section 4 is illustrated for a flowgraph with a feedback structure. The feedback loop has to be broken prior to applying the minimization algorithm; this will result in suboptimal designs, since the performance function $V(b)$ does not take into account feedback effects of the original flowgraph.

The IIR filter [14] to be implemented consists of a real-arithmetic notch filter, with zeros at frequencies $z = \exp(\pm j\theta)$ and poles at $z = p \exp(\pm j\theta)$, $0 < p < 1$, leading to the recursive discrete-time equation $y_n = x_n - 2 \cos \theta x_{n-1} + x_{n-2} + 2 \cos \theta y_{n-1} - p^2 y_{n-2}$. This is shown in Fig. 9a and was used to generate representative data samples for $p = 0.95$ and $\theta = \pi/4$.

Fig. 9b shows the feedforward flowgraph used for wordlength analysis; now, the delay register outputs are treated as algorithm inputs. The optimization approach of

<table>
<thead>
<tr>
<th>No. Sensors $K$</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Arcs $N$</td>
<td>58</td>
<td>118</td>
<td>178</td>
<td>238</td>
<td>298</td>
<td>358</td>
</tr>
<tr>
<td>Calc. Eq. (7) (sec)</td>
<td>0.38</td>
<td>1.85</td>
<td>4.44</td>
<td>7.84</td>
<td>13.36</td>
<td>18.80</td>
</tr>
<tr>
<td>Calc. Eq. (15) (sec)</td>
<td>1.49</td>
<td>4.17</td>
<td>8.01</td>
<td>13.36</td>
<td>19.79</td>
<td>27.28</td>
</tr>
</tbody>
</table>

Timings for 1,000 random inputs, 50 $\alpha$’s, and 2 iterations per $\alpha$. 

Fig. 8. Trade-off dependence on the angle of arrival $\theta$ due to changing input correlation. Results for $K = 5$ sensors.

| Design Times for Wordlength-Optimized Beamformers |
This last expression can give an a priori guarantee of convergence if wordlengths are constrained to be greater than the bound.

For adaptive computing, one could potentially use (15) as the control law governing the update of wordlengths, as a function of changing $a$ and $a_i$'s. The approximate eigenvalue of the Jacobian matrix could be calculated in hardware if necessary to detect divergence in the iteration, allowing for corrective action. The utility of this approach depends on the difficulty of calculating the $a_i$'s. In the adaptive filter in [21], these were simply determined, resulting in a control structure that was so efficient that it could be implemented inexpensively in hardware. Of course, for more complicated flowgraphs, we might not be as fortunate.

### Appendix

Here, the use of the mean-square gradient as the gain coefficient (2) for the linearized transfer function model is justified.

The original flowgraph function is $y = f(z)$, where vector $z$ is regarded as stochastic. Let $x$ be the expanded input vector. The noise-free operating point is denoted by $x_o$. Components of $x_o$ will be functions of the components of $z$. Also, the components of $x_o$ corresponding to the new noise inputs will be identically equal to zero. The flowgraph function of the expanded input vector is written as $y = f(x)$, where $f(x)|_{x=x_o} = f(z)$.

Assume that $x = x_o + e$, where $e = [e_1, \ldots, e_N]^T$ is a vector of statistically independent rounding errors such that $E(e) = 0$ and $E(e^2) = \sigma^2$. Further assume that each component of $e$ is a symmetric random variable, so that any odd-order moment of $e$ is zero. Expectation with respect to errors $e$ and inputs $z$ will be denoted by $E_e\{\}$ and $E_z\{\}$, respectively.

The Taylor series expansion about the operating point is

$$f(x) = f(x_o) + e^T g(x_o) + \frac{1}{2} e^T G(x_o)e + \ldots,$$

where $g(x_o) = \frac{\partial f(x)}{\partial x}|_{x=x_o}$ and $G(x_o) = \frac{\partial^2 f(x)}{\partial x^2}|_{x=x_o}$. 

---

**Fig. 9.** (a) Single pole IIR notch filter. (b) Flowgraph without feedback loop for wordlength analysis.

**Fig. 10.** Notch filter performance for infinite precision and two low-cost implementations.

Section 4 was performed, the resulting wordlengths rounded, and a numerical simulation using the true flowgraph in Fig. 9a was performed. The resulting designs were not Pareto-optimal, and the feedforward flowgraph underestimated the MSE by roughly an order of magnitude. However, the designs could be used as starting points for a search-based optimization technique. Also, the notch depth achieved by the suboptimal designs could be adequate in many situations. For example, Fig. 10 shows the notch depth for infinite precision as well as for two low-cost implementations. In this case, the lowest cost implementation was nearly 70 dB of attenuation.

**6 Summary and Conclusions**

In this paper, a mathematical approach to finding good wordlength combinations for DSP flowgraphs was developed. By defining suitable cost and MSE functions, an optimization problem may be formulated.

The approach of tracing out the Pareto-optimal design curve, which defers the decision of the relative weighting between cost and MSE, was chosen. This is the appropriate approach in several situations: when one is unsure of the overall importance of the DSP flowgraph in relation to a larger total system, when one is populating a static design library, or perhaps in a dynamically reconfigurable system, where the amount of available resources may be changing.

The optimization metric (9) may of course be minimized by many methods. The performance of the interesting iteration (15) was examined. In the example provided, the iteration showed extremely fast convergence, and the experience with other flowgraphs shows similar results.

To monitor convergence, an approximate bound on the largest eigenvalue of the Jacobian matrix of the iteration can be conveniently calculated from (16), (18), and (19). When this is specialized to quadratic cost functions, (22) results. This last expression can give an a priori guarantee of convergence if wordlengths are constrained to be greater than the bound.
We will assume that only the terms of \( f(x) \) up to second order are significant. For notational clarity, the dependence on \( x \) will be mostly suppressed in the notation for \( g \) and \( G \), and the gradient and Hessian elements will be represented by \( g_i = g_i \) and \( G_{ij} = G_{ij} \).

Defining the output error as \( p(x_o, e) = f(x) - f(x_o) \approx e^T g + e^T Ge \), it can be shown that the mean output error is

\[
\sum_i E_i [G_{ii}]^2 / 2.
\]

Thus, there can be a bias in the output even for zero-bias \( e \). For the examples in Section 5, this bias was minute compared to the output MSE.

The output MSE is calculated as

\[
p^2(x_o, e) = e^T g g^T e + (e^T g)(e^T Ge) + \frac{1}{4} (e^T Ge)^2,
\]

\[
E_e \{ p^2(x_o, e) \} = \sum_i g_i^2 \sigma_i^2 + 0 + \frac{1}{4} \sum_i \sum_j \sum_k \sum_l G_{ij} G_{kl} E_e \{ e_i e_j e_k e_l \}
\]

since all odd-order moments of \( e \) are equal to zero. The quadruple summation can be simplified by noting that the zero odd-order moments lead to four distinct summations:

1. \( i = j, k = l, i \neq k \)
2. \( i = j, k = l, i \neq j \)
3. \( i = j, k = l, i \neq j \)
4. \( i = j = k = l \).

Therefore,

\[
E_e \{ p^2(x_o, e) \} = \sum_i g_i^2 \sigma_i^2 + \frac{1}{4} \sum_{i \neq j} G_{ij} G_{ij} \sigma_i^2 \sigma_j^2
\]

\[
+ \frac{1}{4} \sum_{i \neq j} G_{ij}^2 \sigma_i^2 \sigma_j^2 + \frac{1}{4} \sum_{i \neq j} G_{ij} G_{ij} \sigma_i^2 \sigma_j^2 + \frac{1}{4} \sum_i G_{ii}^2 E_e \{ e_i^4 \}
\]

\[
= \sum_i g_i^2 \sigma_i^2 + \frac{1}{4} \sum_{i \neq j} (2G_{ij}^2 + G_{ii} G_{jj}) \sigma_i^2 \sigma_j^2 + \frac{1}{4} \sum_i G_{ii}^2 E_e \{ e_i^4 \}.
\]

Next, reintroduce the \( x \) notation, and take the expectation over the original set of inputs \( z \), giving the MSE in \( y \) as

\[
MSE_y = \sum_i E_x \{ g_i^2(x_o) \} \sigma_i^2 + \frac{1}{4} \sum_i E_x \{ G_{ii}(x_o) \} E_x \{ e_i^4 \}
\]

\[
+ \frac{1}{4} \sum_{i \neq j} E_x \{ 2G_{ij}^2(x_o) + G_{ii}(x_o) G_{jj}(x_o) \} \sigma_i^2 \sigma_j^2.
\]

Thus, for a linearized model, \( MSE_y \approx \sum_i E_x \{ g_i^2(x_o) \} \sigma_i^2 \), and therefore, \( s_i \) in (2) is given by \( s_i = E_x \{ g_i^2(x_o) \}^{1/2} \), the RMS value of the gradient.

The gradient (and Hessian) elements can be calculated using the central difference approximations

\[
g_i(x_o) \approx \left( f(x_o + \epsilon \mathbf{l}_i) - f(x_o - \epsilon \mathbf{l}_i) \right) / 2 \epsilon, \quad G_{ij}(x_o) \approx \left( f(x_o + \epsilon \mathbf{l}_i) - 2 f(x_o) + f(x_o - \epsilon \mathbf{l}_i) \right) / \epsilon^2, \quad G_{ij}(x_o) \approx \left( \sum_{i,j \in \{|x_o| \}} ij f(x_o + \epsilon (i \mathbf{l}_i + j \mathbf{l}_j)) / 4 \epsilon^2, \right.
\]

where \( \epsilon \) is a small perturbation, and \( \mathbf{l}_i \) is a vector with the \( i \)th component equal to one and all other components equal to zero. For the linear transfer function model, the \( s_i \)’s are easily determined via simulation by presenting an average representative set of inputs \( z \) to the flowgraph, perturbing each arc in turn by \( \pm \epsilon \), measuring the resulting output perturbations, calculating the squared gradient, and then averaging over the set of inputs. If \( L \) input vectors are used for the representative set, then at most \( O(LN) \) evaluations of \( f \) are required to evaluate (28). For the second-order transfer function model, a similar procedure can be used by perturbing arcs in pairs and using (28), requiring at most \( O(LN^2) \) function evaluations. Note that for the examples in Section 5, the quadratic perturbation terms were miniscule compared to the linear terms, and therefore, the linear model is justified.

ACKNOWLEDGMENTS

The author would like to thank the anonymous reviewers for their insightful comments.

REFERENCES


Paul D. Fiore received the BS degree in computer system engineering from the University of Massachusetts, Amherst, in 1986, the MS degree in electrical engineering, with specialization in image processing, from Tufts University in 1988, and the PhD degree, with specialization in efficient motion vision algorithms, from the Massachusetts Institute of Technology in 2000. He is currently with the Lincoln Laboratory at the Massachusetts Institute of Technology, Lexington. He was an associate editor of the IEEE Transactions on Signal Processing during 2002-2005. His interests are in the algorithmic development and efficient DSP implementation of sensor array and multichannel processing systems, direction finding, detection, estimation, classification, and computer vision systems. He is a member of the IEEE, the IEEE Signal Processing Society, and an associate member of the society’s Sensor Array and Multichannel (SAM) Technical Committee. He was a program committee member for the IEEE International Workshop on Rapid System Prototyping (RSP) from 1995 to 1996 and a member of the technical committee for the IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM) from 2004 to 2006.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.