Efficient Static Analysis of Fixed-Point Error in DSP Applications via Affine Arithmetic Modeling

Claire F. Fang, Rob A. Rutenbar, Tsuhan Chen
{ffang, rutenbar, tsuhan}@ece.cmu.edu
Electrical and Computer Engineering
Carnegie Mellon University

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 several common DSP kernals.

1. INTRODUCTION

Modern digital signal processing DSP applications are typically prototyped using standard floating-point arithmetic, which offers both large dynamic range and high precision for each numerical computation. However, for hardware implementations, the final form rarely uses a full-precision floating-point unit, given issues of silicon area, power, and speed. This creates the common—and still awkward—problem of transforming the application from its initial "infinite" precision form, into finite-precision fixed-point format.

The two-part problem for any complex DSP task is how to choose the smallest bit-level number formats, and then how to validate that the format choices maintain the necessary level of numerical precision. A commonn solution is to reply on detailed simulations to find the dynamic range and precision of each operand. However, if we choose a sequence of input patterns that are too few or incorrectly distributed, we may fail to find all the extreme values that real-life use will encounter. If we employ instead a more rigorous detailed simulation strategy, the format optimization process becomes unduly expensive.

A more attractive solution is *static analysis*, which guarantees that we will find the extreme values and the maximum error of each operand, but does not require us to consider a potentially unbounded set of input patterns. The obvious approach for such a static analysis draws on techniques from *interval arithmetic* [3], which replaces each scalar value with a bounded interval on the real number line. An algebra of intervals lets us "simulate" the code by performing each operation on the intervals. Unfortunately, conventional interval arithmetic suffers from the problem of range explosion: as more computations are evaluated, the intervals tend to grow without bound.

A recent solution to this problem is a more sophisticated interval model called *affine arithmetic*, introduced in [2]. The approach explicitly captures the correlations among operands, and dramatically reduces the level of pessimism in the final intervals. We apply the idea to the novel problem of finite-precision error estimation for reduced-precision custom floating point formats: we seek to efficiently estimate, for each operand, the maximum *difference* (the error) between a finite-precision implementation and a reference "ideal" version using standard floating point.

The remainder of the paper is organized as follows. Section 2 gives relevant background on affine arithmetic. Section 3 formulates our error analysis model for fixed-point format and offers a refinement based on probabilistic bounds. Section 4 present experimental results for several DSP benchmarks showing the viability of the approach. Finally, Section 5 offers concluding remarks.

2. BACKGROUND

2.1 Interval Arithmetic

Interval arithmetic (IA), also known as *interval analysis*, was invented in the 1960s by Moore [3] 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 [2]. Fixed-point error analysis based on IA is presented [1, 4]. 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 [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

$$\widehat{x} = x_0 + x_1\varepsilon_1 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, \quad -1 \leqslant \varepsilon_i \leqslant 1. \qquad (2)$$

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 [2].

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.

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.

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:

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

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:

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

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, shown as:

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

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. 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):

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

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 $2^{-(f+1)}$. Models on each affine operation are detailed as follows:

<u>Addition:</u>

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

<u>Addition with a constant:</u>

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

Multiplication with a constant:

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

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 Multiplicaiton

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.

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

This product 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 [2], and by using the same *range estimation*, we can approximate Q_1 and Q_2 to linear terms.

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

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).

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

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 $\hat{x} = x_0 + \sum_{i=1}^{n} x_i$ and $\hat{y} = y_0 + \sum_{i=1}^{n} y_i$.

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

According to [2], $\frac{\hat{x}}{\hat{y}}$, $\frac{\hat{x}}{\hat{y}^2}$, $\frac{1}{\hat{y}}$ can all be turned into affine forms using *Chebyshev* approximation, denoted by A_1, A_2, and A_3, respectively. 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

Our AA-based static analysis 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 get very large, it becomes extremely unlikely that all the ε_i's simultaneously take the extreme value 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*, 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.

4. EXPERIMENTAL RESULTS

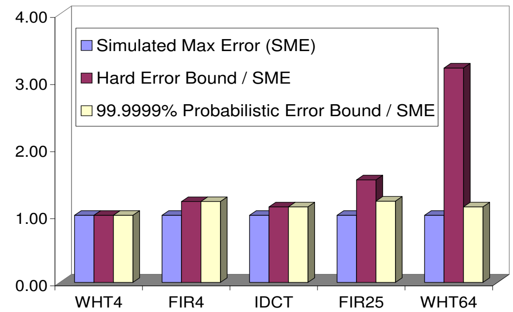
We test the applicability and accuracy of the proposed error model and the probabilistic bounding method on a variety of common signal processing kernels, including WHT (Walsh-Hadamard Transform), FIR (Finite Impulse Response) filter, and IDCT. We assume 16 bits for the fractional part in all the experiments. Since the focus of this paper is the quantization error, 48 bits for the integral part is chosen to assure no overflow occurs. The input range is assumed to be in [-64, 64] for all the kernels. Figure 1 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 the computational complexity. First, the hard error bound always provides an upper bound for the maximum error. However, as the computational complexity increases, this bound becomes looser (see WHT64). Our probabilistic bounding method offers a close estimation of the maximum error, regardless of the 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

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

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.

What makes our static error analysis attractive is not only the accuracy, but also the computational efficiency. Table 1 shows the CPU times needed to compute the estimated bound and the simulated maximum error, the latter estimated from 10^6 input vectors. For these DSP kernels, our static analysis achieves at least five orders of magnitude speed up compared to the simulation.



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

Figure 1: 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 1: Comparison of CPU time (sec)

5. CONCLUSION

We 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 to better capture correlations among dependent operands, and the power of a new probabilistic bounding method to dramatically reduce the pessimism of the estimated error. 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 extending static error analysis to even more complex designs.

REFERENCES

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

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

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

[4] 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.