An Automated, Efficient and Static Bit-width Optimization Methodology Towards Maximum Bit-width-to-Error Tradeoff With Affine Arithmetic Model

Yu Pu and Yajun Ha
Department of Electrical and Computer Engineering
National University of Singapore, Singapore, 119260
{Yu_Pu, elehy}@nus.edu.sg

Abstract – Ideally, bit-width analysis methods should be able to find the most appropriate bit-widths to achieve the optimum bit-width-to-error tradeoff for variables and constants in high level DSP algorithms when they are implemented into hardware. The tradeoff enables the fixed-point hardware implementation to be area efficient but still within the allowed error tolerance. Unfortunately, almost all the existing static bit-width analysis methods are *Interval Arithmetic* (IA) based that may overestimate bit-widths and enable fairly pessimistic bit-width-to-error tradeoff. We have developed an automated and efficient bit-width optimization methodology that is *Affine Arithmetic* (AA) based. Experiments have proven that, compared to the previous static analysis methods, our methodology not only dramatically reduces the fractional bit-width by more than 35% but also slightly reduces the integer bit-width. In addition, our probabilistic error analysis method further enlarges the bit-width-to-error tradeoff.

I. Introduction

Most of the computational-intensive applications are initially developed with high-level standard programming languages where the variables and constants are possible to be described in high precision data types. While being implemented into hardware subjected to fixed-point restriction, these variables and constants prototyped in high precision must be re-defined to the fixed-point. The translation process from full to limited precision must be optimized to tradeoff precision for silicon area, latency, power consumption and etc. Unfortunately, choosing the most appropriate bit-width is a non-trivial task. Moreover, it is too burdensome for designers to translate them manually.

Many researchers have focused their interests in developing tools that may help decide the bit-widths automatically. One category of techniques is based on detailed simulations. Kum et. al. [7] have used Monte Carlo-style statistical simulation to optimize word-length iteratively. This technique can get the optimum results but is quite inefficient since there is often a huge searching space to simulate with a full coverage of input vectors. Chang et al. [2] have developed a design-time tool, *Precis*, which provides designers with area-precision information by doing slack analysis and aids the designer in focusing their manual precision optimization. They have also presented a method that broadens their work in [2] to explore area-to-error tradeoffs by precision steering [3]. However their technique is semi-automated. The other category of techniques is based on static analysis. Stephenson et. al. [1] have formulated the bit-width analysis as a value range propagation problem and introduced a compiler *Bitwise* that extracts the value of each variable by seamlessly performing bi-directional propagation, but their compiler is limited to integers and pointers. Nayak et. al. [4] have used a precision analysis based on the method presented in [1] to infer the minimum number of bits for the integer part and a combined precision and error analysis to infer the fractional bit-width. However, their algorithm is restricted by the assumption that the number of fractional bits is constant for all variables.

We observe that almost all the existing static bit-width optimization methods are *Interval Arithmetic*(IA) based, which may lead to the pessimism because the overestimation accumulates exponentially along the computation path. Fang et. al. [5][6] have introduced the *Affine Arithmetic* model into the verification of the finite-precision effects in DSP applications. Inspired by their work, we have extended the AA model to the bit-width analysis. A closely related but independent research has been simultaneously conducted by Lee et. al. [8], but their method uses the adaptive simulated annealing algorithm to find the optimum number of fraction bits, which is quite time consuming. Compared to their methodology, ours has the following features:

- We use the hard error analysis to insure the output error not to exceed the designer-specified error tolerance.
- In most of the DSP designs, the designer allows certain degree of error rate, which motivates us to explore a larger bit-width-to-error tradeoff than using hard error analysis. Our probabilistic error analysis almost insures that the probability for the output error to lie in the specified error tolerance is higher than the designer specified parameter λ.
- Decision time is greatly reduced since our approach iterates only once.
- Our approach performs on the algorithm level, followed by high-level synthesis which is based on the bit-width optimization results. Therefore, our approach can be easily incorporated into the existing synthesis tools.

The remainder of the paper is organized as follows: Section II briefly gives the background of IA and AA models. Section III goes through our bit-width analysis methodology in detail. Section IV gives and analyzes the experimental results. Section V draws the conclusion of the work.

II. Background

0-7803-9451-8/06/$20.00

Interval arithmetic has been widely used in the range propagation analysis. However, IA model could lead to overestimations and such overestimations are exponentially accumulated along a computation path. As a consequence, the final intervals may be too large to be useful.

Affine Arithmetic (AA) model can overcome the overestimation explosion problem. In AA model, the uncertain variable x is expressed as:

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n, \quad -1 \le \varepsilon_i \le 1$$

where x_0 is the *central value* of the affine form of $\hat{x}$, ε_i is an independent *noise symbol* multiplied by the corresponding *coefficient* x_i. Along the computation path, one symbol of ε_i may contribute to be uncertainties of two or more variables. When these variables are combined, the uncertainties may be cancelled out so as to make the results of range propagation analysis tighter than that of IA model.

In order to present the mathematical reasoning behind our methodology, we briefly introduce the AA based operation models. Assume that there are variables x, y and constant c, we have $x = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + E_x$, $y = y_0 + \sum_{i=1}^{n} y_i\varepsilon_i + E_y$, and $c = c_0 + E_c$, where E_x, E_y and E_c are the error terms generated from quantization. Each of the error terms is expressed in the affine form as $E = |\Delta|\varepsilon_e$, where $|\Delta|$ is the upper bound for the quantization error. If we denote f as the bit-width for the fractional part, we have $|\Delta| = 2^{-f}$ in the case of truncation and $|\Delta| = 2^{-(f+1)}$ in the case of rounding. The noise symbol ε_e indicates the uncertainty of the quantization error.

The AA operation models for the addition, subtraction and multiplication are shown below. With the AA form operation models, the results are also in affine forms. The error that occurs after each operation is linear with respect to errors of its two operands and it shifts with a new quantization error σ.

Addition and Subtraction:

$$x \pm c = (x_0 \pm c_0) + \sum_{i=1}^{n} x_i\varepsilon_i + \Phi,$$

Where $\Phi = E_x \pm E_c + \sigma$

$$x \pm y = (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\varepsilon_i + \Phi,$$

Where $\Phi = E_x \pm E_y + \sigma$

Multiplication:

$$x \cdot c = c_0 x_0 + c_0 \sum_{i=1}^{n} x_i\varepsilon_i + \Phi,$$

Since we know that $E_x \in [-1,1]$ and $E_c \in [-1,1]$, so $E_x E_c \le E_x^2 + E_c^2/2 \le (|E_x| + |E_c|)/2$ then we have

$$\Phi \le E_x \cdot c_0 + E_c(x_0 + \sum_{i=1}^{n} x_i\varepsilon_i) + (|E_c| + |E_x|)/2 + \sigma$$

Similarly,

$$x \cdot y = x_0 y_0 + \sum_{i=1}^{n} (y_0 x_i + x_0 y_i)\varepsilon_i + \sum_{i=1}^{n} |x_i| \sum_{i=1}^{n} |y_i| \varepsilon_k + \Phi$$

$$\Phi \le E_y(x_0 + \left|\sum_{i=1}^{n} x_i\right|) + E_x(y_0 + \left|\sum_{i=1}^{n} y_i\right|) + (|E_x| + |E_y|)/2 + \sigma$$

A quantity in the affine form can be bounded by

$$x = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i \le x_0 + \sum_{i=1}^{n} |x_i| = |x| \qquad (1)$$

where the term $\sum_{i=1}^{n} |x_i|$ is called the *total deviation* of x.

III. Bit-width Analysis Methodology

Figure 1 gives a brief overview of our methodology. First, we accept the designer-specified constraints via a user-interface, and convert the input range into the AA format. Second, we run the program "symbolically" to get the output results in AA form. Third, we use the range analysis to determine the range of each variable (or constant) and the corresponding integer bit-width. The subsequent error analysis will return the optimized fractional bit-width.

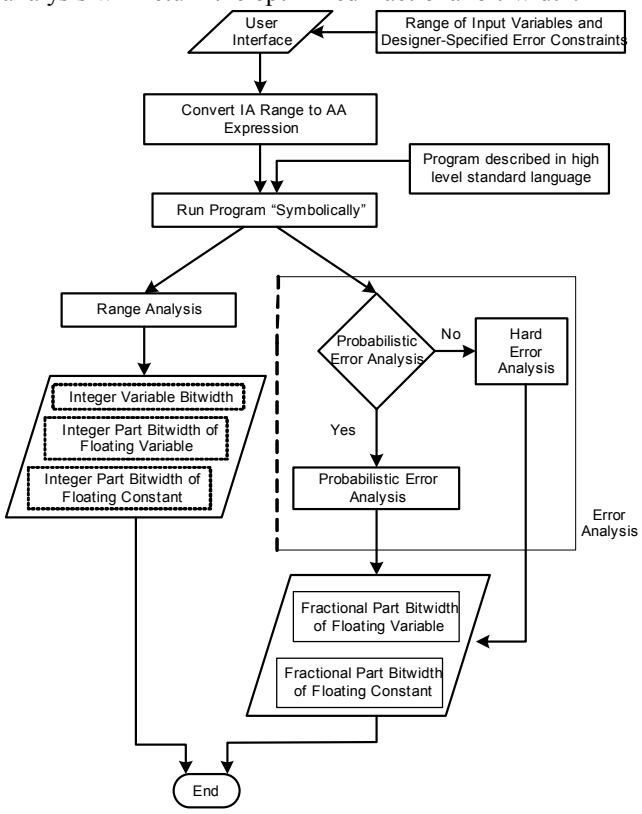

Fig. 1. Bit-width Optimization Flow

Information Provided by Designer:

To limit the complexity, we implemented our method using the multiple-input to single-output computational

model. The designer provides the following 4 types of information using the interface shown in Figure 2: (1) the value range of all the input variables; (2) the output variable; (3)the absolute maximum error tolerance E_{spec} at the output; (4)optionally the probability parameter λ .

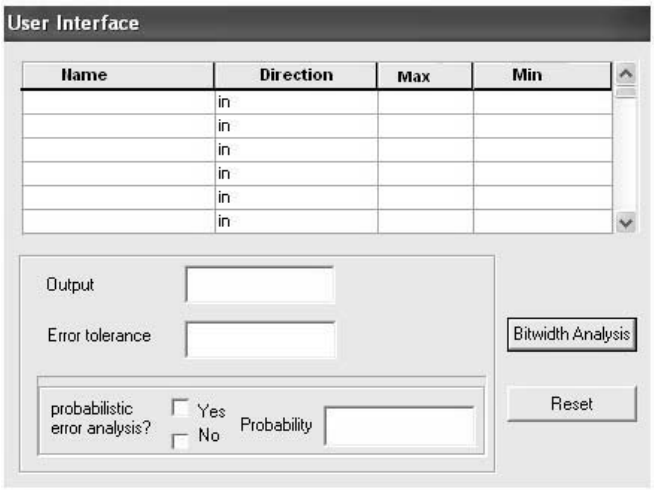

Fig. 2. The User Interface

Convert IA Range to AA Range:

Since the ranges of input variables given by designers are in the form of interval range, for example, an input variable x is ranged in $[a,b]$. We first convert them to the AA form, so x is expressed as $x=(a+b)/2+(a+b)\varepsilon_x/2$, $(-1\le\varepsilon_x\le 1)$, and ε_x is the noise symbol introduced for representing the uncertainty of x .

Run Program "Symbolically":

We "symbolically" traverse the whole program from the entry to the exit in *forward* direction. Assume that there are totally N floating-point variables and constants whose upper quantization error bounds are expressed as $|\Delta_1|,|\Delta_2|,|\Delta_3|\cdots|\Delta_N|$, respectively. When they are transformed to be fixed-point, the actual quantization errors incurred at each of them are expressed in affine forms as $|\Delta_1|\varepsilon_1,|\Delta_2|\varepsilon_2,|\Delta_3|\varepsilon_3\cdots|\Delta_N|\varepsilon_N$. With the operation models outlined in the background part, we get the results in symbolical affine form for all variables and floating-point constants.

There are some rules of running the program:

- For loop construct whose loop count has been prescribed, the variables in the body of the loop are calculated by thoroughly traversing the loop.
- For a variable that is condition dependent, the variable's range is refined based on the outcome of the conditional branch. For example, in Figure 3 the variable x is condition dependent, so at the output of the condition branch($x\le 0?$), the range of x is refined with newly introduced affine form expressions.
- We do not do *backward* propagation because the affine form itself is not an efficient form for performing backward propagation.

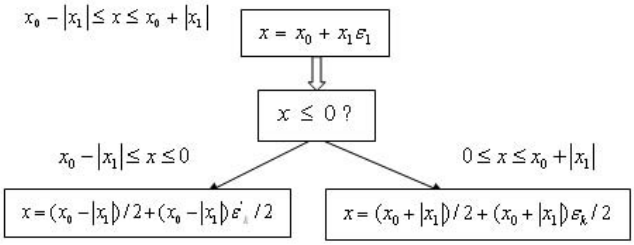

Fig. 3. Range Refinement of Condition Dependent Variable

Range Analysis:

After "symbolically" running the program, we get all the variables and floating-point constants in the affine forms. By Equation (1), their ranges are bounded. Hence, the bit-width of the integer variables, the bit-width for the integer part of the floating-point variables and the floating-point constants are determined. Assume that the quantity x (either variable or constant) is translated to be fixed-point with m bits to be its integer part bit-width, we obtain the minimum m by the following expression:

$$2^{m-1}-1\ge|x| \tag{2}$$

Error Analysis:

After "symbolically" running the program, we also get the error of the output variable. This output error is in the affine form as

$$\Delta_{output}=A_1|\Delta_1|\varepsilon_1+A_2|\Delta_2|\varepsilon_2+A_3|\Delta_3|\varepsilon_3+\cdots+A_N|\Delta_N|\varepsilon_N \tag{3}$$

$$=|A_1||\Delta_1|\varepsilon_1'+|A_2||\Delta_2|\varepsilon_2'+|A_3||\Delta_3|\varepsilon_3'+\cdots+|A_N||\Delta_N|\varepsilon_N' \tag{4}$$

$$\le|A_1||\Delta_1|+|A_2||\Delta_2|+|A_3||\Delta_3|+\cdots+|A_N||\Delta_N| \tag{5}$$

Each independent term $A_i|\Delta_i|\varepsilon_i$ in Equation (3) tells the contribution that a floating point variable or constant's quantization error makes to the overall error at the output, where A_i is an affine form expression. Since the designer-specified output error tolerance E_{spec} is the upper bound of $|\Delta_{output}|$, we have:

$$|\Delta_{output}|\le E_{spec} \tag{6}$$

And we assign non-negative weights to each term in Equation (5):

$$\sum_{i=1}^{i=N}W_i|A_i||\Delta_i|=|\Delta_{output}| \tag{7}$$

$$\sum_{i=1}^{N}W_i=1 \tag{8}$$

Through Equations (5) (6) and (7), once W_i is known, we can infer $|\Delta_i|$. At this moment, we simply assign each term with a equal weight, so $|A_i||\Delta_i|\le E_{spec}/N$ (9)

Now we have $|\Delta_i|\le Espec/(N\cdot|A_i|)$, and each variable's

fractional bit-width $f_1, f_2, ... f_N$ can be decided:

If the quantization is real rounding based,

$$2^{-(f_i+1)} \le |\Delta_i| \Leftrightarrow f_i \ge -\log_2 |\Delta_i| - 1 (f \ge 0) \quad (10)$$

if the quantization is truncation based,

$$2^{-f_i} \le |\Delta_i| \Leftrightarrow f_i \ge -\log_2 |\Delta_i| (f \ge 0) \quad (11)$$

For the experimental results presented in this paper, we employ real rounding.

To further explore the bit-width-to-error tradeoffs, after $f_1, f_2 ... f_N$ are obtained, we use the following "greedy algorithm" to optimize the bit-width:

1. Sort $f_1, f_2, ... f_N$ and map them to a new array $X[N]$ according to their decreasing order.

2. for (j=0;j++j<N)
{
for (i=0;i++;i<N)
{
$E_{residue} = Espec - \sum_{i=1}^{N} 2^{-(X[i]+1)}$;
}
$2^{-(X[j]+1)} = 2^{-(X[j]+1)} + (E_{residue} / |A_K|)$;
// assume that f_k is mapped to $X[j]$
Recalculate $X[j]$ → Recalculate f_k ;
}

We call the aforementioned fractional bit-width analysis method as the "hard" error analysis because it fully insures the output error not to exceed the designer-specified error tolerance.

However, we note that in Equation (4) when N gets large, it is quite unlikely for all the ε_i' to simultaneously take extreme values and push the output error to its upper bound. [6] has introduced a probabilistic model to estimate the error of a affine form variable. Our probabilistic error analysis method is partially based on their idea.

Since we have $|A_i||\Delta_i| = |A_i||\Delta_i| = \cdots = |A_i||\Delta_i| = k$, so Equation (4) depicts a sum of many statistically independent and identically distributed terms. By the *central limit theorem*, the Equation (4) approaches a Gaussian CDF as the parameter N increases. We have

$$\frac{\Delta_{output}}{\sqrt{N}\sqrt{Variance}} \to \mathrm{N}(0,1) \quad (12)$$

Where $\mathrm{N}(0,1)$ is a standard normal distribution, and the

$$Variance = k^2/3 = (|A_i||\Delta_i|)^2/3 \quad (13)$$

We have the designer-specified parameter λ , which bounds the probability that the output error lies within the designer-specified error tolerance. For example, if the designer sets $\lambda = 0.999$, then the chance for the output error to exceed E_{spec} is only once in 10^3 times of simulation.

$$prob(|\Delta_{output}| \le E_{spec}) \ge \lambda \quad (14)$$

The probabilistic error analysis is quite straightforward:

Step 1:

Look up the Gaussian approximation statistical table according to λ . Since $|\Delta_{output}|$ and N are known, we can get the variance and k by Equation (12).

Step 2:

Determine $|\Delta_i|$ by Equation (13), so f_i can be determined.

IV. Experimental Results and Analysis

We use an example to prove the strength of our methodology. The signal flow of a butterfly part in IDCT is shown in Figure 4 and such kinds of computations are common in DSP. To explain explicitly, the data flow graph is shown in Figure 5. We set $Const_1 = Const_2 = \sqrt{2}/2$. Since C/C++ is the most popular general purpose language [13-15], the algorithm is initially described in C++ with all the variables and constants set to be in 64-bit long double precision.

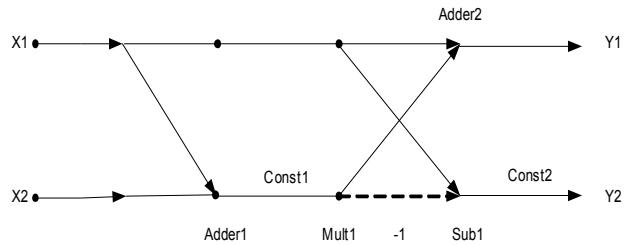

Fig. 4. Signal Flow Graph

We apply our bit-width analysis methodology to this example. We first provide the value range of all the input variables X_1 and X_2 : $X_1 \in [-128,127]$ and $X_2 \in [-128,127]$, then specify the maximum error tolerance at Y_2 to be 1.0. In probabilistic error analysis, the probability λ is 0.999. We compared our bit-width optimization results with the results generated from using the method presented in [4] which is considered so far as the best fully automated IA based static bit-width analysis approach to optimize the word-length for the algorithms described in MATLAB.

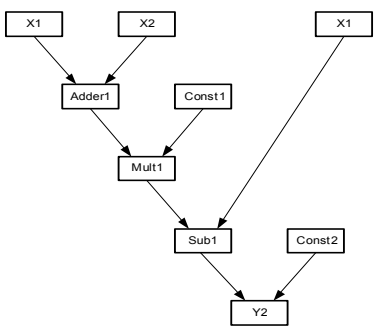

Fig. 5. Data Flow Graph

Bit-width optimization results and Analysis:

TABLE I
Bit-Width Analysis Results

Node		X_1	X_2	$Adder_1$	$Const_1$	$Mult_1$	Sub_1	$Const_2$	Y_2
Integer	IA	8	8	9	1	9	10	1	9
	AA	8	8	9	1	9	8	1	8
Fracti-onal	IA	8	8	8	8	8	8	8	8
	Hard	0	3	3	9	3	3	8	3
	Prob.	0	2	2	10	2	2	9	2

Table I lists the bit-width analysis results. For the integer part, we see that at nodes Sub_1 and Y_2, the IA based range analysis cannot consider the range cancellation among variables such that the pessimistic results are occurred. With our AA based method, the integer bit-width can be optimized. Considering the efficiency reflected by this simple example, it is predicted that in a large scale computation where many variables may have correlations, our method can save considerable integer bit-width. For the fractional part, we see that IA based bit-width analysis returns a large overestimation in terms of the bit-width. At most of the nodes, our method results in much less bit-width than the IA method. However, we notice that at the nodes $Const_1$ and $Const_2$ our method results in a little larger or equal bit-width compared to the IA based method. Note that in the current stage we simply assign the same weight to each node, but actually we can adjust the weights. For example, we assign larger weights to the nodes who need more bit-width, such as $Const_1$ and $Const_2$, and the bit-width for these nodes can be reduced while the bit-width of other nodes only suffer slight increase.

With the similar approach used for the first example, we have experimented on a 4th order polynomial and a FIR low-pass filter to test our methodology. The polynomial equation in our benchmark is $y = x^4 + x^3 + x^2 + x + 1$, where x is initialed as a floating point variable and $x \in [-16,15]$. The FIR filter is implemented according to the architecture shown in Figure 6, where we set all X_n to be in the floating format, $X_n \in [-64,63]$ and the floating constants a0 to a4 to be in *long* format obtained from the command *firpm()* in MATLAB. For each example, we specify the error tolerance to be 1.0.

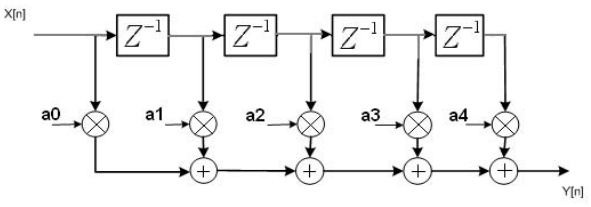

Fig. 6. Architecture of the FIR filter

The average bit-widths for the fractional part of these benchmarks are listed in Figure 7. They are normalized with respect to the IA based fractional bit-width. We see that with our hard error analysis, we can achieve a more than 35% resource saving in terms of the bit-width for the fractional part compared to the IA based fractional bit-width analysis. Therefore, our method has achieved a much larger bit-width-to-error tradeoff than previous IA based method. In addition, with our probabilistic error analysis, the bit-width can be even reduced by 50%. As we gradually release the probabilistic restriction, the tradeoff keeps on going upwardly.

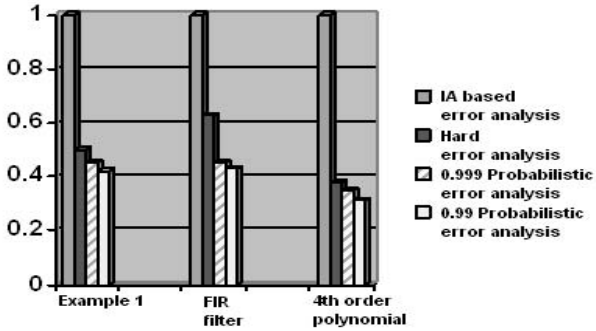

Fig. 7. Average Bit-width for Fractional Part

Verification:

To verify the correctness of our method, for the hard error analysis, we check whether the maximum output error lies within the specified error tolerance after implementing the algorithms into hardware that is subjected to fixed-point restriction. For the probabilistic error analysis, we check that whether the probability for the output error to lie in the specified error tolerance can be higher than the probability specified by the designer. We execute the program written in C++ with 2×10^3 random input data sets that are uniformly distributed within their ranges. Then we modify the C++ programs linking with the compiled SystemC library, set the quantization option to be real rounding and set each variable to be fixed-point data type with the bit-width calculated by our method. We use the same input data sets to get the results. We compare the results to obtain the simulated maximum output error and the simulated probability that the error lies in the specified error. The experimental results for all the examples are shown in Table II.

TABLE II
Verification Results

Test-bench	Bit-width analysis methods	Simulated output error	Specified probability λ	simulated probability λ'
Example1	IA based	0.02	N/A	1
	AA based	0.22	N/A	1
		0.47	0.99	1
		0.33	0.999	1
FIR filter	IA based	0.09	N/A	1
	AA based	0.15	N/A	1
		0.58	0.99	1
		0.58	0.999	1
4th order Poly-nomial	IA based	0.88	N/A	1
	AA based	0.93	N/A	1
		1.10	0.99	0.999
		0.95	0.999	1

In Table II, the 3rd column gives the simulated maximum

output error. The 4th column gives the parameter λ, which is the designer-specified probability for the output error to lie in the specified error tolerance. Note that when the parameter is not available (N/A), we use the hard error analysis. The 5th column gives the simulated probability λ'. Table II shows that, while our hard error analysis fully insures that the output error will not exceed the designer specified error, the probabilistic error analysis can almost guarantee that the probability for the output error to lie in the specified error will be higher than the specified probability. However, since we have used Gaussian approximation during the analysis, theoretically it is difficult to fully guarantee the error probability to be bounded by a designer, even in our experiments it works quite well. Therefore, we suggest that the specified probability should be flexibly restricted.

V. Summary and Conclusions

We have presented an automated and efficient static bit-width optimization methodology which is based on the affine arithmetic model. While our AA based range analysis can slightly reduce the integer part bit-width, the AA based hard error analysis can dramatically reduce the fractional bit-width. In addition, we have proposed the probabilistic error analysis method which can further shorten the bit-width. Our experimental results have proven that our approach can explore a larger bit-width-to-error tradeoff.

However, we also note the limitations of our methodology. First, we bear the same drawback as mentioned in [6] that we assume that all the input variables are independent while this is not always the case. The future work can model the correlations explicitly from start. Second, our method lies at the algorithm level which does not consider the hardware sharing and hardware cost function. Our ongoing research will further explore the solutions for these problems.

References

[1] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," *Proceedings of the SIGPLAN conference on Programming Language Design and Implementation*, June 2000.

[2] Mark L. Chang and S. Hauck, "Precis: A design-time precision analysis Tool," *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp.229-238, 2002.

[3] M. L. Chang and S. Hauck, "Variable Precision Analysis for FPGA Synthesis," Nasa Earth Science Technology Conference 2003.

[4] A. Nayak, M. Haldar, A. Choudhary and P. Banerjee, "Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs," *Design Automation & Test*, March 2001.

[5] C. F. Fang and R. A. Rutenbar and M.Puschel and T. Chen, "Towards efficient static analysis of finite precision effects in DSP applications via affine arithmetic modeling," *Design Automation Conference*, 2003.

[6] C. F. Fang, R. A. Rutenbar, M. Puschel and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," *Proceedings of the International Conference on Computer Aided Design* (ICCAD'03).

[7] K. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Trans. on Computer-Aided Design*, Aug, 2001.

[8] D-U. Lee, A. A. Gaffar, O. Mencer, W. Luk, "MiniBit: bitwidth optimization via affine arithmetic," *Proceedings of the 42nd annual conference on Design automation*.

[9] S. Kim, K.-I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *Workshop on VLSI and Signal Processing*, Osaka, 1995.

[10] N. Shirazi, A. Walters and P. Athanas, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines," *IEEE Symposium on FPGAs for Custom Computing Machines*, April, 1995.

[11] J. Stolfi and L.H. de Figueiredo, "An introduction to affine arithmetic," *TEMA Tend. Mat. Apl. Comput.,* 4, Vol.3, pp. 297-312, 2003.

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

[13] D. Galloway, "The Transmogrifier C Hardware Description Language and Compiler for FPGAs," *FCCM'95*

[14] G. Doncev, M. Leeser and S. Tarafdar, "High-Level Synthesis for Designing Custom Hardware," *Proc. Field-Programmable custom Computing Machines*, April 1998.

[15] B. L. Hutchings and B. E. Nelson, "Using General - Purpose Programming Languages for FPGA Design," *Proc.37th Design Automation Conference*, June 2000.

[16] Mark Stephenson, "Bitwise: Optimization Bitwidths Using Data-Range Propagation," Master's thesis. Massachusetts Institute of Technology, May 2000.

[17] K. Bondalapati and V. K. Prasanna, "Dynamic precision management for loop computations on reconfigurable architectures," *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1999.

[18] W. Sung and K. I. Kum. "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Transactions on Signal Processing*, vol. 43, no.12, pp.3087-3090, December 1995.

[19] J. Patterson, "Accurate Static Branch Prediction by Value Range Propagation," *Proc.of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 67 – 78, June 1995.

[20] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP asic fixed-point refinement," *Design, Automation and Test in Europe Conf.*, 1999.