# Optimization of Imprecise Circuits Represented by Taylor Series and Real-Valued Polynomials

Yu Pang, *Member, IEEE,* Kartazyna Radecka, *Member, IEEE,* and Zeljko Zilic, *Senior Member, IEEE*

***Abstract*—Arithmetic circuits in general do not match specifications exactly, leading to different implementations within allowed imprecision. We present a technique to search for the least expensive fixed-point implementations for a given error bound. The method is practical in real applications and overcomes traditional precision analysis pessimism, as it allows simultaneous selection of multiple word lengths and even some function approximation, primarily based on Taylor series. Starting from real-valued representation, such as Taylor series, we rely on arithmetic transform to explore maximum imprecision by a branch-and-bound search algorithm to investigate imprecision. We also adopt a new tight-bound interval scheme, and derive a precision optimization algorithm that explores multiple precision parameters to get an implementation with smallest area cost.**

***Index Terms*—Arithmetic imprecision, arithmetic transform, fixed-point arithmetic, optimization, polynomials, Taylor series.**

## I. Introduction

ARITHMETIC circuits introduce additional complexity to the already challenging design process. As implementations only approximate the specification, the precision verification and optimization of a given function requires an exploration of the imprecision in the whole domain of the function definition.

It is generally accepted that an imprecision error measured as an arithmetic difference between an implementation and the specification is deemed suitable if it lies within an acceptable error bound. This *margin of implementation correctness* causes many leading verification methods, such as equivalence checking, to be inadequately equipped in cases of imprecise arithmetic implementations. Formal methods have not been helpful for verification with imprecision [1], not to mention that the binary decision diagrams are not appealing for arithmetic functions [2], as well as a variety of bit-level representations.

From the design perspective, however, the imprecision can provide yet another optimization resource, similar in nature to the notion of "don't cares" in logic synthesis. In particular, as implementations match specifications within error bounds, one can search for the least expensive implementation within the allowed imprecision. Fig. 1 illustrates two implementations, *Imp1* and *Imp2*, differing from the specification Spec by errors$e1$ and $e2$, respectively. If the error is within the allowed error bound $E$, then any such implementation is deemed acceptable, but the least costly one is preferable.

As looking into individual bits is not conducive to exploring arithmetic imprecision, suitable forms need to be derived such that they involve *word-level* rather than bit-level function values. One such representation is arithmetic transform (AT), introduced in Definition 2, which we use to reason not only about the precision, but also as a link to real-valued specifications.

Infinite-length Taylor series are a typical real-valued function representations of transcendental and other functions of importance in engineering. Their implementations are inherently approximated, as only a finite number of terms can be realized in hardware. For instance, a common transcendental function $\sin(X)$ is realized by a fixed number of terms of its Taylor series expansion

$$\sin(X) = X - X^3/3! + X^5/5! - X^7/7! + \cdots.$$

Depending on the required precision, the number of terms and the word lengths of inputs/outputs and parameters will vary. For instance, one method can yield a six-term, 16-bit inputs floating-point approximation of $\sin(X)$, while the others ascertain that there is an implementation of $\sin(X)$ by five terms and 18-bit words within the given error bound. The worthy goal is to find the combination of the approximation and precision parameters leading to the optimal implementation. We will primarily be concerned with the area minimization, but the speed and power performance will also gain from the precision optimization.

Various methods have been derived to reason about the approximate realizations of real-valued functions. Traditionally, one mostly relies on *dynamic* methods, which employ simulations to analyze the imprecision between the specification and the implementation [3]–[8]. In this paper, we propose a new AT-based method, which performs the *static analysis* in ascertaining whether the existing implementation is in agreement with the specification. Our approach will explore multiple precision parameters concurrently.

A substantial effort has been directed toward finding adequate bit-width parameters for imprecise fixed-point circuits. Much of the common engineering practice consists of undertaking a sequence of bit-width decisions, leading to the over-estimate of the imprecision and, consequently, to suboptimal solutions. Additionally, the explorations of function approximations are most commonly not an integral part of the precision optimization.

Manuscript received October 23, 2009; revised February 4, 2010. Date of current version July 21, 2010. This paper was recommended by Associate Editor R. Camposano.

The authors are with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 2K6, Canada (e-mail: yu.pang@mail.mcgill.ca; katarzyna.radecka@mcgill.ca; zejko.zilic@mcgill.ca).

Digital Object Identifier 10.1109/TCAD.2010.2049154

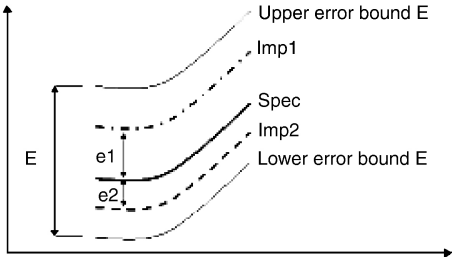


Fig. 1. Comparison of two implementations.

By using real-valued representations such as Taylor series, and having the ability to investigate statically the difference between implementations, we propose a simple scheme to optimize imprecise fixed-point circuits. Toward that, we rely on the earlier demonstrated scheme [20] to efficiently convert Taylor series and other real-valued polynomials to ATs. The AT can efficiently represent both specifications and implementations, making the verification process simpler and more elegant. In consequence, the imprecision represented as the absolute difference between AT polynomials of specifications and implementations can be easily and effectively determined.

In this paper, we first elaborate on the way to compute arithmetic imprecision, in Section III. Next, in Section IV, we define a tight-bound interval analysis which offers more accurate results than the original interval analysis. Finally, we propose an algorithm for an optimized implementation of real-valued specifications, such as Taylor series, with the smallest number of AT polynomial terms.

## II. Background

In optimization of imprecise arithmetic circuits, one needs to repeatedly undertake the precision analysis throughout the search for the optimal solution. The precision analysis has to ascertain whether the implementation matches the specification within the given *error bound* over the whole domain of interest. This question can be addressed by the *dynamic analysis*, based on simulations, or by the *static analysis*, relying on formal representations of a specification and an implementation.

Dynamic analysis has been applied in most precision optimization schemes—work in [3]–[8] present the straightforward simulation-based techniques to assign bit-widths. The simplicity of this approach makes it prevalent; however, one has to enumerate and simulate all possible input values to provably complete the job. Work in [4] presents an approach for combined word-length optimization to minimize the hardware implementation cost. The presented algorithm finds the word-length sensitivity throughout fixed-point simulations of a signal flow graph, and conducts the final word-length optimization by iteratively modifying the word-length of the synthesized hardware model. Gaffar *et al.* [6] utilize the automatic differentiation to compute the sensitivities of outputs to the bit-widths of various operands in a design. Such a sensitivity analysis allows the exploration and comparison of fixed-point and floating-point implementations. Shi *et al.* [7] set up a statistical model to estimate hardware resources in terms of perturbation theory. A tool that automates the floating-point to fixed-point conversion is based on a simulation using Simulink. This method, however, imposes the requirement for a large set of input vectors.

Fang *et al.* [9] take advantage of affine arithmetic (AA) modeling to analyze range and precision. The AA model is a derivation of the *interval arithmetic* (IA) borrowed from numerical analysis. In AA, the quantities of interest are represented as linear combinations (affine forms) of certain primitive variables, which stand for sources of uncertainty in the data or approximations made during the computation. In AA, each quantity $x$ is represented by a formula

$$X = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n$$

where $x_0$, $x_1, \ldots, x_n$ are floating-point numbers, and $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$ are symbolic variables whose values are only known to lie in the range $[-1,+1]$. Pu and Ha [10] apply AA to find the most suitable bit-width-to-error tradeoff. Work in [11] adopts the static analysis to investigate bit-widths due to truncated and rounded data, and explore hardware area and delay in field-programmable gate array (FPGA) for different bit-widths. Work in [12] extends [11] to provide a method for optimizing word-lengths of hardware designs with fixed-point arithmetic based on analytical error models that guarantee accuracy. AA has an advantage of easy computation, but overestimates the error bound like IA and is unnecessarily pessimistic. Our method will be able to get more exact results due to the static analysis from [20] based on AT that verifies whether an imprecise circuit satisfies error bound. Using AT, a *branch-and-bound* algorithm searches effectively for the imprecision error, which we exploit as an underlying static analysis. Therefore, as AA tightens the error bound relative to IA, the AT can get the best analysis.

In [13], authors set up models for error source dependence. In these models, the dependence is approximated by linear functions (AA) or by general polynomials (Taylor series methods). It was shown that the optimal way to decrease the excessive bit-width is to use implicit polynomial dependence.

Constantinides *et al.* [14] propose Synoptix—an optimization technique targeting linear time-invariant digital signal processing systems using their resource binding technique. Synoptix is based on saturation arithmetic to perform the bit-width optimizations.

Sivaram and Kalla [15] focus on polynomials with integer coefficients and integer variables to shrink hardware area. The algebraic properties of polynomials over finite rings are used to express datapaths by its equivalent computation leading to simplified polynomials. An approach is presented to area optimization of arithmetic datapaths. The polynomial feature restricts the application and inherently does not deal with imprecise computation.

Ahmadi and Zwolinski [16] address the bit-width assignment in hardware implementation in the context of high-level synthesis. In order to surpass the pessimism of IA, they introduce a symbolic noise analysis (SNA), which is based on modeling of the error bounds by an assumed probability distribution function over a known range. In comparison to SNA which assumes more localized error distributions, IA is pessimistic by assuming the uniform distribution.

A new technique SAT-Modulo theory is used in [17] to explore range analysis in bit-width allocation which is significant to influence hardware cost. It determines bit-widths for finite precision implementation of numerical calculations, and can get more accurate bounds estimation than IA and AA, in turn yielding smaller bit-widths.

The above works mostly undertake the optimization of a single bit-width with a target of hardware area. There was also a limited consideration of function approximation and its effect on the precision. We note that authors in [18] and [19] have dealt with approximating a function by piecewise polynomials. They evaluate several segments separately for bit-width optimization and precision to reduce area and latency. As many circuits rely on function approximations, in this paper, we demonstrate the imprecise circuit optimization for real-valued specifications that simultaneously considers multiple bit-widths as well as the function approximation by finite Taylor series. Since our main object is Taylor series where the expansions of different degrees lead to the different approximations, we mainly focus on this scheme in this paper.

## III. Introducing Basic Algorithms

In this section, we introduce three algorithms aiming to compute the imprecision in arithmetic circuits.

Real-valued functions can be described by Taylor series as follows:

*Definition 1:* A real and differentiable function $f(X)$ can be represented as a *Taylor series* over an interval $I$ in the neighborhood of the initial value $X_0$

$$f(X) = \sum_{n=0}^{\infty} \frac{1}{n!}(X - X_0)^n f^{(n)}(X_0).$$

The *finite Taylor expansion* restricted to the first $n+1$ terms is

$$f(X_0) + Xf'(X_0) + \frac{X^2}{2!} f''(X_0) + \cdots + \frac{(X - X_0)^n}{n!} f^{(n)}(X_0)$$

and hence the *approximation error* given in terms of a point $\varphi$ in the interval $I$ is

$$R_n(X) = \frac{f^{(n+1)}(\varphi)}{(N+1)!}(X - X_0)^{n+1}. \tag{1}$$

The effect of the Taylor series truncation is easy to evaluate, as Taylor series come with the provable bound on the approximation error. The remainder $R_n(X)$ of the truncation has an explicit expression in terms of the $(n+1)$th derivative at the intermediate point $\epsilon$ in the given interval $I$.

In addition to the function approximation due to the finite number of Taylor terms, it is necessary to replace real-value parameters (inputs, outputs, coefficients, etc.) by their finite word-length approximations. This, in consequence, leads to the *imprecision errors*.

For analysis of the imprecision error due to finite word-length arithmetic, we adopt AT, which has been successfully used for precision verification [24]. AT is an instance of spectral representations, where the spectral coefficients contain function information that is global over the domain of definition, and by that allow a number of properties to be more easily deduced than in the Boolean domain.

*Definition 2:* AT is a polynomial representing a pseudo-Boolean function $f : B^m \rightarrow w$ with an arithmetic operation "+," word-level coefficients $c$, binary inputs $x_1, x_2, \ldots, x_m$ and binary exponents $i_1, i_2, \ldots, i_m$

$$AT(f) = \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} \cdots \sum_{i_m=0}^{1} c_{i_1 i_2 \ldots i_m} x_1^{i_1} x_2^{i_2} \ldots x_m^{i_m}.$$

AT represents functions at *word-level* while the inputs are kept binary. Such I/O format of the transform is well suited, as word-level outputs are instrumental for analyzing the imprecision, while having binary inputs helps in organizing an efficient search for the maximal imprecision. When evaluating the arithmetic imprecision, one can find examples where all the bits are wrong, yet the implementation is considered precise, as well as the cases when few bits are wrong, but the implementation is imprecise. As an example of the former case, two fixed-point values encoded in binary as $011\cdots1$ and $100\cdots0$ differ in all bits, yet they can be made arbitrarily close, with increase in the number of bits. The word-level function values, however, can directly capture the notion of distance used in the underlying arithmetic.

Finding the implementation imprecision involves the search for the maximum value over the function domain. Hence, having inputs at a binary level is useful for organizing an efficient search, leading eventually to a fast *branch-and-bound algorithm* [20].

AT is canonical, and will be used to directly represent approximation and imprecision errors coming from the finite Taylor series function representations. The correspondence between Taylor and AT representation is illustrated by the following lemma:

*Lemma 1:* Consider a finite Taylor polynomial around $X_0 = 0$ where the variable $X$ will be represented as an $m$-bit unsigned fractional number. By denoting $f_0^{(i)} = f^{(i)}(X_0)$, we have

$$f(X) = f_0 + Xf_0' + \frac{X^2}{2!} f_0'' + \cdots + \frac{x^{n-1}}{(n-1)!} f_0^{(n-1)}.$$

The AT of $f(X)$ is expanded from the Taylor polynomial as

$$AT[f(X)] = f[AT(X)]$$

$$= f_0 + f_0' \left( \sum_{i=0}^{m-1} 2^{-(i+1)} x_i \right) + \cdots + \frac{f_0^{(n-1)}}{(n-1)!} \left( \sum_{i=0}^{m-1} 2^{-(i+1)} x_i \right)^{n-1}.$$

*Proof:* The transform of an $m$-bit unsigned fractional number $X$ is $AT(X) = \sum_{i=0}^{m-1} 2^{-(i+1)} x_i$. Since AT is linear, that is, $AT(f_1 + f_2) = AT(f_1) + AT(f_2)$ and $AT(C * f) = C * AT(f)$,

where $C$ is a constant, we can obtain

$$\begin{aligned} AT[f(X)] &= AT\left(f_0 + Xf_0' + \frac{X^2}{2!}f_0'' + \cdots + \frac{X^{n-1}}{(n-1)!}f_0^{(n-1)}\right) \\ &= AT(f_0) + AT(Xf_0') + \cdots + AT\left(\frac{X^{n-1}}{(n-1)!}\right)f_0^{(n-1)} \\ &= f_0 + f_0'AT(X) + \frac{f_0''}{2!}AT(X^2) + \cdots + \frac{f_0^{(n-1)}}{(n-1)}AT(X^{n-1}) \\ &= f[AT(X)]. \end{aligned}$$

Lemma 1 denotes that $AT[f(X)]$ results from substituting expanded bit-level variables for the word-level variable $X$ in $f(X)$. By combining coefficients of isomorphic terms in the expanded polynomial, the AT representation in Definition 2 is obtained, thus leading to the conversion of Taylor expansions to AT.

While Lemma 1 might seem to lead to a simple realization of the conversion between Taylor and AT, in reality the process could be time and memory-consuming. Next, we show an efficient algorithm to realize such a conversion. ■

### A. Algorithm for Taylor Series Conversion

In order to evaluate the imprecision error using AT, the specification should be translated into AT as well. In this section, we describe the conversion of Taylor series into AT by expansion from Lemma 1. A straightforward method for generating $AT[f(X)]$ replaces each monomial in Taylor series $f(X)$ by its defining AT, followed by the *consolidation* of AT terms. Although the overall conversion procedure is conceptually simple, the expansion of the real-valued quantities from Taylor series into word-level AT terms can lead to a large intermediate polynomial, similar to what is known to happen in symbolic computing.

By laws of Boolean algebra, $x_i^n = x_i$ for positive $n$, causing many of the expanded monomials of the intermediate polynomial to be identical. Hence, they are isomorphic and can be combined to form a simplified AT polynomial.

### B. AT Construction from Multi-Variate Polynomials

To extend the AT construction to polynomials over multiple variables, we require the ordering between the variables. For a given order, we define the *index*, which is a unique mapping for each term. The index function will facilitate the combination of isomorphic terms in an intermediate polynomial.

*Definition 3:* Let the term consist of $p$ bit-level literals $b_{p-1} \ldots b_0$. Let every bit $b_r$ belong to the word-level variable $W_r$, that is $m_r$-bit wide. Then, the term index of the AT term is defined as

$$\text{term index} = \sum_{r=0}^{p-1} 2^{(b_r + \sum_{q=0}^{w_r - 1} m_q)}. \tag{2}$$

*Example 1:* Consider AT over three word-level variables $X$, $Y$, and $Z$ consisting of 3, 4, and 3 bits, respectively. Let $X$ be the least significant variable indexed as "0," and $Z$ be the most significant variables indexed as "2." For the three bit-level literal term $z_2z_1x_0$, the word-level variables to which the respective literals belong, are $(W_2, W_2, W_0) = (2, 2, 0)$. The index of the term is obtained as the sum of the three literal indices. First, the computation for $x_0$ produces its index $2^0 = 1$, since $b_0$ is 0 and $W_0$ is 0. Then, $z_1$contributes $2^{1+(3+4)} = 256$, since $b_1$ is 1 and $W_2$ is 2, so $m_0 + m_1 = 3 + 4 = 7$. Finally, $z_2$ produces $2^{2+(3+4)} = 512$, because $b_2$ is 2 and $W_2$ is 2. Therefore, the term index for the AT term $z_2z_1x_0$ is 512 + 256 + 1 = 769.

```
Convert_Multivar_AT(f, term_num, bit_num )
{   for (i=0; i< term_num; i++)
    {   temp_AT = Expand _Term (bit_num);
        sum_AT = Add_AT (sum_AT, temp_AT);     }
    final_AT = sum_AT;          return final_AT;
}
Expand _Term (bit_num)
{   for (p=0; p<word_var_num; p++)
    {   AT_poly[p]=Convert_Univar_AT (f, term_num, bit_num);
        product_AT= Multiply_AT(AT_poly[p], AT_poly[p-1]); }
     Set_index (product_AT);      return product_AT;
}
Add_AT (augend_AT, addend_AT)
{    while (!augend_AT.tail && !addend_AT.tail( ) )
     {   if ( augend_AT.term.index < addend_AT.term.index)
             Copy_AT_term (sum_AT.term, augend_AT.term);
         else if (augend_AT.term.index> addend_AT.term.index )
             Copy_AT_term (sum_AT.term, addend_AT.term);
         else {   Copy_AT_term(sum_AT.term, augend_AT.term);
                  sum_AT.term.coeff = augend_AT.term.coeff
                                      + addend_AT.term.coeff;   }
     }
     Delete (augend_AT, addend_AT);   return sum_AT;   }
}
Multiply_AT (multiplicand_AT, multiplicator_AT)
{   while (!multiplicand_AT.tail)
    {   while (!multiplicator_AT.tail)
        {   product_AT.term.coeff = multiplicand_AT.term.coeff
                                    * multiplicator_AT.term.coeff;
            for (p=0; p<cand_bit_num; p++)
                product_index[p] = cand_index[p];
            for (p=cand_bit_num; p<product_bit_num; p++)
                product_index[p]=cator_index[p-cand_bit_num]; }
    }
    return product_AT;
}
```

Fig. 2. Algorithm to convert a real-valued multivariate polynomial.

Fig. 2 describes the algorithm to produce AT over multiple word-level variables from a real-valued polynomial. The algorithm first generates ATs for each monomial, and then performs additions of the isomorphic intermediate monomials, leading to the final transform. The function `Expand_Term` expands a single word-level polynomial term into its AT. The procedure `Convert_Univar_AT` [21] obtains ATs for all word-level variables in the term. Then, the `Multiply_AT` multiplies out the resulting univariate ATs into the AT for multivariate word-level variables, as per Lemma 1. Note that `Multiply_AT` follows the conversion of a word-level variable that reduces the number of terms. Hence, the size of resulting AT can be kept under control by avoiding storing expanded terms. In each iteration the algorithm adjusts term indices and combines isomorphic terms. Each AT term input to the `Multiply_` AT is assigned a unique index from Definition 3, which guarantees linear ordering among terms.

The function `Add_AT` adds two AT polynomials in a canonical way. In this procedure, the isomorphic term combination and the term ordering by index occur concurrently.

When comparing terms' indices, the AT term with a smaller index is moved forward in the ordered list. If two terms have identical index functions, they are isomorphic, and hence their coefficients are accumulated.

*Example 2:* Consider the polynomial over word-level variables $X$ and $Y$ consisting of 2 and 3 bits, respectively

$$F(X, Y) = 2X^3Y + X^2Y^2.$$

Let $Y$ be the more significant word-level variable, i.e., the input vector is $(YX) = (y_2y_1y_0x_1x_0)$. The AT expansions of $X^3$ and $Y$ present in the first term $2X^3Y$ are

$$AT(X^3) = (2x_1 + x_0)^3 = x_0 + 8x_1 + 18x_1x_0 \text{ and } AT(Y) = 4y_2 + 2y_1 + y_0.$$

The AT transform of the term $2X^3Y$ is then the product of the above two AT polynomials

$$AT(2X^3Y) = 2y_0x_0 + 16y_0x_1 + 36y_0x_1x_0 + 4y_1x_0 + 32y_1x_1 + 72y_1x_1x_0 + 8y_2x_0 + 64y_2x_1 + 144y_2x_1x_0.$$

By Definition 3, the term indices of the expanded $AT(2X^3Y)$ are

$$(5, 6, 7, 9, 10, 11, 17, 18, 19).$$

In the second term $X^2Y^2$ of $F(X, Y)$, expansions of $X^2$ and $Y^2$ are

$$AT(X^2) = x_0 + 4x_1 + 4x_1x_0$$
$$AT(Y^2) = y_0 + 4y_1 + 4y_1y_0 + 16y_2 + 8y_2y_0 + 16y_2y_1.$$

Their multiplication results in the $AT(X^2Y^2)$

$$AT(X^2Y^2) = y_0x_0 + 4y_0x_1 + 4y_0x_1x_0 + 4y_1x_0 + 16y_1x_1 + 16y_1x_1x_0 + 4y_1y_0x_0 + 16y_1y_0x_1 + 16y_1y_0x_1x_0 + 16y_2x_0 + 64y_2x_1 + 64y_2x_1x_0 + 8y_2y_0x_0 + 32y_2y_0x_1 + 32y_2y_0x_1x_0 + 16y_2y_1x_0 + 64y_2y_1x_1 + 64y_2y_1x_1x_0.$$

Its term indices are

$$(5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19, 21, 22, 23, 25, 26, 27).$$

The addition subroutine `Add_AT` matches the terms with the same indices to compute the final $AT(2X^3Y + X^2Y^2)$

$$AT(2X^3Y + X^2Y^2) = 3y_0x_0 + 20y_0x_1 + 40y_0x_1x_0 + 8y_1x_0 + 48y_1x_1 + 88y_1x_1x_0 + 16y_1y_0x_1 + 16y_1y_0x_1x_0 + 24y_2x_0 + 128y_2x_1 + 208y_2x_1x_0 + 8y_2y_0x_0 + 32y_2y_0x_1 + 32y_2y_0x_1x_0 + 16y_2y_1x_0 + 64y_2y_1x_1 + 64y_2y_1x_1x_0.$$

### C. Imprecision Searching Algorithm

As arithmetic imprecision is inevitable, the implementation is considered correct if it fits the specification within some given bound spanning across the whole domain of function definition. Since the arithmetic discrepancy depends on the input value assignments, one cannot, in general, find the imprecision without finding the maximum of arithmetic differences between specifications and implementations over the whole domain.

A static method for value analysis using simulated annealing is proposed in [11]. This method, however, has no means to guarantee that local minima will be avoided. In this paper, we rely on the alternative solution, which utilizes the suitability of AT in the representation and exploration of the imprecision. The expansion from word-level variables to bit-level variables and the binary branch-and-bound searching algorithm [20] find the imprecision exactly, while IA and AA are approximate and based on word-level variables so they cannot account for interaction between the bits. Therefore, the AT method guarantees higher accuracy.

*Definition 4:* The error AT polynomial ($AT_e$) is a difference between AT polynomials of the specification and the implementation. The *imprecision* is defined as the maximum absolute value of $AT_e$ over the domain of the function.

In this paper, we focus on determining the imprecision for an implementation derived from Taylor series; however, the methods apply to all polynomial real-valued specifications. Search for the maximum imprecision will serve as a key component in confirming whether two given implementations $f_1$ and $f_2$ are interchangeable with respect to precision requirements. This problem is stated as follows.

*Problem 1:* Compute imprecision among two implementations.

*Inputs*: $f_1(X), n_1, m_1, f_2(X), n_2, m_2$.
*Output*: *imprecision*.

Here, $n$ and $m$ represent the number of Taylor terms and the input bit-width, respectively.

In the case of AT polynomial, a static precision analysis can be helped by a provable branch-and-bound scheme, as shown in [24] and [21]. The solution to this problem, and the benefits of the static precision analysis, can be then extended to Taylor series specifications. For that, it suffices to convert the Taylor series to AT, Section III-A, and then rely on the algorithm from [21] for the efficient search for the maximum absolute value of an error AT polynomial.

## IV. Implementation Optimization

Given the real-valued specification and a precision error bound, we want to find the area-optimized implementation.

*Problem 2:* Generic imprecise circuit optimization.

*Inputs*: $f, I, X_0, E$.
*Given*: $Distanced(f_1, f_2), Area\_cost(f)$.
*Constraint*: $d(f, \tilde{f}) \geq E \text{ over } I$.
*Goal*: $min(Area\_cost(\tilde{f}))$.
*Output*: $\tilde{f}$.

Our objective is introduced by Problem 2, where $f : R^n \rightarrow R$ and $\tilde{f} : W^n \rightarrow W$ represent the real-valued specification and fixed-point, word-level ($W$) implementation, respectively. The *imprecision* is the maximal absolute difference between $f$ and $\tilde{f}$ over a given interval $I$ around point $X_0$. Also available is the area cost function, $Area\_cost$. The goal is to find an implementation $\tilde{f}$ of minimum area, within the imprecision bound $E$.

The distance $d(f, g)$ between two functions $f$ and $g$ is defined in $L_\infty$ norm as the maximal absolute value of the arithmetic difference

$$d(f, g) = \max_{X \in I} |f(X) - g(X)|.$$

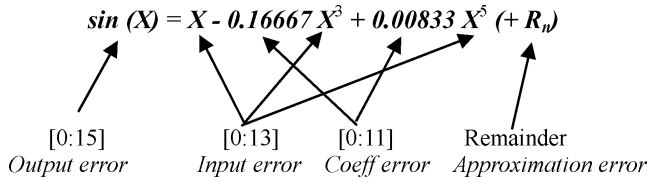


Fig. 3. Imprecision due to the combined sources.

Other distances, such as a *sum of squares*, could be used when dealing with imprecise implementations.

The concrete instance of Problem 2 considered in this paper deals with the case where the specification is given by the Taylor series or real-valued polynomials. The solution will be sought by applying the static precision analysis performed over AT representations. Next, we analyze how the precision parameters influence the overall implementation.

### A. Precision Parameters Analysis

Determining the set of parameters needed to achieve a sufficiently precise circuit is a challenge requiring a well structured precision analysis. The traditional method of using simulations over various values of the parameters is expensive and not guaranteed to produce the optimal result. In this section, we propose the static imprecision analysis. We perform the analysis of an arithmetic imprecision caused by all approximations and finite bit widths in the implementations of real-valued specifications such as Taylor series example in Fig. 3. Here, $\sin(X)$ function imprecision is caused by finite word lengths of the input, coefficients and output, as well as due to the Taylor series truncation. In summing such precision errors, we will repeatedly use the triangle inequality: $d(x, z) = d(x, y) + d(y, z)$.

1) *Errors due to Function Approximation:* In realizing real-valued functions by arithmetic circuits, an algorithm might be employed to *approximate*, rather than *exactly* implement the function. For instance, when using the first $n$ terms of a Taylor series to represent a transcendental function, the approximation error is provably bounded by a remainder $R_n(X)$, (1). For a function defined in interval $I$, this truncation error bound $e_t$ translates into

$$e_t = \max_{X \in I} |R_{n-1}(X)| . \tag{3}$$

*Example 3:* Consider the following function $f(X) = \cos(X)$. In the interval [0, 1], its five-term Taylor approximation calculated around $X_0 = 0$ is

$$\text{Taylor}(\cos X) = 1 - \frac{1}{2}X^2 + \frac{1}{24}X^4.$$

Note that the second and fourth term is zero. The error bound is

$$e_t = \max |R_4(X)| = \max \left| \frac{1}{120} \sin \varphi X^5 \right|$$

$$\leq \left| \frac{1}{120} \sin 1 \right| = 0.007.$$

Given the error bound $E$, the minimal number of Taylor terms is the smallest integer $n$ such that the error $e_t$ resulting from considering only the first $n$ terms of Taylor series lies within the imprecision bound, i.e., $e_t < E$. Such a finite truncation will have the least number of terms within an acceptable imprecision bound over the given interval $I$. Note from (1) that for bounding purposes, instead of finding the exact maximum of the $(n+1)$st derivative in the interval $I$, an upper bound on the derivative can be used.

### B. Input Bit-Width and Quantization Error

In fixed-point implementations, a real-valued input variable $X$ is restrained to a finite word-length vector representation affecting the final outcome. As an insufficiently precise value can result from using too few bits, it is crucial to find an appropriate bit-width for the acceptable overall error. In this section, we present the analysis schemes adopted by us to determine the quantization error.

1) *Effects of Finite-Input Bit-Width—Interval Analysis:* Theoretically, an argument of a real-valued function is an infinitely precise real-valued input $X_{\text{th}}$ which is, in reality, replaced by the quantized input $X$. The resulting quantization error can determined by means of the classical *interval analysis*.

In our scheme, we present the interval analysis in terms of AT. For simplicity, the original interval $I$ is normalized to [0, 1] causing the input $X$ to be scaled down. In consequence, instead of dealing with the input bit-width, we consider the number of *Fractional Bits* (FBs). The input range is divided into uniform $2^{FB}$ intervals of the size $2^{-FB}$. Hence, the value $X_{\text{th}}$ lies between two consecutively quantized numbers and the relation between $X_{th}$ and $X$ is then

$$|X_{th} - X| \leq 2^{-(FB+1)}$$

$$\Rightarrow x - 2^{(FB+1)} \leq X_{th} \leq X + 2^{-(FB+1)}. \tag{4}$$

By replacing $X_{th}$ with $m$ FBs of $X$ in accordance with (4), we obtain the ATs for the theoretical $f_{th}$ and quantized $f$ Taylor functions (given $X_0 = 0$)

$$f_{th} = \sum_{i=0}^{n-1} C_i X_{th}^i = \sum_{i=0}^{n-1} C_i \sum_{k=0}^{m-1} 2^{-(K+1)} x_m \pm 2^{-m-1i} \tag{5}$$

$$f = \sum_{i=0}^{n-1} C_i X^i = \sum_{i=0}^{n-1} C_i \left( \sum_{k=0}^{m-1} 2^{-(K+1)} x_k \right)^i \tag{6}$$

where $C_i$ is a Taylor coefficient equal to $\frac{f_{th}^i(X_0)}{i!}$.

For functions $f_{\text{th}}$ and $f$, the AT polynomials $AT(f_{\text{th}})$ and $AT(f)$ are on the right-hand side of (5) and (6), respectively. The bound $e_{\text{i}}$ on the effects of input quantization of half a *ulp* (*unit in the last place*) to the output precision is obtained by the AT formulation as follows. The *error polynomial* $AT(f_{ei})$ is determined as a difference between $AT(f_{th})$ and $AT(f)$

$$AT(f_{ei}) = AT(f_{th}) - AT(f) =$$

$$= AT \left( \sum_{i=0}^{n} C_i \left[ \left( \sum_{k=0}^{m-1} 2^{-(k+1)} x_k \right) \pm 2^{-m-1} \right]^i \right)$$

$$- AT \left( \sum_{i=0}^{n} C_i \left( \sum_{k=0}^{m-1} 2^{-(k+1)} x_k \right)^i \right). \tag{7}$$

```
Get_input_error (f, n, m)
{  f_th = f(X, 2^-m-1);
   AT(f) = Convert_AT(f, m, n);
   AT(f_th) = Convert_AT(f_th, m, n);
   AT(f_ei) = AT(f_th) - AT(f);
   e_i = Search_Imprecision (AT(f_ei));  }
```

Fig. 4. Computation of input quantization error.

The maximum value of $AT(f_{ei})$ in (7) gives the error bound $e_i$. In practice, $e_i$ can be obtained by an efficient branch-and-bound search algorithm tuned for this application.

Fig. 4 shows the use of interval analysis by AT to estimate the error due to the input quantization. The method invokes the conversion algorithm `Convert_AT` [21]. The maximum mismatch $e_i$ between $f_{\text{th}}$ and $f$ is obtained by the imprecision search algorithm `Search_Imprecision` [21].

2) *Tight-Bound Interval Scheme:* The interval analysis unavoidably overestimates the error bound. Therefore, in order to obtain better results, we propose a *tight-bound interval* scheme for which AT is especially suitable. Our approach employs an auxiliary, higher precision "reference implementation" which has the input bit-width extended beyond the actual implementation. The tight-bound analysis determines the exact mismatch $\epsilon_{\text{hp}}$ between the implementation and the auxiliary reference. Due to the finite bit-width of the reference, there is also a residual error $\varepsilon_{I\_TB}$ of the reference implementation, easily obtained by interval analysis. That error can be made significantly smaller, though, by extending the reference's input bit-width. The overall error bound is obtained by triangle inequality as $\varepsilon \leq \varepsilon_{hp} + \varepsilon_{I\_TB}$.

*Example 4:* Assume that $m = 8$ bits is used to represent a fractional number. Let $f$ and $f_{th}$ represent the quantized function and the theoretical value, respectively. Using the interval analysis the quantization error is estimated to be

$$|f - f_{th}| = \varepsilon_I = \Theta(2^{-8}).$$

Now, we apply the tight-bound method to improve the precision analysis. For this, we use the auxiliary reference implementation with the input bit-width $t = 17$ bits. The error relative to the reference is

$$|f - f_{hp}| = \varepsilon_{hp}.$$

Then, the quantization error of the auxiliary implementation alone is estimated by the interval analysis to be

$$|f_{hp} - f_{th}| \leq \varepsilon_{I\_TB} = \Theta(2^{-17}).$$

Finally, from the triangle inequality, it follows that:

$$|f - f_{th}| \leq |f - f_{hp}| + |f_{hp} - f_{th}| = \varepsilon_{hp} + \varepsilon_{I\_TB}. \quad (8)$$

### C. Feasible Implementation and Optimization for Taylor Series

The parameters analysis technique presented in the previous section can be directly applied to finding a feasible implementation of Taylor series representation of a function $f(X)$ defined in interval $I$ around $X_0$. The problem is stated below.

```
1. Determine Taylor terms
   {  assume n terms and obtain e_t;
      while (e_t ≥ E)   {  n++;  obtain e_t;  }
   }
2. Determine input bit-width
   { assume input bit-width m;
     while
     {  AT(f)_th = Convert_Taylor_AT (f_th, n, m);
        AT(f) = Convert_Taylor_AT (f, n, m);
        e_i = Imprecision_Searching (AT(f_th) - AT(f));
        if (e_i ≥ E-e_t)  m++;  else  break;  }
   }
3. Determine coefficient bit-width
   {  assume bit-width of coefficients q;
      while
      {  AT_ec = Convert_Taylor_AT(f_ec);
         e_c = Imprecision_Searching (AT_ec);
         if (e_c ≥ E-e_t- e_i)  q++;  else  break;  }
   }
4. Determine output bit-width
         o = -log2 (E- e_t - e_i - e_c) + 1
```

Fig. 5. Sequential method to fit error bound.

*Problem 3:* Feasible precision parameters.
*Inputs*: $f(X)$, $X_0$, $I$, $E$.
*Constraints*: $e_t + e_i + e_c + e_o < E, \forall X \in L$.
*Outputs*: $n$, $m$, $q$, $o$.

*Lemma 2:* The algorithm in Fig. 5, consisting of the independent precision parameter selections applied sequentially, such that in each step the total imprecision is smaller than $E$ leads to the solution of Problem 3.

*Proof:* Let the algorithm first choose the number of Taylor terms $n$, such that the approximation error $e_t$ is smaller than $E$. Then, the bit-widths $m$, $q$ and $o$ are selected one at the time, taking into account the sums of imprecision obtained so far. The algorithm clearly terminates with the resulting imprecision being at most $E$. At each step the precision parameter can be chosen to make the error due to that parameter arbitrarily small, and hence the overall error bound not exceeding $E$. Hence, at the end, we obtain $e_t + e_i + e_c + e_o < E$, which is the goal set in Problem 3. ■

This approach is based on sequences of isolated considerations of individual precision parameters. Since the Taylor terms will affect $e_i$ and $e_c$ in terms of (7), it is explored first. The input variable has more area impact than the coefficients because of the higher exponents, so it is explored next. However, the method is clearly insufficient for finding an optimal solution. In the remainder of this section we propose a search algorithm for obtaining precision parameters for minimum cost of implementation under the assumption that the overall imprecision remains within a given bound $E$.

1) *AT Size as a Cost Function:* We want to guide our search algorithm using a *technology-independent area cost function* that will allow us to realistically compare the candidates for the optimal solution. While the exact area of a circuit is not known before mapping it to a given technology, the technology mapping is a time consuming process that cannot be incorporated to a static analysis. It becomes then necessary to find a suitable area cost function.

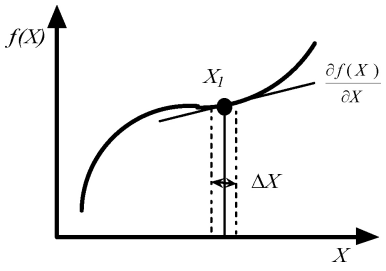


Fig. 6. Basic sensitivity concept.

The size of AT polynomial can be used as a cost function because AT performs bit-level operations by its definition that is the same as arithmetic circuits, so increase of AT size means more binary operations and requires more area. Therefore, AT size is taken to reflect the area.

When implementing Taylor series specifications, the area clearly increases monotonically in both the number of Taylor terms $n$ and the input bit-width $m$. In particular, the larger input bit-width $m$ demands the wider datapath, whose cost grows at least linearly, i.e., as $o(m)$, by the arguments of circuit complexity theory. Furthermore, the number of Taylor terms $n$ affects the implementation cost in two ways. On top of the obvious demands on hardware to realize $n$ Taylor terms, note that inputs are raised to exponents which grow with $n$.

As the number of AT polynomial terms exhibits the same tendency as the one just described, we use $|terms(AT(f))|$ as the cost function to be minimized. The size of AT is obtained by directly expanding the $n$-term Taylor polynomial over the $m$-bit input. One can show that the number of AT terms is

$$|terms(AT(f_{n,m})| = \sum_{i=1}^{n} \binom{m}{i}. \tag{9}$$

It is hard to know in advance which factor has more effect on area since the Taylor terms and the input bit-width are involved in the interplay and determine AT size jointly by (9).

### D. Error Sensitivity

For the number of Taylor $n$, the error bound from (1) can be readily used to determine $e_t$. After that, the input error must be such than $E - e_t$, which helps to explore the suitable input bit-width. The information on error sensitivity will facilitate the calculation of input bit-width.

Traditionally, sensitivity introduced in [6] is defined as (10) and Fig. 6 to describe the influence that a small change $\Delta X$ of $X$ has on the output $Y$

$$\Delta Y \approx f'(X)\Delta X. \tag{10}$$

where $f'(X)$ is the derivative of $f(X)$.

In order to use sensitivity to investigate the input quantization error and find the suitable input bit-width, we re-define the sensitivity.

*Definition 5:* The *sensitivity* is a numerical value to describe the influence that a small change of $X$ has on the output $Y$ in condition of the worst case

$$\Delta Y = AT(f'(X))_{\max} * 2^{-m-1}. \tag{11}$$

The sensitivity reflects the output change in terms of tiny input turbulence. It has the same essence as the representation by (5) and (6), so sensitivity can be used as a substitution.

The performance bottleneck to determine the optimized implementation is the procedure must repeat to invoke the conversion algorithm when searching different Taylor terms and input bit. In each flow, it must require invoking the conversion algorithm twice and need to do subtraction of two AT polynomials as (5) and (6) to get the input error in order to confirm whether the input bit-width is satisfied. Of course, the complex procedure will spend a lot of time and memory. However, if using sensitivity, as long as $f'(X)$ is converted to $AT(f'(X))$ and using branch-bound algorithm finds the maximum value to match the worst case, the sensitivity is calculated by its multiplication with $\Delta X$. Here $\Delta X$ is $2^{-m-1}$, i.e., half of the *ulp*. We can see this procedure only invokes the conversion algorithm one time to transform $f'(X)$ into $AT(f'(X))$. The advantage is obvious. While the sensitivity is obtained, combined with the input error bound, it is easy to conclude the suitable input bit-width.

Similarly, searching for an appropriate bit-width of the Taylor coefficients $C_i$ is guided through the corresponding sensitivity, readily calculated from Taylor series, the conversion algorithm and the searching algorithm.

1) *Algorithm for Optimizing Parameters:* Consider now the problem of finding an implementation of minimum AT size, where all the disparate causes of imprecision are not exceeding the given bound.

*Problem 4:* Precision parameter optimization.

*Inputs*: $f(X)$, $X_0$, $I$, $E$.
*Constraints*: $imprecision < E$, $\forall X \in I$.
*Outputs*: $n$, $m$ /* *terms*; *input bit-width*.
*Goal*: $minimum|\text{terms}(AT(f))|$ /* *polynomial size*.

In deriving a thorough search scheme, we concurrently explore multiple precision parameters. In Section IV-A, we saw that coefficients and output bit-widths impact overall precision via multiplication by a constant and the direct rounding, respectively. As they both have less effect on the area, we focus on the number of Taylor terms and the input bit-widths.

Fig. 7 describes the algorithm optimizing the number of Taylor terms and the input bit-width. A pair ($n$, $m$) is referred to as a *node*, representing a combination of a number of Taylor terms $n$ and an input bit-width $m$ used in each search step. In the first iteration, the algorithm gets the smallest number of Taylor terms for given error bound, and obtains input bit-width by sensitivity computation (Steps 1–5). To explore the search space, it suffices to consecutively increase the set of Taylor terms, while simultaneously exploring alternative the input bit-widths (Steps 6 and 7). If the new node can satisfy the error bound $E$, the newly computed number of Taylor terms is assumed, and the algorithm continues to decrease input bit-width until the current node breaks the bound. When it happens, the algorithm backtracks to the previous node and stores it (Steps 9 and 10). The procedure is repeated until the change of bit-widths are exhausted, while $e_i > E$ (Step 8).

Since Taylor series cannot be compared directly, it is necessary to use AT for comparison because of easy computation of (9), so in above procedure the conversion algorithm is invoked to achieve the goal. The searching algorithm is helpful to find the quantization error represented by AT polynomials. A

```
Design_best_Taylor_imp (f, E)
1. { while (e_t > E)    { ++n;    e_t = Get_Taylor_error (n)   }
2.   AT_derivative = Convert_Univar_AT(f', n, m_0);
3.   sensitivity=Search_Imprecision (AT_derivative)* 2^(-m_0-1);
4.   ini_m = m_0 – log[(E- e_t) / sensitivity];
5.   Store_node (n, ini_m);      m = ini_m;
     while
6.    {   e_t = Get_Taylor_error (++n);
7.        e_i = Get_input_error (f, n, --m);
8.        if (e_i < E)
           {   while (e_i <E - e_t)   e_i = Get_input_error (f, n, --m);
9.             if (++m != ini_m)
                {   Store_node (n, m);   ini_m = m;
10.                 Tight_interval (node);    }   }
11.            else   break;
      }
12.  best_node = Compare_AT_size (nodes);
13.  (e_c, q) = Get_coeff_bit (E, e_t, e_i) ;
14.  o = -log_2 (E- e_t - e_i - e_c) + 1;        return best_node;
}
Get_input_error (f, n, m)              // Using Eqn. (11)
{   AT_derivative = Convert_Univar_AT(f', n, m);
    max_val = Search_Imprecision (AT_derivative);
    e_i = max_val * 2^(-m-1);        return   e_i;   }
Compare_AT_size (nodes)
{   for (i=0; i<nodes_num; i++)
        AT_size[i] = Get_AT (node[i](n), node[i](m) );
    Sort (AT_size);     return the node with smallest AT_size; }
Get_AT (n, m)
{ for (i=1; i<=n; i++)   AT_num += Choose (m, i); }
```

Fig. 7. Algorithm to find optimized parameters for Taylor series.

subroutine `Compare_AT_size` is called to compare AT size of each stored node and select the one with the smallest AT representation. In fact, while the algorithm begins with the largest $e_t$ value (within the total bound $E$), at which time $e_i$ is smallest, in subsequent steps, when $e_t$ is shrunk and concurrently $e_i$ is enlarged until $e_i$ reaches the largest value, the procedure explores the search space, while eliminating nodes that will have larger AT than already obtained solutions.

Finally, the two remaining parameters, i.e., the bit-width of the Taylor coefficients ($q$) and the output bit-width ($o$) are ready to be calculated. The bit-width of Taylor coefficients $q$ is computed using the notion of sensitivity, while the output bit-width $o$ is straightforwardly calculated using the expression $o = -\log_2(E - e_t - e_i - e_c) + 1$ (Steps 13 and 14). Note that at this point all the error parameters in the above equation can be determined using the optimal values of $n$, $m$ and $q$.

The algorithm provides a branch-and-bound exploration of the space of all potential optimized nodes. When the error bound $E$ is exceeded, the complete subtree of the search tree is safely abandoned. Further, the search is guided by the sensitivity function, as a heuristic to speed up the search. At each node, the error $e_i$ from (11) is computed in the subroutine `Get_input_error`, which uses the sensitivity definition. The transform of the first order derivative of $f(X)$ is obtained in terms of the Taylor terms $n$ and input bit-width $m$. Then, the imprecision search algorithm is invoked to get its maximum mismatch, so the sensitivity is calculated through multiplication of the maximum mismatch and $\Delta X$, i.e., $2^{-m-1}$.

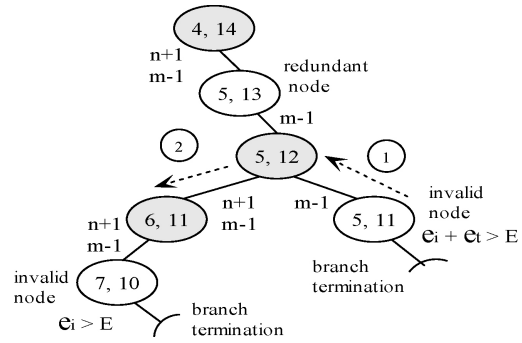


Fig. 8. Search for optimized parameters in Example 5.

As a result, the conversion algorithm is invoked only once to get AT of $f'(X)$, while the use of (5)–(7) would call the algorithm twice. The following example illustrates the use of the precision optimization algorithm.

*Example 5:* Consider an implementation of $\sin(x)$ represented by Taylor series. Due to the given error bound 0.0002, the algorithm finds the least number of Taylor terms to be 4, and the corresponding input bit-width to be 14 on the condition of the Taylor terms. Therefore, the initial node is (4, 14).

The algorithm adds then one Taylor term and cuts one input bit at the same time, hence generating a new node (5, 13). By using the sensitivity, $e_i$ is estimated fast, and as this node satisfies the error bound, input bits are decreased again. However, when the node reached (5, 11), the error addition of $e_t$ and $e_i$ is beyond the bound but $e_i$ is smaller than the bound, and the algorithm backtracks to the previous node (5, 12). The node (5, 13) is redundant because its AT terms number is obviously larger than the node (5, 12), and the node (5, 11) is an invalid node. The procedure is repeated with Taylor terms increased to 6 giving the node (6, 11) which satisfies the bound. The input error $e_i$ of the next node (7, 10) breaks through the error bound so it is an invalid node, which means the smallest input bit-width is 11 regardless of the increase in the number of Taylor terms, so the algorithm stops.

Fig. 8 indicates three nodes (4, 14), (5, 12), and (6, 11) that satisfy the given error bound. Then the procedure `Compare_AT_size` is called to select the node with the smallest AT size, so the node (6, 11) is the optimized parameters for Taylor terms and input bit-width.

From this example, we see that starting from an initial feasible implementation, the algorithm proceeds with generating nodes of improved parameters, and then checks whether such new nodes are within the error bound. In each search step, the sensitivity is used to accelerate calculation of the input quantization error, drastically improving the performance. When the error bound is exceeded, the backtracking technique returns the previously determined feasible solutions, and no solution will be missed.

### *E. Precision Optimization for Multivariate Polynomials*

Although the optimizing algorithm can find the implementation with the smallest area, it can only process Taylor series, that is, limited to one word-level variable. The limitation confines further applications since many real-valued polynomials comprise word-level variables beyond one, the optimization

```
Design_best_poly_imp (f, E)
1. {   for (i=0; i<word_var_num; i++)
2.     {    AT_th = Convert_AT(f(0), i, m0);
3.          AT_real = Convert_Multivar_AT(f(2^-m0-1), i, m0);
4.          error_AT = AT_th – AT_real;
5.          sens [i]=Search_imprecision (error_AT);   }
6.     ini_bit = Get_ini_node (sensitivity);
7.     final_bit = Get_final_node (sensitivity);
8.     for (i=word_var_num-1; i>=0; i--)
9.     {      ini_bit[i]++;   ini_bit[i+1]--;
10.           for (m=word_var_num-1; m>=i; m--)
11.           {   stop_error = Compute_input_error (sens, ini_bit);
12.                ei[0] = pow(2, init_bit[0]-m0) * sens[i];
13.                if (ei[0] >= stop_error)     break;
                   else { while (ei < E)     init_bit[0]--;
                          Store (nodes);   Tight_interval (node); } }
14.           if (ini_bit = final_bit)     break;
          }
15.    Irredundant (nodes);
16.    optimized_node = Compare_AT (nodes); }
```

Fig. 9. Algorithm to find optimized parameters for real-valued polynomials over multiple variables.

algorithm needs extension to process it. In this paper, we solve the problem and present an algorithm to handle the case of specifications given over several word-level variables.

A set of bit-widths for each variable is referred as a *node* in Fig. 9. The algorithm firstly gets sensitivity for each variable as in Steps 1–5, and obtains the initial node and final node by using sensitivity as in Steps 6 and 7. The initial node makes the first variable get minimum bit-width and the final node makes the last variable get minimum bit-width.

Beginning from the initial node, the algorithm shrinks the error generated by the first variable, by increasing its bit-width. At the same time bit-width of the followed variable decreases which will potentially enlarge the error. The procedure propagates the input error within the error bound from the first variable to the last variable in sequence. When the final node is reached, the loop stops and all possible nodes are traversed as in Steps 8–14. While all intermediate nodes are obtained, the redundant nodes are deleted in Step 15.

If two nodes only differ in one variable and other variables have same bit widths, the node which has more bits is the redundant node. For example, if the two nodes have three variables consisting of (12, 13, 12) and (12, 14, 12) bits respectively, only one variable is different and the node of (12, 14, 12) is deleted as a redundant node. The optimized bit-widths for variables are selected by comparing AT sizes of obtained nodes and choosing the smallest one as in Step 16.

*Example 6:* Consider a function $F$ with three word-level variables and the given error bound is 0.006

$$F(X, Y, Z) = 2X^2 + 3YZ - 4Z^3 + XYZ.$$

By using sensitivity the initial node is obtained as (14, 16, 18) which means that the error generated by $X$ has the largest value within the error bound, and the final node is (18, 16, 13) which means that the error generated by $Z$ has the largest value within the error bound. Fig. 10 describes the two nodes and the error is generated by each variable.

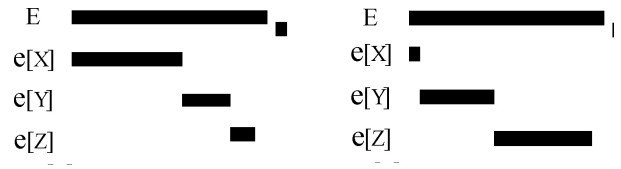


Fig. 10. Error of each variable for the initial node and final node.

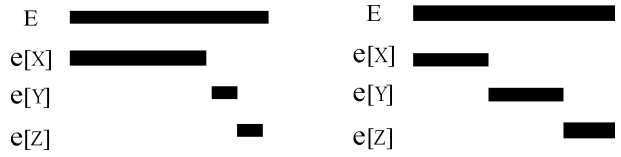


Fig. 11. Two intermediate nodes from the initial node.

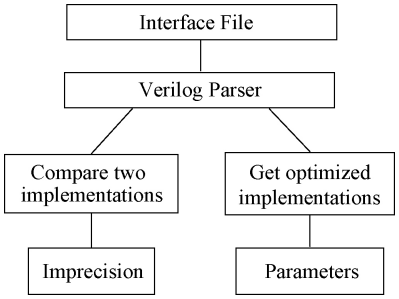


Fig. 12. Overview of the tool ISTSP.

Now the algorithm begins with the initial node to increase bit-width of $Y$ and decrease bit-width of $Z$, etc., $e[Y]$ is shrunk and $e[Z]$ is augmented. The new obtained node is (14, 17, 16) and since the bit-width of $Z$ cannot be cut down any more. Bit-width of $X$ has to be increased to "15" and bit-widths of $Y$ and $Z$ are computed again. Consequently, the node changes to (15, 15, 15). The two nodes are shown in Fig. 11.

The algorithm continues to calculate intermediate nodes until it reaches the final node. It removes the redundant nodes and obtains a search path to represent each node. The path is shown as

$$(14, 16, 18) \rightarrow (14, 17, 16) \rightarrow (15, 15, 15) \rightarrow (15, 16, 14) \rightarrow (16, 14, 16) \rightarrow (16, 15, 14) \rightarrow (17, 14, 15) \rightarrow (17, 17, 13) \rightarrow (18, 16, 13).$$

The AT size of each node is calculated and a node with the smallest size is chosen as the optimized node. In this example the optimized node is (16, 15, 14).

## V. Experimental Results

To help investigate various aspects of imprecise circuits, a tool, Fig. 12, is developed and run on a 512 MB, 2.4 GHz Intel Celeron machine under Linux. The tool addresses Problems 1–4, and the experiments are performed as follows.

### A. Benchmarks

In order to assess the performance of our algorithms, we try several benchmarks, as described next.

TABLE I
ERROR AND PERFORMANCE OF VARIOUS CIRCUITS UNDER DIFFERENT CRITERIA

| Case | Imp Degree 1 | Imp Degree 2 | Imp Bit 1 | Imp Bit 2 | Error AT Terms | Imprecision | Time (s) | Space (MB) |
|---|---|---|---|---|---|---|---|---|
| cos(*x*) | 8 | 8 | 20 | 16 | 224 747 | 1.2e1-5 | 7.98 | 66.6 |
| cos(*x*) | 8 | 8 | 24 | 20 | 1 007 676 | 7.52e−7 | 38.84 | 347.3 |
| cos(*x*) | 10 | 8 | 24 | 20 | 615 115 | 2.75e−7 | 44.16 | 71 |
| cos(*x*) | 10 | 8 | 24 | 24 | 4 533 805 | 2.76e−7 | 214.9 | 523.5 |
| B-splines | 3 | 3 | 20 | 16 | 654 | 2.86e−5 | 0.375 | 0.38 |
| B-splines | 3 | 3 | 24 | 20 | 974 | 1.79e−6 | 6.2 | 0.46 |
| B-splines | 3 | 3 | 28 | 24 | 1356 | 1.12e−7 | 114.4 | 0.55 |
| Chebyshev | 8 | 8 | 20 | 16 | 224 747 | 9.15e−4 | 7.9 | 75.6 |
| Chebyshev | 8 | 8 | 24 | 20 | 1 007 676 | 5.72e−5 | 38.73 | 347 |
| Chebyshev | 9 | 9 | 20 | 16 | 381 267 | 0.0012 | 21.1 | 145 |
| Chebyshev | 9 | 9 | 24 | 20 | 2 147 220 | 7.24e−5 | 132.6 | 599 |
| Filter | 4 | 4 | (16, 16, 16) | (16, 16, 14) | 11 549 | 19.39 | 2.13 | 55.2 |
| Filter | 4 | 4 | (20, 20, 20) | (18, 18, 18) | 307 909 | 3.83 | 23.5 | 221.1 |
| Filter | 4 | 4 | (20, 20, 20) | (20, 18, 18) | 68 156 | 2.36 | 16 | 144.5 |
| DCT | 1 | 1 | 16 | 8 | 512 | 15.62 | 0.08 | 0.24 |
| DCT | 1 | 1 | 16 | 10 | 512 | 3.86 | 0.11 | 0.27 |
| DCT | 1 | 1 | 16 | 12 | 512 | 0.92 | 0.13 | 0.29 |
| Box–Muller | (5, 4) | (4, 4) | (10, 10) | (8, 8) | 219001 | 0.013 | 4.65 | 38.2 |
| Box–Muller | (5, 6) | (5, 4) | (12, 12) | (10, 10) | 613 567 | 0.0068 | 18.3 | 86.5 |

1) *Imprecise Cosine Circuit Implementation:* In ASICs or FPGAs, the pipelined implementation of a cosine circuit represented by finite terms of Taylor series often uses the Horner's polynomial evaluation

$$f(X) = \sum_{i=0}^{N-1} (-1)^i \frac{X^{2i}}{(2i)!} = 1 + X^2 \left( -\frac{1}{2!} + X^2 \left( \frac{1}{4!} + X^2(\cdots) \right) \right).$$

### B. B-splines

Four uniform cubic B-spline basic functions $B_0$, $B_1$, $B_2$ and $B_3$ are defined by

$$B_0(u) = -\frac{1}{6}u^3 + \frac{1}{2}u^2 - \frac{1}{2u} + \frac{1}{6} \quad B_1(u) = \frac{1}{2}u^3 - u^2 + \frac{2}{3}$$

$$B_2(u) = -\frac{1}{2}u^3 + \frac{1}{2}u^2 + \frac{1}{2}u = \frac{1}{6} \quad B_3(u) = -\frac{1}{6}u^3$$

where $u = [0, 1]$. We use different bits to represent $u$ to implement this design and observe imprecision effects.

1) *Chebyshev Polynomial:* A sequence of orthogonal polynomials known as the Chebyshev polynomials is defined by the recurrence relation

$$T_0(X) = 1 \quad T_1(X) = 1 \quad T_{n+1}(X) = 2XT_n(X - T_{n-1}(X).$$

According to the relation, we get

$$T_8(X) = 128X^8 - 256X^6 + 160X^4 - 32X^2 + 1$$

$$T_9(X) = 256X^9 - 576X^7 + 432X^5 - 120X^3 + 9X.$$

2) *Implementations of Cubic Filter:* Cubic filters generally have more than one multiple word-level variable such as benchmarks from University of Utah [15]. Our AT-based method will be used to find the optimal bit-widths and avoid the time-consuming exhaustive simulations. We consider a filter module over three word-level variables

$$F(X, Y, Z) = 16384X^4 + Y^4 + 57344Z^4 + 64767XY^3 + 16127Y^2Z^2 + 8965X^3Z + 19275X^2YZ + 51903XYZ + 32768X^2Y + 40960Z^2 + 32768XY^2 + 49152X^2 + 4869Y.$$

3) *Discrete Cosine Transform (DCT):* We consider the 8 × 8 DCT implementation, where the input is an 8-bit unsigned integer, and the output is encoded by 11 bits. Coefficients for a DCT module are fractional numbers often quantized by 8–16 bits.

4) *Box–Muller Implementation:* Box–Muller algorithm for generating Gaussian random variable is critical to a number of applications such as accurate bit error rate testers. It uses the following expression:

$$Y(X_1, X_2) = (Y_1(X_1) * Y_2(X_2) = \sqrt{-2 \ln X_1} * \cos 2\pi X_2.$$

We represent it by a finite number of Taylor series terms $Y_1(X_1) = \sqrt{-2 \ln X_1}$ around $X_1 = 0.5$ and $Y_2(X_2) = \cos 2\pi X_2$ around $X_2 = 0$

$$Y_1(X_1) = 1.17741 - 1.6984(X_1 - 0.5) + 0.4733(X_1 - 0.5)^2 - 1.582(X_1 - 0.5)^3 + 1.0198(X_1 - 0.5)^4 - 3.3284(X_1 - 0.5)^5 + 2.7848(X_1 - 0.5)^6$$

$$Y_2(X_2) = \sum_{i=0}^{\infty} (-1)^i \frac{(2\pi X_2)^{2i}}{(2i)!}.$$

The implementation consists of two Taylor series and two word-level variables. Imprecisions in two variables affect each other, so it is difficult to evaluate imprecision and get the optimized implementation by past univariate explorations.

### C. Verification of Imprecise Implementations

In this section, we run experiments for comparing implementations in the package shown as Fig. 12. This imprecision search module is critical to assess both the conversion and the imprecision verification algorithms, so it is worth to investigate it standalone. Error AT polynomials are derived from two implemented AT polynomials, from which imprecision can be obtained by a search. The module has advantages of being fast and space-efficient, as we now show.

Table I displays imprecision based on different degrees and input bits. It is obvious that imprecision reduces along with increase of the Taylor degree and input bits. Running time is acceptable even for large number of terms. This module hence supplies a strong capability to calculate imprecision of implementations that engineers can take advantage of, to match any implementations arbitrarily and rapidly. The obtained results also help in understanding whether the existing module implementations can be reused.

### D. Performance of Implementation Optimization

We implemented the precision optimization algorithms as Figs. 7 and 9. More bits imply that the results are more precise, i.e., the implemented function value is closer to

TABLE II

OPTIMIZED IMPLEMENTATIONS AND PERFORMANCE FOR DIFFERENT ERROR BOUNDS

| Circuit | Error Bound | $n$ | $m$ | $q$ | $o$ | $e_t$ | $e_i$ | $e_c$ | $e_o$ | Node | AT Terms | Imprecision | Time [s] | Mem [MB] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\cos(x)/S$ | 5e−4 | 5 | 13 | 14 | 11 | 2.76e−6 | 5.96e−5 | 1.23e−4 | 2.44e−4 | – | 7098 | 4.29e−4 | 1.56 | 1.86 |
| **cos (x)/O** | **5e−4** | **4** | **11** | **15** | **14** | **1.67e−4** | **2.38e−4** | **6.15e−5** | **3.05e−5** | **4** | **1486** | **4.97e−4** | **1.33** | **1.52** |
| $\cos(x)/S$ | 3e−4 | 5 | 14 | 13 | 15 | 2.76e−6 | 3.03e−5 | 2.46e−4 | 1.53e−5 | – | 12 910 | 2.94e−4 | 2.56 | 3.01 |
| **cos(x)/O** | **3e−4** | **4** | **12** | **18** | **17** | **1.67e−4** | **1.19e−4** | **7.69e−6** | **3.8e−6** | **7** | **2509** | **2.97e−4** | **1.58** | **2.13** |
| $\exp(x)/S$ | 3e−4 | 8 | 14 | 15 | 13 | 2.48e−5 | 8.42e−5 | 1.07e−4 | 6.1e−5 | – | 9908 | 2.77e−4 | 1.98 | 2.34 |
| **exp (x)/O** | **3e−4** | **7** | **14** | **18** | **17** | **1.98e−4** | **8.42e−5** | **1.31e−5** | **3.7e−6** | **6** | **6476** | **2.95e−4** | **2.37** | **2.86** |
| B-spline/S | 7e−4 | – | 11 | 11 | 15 | – | 2.45e−4 | 2.43e−4 | 1.5e−5 | – | 231 | 5.03e−4 | 0.09 | 0.18 |
| **B-spline/O** | **7e−4** | **–** | **10** | **12** | **13** | **–** | **4.91e−4** | **1.22e−4** | **6.1e−5** | **1** | **175** | **6.74e−4** | **0.08** | **0.11** |
| Cheby/O | 3e−2 | – | 12 | – | 7 | – | 2.57e−2 | – | 3.91e−3 | 1 | 3797 | 2.96e−2 | 1.42 | 1.53 |
| Cheby/O | 1e−2 | – | 14 | – | 8 | – | 6.54e−3 | – | 1.95e−3 | 1 | 12 911 | 8.49e−3 | 3.84 | 5.14 |
| Cheby/O | 3e−3 | – | 16 | – | 9 | – | 1.64e−3 | – | 9.77e−4 | 1 | 39 203 | 2.62e−3 | 9 | 15.2 |
| DCT/O | 20 | – | – | 8 | – | – | – | 15.71 | – | 1 | 512 | 15.71 | 0.08 | 0.13 |
| DCT/O | 4 | – | – | 10 | | – | – | 3.92 | – | 1 | 512 | 3.92 | 0.11 | 0.14 |
| DCT/O | 1 | – | – | 12 | – | – | – | 0.98 | – | 1 | 512 | 0.98 | 0.13 | 0.15 |
| Filter/S | 50 | (14, 14, 14) | | | | – | 27.6 | – | – | – | 47 865 | 27.6 | 6.7 | 8.9 |
| **Filter/O** | **50** | **(13, 13, 13)** | | | | – | **49.3** | **–** | **–** | **21** | **37 636** | **49.3** | **11.9** | **25.4** |
| Filter/S | 35 | (15, 14, 15) | | | | – | 19.5 | – | – | – | 51 391 | 19.5 | 9.2 | 12.3 |
| **Filter/O** | **35** | **(13, 14, 14)** | | | | **–** | **32.4** | **–** | **–** | **14** | **45 232** | **32.4** | **18.9** | **25.5** |
| Box–Mul/S | 5e−3 | (5, 6) | (12, 12) | 11 | 8 | 1.3e−3 | 5.8e−4 | 6.6e−4 | 1.95e−3 | – | 2 153 903 | 4.5e−3 | 2.68 | 1.58 |
| **Box–Mul/O** | **5e−3** | (5, 6) | **(11, 11)** | **11** | **10** | **1.3e−3** | **2.4e−3** | **6.8e−4** | **4.9e−4** | **13** | **1 620 432** | **4.9e−3** | **5.22** | **6.87** |
| Box–Mul/S | 1e−3 | (7, 6) | (12, 13) | 12 | 11 | 4.2e−5 | 2.8e−4 | 3.3e−4 | 2.5e−4 | – | 9 725 892 | 9e−4 | 7.46 | 4.92 |
| **Box–Mul/O** | **1e−3** | (6, 6) | **(12, 12)** | **13** | **12** | **3.6e−4** | **3.2e−4** | **1.6e−4** | **1.2e−4** | **17** | **5 938 969** | **9.6e−4** | **13.3** | **17.6** |

TABLE III

RESULT COMPARISON WITH THE PAPER [12]

| Case | Precision | Time [12] | Area [12] (Slices) | Time (s) | Area (Slices) |
|---|---|---|---|---|---|
| B-spline | 8 | 0.12 | 1368 | 0.07 | 1132 |
| | 16 | 0.19 | 2188 | 0.15 | 2056 |
| DCT | 8 | 0.89 | 3598 | 0.08 | 857 |
| | 16 | 0.51 | 5069 | 0.17 | 1481 |
| Degree 4 polynomial | 8 | 1.9 | 803 | 0.96 | 763 |
| | 16 | 2.0 | 1921 | 1.55 | 1208 |

TABLE IV

ERROR COMPARISON OF AA AND OUR METHOD

| Case | $n$ | $m$ | AA | Ours |
|---|---|---|---|---|
| $\sin(X)$ | 3 | 9 | 1.52e−2 | 1.1e−3 |
| $\sin(X)$ | 3 | 11 | 1.52e−2 | 2.7e−4 |
| $\sin(X)$ | 4 | 10 | 1.57e−2 | 5.46e−4 |
| $\sin(X)$ | 4 | 12 | 1.57e−2 | 1.37e−4 |
| $\sin(X) * \exp(X)$ | 4 | 8 | 6.7e−2 | 1.5e−2 |
| $\sin(X) * \exp(X)$ | 4 | 11 | 6.7e−2 | 1.9e−3 |
| $\sin(X) * \exp(X)$ | 5 | 8 | 8.9e−2 | 1.48e−2 |
| $\sin(X) * \exp(X)$ | 5 | 11 | 8.9e−2 | 1.87e−3 |

TABLE V

HARDWARE AREA OF OPTIMIZED CIRCUITS

| Circuit | $E$ | Taylor Terms | Input [bits] | Coef. [bits] | Area [Slice] | Saving |
|---|---|---|---|---|---|---|
| $\cos(X)/S$ | 3e−4 | 5 | 13 | 14 | 1037 | – |
| $\cos(X)/I$ | 3e−4 | 5 | 12 | 15 | 965 | 6.9% |
| **cos (X)/O** | **3e−4** | **4** | **12** | **16** | **746** | **28.1%** |
| $\exp(X)/S$ | 3e−4 | 8 | 14 | 15 | 1179 | – |
| $\exp(X)/I$ | 3e−4 | 8 | 14 | 13 | 1136 | 3.6% |
| **exp (X)/O** | 3e−4 | **7** | **14** | **16** | **933** | **20.9%** |
| Cheby/S | 3e−3 | – | 20 | – | 1906 | – |
| **Cheby/O** | **3e−3** | – | **16** | – | **1439** | **24.5%** |
| DCT/S | 4 | – | – | 14 | 1162 | – |
| **DCT/O** | **4** | – | – | **10** | **894** | **23.1%** |
| Filter/S | 35 | – | 15, 15, 15 | – | 3036 | 7.6% |
| **Filter/O** | **35** | – | **13, 14, 14** | – | **2725** | **17%** |
| Muller/S | 1e−3 | (7, 6) | 13, 14 | 13 | 4327 | – |
| Muller/I | 1e−3 | (7, 6) | 12, 11 | 12 | 3986 | 7.9% |
| **Muller/O** | **1e−3** | (6, 6) | **12, 12** | **13** | **3759** | **13.1%** |

the originally specified data output; however, the precision comes at the cost of area, but also of the speed and energy consumption. So choosing an appropriate length to represent coefficients is worth the effort. Two elementary functions ($\cos(x)$ and $\exp(x)$) given by Taylor series, two circuits (*B-spline*, *Chebyshev*) represented by polynomials with one variable are used as benchmarks to assess the effectiveness of Fig. 7. Three circuits (cubic filter, DCT and Box–Muller) verify the algorithm to find optimized implementations of polynomials with multiple input variables in Fig. 9.

Column 2 in Table II gives different error bounds for various functions; Columns 3–10 list obtained parameters and corresponding errors for implementations optimized for the bounds. Columns 11 and 12 show how many nodes are investigated in the whole procedure and the number of obtained AT terms; Column 13 gives total imprecision, always smaller than the given error bound. Time and space requirements are reported in Columns 14 and 15.

For comparison, we invoke the sequential algorithm for solving the Problem 3, a feasible implementation for comparison purpose. By considering the precision parameters sequentially, it mimics often applied schemes for setting precision parameters in isolation. The label "/S" in Column 1 indicates that this sequential assignment algorithm is used while the label "/O" points to the optimization algorithm detailed here. The optimized algorithm traverses more nodes to investigate the real-valued polynomials with multiple variables such as cubic filters and Box–Muller than Taylor series. Please notice that no unique group of parameters satisfies the error bound; changing one parameter would affect others (such as rows 2 and 3, 4 and 5). These rows have different parameters, and all fit the given error bound indicated by Column 2.

It is clear that even when the given error bound is small and bit-widths are large, our algorithm is fast and memory-efficient. In many cases the exact optimization algorithm is faster than the sequential algorithm.

In comparison with the best similar methods, we consider work in [12] that utilizes a multi-stage approach to get 8-bit and 16-bit output precision. Its benchmarks are real-valued polynomials where input word-length is considered—it cannot deal with Taylor series and function approximation. We concern not only the input but coefficients and the output.

Table III compares results with those in [12]. Our algorithm achieves higher speed and smaller area. We also notice that benchmarks in [11] and [12] have lower degrees than ours. We can handle functions with higher degrees, such as Chebyshev polynomials of degree 9, effortlessly. Furthermore, our algorithms are able to process functions with multiple variables. Cubic filters and Box–Muller, which are more difficult for verification and optimization, are used to prove it. We facilitate a more complex exploration of combining as many factors as possible when investigating imprecision and approximation to the specification.

Table IV compares the errors obtained by AA and our method for the same number of Taylor terms and input bit-widths, listed in Columns 2 and 3. The error obtained by our method is far smaller than that of AA, which is an indicator of better accuracy compared to past explorations. Therefore, AT is better to determine whether an implementation is suitable for the specification.

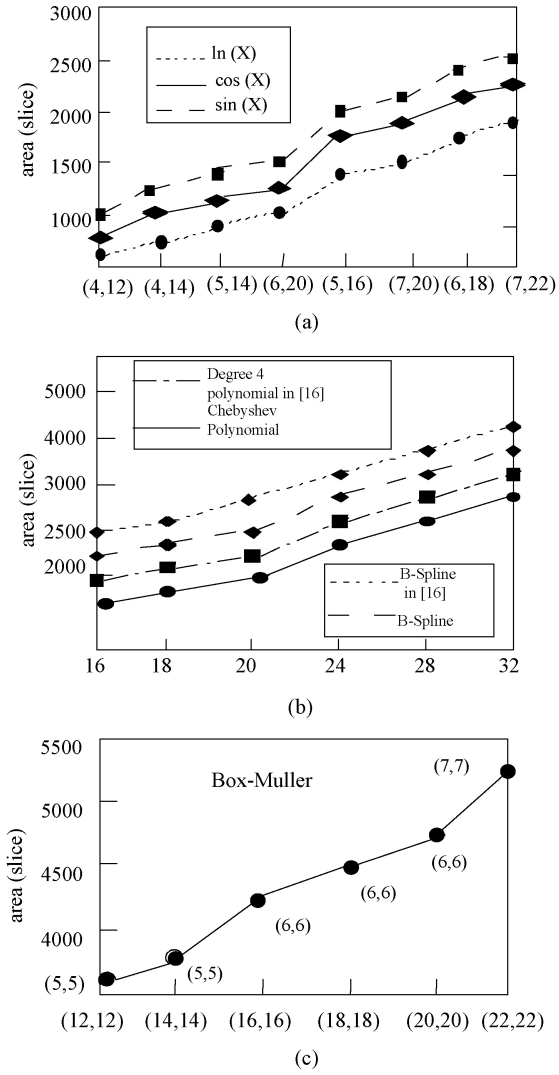


Fig. 13. (a)–(c) Hardware area of Taylor series and real-valued polynomials in different Taylor terms and input bits.

## VI. Area of Mapped Optimized Hardware

While the optimization algorithm produces precision parameters for a minimal size AT polynomial, the exact area of the resulting circuit depends on the technology used in mapping circuits. We perform further experiments with mapping on FPGAs to evaluate the real area impact of the proposed optimization algorithm. In this section, we use the Xilinx Virtex-4 XC4VLX100-12 FPGA, with the ISE tool (version 8.1), the same as in [12], to fairly compare the results.

Table V compares area of the FPGA implementations that satisfy the given error bound $E$, shown in the second column. The rows labeled "/I" use the tight-bound interval method for input and coefficient bit-width, to improve on the sequential algorithm [26], labeled with "/S." This new case produces less input and coefficient bits than the sequential algorithm. The rows labeled "/O" invoke the optimization algorithm that uses the tight interval method from this paper.

The results achieve about 5% area reduction over the optimization algorithm reported in [26] (as "/O"), which uses the plain interval method, for transcendental functions such as $\cos(X)$ and $\exp(X)$. The optimization algorithm in combination with the tight interval method can save the area by up to 30% over the sequential exploration of individual precision parameters.

Fig. 13 describes achievable FPGA hardware area for benchmark circuits using different combinations of Taylor terms and input bits. Such a tabulation facilitates the exploration of tradeoffs between precision and complexity. Shown in Fig. 13(b) are B-spline and Chebyshev polynomial results from [12] for comparison. Our optimization requires less hardware. The benchmarks such as B-splines or Chebyshev polynomial take 80% area of circuits in [12], for the same precision, the same FPGA, which are mapped with the same synthesis tools.

## VII. Conclusion and Future Work

Generally, specifications and implementations do not match exactly for arithmetic circuits, so the flexibility due to the allowed imprecision can be used for the benefit of simplifying the circuits. For real-valued specifications, such as Taylor Series or polynomials, we used AT to explore the solution space within allowed arithmetic imprecision. The basic algorithms were proposed to convert both Taylor Series and multivariate real-valued polynomials into ATs, and seek the maximum mismatch. They compute discrepancy between two implementations efficiently.

An error analysis is a necessary step to match specifications within the given error bound. It has traditionally been complicated, hard to automate, and did not account for disparate approximation and quantization parameters contributing to the overall error, which is what we have overcome here by an AT-based static analysis that is compact and also allows the improvements to the interval analysis, via tight-bound interval analysis.

We achieved optimization of implementations from Taylor series specifications and real-valued polynomial specifications. The work easily incorporates disparate factors that cause the arithmetic error. As traditionally designers apply separate isolated error bounding techniques for mostly the word-length parameters contributing to the imprecision, we are unique in searching for the optimized implementation given a totality of approximation and quantization effects. The use of the intermediate AT is essential to achieve the task through automation. Algorithms are devised to determine precision parameters that guarantee sufficient precision with a minimal AT polynomial size. The proposed techniques can be applied to wider classes of real and complex-valued function specifications, not just the Taylor expansions.

Currently, the exploration only handles a high-level representation so it does not consider truncation error of intermediate signals. The limitation of our approach consists of inability of processing rational functions, since operations such as divisions cannot be handled directly. In the future, the work presented here will add the factor of intermediate variables and be extended to cover rational functions, and incorporate the error range analysis and related optimizations stemming from range constraints, all in a single unified method.

## REFERENCES


[1] J. Smith and G. De Micheli, "Polynomial circuit models for component matching in high-level synthesis," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 6, pp. 783–800, Dec. 2001.

[2] O. Sarbishei, M. Tabandeh, B. Alizadeh, and M. Fujita, "A formal approach for debugging arithmetic circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 5, pp. 742–754, May 2009.

[3] S. Kim and W. Sung, "Fixed-point error analysis and word length optimization of 8 × 8 IDCT," *IEEE Trans. Circuits Syst. Video Tech.*, vol. 8, no. 8, pp. 935–940, Dec. 1998.

[4] K. Kum and W. Sung, "Combined word-length optimization and highlevel synthesis of digital signal processing systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 8, pp. 921–930, Aug. 2001.

[5] M. Willems, V. Bürgens, H. Keding, T. Grötker, and H. Meyr, "System level fixed-point design based on an interpolative approach," in *Proc. Design Autom. Conf.*, 1997, pp. 293–298.

[6] A. Gaffar, O. Mencer, W. Luk, and P. Cheung, "Unifying bit-width optimisation for fixed-point and floating-point designs," in *Proc. IEEE Symp. FCCM*, 2004, pp. 79–88.

[7] C. Shi and R. Brodersen, "Automated fixed-point data-type optimization tool for signal processing and communication systems," in *Proc. Design Automat. Conf.*, 2004, pp. 478–483.

[8] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Precision and error analysis of MATLAB applications during automated synthesis for FPGAs," in *Proc. DATE*, 2001, pp. 722–728.

[9] C. Fang, R. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," in *Proc. ICCAD*, 2003, pp. 275–282.

[10] Y. Pu and Y. Ha, "An automated, efficient and static bit-width optimization methodology toward maximum bit-width-to-error tradeoff with affine model," in *Proc. Asia South Pacific Design Automat. Conf.*, Jan. 24–27, 2006. p. 6.

[11] D.-U. Lee, A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. Constatinides, "Accuracy-guaranteed bit-width optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 1990–2000, Oct. 2006.

[12] W. G. Osborne, J. Coutinho, R. Cheung, W. Luk, and O. Mencer, "Instrumented multi-stage word-length optimization," in *Proc. Field-Program. Technol.*, Dec. 2007, pp. 89–96.

[13] N. S. Nedialkov, V. Kreinovich, and S. A. Starks, "Interval arithmetic, affine arithmetic, Taylor series methods: Why, what next?" *Numerical Algorithms*, vol. 37, nos. 1–4, pp. 325–336, 2004.

[14] G. Constantinides, P. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 10, pp. 1432–1442, Oct. 2003.

[15] S. Gopalakrishnan and P. Kalla, "Optimization of polynomial datapaths using finite ring algebra," *ACM Trans. Design Automat. Electron. Syst.*, vol. 12, no. 4, pp. 1–49, Sep. 2007.

[16] A. Ahmadi and M. Zwolinski, "Symbolic noise analysis approach to computational hardware optimization," in *Proc. 45th ACM/IEEE DAC*, vols. 8–13. Jun. 2008, pp. 391–396.

[17] A. Kinsman and N. Nicolici, "Finite precision bit-width allocation using SAT-modulo theory," in *Proc. DATE Conf. Exhibit.*, Apr. 20–24, 2009, pp. 1106–1111.

[18] D.-U. Lee, A.A. Gaffar, O. Mencer, and W. Luk, "Optimizing hardware function evaluation," *IEEE Trans. Comput.*, vol. 54, no. 12, pp. 1520–1531, Dec. 2005.

[19] D. Lee and J. D. Villasenor, "A bit-width optimization methodology for polynomial-based function evaluation," *IEEE Trans. Comput.*, vol. 56, no. 4, pp. 567–571, Apr. 2007.

[20] K. Radecka and Z. Zilic, "Arithmetic transforms for compositions of sequential and imprecise datapaths," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 7, pp. 1382–1391, Jul. 2006.

[21] P. Yu, K. Radecka, and Z. Zilic, "Arithmetic transforms of imprecise datapaths by Taylor series conversion," in *Proc. IEEE ICECS*, 2006, pp. 696–699.

[22] S. Wadekar and A. Parker, "Accuracy sensitive word-length selection for algorithm optimization," in *Proc. IEEE Int. Conf. Computer Design*, 1998, pp. 54–61.

[23] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement," in *Proc. ACM/IEEE DATE*, 1999, pp. 271–276.

[24] K. Radecka and Z. Zilic, "Specifying and verifying imprecise circuits by arithmetic transforms," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2002, pp. 128–131.

[25] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*. London, UK: Academic Press, 1985.

[26] Y. Pang, K. Radecka, "Optimizing imprecise fixed-point arithmetic circuits specified by Taylor Series through arithmetic transform," in *Proc. 45th ACM/IEEE DAC*, Jun. 2008, pp. 397–402.

[27] D. Menard and O. Sentieys, "Automatic evaluation of the accuracy of fixed-point algorithms," in *Proc. ACM/IEEE DATE Conf.*, 2002, pp. 1530–1591.

[28] S. Kim, K. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," in *Proc. Workshop VLSI Signal Process.*, Nov. 1995, pp. 197–206.

[29] W. Sung and K. I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Signal Process.*, vol. 43, no. 12, pp. 3087–3090, Dec. 1995.

[30] J.-I. Choi, H.-S. Jun, and S.-Y. Hwang, "Efficient hardware optimization algorithm for fixed point digital signal processing ASIC design," *Inst. Elect. Eng. Electron. Lett.*, vol. 32, no. 11, pp. 992–994, May 1996.

[31] M. A. Cantin, Y. Savaria, and P. Lavoie, "A comparison of automatic word length optimization procedures," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2. 2002, pp. 612–615.

[32] K. Han and B. L. Evans, "Wordlength optimization using sensitivity information," *EURASIP J. Appl. Signal Process. Spec. Issue Design Methods DSP Syst.*, vol. 2006, no. 5, pp. 1–14, Jan. 2006.


**Yu Pang** (M'09) received the B.S. degree in electrical engineering from Sichuan University, Sichuan, China, in 2000, and the M.S. degree (honors) in communication and information engineering from the University of Electronic Science and Technology of China, Sichuan, in 2003. He is currently working toward the Ph.D. degree from the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada.

He was a Teaching and Research Assistant throughout his B.S. and M.S. studies. His current research interests include high-performance VLSI design and verification, communications, reversible logic and parallel computing.

**Katarzyna Radecka** (S'00–M'02) received the B.Eng., M.Eng., and Ph.D. degrees from McGill University, Montreal, QC, Canada, in 1995, 1996, and 2003, respectively.

She was with Nortel, Ottawa, Canada, from 1995 to 1996, with Lucent Technologies, Allentown, PA, from 1996 to 1998, and with Concordia University, Montreal, from 2002 to 2007. She is currently with McGill University. Her current interests include arithmetic circuits, verification and test of hardware and software. She has published over 50 publications and has authored the book, *Verification by Error Modeling: Using Testing Methods for Hardware Verification* (Norwell, MA: Kluwer).

**Zeljko Zilic** (S'91–M'97–SM'07) received the Ph.D. degree from the University of Toronto, Toronto, Ontario, Canada, in 1997.

From 1997 to 1998, he was a Technical Staff Member with FPGA Division, Microelectronics Group, Lucent Technologies, Allentown, PA. Currently, he is an Associate Professor with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada. He is researching various aspects of system design, test and verification. He has published over 200 research papers, holds four patents in the area of clock and power management, and has co-authored the books, *Verification by Error Modeling* (Norwell, MA: Kluwer, 2003) and *Generating Hardware Assertion Checkers* (Berlin, Germany: Springer, 2008).

Dr. Zilic is a recipient of a Chercheur Strategique Research Chair from the Province of Quebec. He received the Myril B. Reed Best Paper Award from IEEE International Midwest Symposium on Circuits and Systems in 2001, the Best Paper Award from Design and Verification Conference in 2005, and several Honorary Mention Awards. For his undergraduate teaching, the National Council of Deans of Engineering and Applied Science and Sandford Fleming Foundation awarded him with the Wighton Fellowship in 2006.