Research Journal of Applied Sciences, Engineering and Technology 4(11): 1455-1461, 2012
ISSN: 2040-7467

Submitted: November 17, 2011 Accepted: December 16, 2011 Published: June 01, 2012

An Efficient Method for Imprecise Arithmetic by Taylor Series Conversion

Yu Pang
University of Posts and Telecommunications, Chongqing 400065 China

Abstract: The imprecise circuits engender more complexity for design and verification. The typical circuits are Taylor series. To process the imprecise circuits, we adopt a spectral technique, that is, Arithmetic Transform (AT). A new method based on AT is proposed in this paper to compute representations of imprecise datapaths for purpose of equivalence checking and component matching. From a Taylor Series, we devise an efficient algorithm to produce Arithmetic Transform (AT) which is a compact function representation. Also, a searching algorithm is introduced for calculating the imprecision between the specification and the implementation.

Key words: Arithmetic datapath, arithmetic transform, imprecision, Taylor series

INTRODUCTION

Arithmetic datapath circuits pose important challenges in the development of circuit representations suitable for verification. Modern microprocessors, ASICs and DSP circuits utilize various arithmetic circuits that vary in area, power and delay constraints. Since a lot of different realizations exist, the problems of choosing the best module and verifying whether the module matches specifications become critical.

The mainstream verification methods use Reduced Ordered Binary Decision Diagrams (ROBDD), which have an advantage of being canonical and often compact. However, its limitations are exponential size for circuits such as multipliers Zhou and Burleson (1995). Consequently, it is infeasible to represent a complex circuit including arithmetic by representing functions only at a bit level. Numerous extensions are proposed to overcome these limitations Bryant and Chen (1995), Hamaguchi *et al.* (1995) and Ciesielski *et al.* (2002). Many transcendental arithmetic functions such as sin(*x*) and log(*x*) are realized by Taylor Series. Of course realizations will only have finite terms of Taylor Series, which invariably would lead to an error. Imprecision further comes from a finite-word representation of real numbers, as well as from various other function approximations. Precision analysis is necessary to make use of the fixed-point number representation, which is attractive in balancing complexity, cost and energy consumption. Frequently, engineers care whether the aggregate error is beyond a given upper bound. If not, the implementation is suitable for the specification and engineers may use the existing modules directly.

In mathematics, the Taylor series is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. It may be regarded as the limit of the Taylor polynomials. Let *f(X)* be a real and differentiable function corresponding to an algebraic expression.

Definition 1: The function can be represented as *Taylor Series* using a variableand an initial constant:

$$f(x) = \sum_{n-0}^{\infty} \frac{1}{n!}(x - x_0)^n f^{(n)}(x_0)$$

$$f(x_0) + xf'(x_0) + \frac{x^2}{2} f''(x_0)...+ \frac{(x - x_0)^n}{n!} f^{(n)}(x_0) + R_n(x)$$

=

where *f'(X), f''(X),* etc, are first, second and higher derivatives of *f,* and $R_n(X)$ is a Lagrange remainder.
AT is a canonical polynomial representing uniquely multi-input and multi-output Boolean functions $f: B^n \to B^m$ Radecka and Zilic (2002). To obtain an AT description in a form of a single polynomial, multi-output can be grouped to form a word-level (integer) number , leading to a pseudo Boolean function $f: B^n \to w$. Therefore, the AT representation has Boolean inputs and a word-level output.

Definition 2: The Arithmetic Transform (AT) Radecka and Zilic (2001) is a polynomial representing a pseudo Boolean function $f: B^n \to w$. using an arithmetic operation "+", word-level coefficients $Ci_1i_2mi_n$, binary inputs $x_1, x_2,\ldots x_n$ and binary exponents $i_1, i_2,\ldots i_n$:

$$AT(f) = \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} \ldots \sum_{i_n=0}^{1} ci_1i_2...i_n x_1^{i_1} x_2^{i_2} ... x_n^{i_n}$$

AT has been applied for circuit synthesis, verification and testing by many researches. *Klaus* Heidtmann (1991) develops a new method based on AT for the derivation of

fault signatures for the detection of faults in single-output combinational networks. The signatures do not require exhaustive testing so they provide substantially less work than syndrome testing or the verification of Rademacher-Walsh spectral coefficients. Radecka and Zilic (2006) exploit the algebraic properties of the AT that are used in the compact graph-based representations of arithmetic circuits. Verification time can be shortened under assumption of corrupting a bounded number of transform coefficients. Bounds are derived for a number of test vectors and the vectors successfully verify arithmetic circuits under a class of error models derived from proposed basic design error classes including single stuck-at faults. In Lui and Muzio (1986), it describes a methodology for simulation-based verification in the presence of a fault model. The author proposes an implicit fault model that is based on the AT representation of a circuit and design faults. The proposed approach has the advantage of compatibility with formal verification and manufacturing testing methods. Errors can be modeled implicitly, and such an implicit error model is given by AT of a difference between the correct and faulty circuits.

This study introduces concepts of Taylor Series into the framework based on Arithmetic Transform (AT), including the use of Taylor Series with AT. Bottlenecks of conversion include time and memory requirements. We design efficient algorithms to solve the computational bottlenecks and obtain a good performance. From specified Taylor Series and implemented Taylor Series, an error AT polynomial can be obtained. An upper bound to an error polynomial magnitude is computed by a precision search algorithm. Finally we apply the proposed algorithms to several arithmetic circuits.

Generating at from Taylor series: Given Taylor Series and a binary vector (x_{N-1}, x_{N-2},…x_0), we compute the expression:

$$f(x) = f_0 + Xf(X_0) + \frac{X^2}{2} f(X_0) + \frac{X^3}{3!} f(X_0)...$$

where X is a word-level variable. If X is an unsigned integer, its corresponding AT polynomial form is:

$$AT\left[f(X)\right] = f_0 + \left(\sum_{i=0}^{N-1} x_i 2^i\right) f_0' + \frac{\left(\sum_{i=0}^{N-1} x_i 2^i\right)}{2} f_0'' + ...$$

By the rule that Boolean algebra xin equals xi, lots of expanded terms are identical and they should be combined to form a simplified AT polynomial. A straightforward method multiplies each factor recursively, and gets an intermediate polynomials, then simplifies it by using the Boolean rule, so the AT polynomial is achieved. Although the procedure is easy to comprehend, complexity in the calculation comes from large Taylor degrees and bits number which leads to a large size of the intermediate polynomial since it comprise a great many expanded terms.

For example, with degree k = 7 and input bits N = 16, the number of intermediate terms increases to over 2000000. Consequently, storage and grouping of the same terms are major hurdles and result in low efficiency. We now show how to perform conversion into AT polynomial that handles efficiently the intermediate data swell.

The key problem in converting Taylor Series into an AT polynomial is the calculation of the corresponding:

$$\sum_{i=0}^{n-1} (2^i x_i)^k + C_k^m (2x_1)^m x_0^{k-m}$$

AT terms $\sum_{i=0}^{n-1} (2^i x^i)^k$. Assumed N≤ k., the above sum can be obtained as:

$$\left(\sum_{i=0}^{n-1} 2^i x_i\right)^k = + C_k^m C_{k-m}^p (4x_2)^m (2x_1)^p x_0^{k-m-p.....} \quad (1)$$

where C_k^ms defined as $C_k^m = \binom{k}{m}$. The structure of equation will be explored to reveal the possibility to derive an efficient conversion algorithm. In particular, the following property is used for efficient grouping of common terms.

Property 1: For AT raised to the power k, Eq. (1), the sum of the individual variable's exponent is k for each term.

Proof: The calculation of the sum requires k-1 multiplication, where all bit-level variables in a single factor have a fixed component '1". Through each multiplication procedure, the term's exponent augments one and its beginning exponent is also one, so finally the total exponent is k-1+1= k.

Property 2: If an AT term has p variables, the largest exponent which a variable can obtain is the Taylor degree k subtracting variables number p plus 1 and the least exponent is 1 in all expanded isomorphic terms.

Proof: If a variable appears in an AT term, that's easy to know it has an exponent "1" at the lowest. In terms of Property 1, the summed exponent of the p variables is Taylor degree k, while other p-1 variables all have a least exponent "1", the variable can get the largest exponent, etc., k-p+1.
For example, if:

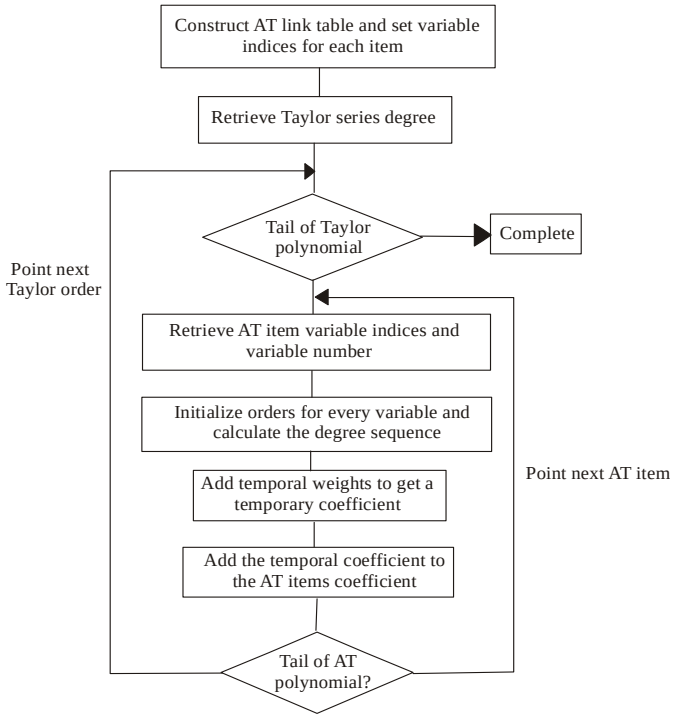

Fig. 1: Algorithm of converting Taylor series into AT

$$(\sum_{i=0}^{2} 2^i x_i)^4 = x_0^4 + (2x_1)^4 + C_4^3(2x_1)^3 x_0 + C_4^2(2x_1)^2 x_0^2 + C_4^1(2x_1)x_0^3 + (4x_2)^4 + C_4^3(4x_2)^3 x_0 + C_4^2(4x_2)^2 x_0^2 + C_4^1(4x_2)x_0^3 + C_4^3(4x_2)^3(2x_1) + C_4^2(4x_2)^2(2x_1)^2 + C_4^1(4x_2)(2x_1)^3 + C_4^2C_2^1(4x_2)^2(2x_1)x_0 + C_4^1C_3^2(4x_2)(2x_1)^2 x_0 + C_4^1C_3^1(4x_2)(2x_1)x_0^2$$

It is clear that the total degree of each term is *k*, and after combination there are 2^n-1=7 items, where each item includes all possible terms. For the item $x_2x_1x_0$ all possible terms are:

$$x_2^2x_1x_0, x_2x_1^2x_0x_2x_1x_0^2$$

Let *msv* and *lsv* represent most significant and least significant variables, respectively. For example, if N = 3,k = 6, the inputs are (x_2, x_1, x_0). For the item $x_2x_1x_0$, x_2 is *msv* and x_0 is *lsv*; for the term x_1x_0, x_1 is *msv* and x_0 is *lsv*. (m,o,p) expresses degrees of x_2, x_1 and x_0. Now we focus to compute AT term $x_2x_1x_0$, at first *msv* is set largest order, etc, 4, so degrees of x_1 and x_0 are 1. Then, first order representation is (4, 1, 1) and temporary coefficient is $C_6^4 * C_2^1 * 4^4 * 2$; after the coefficient is stored, a next degree representation is finished. Beginning with *lsv*, previous variables are searched until one variable with degree not "1" is discovered, so the variable is x_2, its degree decreases 1 and the degree of back variable $x_1$increases 1, after the procession, degree representation changes (3,2,1) and temporary coefficient is $C_6^3 * C_3^2 * 4^3 * 2^2$. The procession continues until *lsv* has set the largest degree to 4, and other variable degrees are 1. At this time, degree presentation turns into (1,1,4) and the ultimate coefficient can be obtained by adding each temporary coefficient. Below is s transformation of the degree sequence:

(4,1,1)→(3,2,1)→(3,1,2)→(2,3,1)→(2,2,2)→(2,1,3)→(1,4,1)→(1,3,2)→(1,2,3)→(1,1,4).

Figure 1 describes the algorithm in detail. The algorithm first constructs a link table and assigns indices for each element, then begins a loop to retrieve the Taylor degree until the last degree is processed. Within each loop, the algorithm picks up each item of AT link table, gets the degree sequence after initialization, and uses Eq. (1) to compute temporary coefficients, before AT coefficients are obtained.

Imprecision computation: Saving costs and speeding up a design are so important to engineers, whenever available, they benefit from reusing a previously designed module. However, these modules usually do not match specifications so they are only approximations. If discrepancy (imprecision) is within an acceptable

```
Search_max (AT_poly)
{ const = Remove_constant(AT_poly);
  var_index = Mpv(AT_poly);
  rev_AT_poly = Reverse(AT_poly);
  rev_var_index = Mpv(rev_AT_poly);
  value_0 = Decompose(AT_poly, var_index);
  value_1 = Decompose(AT_poly, rev_var_index);
  value_2 = Decompose(rev_AT_poly, var_index);
  value_3 = Decompose(rev_AT_poly, rev_var_index);
  max_value = Max(value_0, value_1);
  |min_value| = Max(value_2, value_3);
  mismatch = Max (|max_value+const|, |min_value+const|; }
Decompose(AT_poly, mpv)
{ for (i=0; i<var_num; i++)
   { flag = Preprocess(AT_poly, mpv[i]);
     if (flag = 1)
       { AT_1 = AT(f)_{x_i=1}, ub_1 = Ub(AT_1);
         AT_0 = AT(f)_{x_i=0}, ub_0 = Ub(AT_0);
         if (ub_1> ub_0) AT = AT_1;
         else AT = AT_0;
       }
       Delete_var (mpv[i]); var_num--;
       for (I = 0; i<var_num; i++)
       { flag = Preprocess(AT_poly, mpv[i]);
         if (flag = 0)
           Delete_var (mpv[i]); var_num--;
       }
  }
}
Preprocess (AT_poly, x_i)
{ if (all c_{x_i} > 0) val = 1;
 else if (all c_{x_i} < 0) val = 0; else return 1;
 AT = AT(f)_{x1} = val; return 0;
}
```

Fig. 2: Searching maximum absolute value in AT

boundary, it could be chosen. The approximations come from various aspects and this paper concentrates on restrict input space and finite realization of Taylor series. Therefore, a good solution to find difference between specifications and implementations is significant.

With a known specification and implementation of Taylor Series, specification AT polynomial and implementation AT polynomial is obtained with algorithm mention above. Then, an error AT polynomial is generated by subtracting the two polynomials, to which we add the error due to the finite length Taylor expansions. For the latter, we notice that the Langrange remainder has a provable error bound (as a function of the length of the expansion) that we readily use. Imprecision between the specification and implementation is then the maximum absolute value of error AT. A naive way of trying every input value to compute the error AT is unfeasible, and a much faster algorithm is necessary.

Definition 3: The error AT is a subtraction of specified and implemented AT polynomials, and its maximum absolute value is the maximum mismatch which denotes difference between the specification and the implementation.

A straightforward approach tries every input value to compute its error AT. The procedure requires 2^N calculation because of total 2^N possible inputs. Experiments indicate that such an approach would require an infeasible amount of time, and therefore a fast algorithm is necessary. In this work we propose such an improved algorithm.

For each input variable x_i, s_i is a sum of coefficients multiplying terms with x_i. The *most positive variable (mpv)* is the variable x_j where the sum s_j is largest. An *upper bound ubcoef* of AT polynomial is by summing all coefficients that are positive. Such a bound is calculated as:

$$ub_{coef} = \sum_{c>0} c_{i_1 i_2 \ldots i_n}$$

Algorithm 2 describes the procedure to find the maximum mismatch. To avoid calling the main search loop unnecessarily, the algorithm checks whether there are the input assignments to be made without the search. Such a preprocessing step is used at each call of the search routine. Also a preprocessing step is used at each call of the search routine.

- Assign $x_i = 1$ if coefficients of the AT monomials with x_i present are all positive (or zero).
- Assign $x_i = 0$ if coefficients of the AT monomials with x_i present are all negative (or zero).

The algorithm first removes the constant in the polynomial if it exists, and gets the mpv sequence as the order of decomposition variables, and then the negated AT polynomial and the negated mpv sequence are obtained easily.

A subroutine of Decompose is invoked to compute the maximum value and the minimum value due to the two AT polynomials and two sequences. The preprocessing step deals with a variable to explore whether it can be evaluated directly by probing into its coefficients; if not, the algorithm chooses a path which has a larger upper bound. Figure 2 describes the imprecision searching algorithm in detail.

Example 1: Consider the following AT polynomial:

$$AT(f) = -2 + x_0 - 3x_1x_0 + 3x_2 + 3x_2x_1 - 4x_3x_1 - 2x_3x_2x_0 + 5x3x_2x_1$$

Figure 3 illustrates all the steps taken to compute the maximum mismatch. First remove the constant and get a new AT:

$$AT(f)' = x_0 - 3x_1x_0 + 3x_2 + 3x_2x_1 - 4x_3x_1 - 2x_3x_2x_0 + 5x3x_2x_1$$
$$S_0 = -4,\ S_1 = 1,\ S_2 = 9,\ S_3 = -1$$

so the *mpv* sequence is (x_2, x_1, x_3, x_0).
The reversed AT is:

$$AT(f)'' = -x_0 + 3x_1x_0 - 3x_2 - 3x_2x_1 + 4x_3x_1 + 2x_3x_2x_0 - 5x3x_2x_1$$

The reversed mpv sequence is (x_0, x_3, x_1, x_2).

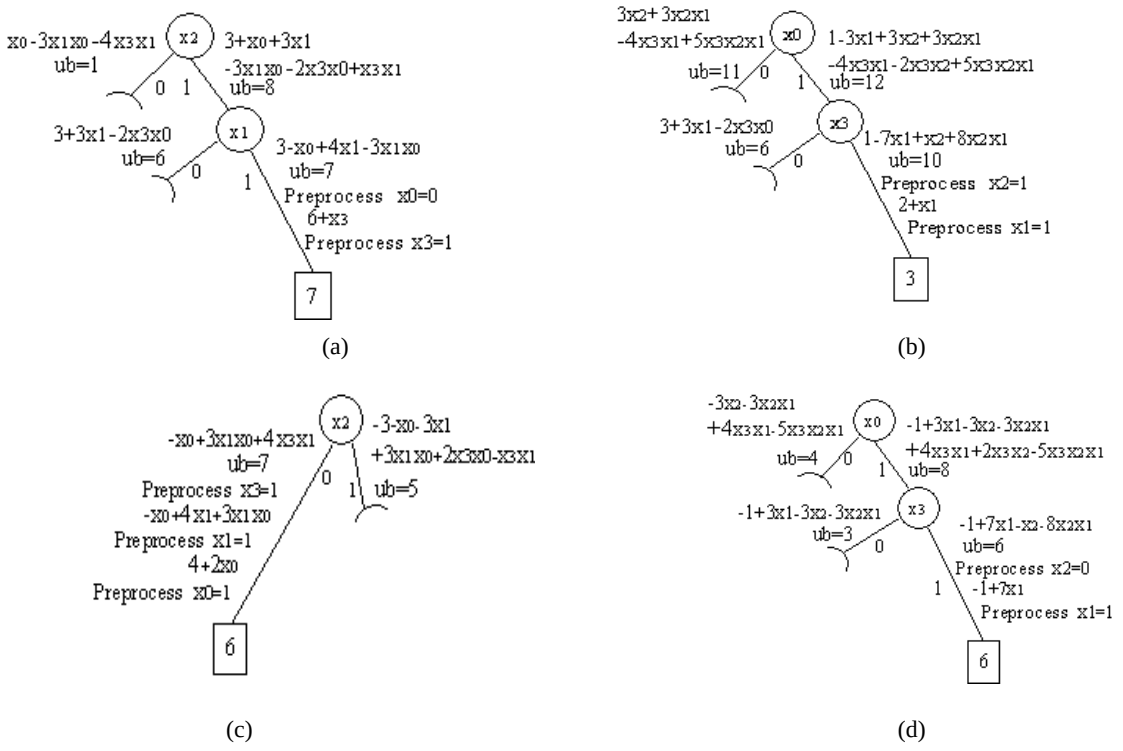

Fig. 3: Steps of example 1 to perform the imprecision algorithm

First $AT(f)^{'}$ is searched by the order of the mpv sequence, due to the ubcoef value, x_2 and x_1 are set to 1, here the decomposed polynomial is 3- x_0 + 4x_1- 3x_1x_0, then the algorithm finds coefficients of all terms with variable x_0 present are negative, so x_0 is preprocessed to 0; and it continues to preprocess x_3 = 1, finally a constant value_0 = 7 is obtained; the procedure is displayed by a) in Fig. 3. Using the negated mpv sequence upon $AT(f)^{'}$, the obtained constant is value_1 = 3, showed by b), so the maximum value of the AT polynomial without the constant "-2" is max_value = max (value_0, value_1) = 7.

Decompose *AT(f)''* by the *mpv* and the negated *mpv* sequences respectively, showed by c) and d), value_2 = value_3 = 6, so the minimum value of the AT polynomial without the constant "-2" is:

min_value =max (value_0, value_1)* −1 = −6.

Eventually the maximum mismatch is computed as:

$$\max(|\text{max\_value}-2|,|\text{min\_value}-2|)= \max(|7-2|,|-6-2|) = 8$$

Compared to the searching algorithm in Radecka and Zilic (2006), the predominance of the algorithm improvement stands to reason. It recursively seeks for variables which can be preprocessed in a decomposition procedure; if successful, complexity is minified much since the computation avoids decomposing the variable and directly sets its value, then the residual polynomial is simplified. For example, only one node, x_2, is searched to determine its value in c), and other three variables are preprocessed, therefore time and space requirements are diminished.

EXPERIMENTAL RESULTS

We demonstrate efficiency of proposed algorithms in this section. Box-Muller circuit and trigonometric circuits are used as benchmarks to be verified.

Results of the conversion algorithm: Table 1 shows results of the algorithm described by Fig. 1. From the table, the conversion algorithm can be performed effortlessly even if Taylor degree and input variables are large, unlike Radecka and Zilic (2006). It has been always the fastest algorithm during experiments.

Imprecise sine circuit implementation: A sinus/cosinus circuit is a widely used block that can be realized in many ways. We consider a Taylor Series specification and pipelined implementations as in Radecka and Zilic (2006). The function sin(x) can be expanded around $x_{1=0}$ as:

Table 1: Performance of Taylor series conversion

Function	Taylor degree	Bits	AT terms	Expanded terms	Run time (s)	Memory (MB)
sin(x)	7	31	3572223	10625591	586.593	156
sin(x)	9	26	5658536	55962920	179.171	247
sin(x)	11	24	7036529	316283264	921.218	293
sin(x)	13	20	988115	409609664	1167.58	59
exp(x)	10	24	4540386	131128139	371.266	239
exp(x)	12	22	3096514	548354039	1633.36	182
exp(x)	14	18	261156	471435599	1497.81	3
exp(x)	14	20	1026876	1391975639	4222.25	59
exp(x)*sin(x)	10	24	4540385	123221864	314.703	254
exp(x)*sin(x)	13	20	988115	429816984	1445.19	88
exp(x)*sin(x)7	15	16	65534	282662144	985.703	18

Table 2: Imprecise sine circuit matching

Spec Taylor degree	Imp Taylor degree	Spec bit	Imp bit	Error at terms	Error	Time(s)
9	7	12	12	3983	2.7-e6	0.36
9	7	16	16	50486	2.8-e8	2.59
9	11	12	12	3991	2.5-e8	1.6
9	11	16	16	62396	2.5-e8	17.6
9	9	20	22	2443163	2.5-e8	305.8
11	11	16	18	230963	2.4-e5	69.3
11	11	20	18	784625	2.1-e6	186.3
13	13	16	14	65398	5e-7	125.6
11	9	16	14	63018	5e-5	19.03
11	9	18	16	230963	1.2e-5	54.45
13	11	18	16	258095	1.2e-5	338.4

Table 3: Performance of box-muller algorithm for different Wordlengths and Taylor term approximations (*k*,*k*)

Spec Taylor degree	Imp Taylor degree	Input bits	Error at terms	Error	Time [s]
(14,14)	(16,16)	(8,8)	65536	0.349	19.3
(22,22)	(20,20)	(8,8)	65536	0.224	112
(26,26)	(24,24)	(8,8)	65536	0.165	437
(16,16)	(14,14)	(9,9)	262144	0.373	69.4
(24,24)	(22,22)	(9,9)	262144	0.209	847
(24,24)	(26,26)	(9,9)	262144	0.182	1674

$$\sin(x)\sum_{i=0}^{\infty}(-1)^i\frac{x^{2i+1}}{(2i+1)!}$$

$$y_2(x_2)=\sum_{i=0}^{\infty}(-1)^i\frac{(2\pi x_2)^{2i}}{(2i)!}$$

Imprecise box-muller algorithm implementation: Box-Muller algorithm Knuth (1998) which generates Gaussian random variable is critical to a number of applications such as accuratebit error rate testers Lee *et al*. (2004). The algorithm uses the following expression:

$$y(x_1,x_2)=y_1(x_1)*y_2(x_2)=\sqrt{-21n(x_1)}*\cos(2\pi x_2)$$

Any implementation is an approximation, therefore it needs to evaluate the error. We represent it by a finite number of Taylor series terms:

$$y_1(x_1)=\sqrt{-21n(x_1)}$$

around the point x_1=0.5 and $y_2=\cos(2\pi x_2)$ around x_2 = 0.

$$y_1(X_1)=1.17741-1.6984(x_1-0.5)+0.4733(x_1-0.5)^2-1.582(x_1-0.5)^3$$

$$+1.0198(x_1-0.5)^4-3.3284(x_1-0.5)^5+2.7848(x_1-0.5)^6$$

Table 3 represents the time in seconds needed to compute errors between specifications and implementations, and gives corresponding imprecision.

Both functions, $y_1(x_1)$*and* $y_2(x_2)$ are approximated with the same number of Taylor terms, shown in Table 2. Times are reported for C++ codes running on a 2.4GHz Intel Celeron processor under Linux.

CONCLUSION

This study proposes a method to convert Taylor Series into an AT expression. An algorithm that manages well the intermediate data swell is presented. From Taylor Series specification and implemented AT polynomials, an error AT polynomial can be obtained. We determine the precision error by an efficient search using the branch-and-bound algorithm and verify the correctness within prescribed error bounds.

Finally, trigonometric and Box-Muller circuits are verified by proposed precision calculation algorithms. As AT expressions with several billion terms are routinely

handled, we conclude conversion and searching algorithms are efficient with respect to memory and running time.

ACKNOWLEDGMENT

This research was supported by the Ministry of Industry and Information Technology of the People's Republic of China under the grant of special projects for internet of things, and by the National Natural Science Foundation of China under the grant No. 61102075.

REFERENCES

Bryant, R.P. and Y.A. Chen, 1995. Verification of Arithmetic circuits with Binary Moment Diagrams Proceeding of 32nd Design Automation Conference, pp: 535-541.

Ciesielski, M., P. Kalla, Z. Zeng and B. Rouzeyre, 2002. Taylor Expansion Diagrams: A Compact, Canonical Representation with Applications to Symbolic Verification, Proceeding Design Automation Test in Europe, DATE, pp: 285-289.

Hamaguchi, K., A. Morita and S. Yajima, 1995. Efficient Construction of Binary Moment Diagrams for Verification of Arithmetic Circuits. Proc. ICCAD, pp: 78-82.

Heidtmann, K.D., 1991. Arithmetic spectrum applied to fault detection for combinational networks. IEEE Trans. Comp., 40(3): 320-324.

Knuth, D., 1998. The Art of Computer Programming. Vol. 2, Addison-Wesley, Reading, MA.

Lee, D.U., W. Luk, J.D. Villasenor and P.Y.K. Cheung, 2004. A gaussian noise generator for hardware-based simulations. IEEE Trans. Comp., 53(12): 1523-1534.

Lui, P.K. and J.C. Muzio, 1986. Spectral signature testing of multiple stuck-at faults in irredundant combinational networks. IEEE Trans. Comp., C-35: 1088-1092.

Radecka, K. and Z. Zilic, 2001. Arithmetic Transforms for Verifying Compositions of Sequential Datapaths. Proceeding IEEE international Symposium on Computer Design, pp: 348-358.

Radecka, K. and Z. Zilic, 2002. Specifying and Verifying Imprecise Circuits by Arithmetic Transforms. Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp: 128-131.

Radecka, K. and Z. Zilic, 2006. Arithmetic transforms for compositions of sequential and imprecise datapaths computer-aided design of integrated circuits and systems. IEEE Trans., 25(7): 1382-1391.

Zhou, Z. and W. Burleson, 1995. Equivalence Checking of Data paths based on Canonical Arithmetic Expressions. Proceedings of 32nd Design Automation Conference, San Francisco. pp: 546-551.