Enabling Practical Uses of Arithmetic Transforms – A Comprehensive Analysis

Zeljko Zilic and Katarzyna Radecka
McGill University
3480 University, Montréal, Québec, Canada
{zeljko.zilic, katarzyna.radecka}@mcgill.ca

Abstract

Arithmetic Transform (AT) has been known under various names, including Inverse Integer Reed-Muller (IRM) transform. We outline the developments that place AT in the center of very practical arithmetic circuit applications, and explain the conditions for making the AT practical. We show that AT allows the only known scheme of treating all the approximation and imprecision sources of arithmetic error in the optimization, through an efficient static method. Finally, the role of AT in the stochastic computing is highlighted.

1. Introduction

Arithmetic Transform (AT) has been known under numerous names, including the Inverse Integer Reed-Muller (IRM), "probabilistic", "adding" and "algebraic" transform. It is an orthogonal expansion defined on pseudo-Boolean functions, where the arguments (inputs) are Boolean n-tuples, while the function value (output) can be over a range of domains, such as integers, rational numbers or rings over integers (i.e. modulo arithmetic). Then, the Reed-Muller (RM) transform is a special case of AT, where the output domain is integer ring Z_2, i.e., integers modulo 2. Although it is more suitable to associate AT with the inverse of RM, the fact is that RM and its inverse are the identical transformations.

AT has been already considered for numerous applications in a number of domains. The applications elaborated up to the end of 20th century are summarized thoroughly by Falkowski [4] they include probabilistic analysis of switching activities, reliabilities in the networks, fault signatures etc. In spite of the large body of literature on AT and its uses, its use has been limited as either the applications were computationally too demanding, or the means to obtain AT were not practical for large functions.

In this paper, we show how AT can be useful for arithmetic circuit verification and optimization, and explain the conditions for making AT practical. A number of properties of AT are shown that facilitate dealing with imprecise fixed-point arithmetic circuits. The practical means are exposed to construct and manipulate AT and the related representations, and the ways are shown towards AT usability into the stochastic computing, believed to be of significance for emerging nanoscale technologies.

2. Background

AT is defined as a transform over pseudo-Boolean functions, $f : B^n \mapsto w$, where it suffices that the output set w is at Abelian group, i.e, a set with an operation of addition and subtraction. The AT is a polynomial:

$$f = \sum_{i_0=0}^{1} \sum_{i_1=0}^{1} \cdots \sum_{i_{n-1}=0}^{1} c_{i_0 i_1 \ldots i_{n-1}} x_0^{i_0} x_1^{i_1} \ldots x_{n-1}^{i_{n-1}}. \quad (1)$$

Hence, AT expresses a function using the set of linearly independent functions defined as: $x_0^{i_0} x_1^{i_1} \ldots x_{n-1}^{i_{n-1}}$, where

$$x_j^{i_j} = \begin{cases} x_j, & \text{if } i_j = 1 \\ 1, & \text{if } i_j = 0 \end{cases}, j = 0, \ldots, n-1.$$

We say that the *arithmetic spectrum* is a set of coefficients $c_{i_0 i_1 \ldots i_{n-1}}$, each of which multiplies an orthogonal basis function $x_0^{i_0} x_1^{i_1} \ldots x_{n-1}^{i_{n-1}}$. A fairly broad and useful framework to consider is when $w = R$, i.e., the functions over real numbers. Further cases, where the arithmetic is restricted to integers, will be used as well. Basis functions can be directly related to the discrete Fourier Transform, such as those in Walsh-Hadamard Transform (WHT) [20, 23]. In the case of AT, the basis consists of monomials, and the transform is simply treated as a polynomial.

An important observation is that AT allow the outputs to be grouped into the word-level quantities. Then, the *encoding* of word-level quantities determines the exact way the calculation is done, i.e., the meaning of the operations "+" and "−", which we already indicated are all that is needed

Table 1. Common arithmetic valuations

Word	Unsigned	Sign-extended	2's Complement
Integer	$\sum_{i=0}^{n-1} x_i 2^i$	$(1-2x_{n-1})\sum_{i=0}^{n-1} x_i 2^i$	$\sum_{i=0}^{n-2} x_i 2^i - x_{n-1}2^{n-1}$
Fractional	$\sum_{i=1}^{n-1} x_i 2^{-i}$	$(1-2x_0)\sum_{i=1}^{n-1} x_i 2^{-i}$	$-x_0 + \sum_{i=1}^{n-2} x_i 2^{-i}$
Generalized Fixed-Point	$\sum_{i=1}^{n-1} x_i 2^{i-m}$	$(1-2x_0)\sum_{i=1}^{n-1} x_i 2^{i-m}$	$-x_0 2^{m-n} - \sum_{i=1}^{n-2} x_i 2^{i-m}$

in the calculation of an AT. In the remainder of this section, AT will be defined constructively, via valuation function and the methods of AT calculation.

Definition 1. *The function* $valuation : B^n \mapsto w$ *is the word-level value calculation undertaken according to the encoding of arithmetic values.*

Simply, valuations provide the "numerical" value that a Boolean vector takes for a given encoding. The valuations for common encodings are summarized in Table 1. For a given word-level value encoding of a Boolean vector, the AT is equal to its valuation, which will be used to assist in arriving at AT in various instances. The valuation function can be easily expanded to describe more complex functions, most notably the arithmetic ones, as shown next.

Example 1. *Consider an unsigned 2-bit adder and a multiplier over word-level quantities* a *and* b*. Their valuation for unsigned integer encoding, and hence, their ATs are:*

Adder: $a + b$ *(for 2-bit unsigned a,b)*

$$valuation(a+b) = valuation(a) + valuation(b) = (a_0 + a_1 * 2) + (b_0 + b_1 * 2) = 2(a_1 + b_1) + (a_0 + b_0)$$

Multiplier: $a * b$ *(for 2-bit unsigned a,b)*

$$valuation(a*b) = valuation(a) * valuation(b) = 4a_1b_1 + 2(a_0b_1 + a_1b_0) + a_0b_0$$

2.1. Calculating AT coefficients

The practical uses of AT are impossible without the efficient ways of obtaining it from the common circuit descriptions. As apparent from the definition, the straightforward method of obtaining the transform coefficient would be exponential in the number of variables in fact, the square of the number of minterms (i.e., points) of a function.

Similarly to RM, there is a number of ways to obtain an AT of a function. The following methods exist in practice:

1. **Matrix transform**: the coefficient vector is $c = Tf$, where T is the lower-triangular matrix that, relative to the RM matrix, has every odd-indexed matrix entry below the diagonal is multiplied by -1. Hence, $T_0 = 0$ and
$$T_n = \begin{bmatrix} T_{n-1} & 0 \\ -T_{n-1} & T_{n-1} \end{bmatrix} \quad (2)$$

2. **Fast transform** that, similar to FFT, employs a butterfly diagram to speed up the multivariate AT through a series of univariate real-valued Davio expansions
$$f = f_{x=0} + x(f_{x=1} - f_{x=0}). \quad (3)$$

3. **Lattice-based (interpolation) construction** [22] is suitable for sparse, incompletely specified functions. Figure 1 shows an instance of Boolean lattice B_{2^4} with the AT of a multiplier from Example 1 inscribed next to the shaded nodes. Coefficients are obtained by evaluating the function in layers 0 and 1 and applying exhaustively the Davio expansion among the pairs of points linked by an edge. In general, the AT of an n-bit multiplier has $O(n^2)$ nonzero coefficients, all of them in Layer 2, and the AT of an adder has $O(n)$ non-zero coefficients in Layer 1.

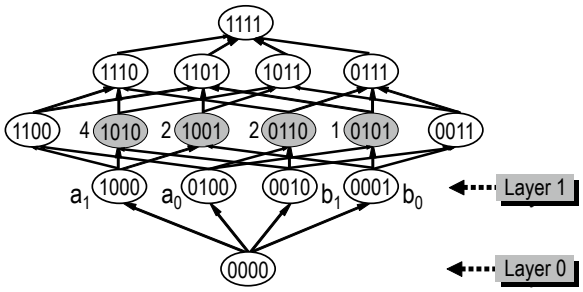

Figure 1. Lattice-based transform of a 2-bit multiplier: non-zero coefficients highlighted

4. **Rewriting of arithmetic expressions** that expands the real-valued variables with their word-level quantities via the valuation function. Expansion from Taylor Series is a specific instance in which arbitrary real-valued functions can be expressed by AT.

5. **Composition from blocks** through mixed AT (MAT) and MAT sequential (MATS) [24] extensions of AT. MAT allows both bit- and word-level inputs, and can be thought of as a hybrid between AT and Taylor series. Because it allows word-level inputs, the outputs of AT or MAT can be directly MAT, to result in a composition of the transforms. MATS additionally allows sequential elements, as well as the sequential unrolling to convert to MAT and eventually AT. Efficient algorithms for the composition were presented in [15].

6. **Graph-based** transforms construct directly graph-based AT representations, such as *BMD [2]. As the decision diagrams are well explained elsewhere, it is only worth pointing out that they are the efficient directed acyclic graph (DAG) representation of the full binary tree-based Davio expansion from Eqn. 3.

Example 2. *AT of the inverter $f(x) = x'$ is obtained by multiplying the matrix T_1 from Eqn. 2 and the vector of values $[1\ 0]^T$ using integer arithmetic. The resulting vector of coefficients $c = [1\ -1]^T$ encodes the function $1 - x$.*

Similarly, the Davio expansion, Eqn. 3, as well as the lattice traversal over B_2, result in $f = f_0 + x(f_1 - f_0) = 1 - x$.

The matrix and even fast AT transforms are mostly impractical, but often offer the way to reason about transforms, and relate to the body of other works in transforms.

3. AT and arithmetic functions

We summarize here the basic properties of AT that are useful in arithmetic circuit optimizations. First, we present the direct way in which the AT can be related to the infinite-precision real-valued specifications. Next, we explain the properties amenable to the efficient precision analysis, and finally we show how a combination of these properties leads to the ability to perform better precision analysis and optimization than the known fixed-point methods, and the mainstream floating-point Remez method.

3.1. Relation to real-valued specifications

Arithmetic circuits most commonly implement real-valued specifications of functions that are in general multivariate, i.e., $f : R^n \mapsto R$. We indicated in Sec. 2.1 that for a number of real-valued specifications, including polynomials and Taylor series, there is a direct and efficient construction of AT. In this section, we show the benefits of the construction of the AT from real-valued specifications, such as Taylor series, and present the details needed for efficiency in algorithms.

For an infinitely differentiable real function f over interval I around point X_0, the infinite Taylor series converging over I are:

$$f(X) = \sum_{i=1}^{\infty} \frac{f^{(i)}(X_0)}{i!} * (X - X_0)^i. \tag{4}$$

In practical implementations, the series become truncated to the first $n + 1$ terms, in which case the remainder is provably bounded by an $(n + 1^{st})$ order derivative:

$$R_n(X) = \frac{f^{(n+1)}(\xi)}{(n+1)!} * (X - X_0)^{n+1} \tag{5}$$

where ξ is a point in the given interval I.

The finite Taylor expansion is straightforwardly converted to an AT. For an instance of m-bit fixed-point presentation the arithmetic rewrite scheme from Sec. 2.1 replaces all instances of the real-valued X with the valuation function over a m-tuple of bits. For instance, with unsigned encoding, the replacement is:

$$X \leftarrow \sum_{j=0}^{m-1} x_j 2^j$$

and the AT is obtained by expanding the $(n+1)$-term Taylor polynomial (where without loss of generality $X_0 = 0$ and $f_0 = f(X_0)$):

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

The single most challenging issue in the conversion to AT is that of the intermediate polynomial swell during the transformation. The techniques for the efficient rewrite of arithmetic expressions are well studied and elaborated in, for instance, symbolic computing area, where such rewrites are often met.

Having the efficient construction of AT from real-valued specification, we can simply apply the precision analysis from Sec. 3.2, which allows us to efficiently verify whether a fixed-point implementation of a real-valued specification achieves the required precision over the subset of the domain of the function that is of interest.

3.2. AT and arithmetic precision

Arithmetic circuits that implement algorithms conceived on infinite-precision real-valued functions are bound to be imprecise. There are two major causes for that. First, there are *approximations* applied to the real-valued specification.

The second cause of the *imprecision* is based on a finite-word implementation. For example, real numbers are realized using finite-size words as fixed-point data representations.

The *imprecision error* is a distance between the specification and implementation, most commonly measured as the maximal absolute difference between the two, i.e., their distance in the uniform norm L_∞. The error is commonly expressed in the terms of the *units in the last place* (ULP), i.e., relative to the length of the fixed-point implementation.

The precision analysis cannot be dealt with at the bit-level, and the explicit representation of output word-level values is required. Consider an example of arithmetic computation where all output bits are incorrect, while the imprecision can be made arbitrarily small by extending the wordlength.

Example 3. *Assume that the exact n-bit fractional result is*

$$value_{exact} = 1.00\ldots0,$$

while the approximation is

$$value_{appr} = 0.11\ldots1.$$

While all bits are incorrect, the error is one ULP, which can be arbitrarily small by increasing n.

Since AT represents the function output at a word level, it can be used for reasoning directly about the imprecision. On the other hand, keeping the input variables at a bit level allows a convenient way to explore the domain of the function definition while searching for the maximal difference. In other words, if there is a function implementation f that includes an error e, then its AT is

$$valuation(f + e) = valuation(f) + valuation(e),$$

by linearity of AT. Furthermore, in order to compare the arithmetic values for two functions, we just compute

$$valuation(f - g) = AT(f) - AT(g).$$

The maximum absolute value of the difference

$$||AT(f) - AT(g)||_\infty = \max_I |f - g|$$

over the domain of the definition I will be an error between the two. No bit-level representation can deal so succinctly and directly with the imprecision in the implementation.

Expressing precision by AT. The AT, understood and used as valuation function, offers a direct way to express the imprecision properties of the circuit. For the same intended arithmetic function f, given two word-level implementations with p and q bits denoted as f_p and f_q, respectively, the two valuations will be denoted as $valuation_p(f), valuation_q(f)$. Further, the arithmetic mismatch between the two will be:

$$\begin{aligned} valuation_p(f) - valuation_q(f) &= AT(f_p) - AT(f_q) \\ &= AT(f_p - f_q) \end{aligned}$$

For a given reference implementation f_{ref}, the imprecision due to the n-bit wordlength is then directly expressed as

$$AT(f_{ref}) - AT(f_n).$$

Hence, the imprecision to the finite wordlength n is expressed in terms of maximum absolute distance in L_∞ as

$$||AT(f_{ref}) - AT(f_n)||_\infty = ||AT(f_{ref} - f_n)||_\infty. \quad (6)$$

In consequence, to find the imprecision in the L_∞ norm, it suffices to search for a maximum absolute value that the AT of a difference between the reference and the implementation functions take.

Furthermore, the other distance functions, like the sum of squares (i.e., distance in L_2 norm) could be equally applied as well. Since the maximum absolute value of the imprecision is the common metric for usual arithmetic and DSP implementations, in the rest of the paper, we assume the L_∞ norm and its induced distance.

3.3. Relating arithmetic circuit techniques

In practice, arithmetic and DSP circuit designers usually perform the individual approximation and wordlength imprecision analyses sequentially, where each step usually gives a pessimistic bound. As they assume that the overall imprecision error is a sum of error bounds obtained in each step, this introduces further serious pessimism and forces the designers to waste the circuit resources (and consequently, speed and energy consumption). The more efficient, state-of-art fixed-point and floating-point techniques are outlined next.

Fixed-point analysis and optimization often rely on the dynamic (i.e. simulation-based) evaluation schemes [9, 31, 28]. Up to our best knowledge, all such schemes deal only with the bit-width assignment, and the function approximation was not tackled in an integral manner. For instance, Kim and Sung [9] develop a tool for the area optimization via the word-length exploration through simulation, where the word-length sensitivity is obtained by simulating a signal flow graph and the cost is assessed through the model of the hardware.

Interval, affine and symbolic noise analysis Most often, the error beyond the ULP is modeled by the *interval analysis* (IA), by which the error is assumed to be growing linearly with the size of the interval beyond the ULP, with the

worst case being half of the ULP. Nayak et al. [12] develop a tool for Matlab specifications, while [5] use static affine arithmetic (AA) modeling, but only for the word-length determination, similar as Pu and Ha [17]. Affine arithmetic assumes that the imprecision is a linear combination of the sources of uncertainity, contributing to the imprecision. While AA is relatively simple, it tends to be overly pessimistic. A good comparison of IA, AA and the Taylor series-based methods is performed in [13]. Just recently, Ahmadi and Zwolinski [1] have introduced the symbolic noise analysis, which assumes a probability distribution in representing an error over an interval, which is useful in overcoming the pessimism of IA and AA, but relies on an unknown error distribution.

Function approximations The work dealing with function approximation, especially in conjunction with transcedental functions, includes the the methods that use general polynomial representations [10], as well as Taylor series-based lookup table approximations [27]. In the latter, the detailed analysis is performed on Taylor series truncation, while the word-length is determined as a final step. Such schemes are similar to the floating-point design, outlined next.

Floating-point circuit designers often rely on Remez algorithm [11], which is a scheme for finding the best function approximation in the uniform norm L_∞ by a fixed-degree polynomial. The Remez algorithm is preferred to Chebyshev approximations and other schemes. Since it essentially addresses only the approximation component of the imprecision, which is an acceptable simplification in the case of the floating point circuits, especially with large mantissa and exponent. In other words, the classical polynomial approximation, such as Remez algorithm is not that useful for fixed-point circuits.

Instead, by relying on the provable error bounds on Taylor series, followed by the AT construction and error analysis from Sec. 3.2 has a number of advantages, most notably the uniform way of dealing with all sources of imprecision and approximation. In essence, our construction is tighter as it locates exactly the worst case imprecision, rather than adding conservatively the error bounds. With AT, we can also get tighter error bounds than with the interval and affine analysis.

4. Arithmetic circuit applications

4.1. Verification of imprecise circuits

With the properties of AT outlined so far, we have all the elements to present an elegant and efficient precision verification algorithm. All that is needed is to construct the AT and evaluate whether the imprecision from Eqn. 6 is acceptable, i.e., smaler than a given bound ε. The problem is specified as follows.

Problem 1 PRECISION VERIFICATION
Input: $f_{ref}, f_{impl}, \varepsilon$
Output: TRUE *iff* $||AT(f_{ref}) - AT(f_{impl})||_\infty < \varepsilon$

This formulation is also practical if a reference f_{ref} by itself is imprecise within a bound, say δ. For instance, the transcendental functions can only have the approximate reference implementations, such as with Taylor series. Then, we can apply the triangle inequality

$$||AT(f_{abs}) - AT(f_{impl})||_\infty \leq ||AT(f_{abs}) - AT(f_{ref})||_\infty + ||AT(f_{ref}) - AT(f_{impl})||_\infty$$

among the absolutely precise f_abs, the reference and the implementation, to guarantee that $|\varepsilon + \delta|$ is an acceptable imprecision.

A very important fact is that AT allows us to devise a completely *static verification* scheme that does not require any circuit simulation (i.e., dynamic verification) towards establishing the correctness of the circuit under precision constraints. For that, a branch-and-bound method was presented in [21] that finds whether the worst case imprecision of a fixed-point implementation is smaller than the allowed imprecision. The exact branch-and-bound search for the maximal imprecision (hence, the distance in L_∞) is reduced to a search for a maximal value that an AT polynomial takes, as per Eqn. 6. The algorithm explores the assignments of variables in an implementation AT, by means of a tree-like traversal. In fact, the classical traversal is shortened when the bounding techniques is developed that allows us to abandon the inutile subtrees.

To devise and efficient search tailored for AT representation, such static verification scheme rely on the further useful properties of AT polynomials, explained in more detail next. First, we recall that for any efficient branch-and-bound search, one needs

- easily computable bounds to terminate early the search that will not contribute to the solution,
- the heuristic guidance functions that help with the performance and
- the easily identifiable special cases where a solution is known in advance, without performing a search.

In the case of the precision verification, to apply each of these three techniques to the AT polynomial maximal absolute value search, we outline three further easily obtainable algebraic properties of AT.

Bounds on AT polynomials The key observation leading to the bounding stage in branch-and-bound search are based

on the easy lower and upper bounds, lb and ub that an AT can take. Since the AT coefficients are multiplied by 1 or 0, the extreme cases occur when only the coefficients in one polarity have a non-zero multiplier. Then, an upper bound is attained for all positive coefficients, denoted $c > 0$

$$ub = c_{00\ldots0} + \sum_{c>0} c_{i_1 i_2 \ldots i_n}, lb = c_{00\ldots0} + \sum_{c<0} c_{i_1 i_2 \ldots i_n}.$$

One can further notice that

$$lb + ub = f_{00\ldots0} + f_{11\ldots1},$$

i.e., one bound can be derived from the other by sampling the function at only two points, when variables are all 0 or all 1.

Most positive variables for search ordering heuristic (splitting variable) To increase the chance of early terminations due to a too small upper bound on achievable value, it is desired that the early stages of the search produce as high value of AT as possible. For that, for each variable x_i, we examine the coefficients c_{*i*} that multiply the AT terms with variable x_i being present. Without performing a more detailed search for this heuristic step, one can identify the variable mpv with the highest absolute value sum of the coefficients c_{*mpv*}

$$mpv = i \; s.t. | \sum c_{*i*} | \rightarrow max.$$

While this selection clearly ignores the role that the assignments of the other variables take, as a heuristic guidance, it suffices, and can be considered as the best candidate leading to the sought maximal value. In the case of a tie between the two variables, the second order statistics are applied. In this case, we note that the lower bound provides an estimate "from below", as opposed to the initial estimate from above. The lower bound will provide additional information and will help to narrow down the uncertainty inherent in such heuristic guidance.

Unate functions for easy solutions Other properties of AT lead to speedup of the search algorithm by preprocessing and handling of special cases. If at any stage a function is positive (negative) *unate*, i.e., monotonously growing (decreasing) in any variable, then the variable is safely assigned 1(0) in the search for the maximum. Unateness in a variable is another property that is easily computed on an AT: it suffices to check whether the coefficients c_{*i*} multiplying all occurences of the variable i are in one polarity. The preprocessing step for an unate variable i is then denoted as

$$\forall i \; c_{*i*} \geq 0 \Rightarrow x_i \leftarrow 1$$

$$\forall i \; c_{*i*} < 0 \Rightarrow x_i \leftarrow 0.$$

In every step of the search algorithm, the preprocessing is applied to assign the variables by checking unateness. Furthermore, an assignment of a variable due to unateness can possibly result in other variables becoming unate, so the preprocessing needs to be applied in a fixpoint manner, until there is no change in unateness.

Algorithm 2 outlines the precision search. The recursively called procedure MAX_ABS explores the search tree. A subtree corresponding to an assignment of a variable to 1 or 0 is examined if the upper bound ub is not smaller than an already obtained best value. When all variables are traversed, the maximal absolute value for that variable assignment is compared against the best one obtained so far. The searches are backtracked when the ub for the subtree is too small relative to the best currently obtained value.

Algorithm 2 PRECISION (f, ε)

```
1: ub = c_{00...0} + Σ_{c>0} c_{i1 i2 ... in}; lb = f_{00...0} + f_{11...1} − ub
2: CurrentBest = lb; Current = ub;
3: return(MAX_ABS (AT(f), Current) < ε)
4: MAX_ABS (AT(f), Current)
5: {
6: FIXPOINT(PREPROCESSUNATEVARS(AT))
7: if UnassignedVars then
8:    x = MostPositiveVar;
9:    if ub(AT(f_{x=1}) > CurrentBest then
10:      Current = MAX_ABS (AT(f_{x=1}), Current)
11:   end if
12:   if ub(AT(f_{x=0}) > CurrentBest then
13:      Current = MAX_ABS (AT(f_{x=0}), Current)
14:   end if
15: else
16:   Current = AT;
17:   CurrentBest = |max(Current, CurrentBest)|
18:   return CurrentBest
19: end if
20: }
```

The description of the verification presented here is complemented with the implementation details of the efficient algorithm presented in [16].

4.2. Component matching

Another application where AT can play useful role is that of finding the most suitable library components for imprecise arithmetic, and here we demonstrate how the above body of techniques can be adjusted to deal with this problem. The component matching has surfaced as a problem in conjunction with IP library reuse. The work such as [29] has attempted to use real-valued polynomials to deal with the matching of real-valued functions. A significant limitation of that approach is that the known apparatus for such

polynomial model manipulation is limited to polynomials in one variable, and the work [29] is applicable fully only to univariate polynomials.

With AT, being multivariate to start with, the transition to multivariate real-valued polynomial specifications is straightforward. Further, the search for a suitable library component from a given library lib is reduced to the problem PRECISION VERIFICATION, applied sequentially to all the elements of lib.

The problem is concisely defined as that of finding a precise enough library cell element for a given arithmetic function.

Problem 3 PRECISE COMPONENT MATCHING
Input: $f_{ref}, lib, \varepsilon$
Goal: $||AT(f_{ref}) - AT(f_{impl})||_\infty < \varepsilon$
Output: $f_{impl} \in lib$

4.3. Optimization of fixed-point arithmetic

We show next how AT plays a role as a very practical mechanism for optimizing the fixed-point implementations of real-valued specifications. Starting from a polynomial or Taylor series description, the algorithm automatically selects the multiple approximation and imprecision parameters by a scheme that is essentially branch-and-bound, with a number of optimizations specific to this case.

While this approach could be applied to a variety of real-valued representations, we present the main ideas applied to the case of finite Taylor expansion. The optimization problem is defined as that of finding the parameters such as the number of Taylor terms, n, and the wordlength m of the input representation, such that the cost is miminal.

Problem 4 TAYLOR PRECISION OPTIMIZATION
Input: f_{ref}, ε
Goal: $min\ terms(AT(f_{n,m}))$
Goal: $||AT(f_{ref}) - AT(f_{n,m})||_\infty < \varepsilon$
Output: $m,\ n$

Before proceeding with elaborating the automated precision optimization, we first present few additional properties of AT that make it suitable for this application. First, the AT is shown as usable as an technology-independent area cost function measure, and then the properties suitable for the search heuristic order.

AT as a cost function We want to accurately predict the area of an arithmetic circuit within the optimization procedure, such that the method can be again be completely static, rather than involving simulations and technology mapping. In considering the pre-mapping, technology-independent area cost estimate suitable for arithmetic optimization, we want the cost function that is simple, yet accurate in terms of relative comparison among the alternatives. In the case of Taylor functions with parameters n and m, the area cost is clearly exhibiting a monotonous growth relative to both. More input bits m implies wider datapaths and arithmetic operators whose cost grows at least linearly in m. On the other hand, with more Taylor coefficients, the input argument X gets raised to a larger exponent and, as there is $O(n)$ terms in the Taylor series summation, with at least $O(n)$ operations for each terms, the growth in n is at least quadratic, so the cost grows higher in n.

Consider now the number of AT terms of an expanded nth order Taylor polynomial over m-bit fixed point numbers as a cost function in the optimization procedure. By enumeration, it can be shown that the number of AT terms is then equal to

$$cost = terms(AT(f_{n,m})) = \sum_{i=1}^{n} \binom{m}{i}$$

In this case, the cost function exhibits the same monotonicity in the two variables. By using this cost function, we will effectively be searching for a minimal size AT polynomial implementing a function within a given precision.

Search guidance - sensitivity Similar to the previous instance of branch-and-bound, providing a search heuristic is a significant way to speed up the algorithm. In this case, we can apply the domain-specific knowledge about the *sensitivity* of a function relative to the change in either n or m.

Sensitivity of function f with respect to variable x is defined as a derivative of a function $\frac{df}{dx}$, and as such is used in a wide range of continuous function optimizations. Here, variables n and m are discrete, and need to be mapped to continuous variables that they present.

In the case of m, the wordlength, an increase of one bit results in the added shift of half a unit in last place, ulp.

$$\Delta X = 2^{-(m+1)}.$$

Then, the influence of the wordlength change to the function is

$$\Delta f = \frac{df}{dX}\Delta X = \frac{df}{dX}2^{-(m+1)}.$$

Furthermore, by recalling that the first Taylor term is a derivative at the expansion point X_0, the sensitivity is calculated as an $m+1$-fold shift of the first coefficient. Since this is true only around X_0, a more accurate sensitivity is obtained by differentiating the function. AT can be used here as well, to easily differentiate the given function. By finding the maximal value over the interval, as in the precision search, the worst case influence of the variable change is obtained.

In the case of sensitivity to n, we can apply the error bound from Eqn. 5. When a new n is to be selected in the search, the difference between the two remainder functions, bounding approximation error, can be easily computed. Further, when going, say, from n to $n+1$ terms, the function difference is

$$\Delta f = Taylor_{n+1}(f) - Taylor_n(f) + R_n(f) - R_{n+1}(f)$$

The algorithm takes into account in an unified way all the approximation and imprecision sources. For simplicity, it is instructive to think of it as an optimization algorithm for real-valued specifications given by Taylor series, where only the parameters n and m are to be chosen. The pseudocode is given in Algorithm 5. It starts with the number of terms given by Eqn. 5 and then explores m and n according to the sensitivity.

Algorithm 5 OPTIMIZE (f, ε)

```
1: lb = MINTAYLORTERMS
2: CurrentBest = lb; Current = ub;
3: MIN_AREA (AT(f), Current)
4: {
5: if UnexploredPosibilities then
6:    direction = FINDMOSTSENSITIVEVAR
7:    if (direction ==m) then
8:       Current = MIN_AREA (AT(f_m), Current)
9:    else
10:      Current = MIN_AREA (AT(f_n), Current)
11:   end if
12: else
13:   Current = AT;
14:   CurrentBest = |min(Current, CurrentBest)|
15:   return CurrentBest
16: end if
17: }
```

Example 4. *Consider $sin(x)$ with $\varepsilon = 0.0002$. By Eqn. 5, the initial n is 4. To satisfy the error bound the initial m is set to 14, so the starting node is $(4, 14)$, Figure 2. The algorithm then adds one Taylor term and explores the decrease in m, to reach node $(5, 13)$. By applying sensitivity, the value of m can be decreased twice, to reach $(5, 11)$, which, upon closer examination, is beyond the error bound ε. The algorithm then backtracks to $(5, 12)$ and finds that $(6, 11)$ also satisfies the bound. All other nodes exceed the bound, and the search stops.*

4.4. Performance of AT precision tasks

While we find no other method that treats all the approximation and imprecision sources, the presented methods compare well to the best and most comprehensive modern methods [14], which by itself improves over the work by the same group of authors. Table 2 shows the FPGA mapping results, done with our fixed-point optimization on the same FPGAs and the tools as [14]. The experiments were run on a 2.4GHz Celeron PC with 512MB of main memory.

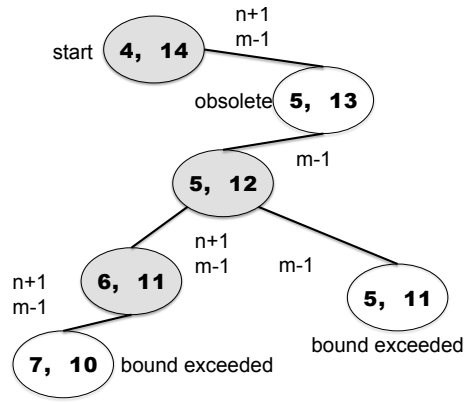

Figure 2. Optimization of sin function

Table 2. FPGA area comparison with [14]

Case	Bits	Time [14]	Area [14]	Time	Area
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
Deg-4	8	1.9	803	0.96	763
Poly	16	2.0	1921	1.55	1208

5. AT and reliable nanoscale computing

We present in this section a few areas where AT can be applied to timely problems, such as resilience to errors with emerging nanoscale integration technologies. An interesting proposal combines the benefits of the traditional uses of the AT for reliable computing with the arithmetic circuit treatment as in the previous sections.

The arithmetic circuits, as considered in previous sections, could be assisted with different number encodings, such as a well-explored trigonometric CORDIC scheme, suited for avoiding hardware multipliers. In recent years, several authors [18, 30] have revisited the stochastic encoding [7], by which the arithmetic quantities are expressed as pseudo-random series of ones and zeros, where the encoded value corresponds to the probability of ones. Stochastic circuits are now interesting because they can tolerate better the errors arising with emerging technologies: in a fault model of a random bit flip, any such error is smaller than or equiv-

alent to an ULP, i.e., an error in a least significant bit.

5.1. AT and reliability analysis

Numerous authors (see [4, 32] for a more detailed account) have used AT in reliability and testing studies. While others have applied AT to the reliability analysis [26] and the manufacturing fault testing signatures [8], we have focused on verification by testing methods, by schemes such as mutation testing [23]. As for our most valuable lesson, seeing that the spectrum of common design errors are relatively small, which leads to the use of techniques such as polynomial interpolation for error correcting codes, to extend the known interpolation decoding for RM codes [25]. Such techniques are capable of bringing fast a set of test vectors that guarantee high coverage of injected faults.

5.2. AT application to stochastic computing

We show now that AT is a natural description for operations over stochastic computing number representations. In stochastic computing, the real-valued quantities, normalized in absolute value by one, are represented by a stream of pseudo-random bits, where the encoded quantity V is equal to the probability of one for the stream. There are only a few basic circuit blocks for this logic, which nevertheless are sufficient, i.e., complete in the sense of generating a real-valued function.

The operations possible with stochastic logic are:

- **Multiplication**. For two input streams, A and B, the output O is a bitwise multiplication of the two. For every time instance i,

  $$O_i = A_i * B_i$$

  and the corresponding signal probabilities are $p_O = p_A * p_B$. It is easy to observe that a single 2-input AND gate suffices to perform this operation.

- **Scaled Addition**. Here, two input streams, A and B, are selected according to the probability that a third input stream x takes 0 or 1, respectively. Then, in every time instance i, the output O_i is

  $$O_i = x_i * A_i + (1 - x_i)B_i$$

  with the corresponding bit probability values calculated likewise as $p_0 = p_x P_A = (1 - p_x)p_B$.

Even more importantly to us, the scaled addition is the basic building block for AT, i.e., a real-valued Davio expansion. By rewriting the scaled addition definition, we obtain:

$$O_i = x_i * A_i + (1 - x_i)B_i = B_i + x_i(A_i - B_i)$$

which is exactly as the definition of the expansion for AT in Eqn. 3, with the two cofactors being functions A and B, and the splitting variable x.

5.3. Constructing stochastic circuits by AT

It was noted earlier [19] that a stochastic logic computes a multivariate polynomial in a serial manner, but we are not aware that the link between stochastic computing and AT has been made earlier. The polynomial expressing the signal probability is

$$p_f = \sum_{i_0=0}^{1} \sum_{i_1=0}^{1} \cdots \sum_{i_{n-1}=0}^{1} \alpha_{i_0 i_1 \ldots i_{n-1}} p_{x_0}^{i_0} p_{x_1}^{i_1} \cdots p_{x_{n-1}}^{i_{n-1}} \quad (7)$$

Furthermore, by considering a single time instance, where probability p_x is replaced by variable x in Eqn. 7, the obtained expression takes the exact form as the AT definition in 1.

More important issue is that of converting between AT and a stochastic computing circuits and formulas. Relying on the basic primitives in stochastic computing, the two rules are applied

Stochastic circuit to AT. The operation of obtaining the AT definition for a given stochastic circuit topology is performed by the following two rules, applied in the topological order over the network.

1. Each AND gate in stochastic circuit is represented as a product of the two input ATs
2. Each scaled addition (mux) gate is a Davio Expansion, with the control variable to mux being the expansion variable in AT.

AT to stochastic circuit. The operation is performed exactly opposite to the previous case. Please notice that in this case, it is advantageous to use the decision diagram-based expressions, because the factorizations and decompositions were already performed, to result in more efficient circuits.

6. Conclusions and future directions

Arithmetic Transform is a natural match for reasoning about, verifying and optimizing fixed-point arithmetic circuits. We have presented the basic methods behind several practical applications of AT in fixed-point verification and synthesis, as well as in the stochastic logic circuitry. Our emphasis was on discerning the basic algebraic properties of AT that become useful in a variety of ways, rather than the algorithm implementation efficiency. Among the points completely omitted are the graph-based representations of AT and derived forms, as well the real-valued specifications, such as Taylor Expansion Diagrams [3, 6].

In future, the AT-based methods could be extended to deal with the overflow/underflow cases, as well as towards providing assistance in the debug of arithmetic circuits, i.e.,

finding the root cause in the case of failures associated with arithmetic circuits.

References

[1] A. Ahmadi and M. Zwolinski. Symbolic noise analysis approach to computational hardware optimization. *Proceedings of Design Automation Conference*, pages 391–396, June 2008.

[2] R. E. Bryant and C.-Y. Chen. Verification of arithmetic functions with binary moment diagrams. *Proceedings of Design Automation Conference*, pages 535–541, May 1995.

[3] M. Cieselski, P. Kalla, Z. Zeng, and B. Rouzeyre. Taylor expansion diagrams: a compact, canonical representation with applications to symbolic verification. *Proc. Design Automation and Test in Europe*, pages 285–289, March 2002.

[4] B. J. Falkowski. A note on the polynomial form of Boolean functions and related topics. *IEEE Transactions on Computers*, 48(8):860–864, August 1999.

[5] C. Fang, R. Rutenbar, and T. Chen. Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs. *Proceedings of International Conference on Computer-Aided Design*, pages 275–282, November 2003.

[6] G. Fey, R. Drechsler, and M. Cieselski. Algorithms for Taylor expansion diagrams. *Proc. International Symposium on Multiple-Valued Logic*, pages 155–161, May 2004.

[7] B. R. Gaines. Stochastic computing systems. *Advances in Information Systems Sciences*, 2:33–172, 1969.

[8] K. D. Heidtman. Arithmetic spectrum applied to fault detection for combinational networks. *IEEE Transactions on Computers*, 40(3):217–221, March 1991.

[9] S. Kim and W. Sung. Combined word-length optimization and high level synthesis of digital signal processing systems. *IEEE Transactions on Computer-Aided Design*, 20(8):921–930, August 2001.

[10] D.-U. Lee and J. Villasenor. A bit width optimization for polynomial-based function evaluation. *IEEE Transactions on Computers*, 25(4):567–571, April 2007.

[11] J.-M. Muller. *Elementary Functions Algorithms and Implementations*. Birkhauser, Berlin, 1997.

[12] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee. Precision and error analysis of Matlab applications during automated synthesis for FPGAs. *Proceedings of Design Automation and Test in Europe*, pages 722–728, April 2001.

[13] N. S. Nedialkov, V. Kreinovich, and S. A. Starks. Interval arithmetic, affine arithmetic, Taylor series methods: What comes next? *Numerical Algorithms*, 37(1-4):325–336, March 2004.

[14] W. G. Osborne, J. Coutinho, R. Cheung, W. Luk, and O. Mencer. Implementing multi-stage word-length optimization. *Proceedings of Field Programmable Technology*, pages 158–165, December 2007.

[15] Y. Pang, K. Radecka, and Z. Zilic. Fast algorithms for compositions of arithmetic transforms and their extensions. *Proc. of NEWCAS-TAISA*, pages 232–235, June 2008.

[16] Y. Pang, K. Radecka, and Z. Zilic. Verification of fixed-point circuits specified by taylor series using arithmetic transform. *Proc. of NEWCAS-TAISA*, pages 422–425, June 2008.

[17] Y. Pu and Y. Ha. An automated, efficient and static bit-width optimization methodology towards maximum bit-width-to-error tradeoff with affine arithmetic model fixed-point data type optimization tool for signal processing and communication systems. *Proceedings of Asia South Pacific Design Automation Conference*, page 6pp, January 2006.

[18] W. Qian, J. Backes, and M. D. Riedel. The synthesis of stochastic circuits for nanoscale computation. *Proceedings of International Workshop on Logic Synthesis*, pages 97–104, June 2007.

[19] W. Qian and M. D. Riedel. Synthesis of robust polynomial arithmetic with stochastic logic. *Proceedings of Design Automation Conference*, pages 567–572, June 2008.

[20] K. Radecka and Z. Zilic. Relating arithmetic and Walsh spectra for verification by error modeling. *Proceedings of 5th International Workshop on Applications of Reed-Muller Expansions in Circuit Design*, pages 205–214, August 2001.

[21] K. Radecka and Z. Zilic. Specifying and verifying imprecise sequential datapaths by arithmetic transforms. *Proceedings of International Conference on Computer-aided Design, ICCAD'02*, pages 128–131, November 2002.

[22] K. Radecka and Z. Zilic. *Verification by Error Modeling: Using Testing Methods for Hardware Verification*. Kluwer Academic Publishers, Boston, Massachussets, 2003.

[23] K. Radecka and Z. Zilic. Design verification by test vectors and arithmetic transform universal test set. *IEEE Transactions on Computers*, 53(5):628–640, May 2004.

[24] K. Radecka and Z. Zilic. Arithmetic transforms for compositions of sequential and imprecise datapaths. *IEEE Transactions on Computer-Aided Design*, 25(7):1382–1391, July 2006.

[25] R. M. Roth and G. M. Benedek. Interpolation and approximation of sparse multivariate polynomials over GF(2). *SIAM Journal of Computing*, 20(2):301–309, April 1991.

[26] W. G. Schneeweiss. On the polynomial form of Boolean functions: Derivations and applications. *IEEE Transactions on Computers*, 47(2):217–221, February 1998.

[27] M. J. Schulte and J. E. Stine. Symmetric bipartite tables for accurate function approximation. *Proc. International Symposium on Computer Arithmetic*, pages 146–154, May 1997.

[28] C. Shi and R. Brodersen. Automated fixed-point data type optimization tool for signal processing and communication systems. *Proc. Design Automation Conference*, pages 478–483, June 2004.

[29] J. Smith and G. De Micheli. Polynomial circuit models for component matching in high-level synthesis. *IEEE Transactions on VLSI*, 9(6):783–800, December 2001.

[30] S. Tehrani, S. Mannor, and W. J. Gross. Survey of stochastic computation on factor graphs. *Proc. International Symposium on Multiple-valued Logic*, pages 112–117, June 2007.

[31] M. Willems, V. Burgens, H. Keding, T. Grotker, and H. Meyr. System-level fixed-point design based on interpolative approach. *Proceedings of Design Automation Conference*, pages 293–298, June 1997.

[32] S. Yanushkevich. *Logic Differential Calculus in Multi-Valued Logic Design*. Technical University of Szczecin Press, Szczecin, 1998.