FAST, ACCURATE STATIC ANALYSIS FOR FIXED-POINT FINITE-PRECISION EFFECTS IN DSP DESIGNS

Claire F. Fang, Rob A. Rutenbar, Tsuhan Chen

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{ffang, rutenbar, tsuhan}@ece.cmu.edu

ABSTRACT

Translating digital signal processing (DSP) software into its finite-precision hardware implementation is often a time-consuming task. We describe a new static analysis technique that can accurately analyze finite-precision effects arising from fixed-point implementations of DSP algorithms. The technique is based on recent interval representation methods from affine arithmetic, and the use of new probabilistic bounds. The resulting numerical error estimates are comparable to detailed statistical simulation, but achieve speedups of four to five orders of magnitude by avoiding actual bit-true simulation. We show error analysis results on both feed forward and feedback DSP kernels.

1. INTRODUCTION

Finite-precision arithmetic is the bane of DSP hardware. DSP designs are routinely prototyped in high-precision floating point for ease and flexibility. But they are always re-implemented in some hardware-efficient finite-precision format for better silicon area, power, and speed tradeoff. For applications that require large dynamic range, custom floating point formats (e.g., smaller mantissas or exponents, simplified rounding modes) are one option. These so-called "lightweight float" formats [7] can mitigate some of the pain of translating from full to limited precision.

However, more commonly, fixed-point formats are employed in custom implementation of DSP algorithms, in which each operand is modeled with both an integer and a fixed-length fraction. Fixed-point formats have three important advantages. First, they behave mostly like integers, except for the need to round the fractional parts after each operation. Second, it is easy to transform one fixed-point format into another of different bit-width via shifting and zero-padding. Third, as a result of this ease of format translation, it is possible to assign each operand in a complex DSP task a *unique*, minimal bit-width, and thus minimize overall area, power, and delay.

Of course, the problem is how to do this efficiently. Translation from floating point to small, finite-precision formats is still often done by hand. The issue is how to analyze the finite-precision redesign, and guarantee both the necessary dynamic range for each operand (to avoid overflows) and the right fraction precision (to bound accumulated round-off errors). Attacks on this analysis problem to date have focused on three techniques: Monte Carlo-style statistical simulation [4, 5, 8, 11], transfer-function-based error analysis [13, 14], and interval analysis [1, 17]. The simulation techniques work well, but can be very inefficient if we need to rerun many input vectors for each small proposed format change. Further, full coverage of input vectors is not guaranteed. Error analysis based on system transfer functions is mathematically attractive, but in practice, limited to applications which can be abstracted as a linear transfer function. Interval methods would seem more efficient and general, given that they strive to replace discrete operand values with a representation of a *range* of values, and propagate interval ranges instead of numbers through the computation. The problem is the *pessimism* in the computed result: for nontrivial applications, the intervals explode in size as they propagate, because the correlations among operands are no longer carried in the intervals, and conservative assumptions are made for each interval operations. Such overestimated intervals may provide little insight about precision.

Recently, an entirely new mode of analysis for finite-precision effects was introduced in [2]. Replacing simple interval models with recently developed *affine arithmetic* models [6], it was shown that tight bounds on the errors could be quickly estimated for custom-precision floating point. The style of the analysis is that of symbolic interval methods; but the error bounds rival expensive statistical simulation. Unfortunately, the formulation of [2] is inapplicable to the much more common case of fixed-point arith-

This work was supported by the Semiconductor Research Coorporation and Pittsburgh Digital Greenhouse. The authors would like to thank Markus Püschel for many fruitful discussions.

metic, which have very different error propagation models for each common arithmetic computation.

In this paper, we extend the "static analysis" models of [2] to the more practical case of fixed-point arithmetic. The use of term "static analysis" is borrowed from the similar usage in static timing analysis for logic netlists. We seek an analysis strategy in which a single pass of "simulating" the computational network produces all the information we need for error analysis. For static timing analysis, instead of logic values, one propagates best and worst case timing information through the logic network. For fixed-point finite-precision error analysis, instead of discrete values, we propagate affine interval models of each operand, and in particular, use the algebra of affine intervals to deal with many of the *correlations* among operands that have thwarted all earlier attempts at accurate interval analysis. Like static timing, static error analysis lets us "run" the computation just once, but produces all the information necessary for accurate analysis of the numerical impacts of each finite-precision format choice.

The remainder of the paper is organized as follows. Section 2 gives necessary background on interval methods in general and affine models in particular. Section 3 introduces a new affine error model suitable for fixed-point data formats, and analyzes how errors propagate differently in each of the common DSP arithmetic computations. This section gives both a deterministic model, and an essential probabilistic estimator based on *confidence intervals* that dramatically reduces the pessimism of the estimations in practical applications. Section 4 offers a range of experimental results showing the utility of the approach. We routinely rival the accuracy of statistical methods, but with four to five orders of magnitude less CPU time. We also apply the static error analysis technique to a typical DSP feedback system in which outputs circulate back to inputs. As with static timing analysis, such closed-loop systems require careful analysis. We describe how to handle this difficult case. Finally, Section 5 offers concluding remarks.

2. BACKGROUND

The modeling tool in this paper is *affine arithmetic*, which is an efficient and recent variant of range arithmetic. In this section, we begin with introducing its predecessor, interval arithmetic, and then emphasize the advantage of affine arithmetic.

2.1. Interval Arithmetic

Interval arithmetic (IA), also known as *interval analysis*, was invented in the 1960s by Moore [16] to solve range problems. The uncertainty in a variable x is represented by the interval $\overline{x} = [\overline{x}.lo, \overline{x}.hi]$, meaning that the true value of x is known to satisfy $\overline{x}.lo \leqslant x \leqslant \overline{x}.hi$. For each operation $f : R \leftarrow R^m$, there is a corresponding range extension $\overline{f} : \overline{R} \leftarrow \overline{R}^m$. Taking addition as an example, the corresponding IA operation is obtained as

$$\overline{z} = \overline{x} + \overline{y} = [\overline{x}.lo + \overline{y}.lo, \quad \overline{x}.hi + \overline{y}.hi] \tag{1}$$

Similar formulas can be derived for multiplication, division, square root, and other common mathematical functions [6]. A floating-point error model based on IA was introduced in [10]. The main problem of IA is overestimation, especially for correlated variables. To illustrate this problem, suppose that in (1) $\overline{x} = [-1, 1]$, $\overline{y} = [-1, 1]$, and that x and y have the relation $y = -x$. Using (1), we get $\overline{z} = [-2, 2]$, yet in reality $z = x + y = 0$. The effect of overestimation accumulates along the computation chain, and may result in an exponential range explosion. This problem is alleviated by *affine arithmetic*.

2.2. Affine Arithmetic

Affine arithmetic (AA), or *affine analysis*, is a recent refinement in range arithmetic [6]. It has been used in areas such as computer graphics [6], analog circuit sizing [12], and floating point error modeling [2]. In contrast to IA, AA preserves correlations among intervals. In affine arithmetic, the uncertainty of a variable x is represented as a range in an *affine form* $\widehat{x}$, given by

$$\widehat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n, \quad -1 \leqslant \varepsilon_i \leqslant 1. \tag{2}$$

Each *uncertainty symbol* ε_i stands for an independent component of the total uncertainty of the variable x; the corresponding coefficient x_i gives the magnitude of that component. For the affine functions $\widehat{x} \pm \widehat{y}, \widehat{x} \pm c$, and $c\widehat{x}$, the resulting affine forms are easily obtained using (2). For other operations (e.g., multiplication), the result, as a function $f(\varepsilon_1, \ldots, \varepsilon_n)$, is no longer affine. Thus, to obtain the affine form for the result, first a linear function $f^*(\varepsilon_1, \ldots, \varepsilon_n)$ is selected as an approximation of $f(\varepsilon_1, \ldots, \varepsilon_n)$, and a new uncertainty term indicating the approximation error is estimated and added to the final affine form [6].

The key feature of AA is that one symbol ε_i may contribute to the uncertainties of two or more variables, indicating correlations among them. When these variables are combined, uncertainty terms may cancel out, known as *range cancellation*. This advantage is especially noticeable in computations that are highly correlated or of great computational depth. Returning to our previous simple example, suppose that x and y have affine forms $\widehat{x} = 0 + 1\varepsilon$ and $\widehat{y} = -\widehat{x} = 0 - 1\varepsilon$. In this case, the affine form of the sum $\widehat{z} = \widehat{x} + \widehat{y} = 0$ perfectly coincides with the actual range of the variable z.

Range arithmetic provides a tool for problems in which precise information is unavailable and an estimation of range

offers a good approximation of the solution. In the next section, we apply affine arithmetic to fixed-point error analysis.

3. FIXED-POINT ERROR MODELING VIA AFFINE ARITHMETIC

When designers transform a high level algorithmic description to its fixed-point implementation, two distinct problems may arise: overflows, which appear if the integer bit-width is too small, and precision loss, commonly known as *quantization error* or *round-off error*, due to the finite fraction bit-width. Those problems require us to quantify the range growth, as well as the quantization errors. This paper focuses on the quantization error and the determination of the fraction bit-widths, although the modeling itself also catches the range growth, as a byproduct of the formulation.

In this section, we start with the AA-based models for fixed-point numbers and elementary computations, then we develop a probabilistic bounding method to estimate the error bound.

3.1. Affine Forms for Fixed-Point Numbers

To build the affine model for fixed-point arithmetic, we first write the fixed-point representation of a real number as an affine interval. Suppose a real number x is represented by the (i, f) fixed-point format, where i is the bit-width for the integral part, and f is the bit-width for the fractional part. The quantization error is bounded by $2^{-(f+1)}$ in case of real rounding, or 2^{-f} in case of truncation. Throughout the paper, we assume real rounding is employed. Therefore the fixed-point representation x_f can be written as:

$$x_f = x + 2^{-(f_x+1)}\varepsilon, \quad \text{with } \varepsilon \in [-1, 1] \tag{3}$$

Note that (3) is an affine interval, whose uncertainty, expressed by the random variable ε, is caused by rounding.

There are typically two types of fixed-point numbers in a program: constants and variables. (3) can be more precisely specified for each case. For a constant, (3) is reduced to the following:

$$\begin{aligned} c_f &= c + e, \\ E_c &= e, \quad |e| \leqslant 2^{-(f_c+1)} \end{aligned} \tag{4}$$

The quantization error E_c is known, given the fraction bit-width. For a variable that lies in a certain range $[v_0 - v_1, v_0 + v_1]$, its fixed-point representation has one more uncertainty term $v_1\varepsilon$, shown as:

$$\begin{aligned} v_f &= v_0 + v_1\varepsilon + 2^{-(f_v+1)}\varepsilon_e \\ E_v &= 2^{-(f_v+1)}\varepsilon_e, \quad \varepsilon, \varepsilon_e \in [-1, 1] \end{aligned} \tag{5}$$

Both the error term E_v and the fixed-point representation v_f are in affine forms. Two independent random variables, ε and ε_e, indicate the uncertainties from the input range and quantization error, respectively. Later, we show that our analysis is not affected by their actual distributions. It is these random variables that capture the sources of the uncertainties and keep track of the correlations among a large number of of fixed-point variables.

3.2. Fixed-Point Computation and Error Propagation with AA

In order to symbolically simulate a fixed-point program with AA, we must replace each elementary operation on fixed-point numbers with the corresponding operation on affine forms, returning an affine form. Further, there is a fixed-point error associated with the output of each computation, coming from the input error and the quantization during the computation. We need to model the error propagation in AA as well. For affine functions, namely $x \pm y$, cx, and $x \pm c$, we show that the resulting affine form can be deduced from the input affine forms. For non-affine functions, xy and x/y, we also obtain the results as affine forms with certain approximations.

For a binary operation $z \leftarrow f(x, y)$ or $z \leftarrow f(x, c)$, where x and y are variables, and c is a constant, the challenge is to turn z into an affine form. First, we generalize the operands in the following affine forms, based on (4) and (5):

$$\begin{aligned} x_f &= x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + E_x, \quad E_x = x_{e0} + \sum_{i=1}^{m} x_{ei}\varepsilon_{ei} \\ y_f &= y_0 + \sum_{i=1}^{n} y_i\varepsilon_i + E_y, \quad E_y = y_{e0} + \sum_{i=1}^{m} y_{ei}\varepsilon_{ei} \\ c_f &= c + E_c, \quad E_c = e \end{aligned}$$

where x_{e0}, y_{e0} and e are errors resulted from quantization on constants.

All the ε_i's and the ε_{ei}'s are independently distributed in [-1, 1], the former capturing the range uncertainty due to the insufficient knowledge of the exact value of the variables, and the latter capturing the error uncertainty caused by fixed-point quantization. Then, we develop the affine computation and error propagation models for the following types of operations.

3.2.1. Affine operations

For affine operations, the result is the combination of two input affine forms and a quantization error term. The existence of this term depends on the fraction bit-width. Taking addition as an example, quantization error is only introduced when the result has smaller fraction bit-width than either of the operands. This newly introduced error, denoted by ϕ, is bounded by $2^{-(f+1)}$. Here we give details on addition, addition with a constant, and multiplication with a constant.

Addition:

$$
\begin{aligned}
x_f \pm y_f &= (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\varepsilon_i \\
&\quad + E_x \pm E_y + \phi \\
E_z &= E_x \pm E_y + \phi \\
&\text{where } \phi = \begin{cases} 2^{-(f_z+1)}\varepsilon_z, & f_z < max(f_x, f_y) \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{6}
$$

Addition with a constant:

$$
\begin{aligned}
x_f \pm c_f &= x_0 \pm c + \sum_{i=1}^{n} x_i\varepsilon_i + E_x \pm e + \phi \\
E_z &= E_x \pm e + \phi \\
&\text{where } \phi = \begin{cases} 2^{-(f_z+1)}\varepsilon_z, & f_z < max(f_x, f_c) \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{7}
$$

Multiplication with a constant:

$$
\begin{aligned}
c_f x_f &= (c + E_c)(x_0 + \sum_{i=1}^{n} x_i\varepsilon_i) + E_x \cdot c + \phi \\
&\quad (E_x E_c \text{ is neglected}) \\
E_z &= E_x \cdot c + E_c(x_0 + \sum_{i=1}^{n} x_i\varepsilon_i) + \phi \\
&\text{where } \phi = \begin{cases} 2^{-(f_z+1)}\varepsilon_z, & f_z < f_x + f_c \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{8}
$$

$E_x E_c$ is neglected because it is the product of two small terms.

The final error E_z includes the errors propagated from the operands and the new quantization error ϕ. Note that since E_x, E_y, and E_c are all affine forms, this final error term E_z is also an affine form.

3.2.2. Multiplication

Let us now consider multiplication of two fixed-point variables x_f and y_f. Similar to affine operations, a new quantization error ϕ is introduced after the multiplication.

$$
\begin{aligned}
x_f y_f &= (x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + E_x)(y_0 + \sum_{i=1}^{n} y_i\varepsilon_i + E_y) + \phi \\
&= x_0 y_0 + \sum_{i=1}^{n} (y_0 x_i + x_0 y_i)\varepsilon_i + Q_1 + Q_2 + \phi \\
&\text{where } \phi = \begin{cases} 2^{-(f_z+1)}\varepsilon_z, & f_z < f_x + f_y \\ 0, & \text{otherwise} \end{cases} \\
Q_1 &= \sum_{i=1}^{n} x_i\varepsilon_i \sum_{i=1}^{n} y_i\varepsilon_i \\
Q_2 &= (\sum_{i=1}^{n} x_i\varepsilon_i)E_y + (\sum_{i=1}^{n} y_i\varepsilon_i)E_x \\
&(E_x E_y \text{ is neglected})
\end{aligned}
$$

$E_x E_y$ is neglected because it is the product of two small terms.

The challenge is that the product $x_f y_f$ is *not* an affine form any more, due to the quadratic terms Q_1 and Q_2. Moreover, each quadratic component is *not* independent of the others, which makes it impossible to replace the random variable $\varepsilon_i\varepsilon_j$ or $\varepsilon_i\varepsilon_{ej}$ with a new random variable ε_k.

This issue is discussed in [6], and by using the same *range estimation*, we can approximate Q_1 and Q_2 to linear terms.

$$
\begin{aligned}
\widetilde{Q_1} &= B(\sum_{i=1}^{n} x_i\varepsilon_i)B(\sum_{i=1}^{n} y_i\varepsilon_i)\varepsilon_k \\
\widetilde{Q_2} &= B(\sum_{i=1}^{n} x_i\varepsilon_i)E_y + B(\sum_{i=1}^{n} y_i\varepsilon_i)E_x
\end{aligned}
$$

where ε_k is a new random variable in $[-1,\ 1]$, and the bounding operator B is defined by

$$
B(\sum_{i=1}^{n} x_i\varepsilon_i) = \sum_{i=1}^{n} |x_i|, \tag{9}
$$

which computes a hard upper bound of its argument. This is a conservative approximation, in the sense that the new interval always includes the original true interval.

By range estimation, we turn the product and its fixed-point error into affine forms, shown by (10).

$$
\begin{aligned}
x_f y_f &\approx x_0 y_0 + \sum_{i=1}^{n} (y_0 x_i + x_0 y_i)\varepsilon_i + r\,\varepsilon_k \\
&+ \quad B(\sum_{i=1}^{n} x_i)E_y + B(\sum_{i=1}^{n} y_i)E_x + \phi \\
E_z &\approx B(\sum_{i=1}^{n} x_i)E_y + B(\sum_{i=1}^{n} y_i)E_x + \phi
\end{aligned} \tag{10}
$$

where r equals $B(\sum_{i=1}^{n} x_i\varepsilon_i)B(\sum_{i=1}^{n} y_i\varepsilon_i)$.

3.2.3. Division

To derive the model for division $z \leftarrow \frac{x}{y}$, we assume the range of y does not include zero. We set $\hat{x} = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i$ and $\hat{y} = y_0 + \sum_{i=1}^{n} y_i \varepsilon_i$.

$$\begin{aligned}
\frac{x_f}{y_f} &= \frac{\hat{x} + E_x}{\hat{y} + E_y} + \phi \\
&= (\frac{\hat{x} + E_x}{\hat{y}})(1 + \frac{E_y}{\hat{y}})^{-1} + \phi \\
&\approx (\frac{\hat{x} + E_x}{\hat{y}})(1 - \frac{E_y}{\hat{y}}) + \phi \\
&= \frac{\hat{x}}{\hat{y}} - \frac{\hat{x}}{\hat{y}^2} E_y + \frac{1}{\hat{y}} E_x + \phi \\
&\text{where } \phi = 2^{-(f_z+1)} \varepsilon_z
\end{aligned}$$

According to [6], $\frac{\hat{x}}{\hat{y}}$, $\frac{\hat{x}}{\hat{y}^2}$, $\frac{1}{\hat{y}}$ can all be turned into affine forms using *Chebyshev* approximation, denoted by A_1, A_2, and A_3 in (11). By further applying the same bounding operator as in the multiplication model, the result of fixed-point division and its error also become affine forms:

$$\begin{aligned}
\frac{x_f}{y_f} &= A_1 - B(A_2) E_y + B(A_3) E_x + \phi \\
E_z &= -B(A_2) E_y + B(A_3) E_x + \phi
\end{aligned} \tag{11}$$

where the A_i's are affine forms.

3.3. Error Bound Estimation

With the affine computation model and the error propagation model developed above, we are able to simulate the fixed-point program once, "symbolically", with not only the full coverage of all cases, but also the consideration of all correlations on the data path. Such symbolic simulation requires the input ranges as well as the fixed-point fraction bit-widths for all the computations, and returns the output error expressed as an affine form:

$$E(out) = e_0 + \sum_{i=1}^{n} e_i \varepsilon_i$$

The estimated upper bound of the error is

$$\bar{E}(out) = B(e_0 + \sum_{i=1}^{n} e_i \varepsilon_i) = \sum_{i=0}^{n} |e_i| \tag{12}$$

The bounding operator B is the same as defined in (9).

Unfortunately, a disadvantage of using this bounding operator is that the estimation may be too pessimistic. When n gets very large, it becomes extremely unlikely that all the ε_i's simultaneously take the extreme values and cause the true interval to reach this upper bound. To formalize this behavior, we set $E_n = \sum_{i=1}^{n} e_i \varepsilon_i$ and denote with $\mathcal{N}(0,1)$ a standard normally distributed random variable. Then, by the *central limit theorem* [15], as n increases,

$$\frac{E_n}{\sqrt{n}\sqrt{variance(E_n)}} \rightarrow \mathcal{N}(0,1).$$

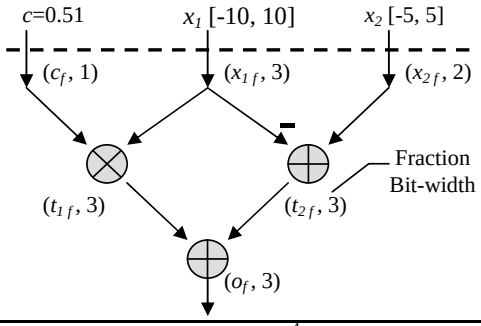
We use this interpretation to develop a refinement of the AA-based error analysis, which is based on *probabilistic bounds*. To achieve this, we modify the bounding operator B such that it returns a *confidence interval* that bounds the true interval with a specified high probability λ. We denote this new operator B_λ and define it by

$$prob(E_n \leqslant B_\lambda(E_n)) \geqslant \lambda. \tag{13}$$

This new probabilistic bounding operator B_λ also replaces the hard bounding operator in the multiplication and division models in (10) and (11).

To calculate $B_\lambda(E_n)$ for a given λ, we use the inverse CDF (Cumulative Density Function) of E_n for $n = 1, 2, 3$, and Gaussian approximation for $N > 3$. Since the central limit theorem applies to any distribution, our probabilistic bound analysis is independent of the input range distributions and the quantization error distribution when N is larger than 3, which is usually the case for real applications. In our experiments, $\lambda = 0.999999$ is chosen. The original hard upper bound corresponds to $\lambda = 1$. In the next section, we show the difference between these two bounds, and the dramatic impact of the probabilistic relaxation.

Finally, we illustrate the AA-based fixed-point error analysis by a simple program shown in Figure 1. The fraction



x_{1f}	$= 10\varepsilon_1 + 2^{-4}\varepsilon_{e1}$
x_{2f}	$= 5\varepsilon_2 + 2^{-3}\varepsilon_{e2}$
c_f	$= 0.5$
$t_{1f} = c_f x_{1f}$	$= 5\varepsilon_1 + 2^{-5}\varepsilon_{e1} + 2^{-4}\varepsilon_{e3}$
$t_{2f} = x_{2_f} - x_{1f}$	$= -10\varepsilon_1 + 5\varepsilon_2 - 2^{-4}\varepsilon_{e1} + 2^{-3}\varepsilon_{e2}$
$o_f = t_{1f} + t_{2f}$	$= -5\varepsilon_1 + 5\varepsilon_2 - 2^{-5}\varepsilon_{e1} + 2^{-3}\varepsilon_{e2} + 2^{-4}\varepsilon_{e3}$
E_o	$= -2^{-5}\varepsilon_{e1} + 2^{-3}\varepsilon_{e2} + 2^{-4}\varepsilon_{e3}$
$B_1(E_o)$	= 0.2188
$B_{0.999999}(E_o)$	= 0.2038

Figure 1: A simple example of error analysis

bit-widths are labeled beside the variables in the data-flow graph. According to (4) to (8), (12) and (13), the fixed-point affine forms for all the variables and the error bound for the final output are derived. In Figure 1, ε_i represents the input range uncertainty, and ε_{ei} denotes the quantization error uncertainty. Note that there is range cancellation in the last addition, which exhibits the advantage of affine arithmetic modeling.

4. EXPERIMENTAL RESULTS

4.1. Methodology

Recall our original goal: estimate quickly the numerical errors that accrue given a candidate fixed-point format. The actual "error" we seek is the maximum difference between the fixed-point value computed in this format, and the "ideal" real value, which we take to be the IEEE double-precision version of the computation. With bit-true simulation over a suitably large set of inputs, we can calculate this error. In all the experiments in this section, we compare our estimated error bound against this simulated maximum error.

Our experiment relies on two C++ libraries. The SystemC fixed-point library [18] is used to assist bit-true simulation, which provides a comparison baseline for our static analysis. The maximal error is obtained by comparing the output of the fixed-point and double-precision versions of the code simulated with 10^6 random independent inputs. We assume 16 bits for the fractional part in all the experiments. Since the focus of this paper is the quantization error, 48 bits for the integral part is chosen to assure no overflow occurs. The other library is an AA-based fixed-point computation library, developed based on the models presented in this paper. It overloads the C++ arithmetic operators and computes the affine form for each variable in the code, and the relevant bound. This strategy allows us to analyze the program with minimal modification to the source code.

4.2. Comparing AA to IA

To verify the advantage of range cancellation enabled by the novel affine modeling, we compare the AA-based error bound according to (4)–(13) with the conventional IA-based error bound [17, 19] in a DSP application— the 8-input IDCT (Inverse Discrete Cosine Transform), which is widely used in image and video processing. The inputs are assumed to lie in the range [-64, 64]. As shown in Table 1, the AA-based error bound ($\lambda = 1$) is much tighter compared to the IA-based error bound. IA overestimates because it fails to consider the correlations among variables. We highlight an example of such correlations in the IDCT diagram in Figure 2.

Since correlations on the data path are very common in DSP applications, our AA-based error model significantly improves accuracy compared to the IA-based model, while incurring virtually the same computational cost.

AA error bound	IA error bound	Simulated max error
0.00122	0.00337	0.00109

Table 1: Error analysis results on IDCT

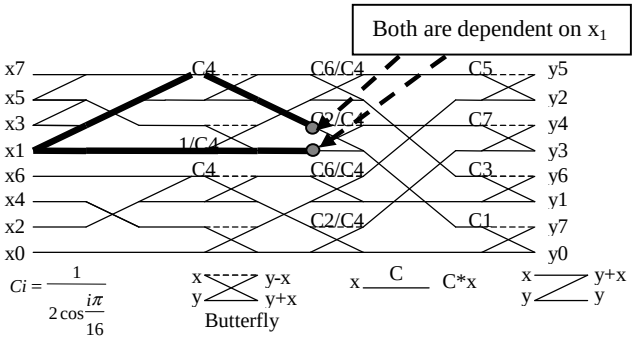


Figure 2: Data-flow of the IDCT algorithm

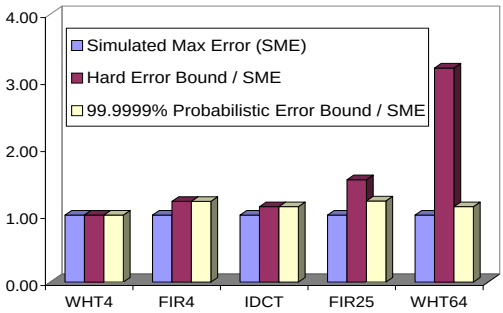
4.3. Results on Feed Forward Systems

We test the applicability and accuracy of the proposed error model and the probabilistic bounding method on a variety of common signal processing kernels, including WHT (Walsh-Hadamard Transform), FIR (Finite Impulse Response) filter, and IDCT, all of which are feed forward systems, meaning that there is no feedback loop on data path. We assume the input range is [-64, 64] for all the kernels. Figure 3 shows the comparison among the simulated maximum error, the hard error bound ($\lambda = 1$) and the probabilistic error bound with $\lambda = 0.999999$. They are normalized with respect to the simulated maximum error (SME). The x-axis is ordered by computational complexity. First, the hard error bound always provides an upper bound for the maximum error. However, as computational complexity increases, this bound becomes looser (see WHT64). This verifies the asymptotic behavior expressed by the central limit theory. Our probabilistic bounding method offers a tight–yet reasonably accurate (with 99.9999% confidence)–bound to the maximum error, regardless of computational complexity.

Indeed, the simulated maximum error is not the true maximum error. There is also a confidence value associated with it, because the maximum error seen in 10^6 simulations only guarantees that

$$prob(error > \textit{simulated maximum error}) \leqslant \frac{1}{10^6},$$

if we assume uniform error distribution. Therefore the corresponding confidence of the simulated maximum error is 0.999999, which is the same as the confidence of our estimated bound.



* WHT n : n-input WHT
* IDCT : 8-input IDCT
* FIR n : n-tap FIR filter

Figure 3: Accuracy of the error bounds

	CPU time for probabilistic bound	CPU time for max error
WHT4	0.001	28.7
WHT64	0.05	1147
FIR4	0.001	50
FIR25	0.01	355.8
IDCT8	0.01	155.2

Table 2: Comparison of CPU time (sec)

What makes our static error analysis attractive is not only the accuracy, but also the computational efficiency. In contrast to bit-true simulation, which requires millions of complete program executions, our method can predict error statically via a simple program execution with overloaded operators. Table 2 shows the CPU times on a 1.6GHz Pentium4 machine needed to compute the estimated bound and the simulated maximum error, the latter estimated from 10^6 input vectors. Note that the CPU time required to compute the probabilistic bound is independent of λ. For these DSP kernels, our static analysis achieves four to five orders of magnitude speedup compared to simulation.

4.4. Results on a Feedback System

In feed forward systems, the output only depends on a finite number of inputs and operations, while in feedback systems, all the inputs and operations in the past have influence on the current output. This raises an important question. Since our AA-based error model preserves all the uncertainty terms that are related to the current output, will the estimated output error keep growing endlessly, as more iterations are analyzed?

To investigate this, we consider a basic DSP feedback system— IIR (Infinite Impulse Response) filter, specified by

$$y_n = x_n + c_1 y_{n-1} - c_2 y_{n-2}, \quad \text{where } c_1 = \frac{1}{\sqrt{2}}, c_2 = \frac{1}{2} \tag{14}$$

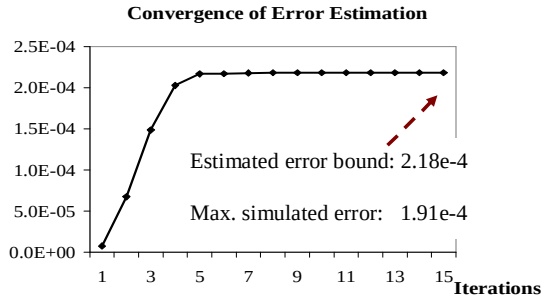


Figure 4: Convergence of Error Estimation in the IIR filter

The values of c_1 and c_2 are chosen to ensure that the IIR filter is a stable system. x_n is distributed in [-64, 64], and 16 bit fractional part is assumed for all the computations. As more iterations are analyzed, the output error bound gets larger and larger. Interestingly, after a certain number of iterations, the error bound *converges*, although more uncertainty terms are constantly added to the error affine form. This behavior is shown in Figure 4. The maximum simulated error is tightly bounded by the 99.9999% probabilistic error bound after convergence.

The reason for the convergence is that *for a stable feedback system, the error itself is also stable*. By applying the AA-based error model to (14), we get

$$E_n = \Phi + c_1 E_{n-1} - c_2 E_{n-2} \tag{15}$$

meaning the error E_n depends on the errors propagated from the previous two time steps and the errors introduced in the current step. Φ includes all the quantization errors in the multiplications and the additions, and the error from x_n. Note that (15) has exactly the same system parameters as those in (14). Thus, E is also stable, and so the range of E does not grow to infinity. Therefore, error analysis only needs to be conducted until it converges.

4.5. Context and Analysis

The techniques presented here, and for custom floating-point arithmetic in [2], together constitute the first highly efficient, accurate set of static analysis methods for finite-precision effects in practical DSP system design. It is worthwhile to place these efforts in context. Figure 5 shows a simple taxonomy of ongoing work in this area. One axis of classification is simply the floating-point versus fixed-point implementation target. We focus on fixed-point in this paper. The second axis separates techniques into those based on statistical simulation and sampling to measure errors, and techniques based on more symbolic, static analysis style. Our work obviously falls in the second category. Finally, the third axis captures the purpose for which the analysis is requested: synthesis of optimal bit-widths, or verification that an existing set of format choices has acceptable numerical precision. We have focused on the verification task

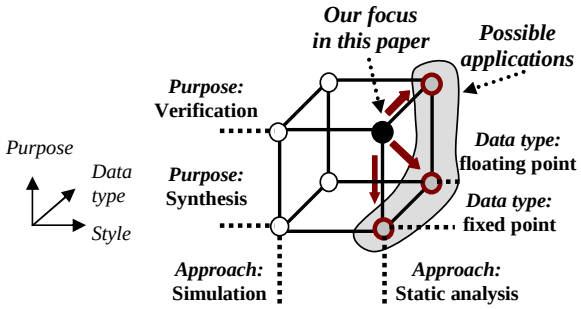


Figure 5: Research on finite-precision arithmetic

in this paper, but note that the development of techniques that are four to five orders of magnitude faster than statistical simulation bodes well for the likelihood that they might be incorporated in simulation-based DSP optimization and synthesis tasks, e.g., the systems presented in [3, 8, 9, 11].

Finally, we note some limitations of the current technique that are the focus of ongoing work. First, in the development of the AA-based error model, we assumed all the uncertainty symbols ε_i's are independent. This is not always true for the inputs. Strong correlations among inputs may lead to over-estimations. The appropriate solution would seem to be to model these correlations explicitly from the start. Second, we note that models for non-affine operators (e.g., multiplication (10) or division (11)) require approximations. In the case of a long chain of exclusively non-affine operations, the final estimate might again be overly pessimistic. Luckily, this seems to be rare in our experience in common DSP applications. Finally, our static analysis methods can estimate error upper bounds , moments of the error, and their functions. But for DSP applications that rely on measurements not closely related to the error, such as the convergence rate in adaptive filtering, or the perceptual quality in audio processing, our methods must be augmented with transformations that map these quality criteria back into specific errors that we can track.

5. CONCLUSION

We extended the work of [2] and developed new static analysis methods to accurately analyze finite-precision effects that arise from fixed-point implementation of DSP tasks. The technique exploits advances in interval representation methods from affine arithmetic [6] to better capture correlations among dependent operands, and the power of a new probabilistic bounding method to dramatically reduce the pessimism of error estimation. The combination lets us analyze larger designs and feedback systems accurately that were not previously tractable by any symbolic analysis methods. This technique offers error estimates comparable to statistical simulation, with four to five orders of magnitude speedup. Our ongoing work focuses on studying error modeling of non-affine functions, such as sin, cos, exp, etc. and extending static error analysis to even more complex designs.

6. REFERENCES

[1] A. Benedetti and P. Perona. Bit-width optimization for confi gurable DSP's by multi-interval analysis. In *34th Asilomar Conference on Signals, Systems, and Computers*, 2000.

[2] C. F. Fang and R. A. Rutenbar and M. Püschel and T. Chen. Towards effi cient static analyis of fi nite precision effects in DSP applications via affi ne arithmetic modeling. In *Design Automation Conference*, 2003.

[3] M. L. Chang and S. Hauck. Précis: A design-time precision analysis tool. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002.

[4] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP asic fi xed-point refi nement. In *Design, Automation and Test in Europe Conf.*, 1999.

[5] L. D. Coster, M. Ade, R. Lauwereins, and J. Peperstraete. Code generation for compiled bit-true simulation of DSP applications. In *International Symposium on System Synthesis*, 1998.

[6] L. H. de Figueiredo and J. Stolfi . Self-validated numerical methods and applications. *Brazilian Mathematics Colloquium monograph, IMPA, Rio de Janeiro, Brazil*, July 1997.

[7] F. Fang, T. Chen, and R. Rutenbar. Lightweight fbating-point arithmetic: Case study of inverse discrete cosine transform. *EURASIP J. Sig. Proc.; Special Issue on Applied Implementation of DSP and Communication Systems*, 2002(9):879–892, Sept. 2002.

[8] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: A fi xed-point design and simulation environment. In *Design, Automation and Test in Europe Conf.*, 1998.

[9] S. Kim and E. A. Lee. Infrastructure for numeric precision control in the ptolemy environment. In *40th Midwest Symposium on Circuts and Systems*, 1997.

[10] W. Kramer. A priori worst case error bounds for fbating-point computations. *IEEE Trans. Comp.*, 47:750–756, July 1998.

[11] K. I. Kum and W. Sung. Combined word-length optimization and high-level synthesis of digital signal processing systems. *IEEE Trans. Computer Aided Design*, 20(8), Aug. 2001.

[12] A. Lemke, L. Hedrich, and E. Barke. Analog circuit sizing based on formal methods using affi ne arithmetic. In *International Conf. on Computer Aided Design*, Nov. 2002.

[13] B. Liu. Effect of fi nite word length on the accuracy of digital fi lters - a review. *IEEE Trans. Sig. Proc.*, 18, Nov. 1971.

[14] D. Menard and O. Sentieys. Automatic evaluation of the accuracy of fi xed-point algorithms. In *Design, Automation and Test in Europe Conf.*, 2001.

[15] D. S. Moore and D. P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman and Company, New York, 1989.

[16] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[17] A. Nayak, A. C. M. Haldar, and P. Banerjee. Precision and error analysis of matlab applications during automated hardware synthesis for FPGAs. In *Design, Automation and Test in Europe Conf.*, 2001.

[18] http://www.systemc.org.

[19] S. A. Wadekar and A. C. Parker. Accuracy sensitive word-length selection for algorithm optimization. In *International Conf. on Computer Design*, 1998.