the resulting global state–space model. Pertinent examples include the construction of a three-band parallel cascade of Butterworth filters, and the realization of a global system model for a pure-delay transfer function given bandlimited interpolation data in three different frequency bands. Passivity-enforcement techniques can be performed in a postprocessing step.

References

[1] T. Dhaene, J. Ureel, N. Fache, and D. De Zutter, "Adaptive frequency sampling algorithm for fast and accurate S-parameter modeling of general planar structures," in *IEEE MTT-S Int. Microwave Symp. Dig.*, Orlando, FL, May 1995, vol. 3, pp. 1427–1430.

[2] R. Lehmensiek and P. Meyer, "An efficient adaptive frequency sampling algorithm for model-based parameter estimation as applied to aggressive space mapping," *Microw. Opt. Technol. Lett.*, vol. 24, no. 1, pp. 71–78, Jan. 2000.

[3] R. Pintelon and J. Schoukens, *System Identification, A Frequency Domain Approach*. New York: IEEE Press, 2001.

[4] E. Chiprout and M. S. Nakhla, "Analysis of interconnect networks using complex frequency hopping," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 2, pp. 186–200, Feb. 1995.

[5] S.-H. Min and M. Swaminathan, "Construction of broadband passive macromodels from frequency data for simulation of distributed interconnect networks," *IEEE Trans. Electromagn. Compat.*, vol. 46, no. 4, pp. 544–558, Nov. 2004.

[6] J. Choi, S.-H. Min, J.-H. Kim, M. Swaminathan, W. Beyene, and X. Yuan, "Modeling and analysis of power distribution networks for Gigabit applications," *IEEE Trans. Mobile Comput.*, vol. 2, no. 4, pp. 299–313, Oct.–Dec. 2003.

[7] H. Akçay and B. Ninness, "Orthonormal basis functions for modelling continuous-time systems," *Signal Process.*, vol. 77, no. 3, pp. 261–274, Sep. 1999.

[8] B. Wahlberg, "System identification using Kautz models," *IEEE Trans. Autom. Control*, vol. 39, no. 6, pp. 1276–1281, Jun. 1994.

[9] L. Knockaert, "On orthonormal Müntz-Laguerre filters," *IEEE Trans. Signal Process.*, vol. 49, no. 4, pp. 790–793, Apr. 2001.

[10] B. Gustavsen and A. Semlyen, "Enforcing passivity for admittance matrices approximated by rational functions," *IEEE Trans. Power Syst.*, vol. 16, no. 1, pp. 97–104, Feb. 2001.

[11] D. Saraswat, R. Achar, and M. Nakhla, "A fast algorithm and paractical considerations for passive macromodeling of measured/simulated data," in *Proc. Topical Meeting Electrical Performance Electronic Packaging (EPEP)*, Monterey, CA, 2002, pp. 297–300.

[12] C. P. Coelho, J. R. Phillips, and L. M. Silveira, "A convex programming approach for generating guaranteed passive approximations to tabulated frequency data," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 2, pp. 293–301, Feb. 2004.

[13] S. Grivet-Talocia, "Passivity enforcement via perturbation of Hamiltonian matrices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 9, pp. 1755–1769, Sep. 2004.

[14] P. Feldmann and R. W. Freund, "Efficient linear circuit analysis by Pade approximation via the Lanczos process," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 5, pp. 639–649, May 1995.

[15] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodeling algorithm," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 8, pp. 645–654, Aug. 1998.

[16] L. Knockaert and D. De Zutter, "Laguerre-SVD reduced order modeling," *IEEE Trans. Microw. Theory Tech.*, vol. 48, no. 9, pp. 1469–1475, Sep. 2000.

[17] J. R. Phillips, L. Daniel, and L. M. Silveira, "Guaranteed passive balancing transformations for model order reduction," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 8, pp. 1027–1041, Aug. 2003.

[18] L. Daniel, O. C. Siong, L. S. Chay, K. H. Lee, and J. White, "A multiparameter moment-matching model-reduction approach for generating geometrically parameterized interconnect performance models," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 5, pp. 678–693, May 2004.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: The Johns Hopkins Univ. Press, 1996.

[20] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran, The Art of Scientific Computing*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1992.

Arithmetic Transforms for Compositions of Sequential and Imprecise Datapaths

Katarzyna Radecka and Zeljko Zilic

***Abstract*—This paper addresses the issue of obtaining compact canonical representations of datapath circuits with sequential elements for the purpose of equivalence checking and component matching. First, the authors demonstrate the mechanisms for an efficient compositional construction of the arithmetic transform (AT), which is the underlying function representation used in modern word-level decision diagrams (WLDDs). Second, presented is a way of generating the canonical transforms of the sequential and imprecise datapath circuits.**

***Index Terms*—Arithmetic transform (AT), hardware verification, imprecise arithmetic, sequential circuits.**

I. Introduction

Arithmetic datapath circuits have always been important in the development of circuit representations suitable for formal verification. Early bit-level methods, including the binary decision diagram (BDD), proved to be insufficient in describing common arithmetic circuits like multipliers. The emergence of word-level decision diagrams (WLDDs) overcame this problem to some extent. Indeed, the interest in the verification of arithmetic circuits has been renewed [1]–[3]. Therefore, there is a shift in interest from bit-level to word-level forms. However, some caution is in place, as not all word-level representations are beneficial for arithmetic circuits. Good candidates are the descriptions based on the arithmetic transform (AT) [4], such as binary moment diagrams (BMDs) [5], [6] and their extensions [7] that are compact for common arithmetic circuits.

This paper aims at enhancing the capability of basic AT forms. To reduce the complexity and to capitalize on the block-level structure, we consider the means to compose AT descriptions from those of the smaller blocks. However, unlike bit-level representations, the existing AT-based word-level forms cannot be composed because of the datatype incompatibility—the inputs are binary, while the outputs are word-level quantities. To address this problem, we propose AT extensions that facilitate the combination of ATs describing the combinational, sequential, and imprecise arithmetic blocks. We notice that the recently introduced Taylor expansion diagrams (TEDs) [8] also have basic composition properties, making the addition and multiplication of two TEDs feasible although apparently not simple [9].

Sequential elements in datapaths present another difficulty for word-level forms, which are only suitable for combinational functions. In general, the notion of a sequential equivalence [10] becomes fairly complex when allowing for manipulations such as retiming, pipelining, and FSM optimization. In this paper, we provide an extension to the AT for sequential datapaths. Using these forms, we show, for instance, how to match a specification with the sequential implementation in circuits employing distributed arithmetic (DA).

The final issue explored in this paper is the representation of imprecise arithmetic circuits. The major challenge here lays in the assertion of circuit correctness under the presence of an error due to the arithmetic imprecision. Imprecision naturally comes from the

Manuscript received October 6, 2004; revised February 7, 2005 and May 9, 2005. This paper was recommended by Associate Editor R. F. Damiano.

K. Radecka is with Concordia University, Montreal, QC H3G 1M8, Canada.

Z. Zilic is with McGill University, Montreal, QC H3A 2A7, Canada (e-mail: zeljko@ece.mcgill.ca).

Digital Object Identifier 10.1109/TCAD.2005.855935

finite-word representation of real numbers, as well as from various approximations. Precision analysis is critical to using the fixed-point number representation, which is attractive in balancing complexity, cost, and energy consumption [11]. Furthermore, the component matching [12] of arithmetic circuits, which is increasingly important in systems-on-chip (SoC), also requires efficient dealing with imprecision. Issues regarding the verification of such datapaths are dealt with in [13]–[17].

Although data representations that tackle each of the above issues exist, none are capable of addressing them simultaneously. The AT extensions that facilitate the composition of sequential blocks are presented in Section II-A. Section III deals with the imprecision of implementation. In Section IV, we apply the proposed algorithms to several datapath circuits.

II. Arithmetic Transform Extension for Composition

The AT is a canonical polynomial representation of multi-input/multi-output Boolean functions $f : B^n \to B^m$. To describe such functions with a single polynomial, function outputs are "grouped together" into word-level (W) quantities (e.g., integers) resulting in a pseudo-Boolean function $f : B^n \to W$. Therefore, the representation should have Boolean inputs and word-level outputs.

Definition 1: The AT of a pseudo-Boolean function $f : B^n \to W$ is a polynomial with an arithmetic "+" operation; word-level coefficients $c_{i_1 i_2 \cdots i_n}$; binary inputs $x_1, x_2, \ldots, x_n$; and binary exponents $i_1, \ldots, i_n$

$$\mathrm{AT}(f) = \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} \cdots \sum_{i_n=0}^{1} c_{i_1 i_2 \cdots i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \tag{1}$$

that uniquely and exactly interpolates "f."

The AT can be obtained in several ways, ranging from a transform matrix multiplication, to the fast transforms, decision diagram transformations, and polynomial interpolation [18]. The 2^n-word vector $C = \{c_{i_1 i_2 \cdots i_n}\}$ of coefficients from (1) is obtained by multiplying the 2^n-word vector f of word-level function outputs with the $2^n \times 2^n$ matrix T_n, $C = T_n * f$. T_n is defined recursively as

$$T_n = \begin{bmatrix} T_{n-1} & 0 \\ -T_{n-1} & T_{n-1} \end{bmatrix}, \quad T_0 = 1. \tag{2}$$

The AT generates outputs in the word-level form. A word-level encoding is explicitly expressed by the number-norm function $|\,| : B^m \to W$, defining a Boolean vector interpretation in the word-level domain. Table I summarizes common integer and fractional number norms for a vector of Boolean values x_i.

Definition 2: Binary encoding $(x_1, x_2, \ldots, x_n)$ of a word w is the inverse of the norm function $|w|^{-1} = (x_1, x_2, \ldots, x_n)$.

Lemma 1: Consider a pseudo-Boolean function $f : B^n \to W$ with a norm $|\,| : B^m \to W$. Then

$$\mathrm{AT}(f) = |f|.$$

Proof: By applying the transform to quantities in W, an interpolation polynomial is obtained, such that for all inputs $\mathrm{AT}(f(x_1, x_2, \ldots, x_n)) = |f(x_1, x_2, \ldots, x_n)|$. ■

Defining $\mathrm{AT}(f)$ in terms of a norm function $|f|$ helps to illustrate problems with compositions of ATs. Consider the circuit consisting of two blocks $B1$ and $B2$ in Fig. 1. The composition of the two ATs [$R = \mathrm{AT}(B1)$ and $S = \mathrm{AT}(B2)$] would require the binary encoding, i.e., the conversion of the word-level output R of the first AT into the bit-level values T acceptable as inputs to the second AT (Fig. 1).

TABLE I
Norm Functions for Common Word Encoding

Word	Number Norm $\lvert x \rvert$		
	Unsigned	Sign Extended	2's Complement
Integer	$\sum_{i=0}^{n-1} x_i 2^i$	$(1-2x_{n-1})\sum_{i=0}^{n-2} x_i 2^i$	$\sum_{i=0}^{n-2} x_i 2^i - x_{n-1} 2^{n-1}$
Fractio-nal	$\sum_{i=1}^{n-1} x_i 2^{-i}$	$(1-2x_0)\sum_{i=1}^{n-1} x_i 2^{-i}$	$-x_0 + \sum_{i=1}^{n-1} x_i 2^{-i}$
Fixed Point	$\sum_{i=0}^{n-1} x_i 2^{i-m}$	$(1-2x_0)\sum_{i=1}^{n-1} x_i 2^{i-n}$	$\sum_{i=1}^{n-1} x_i 2^{i-m} - x_0 2^{m-n}$

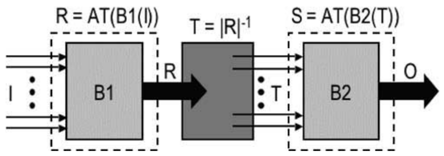


Fig. 1. Composition of ATs—binary encoding use.

Note that there is no closed-form expression for binary encoding; instead, an integer-to-binary conversion algorithm must be applied to the AT polynomial to obtain $|w|^{-1}$. Since there is no simple $\mathrm{AT}(|w|^{-1})$, we need an AT extension that accepts word-level inputs.

A. Mixed AT (MAT)—First AT Extension

Since inputs to the AT are bit-level quantities, while outputs are words; we cannot use outputs of an AT as inputs to the other transform. Therefore, the AT cannot be applied to compose an overall transform of a circuit from the ATs of its basic blocks. Similarly, the AT at its original form is not equipped for representing sequential elements.

In this section, we propose two extensions to basic ATs—allowing the description of sequential components and their efficient integration into the overall circuit representation.

Our first extension facilitates the compositional approach to representing datapaths. We permit inputs to be a mix of binary (x_i) and word-level (w_j) quantities, i.e., $f(x_1 \cdots x_n, w_1 \cdots w_k)$.

Definition 3: The MAT polynomial interpolating the function $f : B^n \times W^k \to W$ uses the arithmetic "+" operation, word-level coefficients $c_{i_1 i_2 \cdots i_n}$, binary $x_1, x_2, \ldots, x_n$, and word-level $w_1, w_2, \ldots, w_k$ inputs; as well as the binary exponents $i_1, \ldots, i_n$ and $e_1, \ldots, e_k$

$$\mathrm{MAT}(f) = \sum_{i_1=0}^{1} \cdots \sum_{i_n=0}^{1} \sum_{e_1=0}^{1} \cdots \sum_{e_k=0}^{1} c_{i_1 \cdots i_n e_1 \cdots e_k} x_1^{i_1} \cdots x_n^{i_n} w_1^{e_1} \cdots w_k^{e_k}.$$

We apply the MAT when some sets of inputs to a combinational function are known to be word-level quantities, such as outputs coming from previous blocks. More than one MAT can be derived from a single AT, depending on which variables are considered at word level. For simplicity, we omit the explicit word-level designation in MAT.

Lemma 2: The coefficients of an MAT can be calculated using the transformation given by (2), expanded around binary input variables, with word-level input quantities unassigned, i.e., treated as symbols

$$C(w_1, w_2, \ldots, w_k) = T_n * f. \tag{3}$$

Proof: The application of the above transform to the quantities in B^n results in an interpolation polynomial: $\mathrm{MAT}(f(x_1, x_2, \ldots, x_n, w_1, w_2, \ldots, w_k)) = |f(w_1, w_2, \ldots, w_k)|$. Allocating concrete values to unassigned word-level variables preserves the norm, and this transform then equals $|f|$, which implies that the resulting MAT exactly interpolates f. ■

Example 1—MAT Through Matrix Multiplication: Consider the MAT of a function $f = 2a + 3b$, where a and b are 2-bit unsigned integers. We treat $a = a_1a_0$ as a bit vector, and b as a single word-level quantity. Then, for all assignments of binary variables, we obtain the truth table

$$f = [3b \quad 2+3b \quad 4+3b \quad 6+3b]^T$$

from which, the AT transform application yields

$$\mathrm{MAT}(f) = T_2 * f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 3b \\ 2+3b \\ 4+3b \\ 6+3b \end{bmatrix} = \begin{bmatrix} 3b \\ 2 \\ 4 \\ 0 \end{bmatrix}.$$

The resulting polynomial is $G(a_1, a_0) = 3b + 2a_0 + 3a_1 + a_1a_0$.

By treating input b as word-level values, we shrink the size of the matrix T_n from 16 × 16 to 4 × 4. Therefore, as the above example shows, an MAT allows a compact way of generating the AT.

To obtain the final $\mathrm{AT}(f)$ from $\mathrm{MAT}(f)$, it suffices to syntactically replace the word-level variables in $\mathrm{MAT}(f)$ with their binary encoding at the end of the process. In the previous example, substituting the word-level variable b with its binary encoding $b = b_0 + 2b_1$ results in $\mathrm{AT}(f) = 3b_0 + 6b_1 + 2a_0 + 3a_1 + a_1a_0$.

Once the MAT of a circuit is constructed, we can provide bounds on the size of the AT obtained from it.

Lemma 3: Consider the MAT of f that has t terms and one word-level input w consisting of n bits. The size of the AT obtained from this MAT would be bounded by nt terms.

Proof: To obtain $\mathrm{AT}(f)$ from $\mathrm{MAT}(f)$, all occurrences of the word-level input variable w must be replaced by its AT representation. Since the size of w is n bits, at most n terms are used to substitute w in each $\mathrm{MAT}(f)$ monomial. Hence, the total number of terms in the resulting $\mathrm{AT}(f)$ will be nt. ■

Corollary 1: If the MAT of a given function f has t terms and k word-level inputs, then the replacement of all such word-level inputs results in an AT with at most $n^k t$ terms.

Lemma 3 and Corollary 1 prove that an MAT of the composition of blocks can always be converted to the AT with polynomial size increase in word length n.

B. MAT Features and Scope Limitations

The primary application of the MAT is in compositions of representations, where word-level outputs of a block are being passed to another block, Definition 3. There are some limitations of an MAT regarding the inputs it can deal with and the classes of functions it can represent. Sometimes, selected bits of a word output have to be split off before inputting them to an MAT (for example, buses $w1$ and $w2$ of the word w in circuit C, Fig. 2). Since obtaining individual bits from a word w (i.e., $|w|^{-1}$) is computationally expensive, such operations are not easily performed through an MAT.

Instead of splitting the words, an alternative approach is used. For each part $w1$ and $w2$ of the bus w, a separate function transform is created. This amounts to splitting the block B into $B1'$ and $B1''$ in circuit D, Fig. 2. The individual word outputs can then be readily used; and if the whole bus is needed, the bus w is created by concatenating a word $w1$ (least significant part of w) with a word $w2$. The result is $\mathrm{MAT}(w) = 2^{\mathrm{length}(w1)}|w2| + |w1| = 2^{\mathrm{length}(w1)}\mathrm{MAT}(B1') + \mathrm{MAT}(B1'')$.

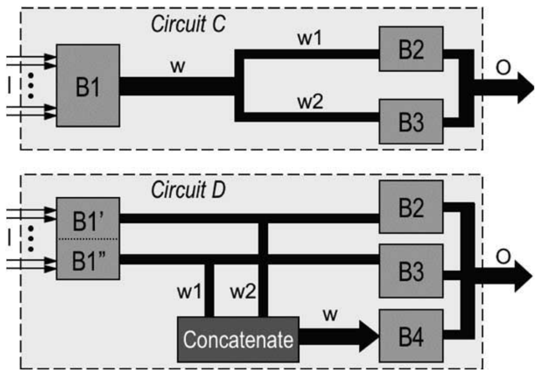


Fig. 2. Composition of specifications using MAT.

An MAT is primarily a tool for composing ATs by means of its word-level input variables, rather than for representing all functions. A function must be explicitly expressed in terms of designated word-level inputs to possibly be represented by an MAT. More precisely, such a function must be disjointly decomposable [19] along designated word-level inputs.

Lemma 4: The MAT of f exists if there is a disjoint decomposition of f with respect to the word-level input variables.

Proof: By Definition 3, the application of the transform to the quantities in B^n results in an interpolation polynomial $\mathrm{MAT}(f(x_1, x_2, \ldots, x_m, w_1, w_2, \ldots, w_k)) = |f(w_1, w_2, \ldots, w_k)|$. Since it is expressed by designated word-level inputs alone, it is disjointly decomposable with respect to such variables.

The decomposable function can be expressed as

$$f = g\left(x_1, x_2, \ldots, x_m, h(w_1, w_2, \ldots, w_k)\right).$$

Then, for each assignment of Boolean variables, the output is a constant or a function of word-level variables. By applying the AT construction symbolically, we obtain an MAT. ■

Since an MAT is used for compositions of AT representations of individual blocks, the MAT for a given block is assumed to exist, and there is no need to seek word-level decompositions.

C. Sequential at Extensions—ATS and MATS

Sequential elements cannot be represented by either an AT or by an MAT, as there is no notion of time provided by these transforms. In contrast to the AT and MAT, which are time invariant, we introduce new AT extensions for sequential implementations that allow variables to change over time. We refer to such variables as timed variables.

Definition 4: The timed variable $v[n]$ is a variable v to which a time tag $[n]$ is assigned to indicate that the function generating the value of v changes with a time instance n.

We use timed variables to abstract away the clock in the sequential implementation of datapaths. A timed function $f[n]$ represents the value of f in the nth clock period. We assume the finite clock duration, i.e., the function $f[n]$ is executed in a finite number of clock cycles. In terms of circuit implementations, we restrict our attention to synchronous sequential systems.

The simplest role that the timed transform plays is in representing the state information kept in memory.

Example 2: Consider a memory element such as a flip-flop, whose contents is reloaded every clock cycle. Its defining timed equation is $m_{\mathrm{out}}[n] = m_{\mathrm{in}}[n-1]$.

The other application of the timed function is for the modeling of multicycle latency incurred in executing a function. In this case, for verification purposes, it might be needed to validate conditions such as achieving the exact latency or to guarantee a fixed upper bound on latency.

Finally, when the MATS function description combines occurrences of various intermediate variables at different instances of time, the elimination of internal timed variables is required to express the function solely in terms of input values, which are possibly timed.

Definition 5: The AT sequential (ATS) is the $\mathrm{AT}(f)[n]$ of a timed function f at a time instance n while the MAT sequential (MATS) is analogously the $\mathrm{MAT}(f)[n]$ of a timed function with word- and bit-level inputs.

Lemma 5: When a sequential element is fed entirely by bit-level combinational (primary) inputs, then the transform describing it is the ATS.

Proof: Combinational blocks with bit-level inputs only are given by the AT. However, by Definition 5, if such blocks realize sequential functions; then they should be described by the ATS to indicate timing dependencies among inputs. ■

Example 3—ATS of Flip-Flop: We use the ATS to represent a standard flip-flop with input D, reset signal reset, and an enable signal—all bit type

$$\mathrm{ATS}(ff)[n] = (1 - \mathrm{reset})(\mathrm{En} * D + (1 - \mathrm{En}) * ff[n-1]).$$

Note that for notational convenience, the syntax of the ATS is simplified in the preceding example. First, a time tag $[n]$ is omitted on the right-hand side whenever it is the same as on the left-hand side. Second, the explicit reference to the transform of the same block is dropped from the right-hand side.

In fact, when intermediate variables generated by sequential elements are word-level quantities, then the only appropriate sequential transform is an MATS. In addition to the MATS being required to describe sequential elements, combinational blocks represented by an MAT may need to use an MATS, following the composition with sequential blocks.

Lemma 6: The MATS of a composition of a combinational element described by function f with sequential blocks can be obtained from $\mathrm{MAT}(f)$ by replacing each input that is generated by a sequential element with its defining MATS/ATS.

Proof: An MAT describes the combinational function f of a block. Consider the inputs to f to be fed by sequential elements. A syntactical replacement of these inputs with timed variables defining corresponding sequential blocks produces MATS of the overall function composition. ■

Corollary 2: The MATS of a sequential function f can be obtained from the MAT of the combinational part of f by replacing each MAT input that is generated by a memory element with its defining MATS.

Corollary 3: If at least one input variable of a combinational function f is generated by a sequential block, then the transform of the composition (instead of the MAT) needs to be presented as the MATS

$$\begin{aligned}\mathrm{MATS}\left((f(x_1, \ldots, w_i, \ldots, w_k)\right)&\\ &= \mathrm{MAT}\left(f\left(x_1, \ldots, \mathrm{MATS}(w_i), \ldots, w_k\right)\right).\end{aligned}$$

All inputs to the MATS must be assigned a time tag, regardless of whether they are from a sequential or a combinational block.

From the preceding corollary, we conclude that the overall transform of any composition of a sequential block $B1[\mathrm{MATS}(B1)]$, with a combinational element $B2$, requires the use of an MATS rather than an MAT.

1) Converting MATS to MAT or ATS: $\mathrm{MATS}(f)$ can take two forms. A type-I MATS presents a case where the timed output variable f is expressed only in terms of timed input values. A type-II MATS describes a recurrence equation, where a symbol of a considered function f appears on both sides of a definition. The left-hand side of the MATS has only the symbol $f[n]$, while the expression on the right-hand side is the function of various $f[k]$s, where $k < n$, as well as other timed inputs. A solution to such a type-II MATS eliminates the instances of f at the right-hand side and expresses the outputs only in terms of the inputs, possibly delayed.

We want to remove timed variables whenever possible, as transforms are then purely combinational (MAT). When the overall circuit specification is given in terms of the AT (ATS), then the implementation must be represented in the same way. To achieve that, all MATS forms must be translated either as an MAT or an ATS. In the case of an MAT, such expressions are then converted into AT through symbolic replacements of all word-level inputs. Therefore, the conversion between an MATS and an MAT or the ATS (when appropriate) is a crucial step.

Lemma 7: If a specification of a function f implemented by a given block is expressed in terms of the AT, i.e., f is a combinational function; then the MATS description of an implementation of function f must be transferable into the MAT (and eventually to the AT).

Proof: The AT and its derivatives are canonical. As a combinational function f is represented in terms of the AT, its implementation must also be given in the form of a combinational transform (MAT or AT). Therefore, the translation from its initial $\mathrm{MATS}(f)[n]$ to $\mathrm{MAT}(f)$ exists and acts by removing the timing variable n from an MATS. ■

In consequence, time tags cannot be completely eliminated when the function is sequential. In that case, comparison with a sequential specification is made on the ATS rather than the AT.

Corollary 4: Given a combinational specification and assuming that the implementation and its MATS description is correct, then there exists a procedure to eliminate all timing information in an MATS and transforms it to the AT.

A type-II MATS can be converted into an MAT either by solving an MATS as a recurrence equation or by the procedure that we refer to as a "symbolic loop unrolling." When considering the loop unrolling, each value of an output function present in the right-hand side is symbolically expressed in terms of its output variables at earlier time instances. The process continues until the terminal case is reached and the output variables have been eliminated from the right-hand side. Recurrence equation solvers are generally faster than the loop unrolling procedures; however, they require known closed-form solutions and can be relatively costly by themselves.

Example 4—MATS of Circuits With Accumulative Loops: In Fig. 3(a), block $A1$ represents an N-bit adder. In the nth step, one summand is taken from the primary inputs, while the other is supplied from the register storing the values of the previous $n-1$ additions. The register has been initially reset.

The MATS of this loop is obtained by considering the register input $f[n]$, with the value given by the recurrence

$$\mathrm{MATS}(f)[n] = a[n] + \mathrm{MATS}(f)[n-1], \quad \mathrm{MATS}(f)[0] = 0.$$

Its solution is $\mathrm{MATS}(f)[n] = \sum_{i=1}^{n} a[i]$.

Next, consider Fig. 3(b), where block $B1$ represents an $N \times N$-bit multiplier, and block $B2$ is a $(2N+1)$-bit adder creating a multiply-and-accumulate loop. The MATS results from using the previously derived MAT transforms of its individual blocks. The inputs to the multiply-and-accumulate loop at the time instance i are the N-bit

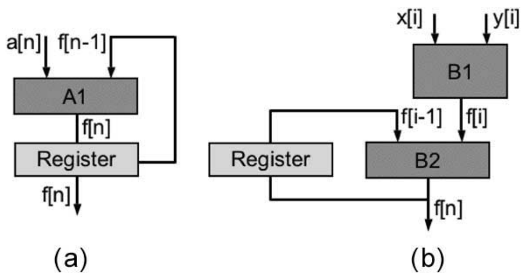


Fig. 3. Add- and multiply-accumulate loops.

binary vectors $x[i]$ and $y[i]$, the output $f[i]$ is a binary vector of size $(2N+1)$. The ATS (all inputs are bits) of the multiplier $B1$ is defined for inputs at a time instance i

$$\text{ATS}(B1 = x * y)[i] = \sum_{k=0}^{N} x_k[i]2^k * \sum_{k=0}^{N} y_k[i]2^k = a[i].$$

The transform of this loop equals the recurrence solution

$$\text{MATS}(f)[n] = \sum_{i=1}^{n} a[i] = \sum_{i=1}^{n} \left(\sum_{k=0}^{N} x_k[i]2^k * \sum_{k=0}^{N} y_k[i]2^k \right).$$

D. Composition of Transforms

Once each block is expressed in terms of its appropriate transform (AT, ATS, MAT, or MATS), the description of the overall circuit is generated through the composition of individual block transforms. If the specification is combinational, then the final circuit representation must be the AT; otherwise, it will require the ATS form.

The effort to obtain a final transform of the implementation expressed by the AT (ATS) is justified since the specification is a function of bit inputs only and is therefore given in terms of the AT (ATS). If this were not the case, the description of the implementation using an MAT or an MATS would be also suitable for further equivalence checking.

Constructing the overall AT (ATS) of a given complex design begins from blocks fed entirely by primary inputs and continues in the forward direction until all primary outputs are reached, as shown in Algorithm 1. According to Definitions 1 and 5, all elements with bit-level only inputs are expressed in terms of their AT (combinational) or ATS (sequential). These operations correspond to steps 4 and 5 in Algorithm 1.

```
Algorithm 1: Composition of Block Transforms
Generate Transform of the network of blocks
1.  **for each** block $\mathrm{B_i}$ in topological order
2.  {
3.  Assign: inputs $(\mathrm{B_i})$ to output(predecessor$(\mathrm{B_i})$)
4.     **if** (combinational$(\mathrm{B_i})$ && all_inputs_primary)
5.        $f_\mathrm{i} = \mathrm{AT}(\mathrm{B_i})$;
6.     **if** (combinational$(\mathrm{B_i})$ && no_seq_input)
7.        $f_\mathrm{i} = \mathrm{MAT}(\mathrm{B_i}$, assigned_input_list);
8.     **else** /*sequential$(\mathrm{B_i})$*/
9.        $f_\mathrm{i} = \mathrm{MATS}(\mathrm{B_i}$, assigned_input_list);
10.          $f_\mathrm{i}$ = unroll_or_reccurence_solve$(f_\mathrm{i})$;
11. }
12. **if** (all_time_variables_eliminated)
13.    Overall_AT = $f_\mathrm{i}$
14. **else** Overall_ATS = $f_i$
```

The remaining blocks, which have both binary- (from primary inputs) and word-level inputs (outputs from other blocks), are represented as an MAT (Definition 3 for combinational circuits) or an MATS (Definition 5 for sequential elements). The above operations are executed in steps 6–9 of Algorithm 1. Additionally, the AT representations of previously traversed components are assigned to the current block inputs (step 7). Since these forms are functions of primary inputs, the resulting new composition is the AT (ATS if sequential blocks are in the fanin). Finally, an MATS of type II is converted to an MAT by loop unrolling or solving a recurrence equation (step 10). Upon traversing all blocks, the final AT (combinational specification) or ATS (sequential case) is completed (lines 12–14). While a single pass over the blocks suffices, the algorithm running time is dominated by the recurrence-solving step, followed by the transform generation and input substitution. For the last part, the running time is proportional to the size of the transform obtained (Lemma 3).

III. Verifying Imprecise Datapaths

The majority of methods for datapath verification are suitable for circuits that are implemented exactly. However, there is a serious shortcoming in this approach. It is a common design practice to approximate, round, or truncate some of the intermediate data, and thus obtain approximate results. In such cases, the exact match between the realization and the specification might not exist, but we still need to assert the circuit correctness under the introduced imprecision [17]. The goal is to distinguish between implementations that are incorrect and those that are correct within the allowed imprecision.

A. Imprecise Arithmetic Computations

We consider two major causes of imprecision in a circuit implementation. The first source is the approximations of specifications in hardware realization. The second is due to a finite-word implementation of an infinite length of specification data. For example, real numbers are realized using finite-size registers and buses, i.e., viewed as fixed-point data representations.

Definition 6: The error is a difference between the quantity obtained in the implementation and the result required by the circuit specification. The measure used for evaluating the error is the unit in the last place (ULP). This is the least significant bit for a given number encoding.

Definition 7: The precision is the total number of bits used in the fixed-point representation.

Definition 8: The arithmetic circuit approximation is the implementation that is inexact regardless of the precision.

1) Truncation and Rounding of Intermediate Data: The conversion between infinite-length real numbers and their finite precision implementations is routinely performed using truncation and rounding. These are the operations that restrict the bit width of the output, causing an imprecision.

Example 5—Truncation and Rounding: Consider a circuit where four n-bit fractional inputs a, b, c, and d are joined in the expression $ab + cd$. The exact $(2n+1)$-bit result is

$$\text{AT}(a*b+c*d) = \sum_{k=1}^{n} a_k 2^k * \sum_{k=1}^{n} b_k 2^k + \sum_{k=1}^{n} c_k 2^k * \sum_{k=1}^{n} d_k 2^k.$$

Assume that the implementation of the n-bit datapath forces the result to be restricted to n most significant bits. We consider two cases.

1) With rounding to the nearest value, the result is calculated in the full precision first, followed by the rounding step. The error is bounded to half of the ULP, i.e., $2^{-(n+1)}$.
2) For truncation to "n" bits, the error is bounded by one ULP, which is 2^{-n}.

In general, precision verification requires the explicit representation of output word-level values, as one cannot reason about the precision on a per-bit basis. We can easily construct an example of arithmetic computation where all output bits are incorrect, while the imprecision can be made arbitrarily small. If the exact n-bit result is $100 \cdots 0$, while the approximation is $011 \cdots 1$, then all bits are incorrect. However, the error is one ULP, which becomes negligible for large n.

B. Arithmetic Transforms and Imprecise Datapaths

We now show that the AT and its related forms are suitable for the verification of imprecise datapaths. While error effects can sometimes be easily found in terms of ULP (as in the previous example), in general, there would be an error accumulation in a circuit. The value of an error observed at circuit outputs is further dependent on the circuit inputs and the internal states.

One of the AT properties that is directly applicable to imprecision verification is linearity. The transform of an imprecise circuit f, $\mathrm{IAT}(f)$, can be represented as a linear superposition of the exact AT form (specification) and an imprecision error. Hence, various errors throughout the circuit f can be abstracted away and expressed in terms of an error e added to the fault-free AT of a specification $\mathrm{AT}(f)$

$$\mathrm{IAT}(f) = \mathrm{AT}(f) + e.$$

AT extensions preserve the linearity-based mechanisms dealing with the error accumulation in datapaths. They allow the use of word-level inputs, which are required to propagate the error e through the network. Additionally, through the sequential extensions to the AT, the propagation and the accumulation of an error are explicitly described and can be found by solving recurrence equations. The following example shows the use of MATS in computing the accumulated error.

Example 6—Using MATS for Calculating Error Accumulation: Consider the circuits in Fig. 3. Assuming that blocks $A1$ [Fig. 3(a)], $B1$, and $B2$ [Fig. 3(b)] describe an $n \times n$ multiplier. Inputs to all multipliers are n-bit wide, unsigned fractional binary vectors. The original $(2n+1)$-bit outputs in each case are rounded to n most significant bits. Rounding of each output introduces the error of a single multiplication bounded by $e = 1/2$ ULP.

In the case of the circuit in Fig. 3(b), the effect of rounding is incorporated in the MATS expression

$$\mathrm{MATS}(f)[i] = f[i-1] + x[i] * y[i] + e.$$

By solving the recurrence equation after k steps, the overall error Err, is bounded by $\mathrm{Err} = k * e$.

The circuit in Fig. 3(a) (a loop around a single multiplier) is represented by

$$\mathrm{MATS}(f)[i] = f[i-1] * a[i] + e.$$

The overall error Err after k steps amounts to

$$\mathrm{Err} = \sum_{i=2}^{k} \prod_{j=1}^{k} a[j] * e.$$

The worst case occurs for inputs encoded by all binary ones. The maximum overall error bound is then

$$\max_{\mathrm{Err}} = e * 2^n \left(1 - (1 - 2^{-n})\right)^k.$$

By considering the case when n goes to infinity, this worst case error still grows linearly in k. Hence, both calculations are producing tight bounds on the accumulated error. Such bounds are applicable when using error-affected blocks in larger circuits.

1) Imprecision Error in Terms of Arithmetic Transform: The major challenge of the imprecision verification is to distinguish between cases of an erroneous circuit behavior and circuit outputs that are correct within some error bounds due to imprecise arithmetic. Note that verification by equivalence checking deals only with exact computations and such imprecise cases are declared incorrect.

The AT of an error (AT_e) is defined independently as a source of imprecision, which can be caused, for example, by approximations of a specification function or restrictions in the size of the intermediate data in an implementation.

Definition 9: The AT error (AT_e) is the absolute value of the difference between arithmetic transforms of implementation (IAT) and its corresponding specification (SpecAT).

Example 7—Imprecision Error as AT Polynomial: A product $a * b$ is approximated by calculating only the n most significant bits and disregarding all partial products needed for obtaining n least significant bits. This approximation will save half the circuit area, but causes the AT error

$$\begin{aligned}\mathrm{AT}_e &= \mathrm{AT}(a*b) - \mathrm{AT}(a*b)_{\mathrm{trunc}} \\ &= \sum_{i=1}^{n} a_i \sum_{j=1}^{n} b_j 2^{-i-j} - \sum_{i=2}^{n+1} \sum_{j=1}^{i-1} a_{i-j} b_j 2^{-i} \\ &= \sum_{j=2}^{n} \sum_{i=n-j-2}^{n} a_i b_j 2^{-(i+j)}.\end{aligned}$$

After summation, we find that the worst case error is bounded by $((n-2)2^n + 2)/2^{2n+1}$, which is $O(n2^{-n-1})$.

2) Verification Formulation: Once the overall AT is constructed for an imprecise circuit, determining the value of an error amounts to the search for the maximum value of an error polynomial (AT_e).

When an input/output size of an implementation differs from that of a specification, the precision of the implementation is a required parameter. We then require that an implementation (IAT) agrees with a specification (SpecAT) within a precision error bound ε. The maximum absolute value of AT_e needs to be

$$\max |\mathrm{AT}_e(X)| = \max |\mathrm{SpecAT}(X) - \mathrm{IAT}(X)| \le \varepsilon. \quad (4)$$

The maximum AT error can be found by searching the set of circuit inputs X that maximize the mismatch. As the AT is linear (the difference of two ATs is still an AT polynomial), the problem reduces to subproblems of finding the maximal positive and the minimal negative values of the AT mismatch

$$\max |\mathrm{IAT} - \mathrm{SpecAT}| = \max \left\{ \begin{array}{l} \left|\max\left(\sum c_{i_1 i_2 \cdots i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}\right)\right| \\ \left|\min\left(\sum c_{i_1 i_2 \cdots i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}\right)\right| \end{array} \right\}.$$

When SpecAT is in itself imprecise and represents a function f up to an absolute precision of δ, the following condition holds. By applying triangle inequality to (4), we obtain

$$\begin{aligned}\max |\mathrm{IAT}(X) - \mathrm{AT}(f(X))| &\le \max |\mathrm{IAT}(X) - \mathrm{SpecAT}(X)| \\ &+ \max |\mathrm{SpecAT}(X) - \mathrm{AT}(f(X))| \le \varepsilon + \delta. \quad (5)\end{aligned}$$

With the value of δ known, it suffices to verify the imprecision of an IAT relative to its SpecAT as in (4).

C. An Algorithm for Precision Verification

Once an error description is represented in terms of the AT error AT_e, the next step is to establish the bounds of this error by determining input values that maximize the AT_e. By applying a binary search, finding suitable input assignments is done.

1) Branch-and-Bound Search for Imprecision Error: An input assignment that maximizes (minimizes) the AT_e polynomial is found through a branch-and-bound search. One approach to establish an upper bound $\mathrm{ub}_{\mathrm{coef}}$ of the AT_e is by summing all AT_e polynomial coefficients that are positive and the coefficient $c_{00\cdots0}$ that contributes a constant offset for all input assignments. Such a bound is calculated as

$$\mathrm{ub}_{\mathrm{coef}} = c_{00\cdots0} + \sum_{c>0} c_{i_1i_2\cdots i_n}.$$

By analogy, a lower bound $\mathrm{lb}_{\mathrm{coef}}$ is the sum of all negative coefficients and the constant $c_{00\ldots0}$. These two bounds are related, as shown by the following lemma.

Lemma 8: The sum of the bounds $\mathrm{lb}_{\mathrm{coef}}$ and $\mathrm{ub}_{\mathrm{coef}}$ is equal to the sum of the function values $f_{00\cdots0}$ and $f_{11\cdots1}$

$$\mathrm{ub}_{\mathrm{coef}} + \mathrm{lb}_{\mathrm{coef}} = f_{00\cdots0} + f_{11\cdots1}.$$

Proof: From the definitions of $\mathrm{lb}_{\mathrm{coef}}$ and $\mathrm{ub}_{\mathrm{coef}}$, their sum is

$$\mathrm{ub}_{\mathrm{coef}} + \mathrm{lb}_{\mathrm{coef}} = c_{00\cdots0} + \sum_{c>0} c_{i_1i_2\cdots i_n} + c_{00\cdots0} + \sum_{c<0} c_{i_1i_2\cdots i_n}$$

which is equal to $c_{00\cdots0}$ and the sum of all coefficients. The former is equal to the function value at the point (minterm) $00\cdots0$, and the latter is equal to the value at $11\cdots1$. ■

The AT_e attains the upper bound if there is a combination of inputs for which all positive coefficients are multiplied by one and all negative coefficients are multiplied by zero. The constant coefficient $c_{00\cdots0}$ is not affected by any input combination.

2) Search Procedure: Algorithm 2 describes the procedure for finding the maximum mismatch. First, to avoid calling the main search loop unnecessarily, the algorithm checks whether there are any input assignments to be made without the search. Such a preprocessing step is used at each call of the search routine.

```
Algorithm 2: Finding a Maximum Mismatch
 1. $\mathrm{ub}_{\mathrm{coef}} = c_{00\cdots0} + \sum_{c>0} c_{i_1i_2\cdots i_n}$;
    $\mathrm{lb}_{\mathrm{coef}} = f_{00\cdots0} + f_{11\cdots1} - \mathrm{ub}_{\mathrm{coef}}$
 2. CurrentBest = $\mathrm{lb}_{\mathrm{coef}}$; Current = $\mathrm{ub}_{\mathrm{coef}}$;
            ...
 3. Max_Abs (AT, Current){
 4. **fixpoint**{ **for each** variable $x_i$      /*preprocessing*/
 5.    **if** (all $c_{*_{xi}*}$ >= 0) $x_i = 1$
          **else**
          **if** (all $c_{*_{xi}*}$ <= 0) $x_i = 0$};
 6. if UnasignedVars {
 7.    $x$ = MostPositiveVariable;
 8.    $\mathrm{CAT}_e = \mathrm{AT}_e(f)_{x=1}$      /*try $x = 1$*/
 9.    **if** ($\mathrm{ub}_{\mathrm{coef}}(\mathrm{CAT}_e)$ < CurrentBest) backtrack;
         /*cut search branch*/
10.    **else** Current = Max_Abs($\mathrm{CAT}_e$, Current); /*recur further*/
11.    $\mathrm{CAT}_e = \mathrm{AT}_e(f)_{x=0}$      /*try $x = 0$*/
12.    **if** ($\mathrm{ub}_{\mathrm{coef}}(\mathrm{CAT}_e)$ < CurrentBest) backtrack;
13.    **else** Current = Max_Abs($\mathrm{CAT}_e$, Current);}
14. **else**{
15.    Current = $\mathrm{AT}_e$; /*all variables assigned, leaf case*/
16. **if** (Current > CurrentBest) **return**(CurrentBest = Current);
17. }}
```

Preprocessing: The reduction of the search space is based on two rules for assigning a value to a binary input.

1) If coefficients of the AT_e monomials with x_i present are all positive (or zero), assign $x_i = 1$.
2) If coefficients of the AT_e monomials with x_i present are all negative (or zero), assign $x_i = 0$.

Since each such assignment can make additional variables eligible, the given rules are applied in the fix-point manner until no changes are found. Preprocessing is done in lines 4–5.

Main Search Loop: The search is guided by a heuristic most positive variable as follows. For each input variable x_i, we obtain the sum S_{xi} of all coefficients multiplying terms with x_i. The most positive variable is the variable x_j, for which the sum S_{xj} is the largest. In the case of a draw, the easily obtainable lower bounds (by Lemma 8) are compared as well.

The function Max_Abs (line 3) is called recursively until all variable assignments have been explored. When the function UnasignedVars (line 6) indicates some unassigned variables, the allocation of the value of the most positive variable is attempted first. Otherwise, the leaf nodes are reached (line 15–16), and the value for the current input assignment is returned.

The search is terminated when a current upper bound is not larger than the currently largest value (lines 9 and 12, for two possible variable assignments). The upper bound of the current function restriction $\mathrm{ub}_{\mathrm{coef}}(\mathrm{AT}_e(x = v))$ is the sum of all positive AT_e coefficients when a variable x is set to v (line 1).

Example 8—Finding a Maximum Mismatch: Consider the error AT of an imprecise implementation of $f(x_1, x_2, x_3, x_4, x_5)$

$$\mathrm{AT}_e(f) = -10x_1 + 4x_2 - 5x_3 - 6x_4 - x_1x_5 + 8x_2x_3x_4 + 2x_1x_3x_5.$$

To find the maximum value $\max|\mathrm{SpecAT}(f) - \mathrm{IAT}(f)|$ of the $\mathrm{AT}_e(f)$, we apply the procedure described in Algorithm 2.

1) The upper bound (line 1) $\mathrm{ub}_{\mathrm{coef}} = 4 + 8 + 2 = 14$; the lower bound $\mathrm{lb}_{\mathrm{coef}} = 0 + (-10 + 4 - 5 - 6 - 1 + 8 + 2) - 14 = -22$.
2) CurrentBest $= \mathrm{lb}_{\mathrm{coef}} = -22$ (line 2).
3) Most positive variables (in the decreasing order) are

$$x_2 = 12; \quad x_3 = 5; \quad x_4 = 2; \quad x_5 = 1; \quad x_1 = -9.$$

4) First, preprocessing assigns $x_2 = 1$ (line 5).

The search is summarized in Fig. 4. By assigning the most positive variable x_3 to 1, followed by preprocessing, the first terminal node provides CurrentBest $= 1$. After that, all the branches where $\mathrm{ub}_{\mathrm{coef}}$ is not larger than 1 are cut. Upon setting $x_3 = 0$, the preprocessing consequently assigns zero to remaining variables, reaching CurrentBest $= 4$. Hence, the maximum is found after checking function value at only two points.

IV. Experimental Results

In this section, we demonstrate the efficiency of the proposed methods. First, we deal with transcendental functions, where the imprecision cannot be avoided. In the case of DA circuits, we match a specification with the sequential implementation. The experiments evaluating the efficiency of the precision verification conclude this section.

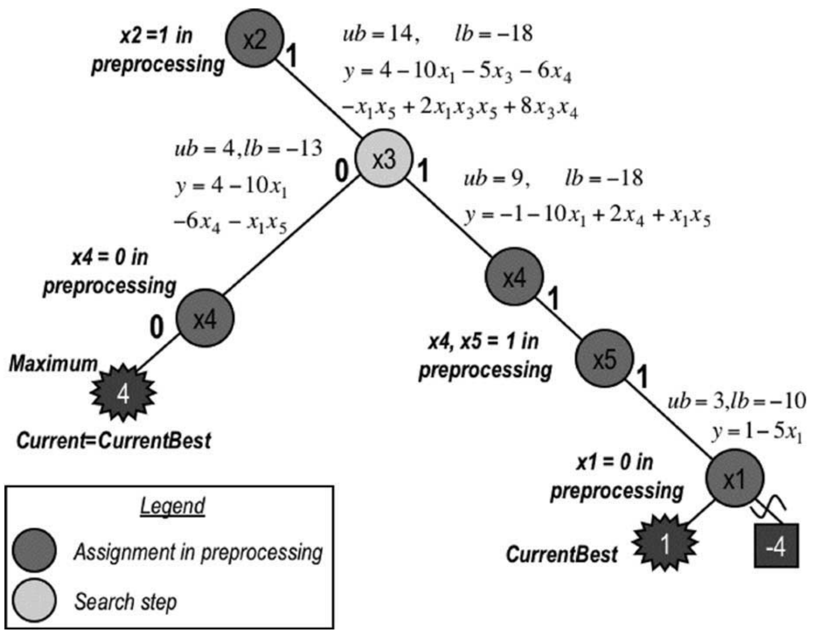


Fig. 4. Example of the calculation of maximum AT$_e$.

TABLE II
$\sqrt{-2\ln(x_1)} * \cos(2\pi x_2)$—AT GENERATION TIMES FOR DIFFERENT WORDLENGTHS n AND TAYLOR TERM APPROXIMATIONS (k, k)

n	(4,4)	(6,6)	(8,8)	(10,10)
8	0.02	0.22	1.45	3.8
12	0.08	0.61	5.78	44.0
16	0.11	5.0	45.0	230.5
24	0.74	19.0	124.1	735.3
32	1.95	49.3	583.2	2342.3

A. Imprecise Box–Muller Algorithm Implementation

The circuit for generating Gaussian random variables by Box–Muller algorithm [20] is critical to a number of applications such as accurate bit-error-rate testers [21]. The core of the algorithm uses the following expression:

$$y(x_1, x_2) = y_1(x_1) * y_2(x_2) = \sqrt{-2\ln(x_1)} * \cos(2\pi x_2). \tag{6}$$

Any implementation of this transcendental expression will only be an approximation—hence, the need to assess the error associated with it. We consider a case, where each product term is represented by a finite number of Taylor series terms: $y_1(x_1) = \sqrt{-2\ln(x_1)}$ around the point $x_1 = 0.5$ and $y_2(x_2) = \cos(2\pi x_2)$ around $x_2 = 0$

$$\begin{aligned} y_1(x_1) &= 1.17741 - 1.6984(x_1 - 0.5) + 4.73336(x_1 - 0.5)^2 \\ &\quad - 1.58199(x_1 - 0.5)^3 + 1.01982(x_1 - 0.5)^4 \\ y_2(x_2) &= \sum_{i=0}^{\infty} (-1)^i \frac{(2\pi x_2)^{2i}}{(2i)!}. \end{aligned} \tag{7}$$

Table II represents the time in seconds needed to obtain the AT expression for $y(x_1, x_2)$, composed from Taylor series approximation of compound functions $y_1(x_1)$ and $y_2(x_2)$ from (7). Both functions were approximated with the same number of Taylor terms, shown in the heading of Table II. Times in seconds are reported for Mathematica 5.1 code running on a 2-GHz AMD Opteron processor under Linux.

1) AT of Specification: The Box–Muller specification is given as a real-valued function $f : R \to R$, requiring infinite precision. To obtain the approximate specification that can be made arbitrarily precise, we note that Taylor expansions come with provable error bound $\delta = O(x^n)$ due to the truncation of the nth and higher order terms. Then, the reference AT can have a finite number of terms, plus arbitrarily small error bound δ. We then use the triangle inequality and error bounding from Section III-B2.

2) AT of Imprecise Implementation: In this case, the implementation of the circuit consists of three blocks: the two approximations of Taylor expansions representing functions $y(x_1)$ and $y(x_2)$, followed by a multiplier, in (6) and (7). For each Taylor polynomial, we use the shorthand notation

$$y_j(x_j) = \sum_{i=0}^{l_j} a_{ji} x_j^i, \quad j \in \{1, 2\}.$$

The lumped Taylor series coefficient of function y_j is a_{ji}, and the number of terms in the Taylor expansion is l_j. The AT is obtained by replacing x_j in the Taylor expansion by its norm function. For unsigned fractional encoding, we have

$$\mathrm{IAT}[y_j(x_j)] = \sum_{i=0}^{l_j} a_{ji} \left(\sum_{k=0}^{n} x_{jk} 2^{-k} \right)^i, \quad j \in \{1, 2\}. \tag{8}$$

The pipelined implementation of the cosine circuit in Fig. 5, (8) for $j = 2$, uses the Horner polynomial evaluation

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

The error polynomial is a difference between the SpecAT$[y(x_1, x_2)]$ and the IAT$[y(x_1, x_2)]$. The application of Algorithm 2 then results in the maximum error values, Table III.

B. Imprecise Distributed Arithmetic Circuit

DA refers to a datapath where the inner product of an input vector x with a vector of constants A is performed as a bit-serial operation [22].

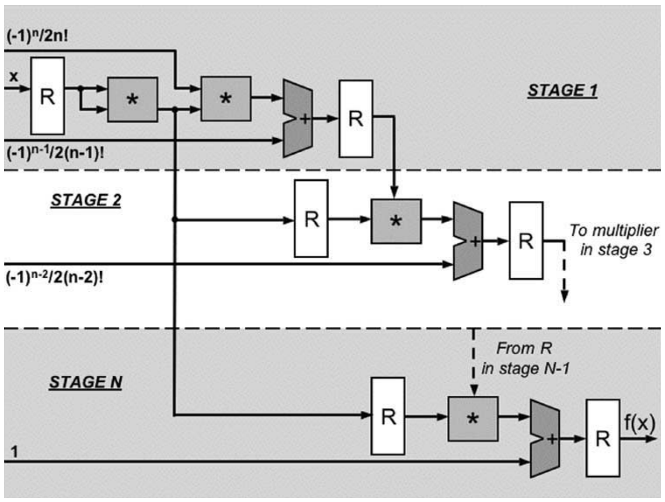
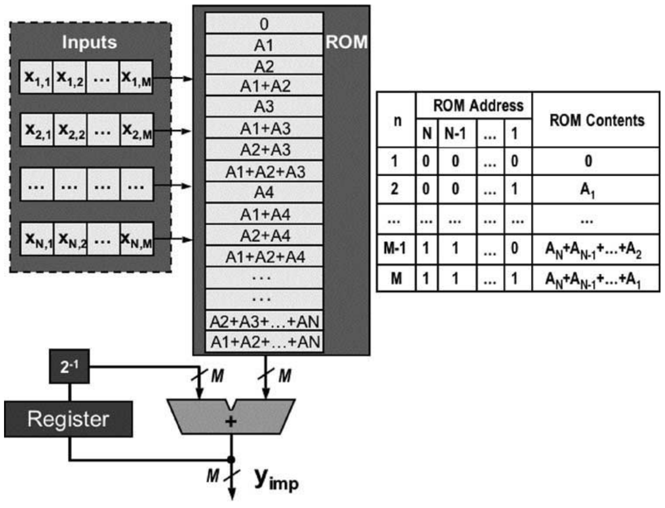


Fig. 5. N-stage pipelined cosine circuit implementation.



n	ROM Address				ROM Contents
	N	N-1	...	1	
1	0	0	...	0	0
2	0	0	...	1	A_1
...	...	...	...	...	...
M-1	1	1	...	0	$A_N+A_{N-1}+...+A_2$
M	1	1	...	1	$A_N+A_{N-1}+...+A_1$

Fig. 6. Block structure of the DA implementation.

TABLE III
PRECISION ERROR MAGNITUDES

n	**(4,4)**	**(6,6)**	**(9,6)**	**(6,9)**
8	0.02	0.018	0.0145	0.016
12	0.0014	0.00083	0.000058	0.00082
14	0.0011	0.00082	0.000057	0.00082
16	0.0008	0.00072	0.000053	0.00071

TABLE IV
GENERATION OF AT COEFFICIENTS FOR DA CIRCUITS

#	Unsigned		Signed		Encoded		Time
Terms	ROM	AT	ROM	AT	ROM	AT	[s]
8	256	120	512	128	128	128	0.5
16	16k	240	32k	256	8k	256	2.0
32	4G	480	8G	512	2G	512	8.1
64	4G*4G	960	8G*8G	1024	8G*8G	1024	32.4

It is assumed that all inputs are M-bit finite-precision words and that the vector of constants A contains N values. The inner product over unsigned fractional input vectors x_k, $k = 1, 2, \ldots, N$ is

$$y_{\text{spec}} = \sum_{k=1}^{N} A_k x_k, \qquad \text{where} \quad x_j = \sum_{i=1}^{M} x_{ji} 2^{-i}.$$

The above multiplication is performed in DA as a bitwise operation, taking M steps. In the ith cycle, the shifted ith bit of the vector x_k (i.e., x_{ik}) is "multiplied" in parallel by its constant A_i. In essence, the order of summations is changed, to result in

$$\text{AT}(y_{\text{spec}}) = \sum_{i=1}^{M} \left(\sum_{j=1}^{N} A_j x_{ji} \right) * 2^{-i}. \tag{9}$$

The implementation of a circuit given by (9) is proposed in Fig. 6. The hardware components are: N M-bit shift registers providing input vectors x_i, an $M \times N$ ROM storing constant coefficients of vector A, and an add-and-accumulate loop consisting of an M-bit register with a delay element 2^{-1} and an M-bit unsigned adder.

The overall AT transform of the scalar product circuit is generated following Algorithm 1. The shift registers sequencing input vectors to the ROM (M-bit data addresses) are expressed in terms of ATS: $\text{ATS}(SR(x_j))[k] = x_{j(M-k)}$.

Data selected from the ROM based on the value of input vectors is represented as $\text{MATS(ROM)}[k] = \sum_{j=1}^{N} A_j x_{j(M-k)}$. A new partial product in step j, $j = 1, \ldots, N$ is accumulated next in the register. The result $y_{\text{imp}}[M]$ is obtained after M steps

$$\text{AT}(y_{\text{imp}}) = \text{MATS}\,(y_{\text{imp}}[M]) = \sum_{k=1}^{M} \sum_{j=1}^{N} \boldsymbol{A}_j x_{j(M-k)} 2^{-(M-k)}.$$

The final $\text{AT}(y_{\text{imp}})$ is given by the expression that is equivalent to that of the specification (y_{spec}) in (9)

$$\text{AT}(y_{\text{imp}}) = \sum_{k=1}^{M} \sum_{j=1}^{N} A_j x_{j(M-k)} 2^{-(M-k)} |_{i=M-k}$$
$$= \sum_{i=1}^{M} \left(\sum_{j=1}^{N} A_j x_{ji} \right) * 2^{-i}.$$

The AT generation of various DA circuits was implemented in Mathematica 5.1—its performance is summarized in Table IV.

1) Error Analysis: The source of imprecision is a divider by two (2^{-1}) (Fig. 6), where the error is incurred by keeping the word length constant. The error e can be due to either truncation or rounding. For the add-and-accumulate loop, the recurrence representing the bottom blocks in Fig. 6 is

$$\text{MATS}(y_{\text{imp}})[i] = \text{ROM}[i] + \frac{1}{2}\, y_{\text{imp}}\, [i-1] + e.$$

The solution to this recurrence equation determines that the overall error is twice that of the error in a single step. Unlike previously considered loops in Fig. 3, where the error grows linearly with the number of steps; here, it remains constant. Furthermore, as the error function is unate, the search is avoided. Algorithm 2 only needs the preprocessing step to find all the input values.

C. Benchmarking Precision Search

The precision search from Algorithm 2 has been implemented in C++. To illustrate its performance, several imprecise circuit realizations have been considered. Two major sources of imprecision were examined: a finite-word implementation of real-valued input/output data and the approximations to the given function. The results are reported in Table V.

TABLE V
PRECISION SEARCH TIMES

Circuit	Error AT Size	# Inputs	Time (s)
Alu64	4032	130	<1
NTSC Filter	300	300	<1
Cos(12,12)	4095	12	1
Cos(16,10)	58650	16	126
Cos(32,6)	125407	32	402
Mu-Law (6)	40098	16	94
A-Law (6)	40098	16	101
BoxMiller(6)	61009	16	182

64-Bit Arithmetic-Logic Unit (ALU): The imprecision in the 64-bit version of the 74181 ALU was introduced in the $n \times n$ multiplication function by restricting the result to n most significant bits. All remaining ALU functions were exact.

49-Tap Low-Pass Channel Filter for the National Television System Commission (NTSC) Video Coding Standard: The filter [23] has the following characteristics: passband from 0 to $f_p/f_{sampl} = 0.2933$ and the stopband from $f_s/f_{sampl} = 0.3344$ to 0.5. The normalizing sampling frequency f_{sampl} is 13.5 MHz. The imprecision in this case is due to the finite word length of the input variables. The considered precision of the based filter is 14 bits, while the implementation uses 8 bits.

Pipelined Cosine Circuits: The cosine function is of particular interest in DSP due to its use in MPEG encoding. The circuit is outlined in Fig. 5 and also referred to in [24]. For the three cases reported in Table V, the imprecision was due to the finite word length (12, 16, and 32 bits, respectively) as well as the use of Taylor expansion with finite number of terms. In the three cases mentioned, the number of Taylor expansion terms was 12, 10, and 6 terms, respectively.

Communication Circuits—A-Law and Mu-Law Companders: These elements are used in different versions of the pulse-code-modulation standard CCITT G.711 for voice coding. Imprecision was due to the finite words (8 bits) and the transcendental function approximations by six Taylor terms.

Box–Miller Circuit: The design introduced in Section IV-A was checked for imprecision due to the Taylor expansion with six terms and finite-word approximations (8 bits).

In all cases, the preprocessing and the early termination due to the use of bounds were extensively applied. In fact, for the first two circuits, the preprocessing was able to detect and assign all positive and negative variables.

V. CONCLUSION

In this paper, we proposed circuit representations that extend the bit-level input/word-level output AT to allow a composition of individual blocks (MAT) and handling sequential (MATS, ATS) and imprecise datapaths. The above extensions provide equally compact circuit descriptions as the AT in a fraction of time and are especially suitable for arithmetic and DSP circuits.

Finally, the proposed representations can explicitly express the imprecision error of the implementation. Once the transform of such an error is generated, we determine its maximum value using the proposed branch-and-bound search and verify the correctness within prescribed error bounds.

REFERENCES

[1] Z. Zhou and W. Burleson, "Equivalence checking of datapaths based on canonical arithmetic expressions," in *Proc. 32nd Design Automation Conf.*, San Francisco, CA, 1995, pp. 546–551.

[2] D. K. Pradhan, S. Askar, and M. Ciesielski, "Mathematical framework for representing discrete functions as word-level polynomials," in *Proc. 8th Int. High-Level Design Validation and Test Workshop*, San Francisco, CA, 2003, pp. 135–139.

[3] Y.-T. Chang and K.-T. Cheng, "Self-referential verification for gate-level implementations of arithmetic circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 7, pp. 1102–1112, Jul. 2004.

[4] M. A. Thornton, M. D. Miller, and R. Drechsler, *Spectral Methods in VLSI CAD*. Dordrecht, The Netherlands: Kluwer, 2002.

[5] R. P. Bryant and Y. A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *Proc. 32nd Design Automation Conf.*, San Francisco, CA, 1995, pp. 535–541.

[6] K. Hamaguchi, A. Morita, and S. Yajima, "Efficient construction of binary moment diagrams for verification of arithmetic circuits," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, CA, 1995, pp. 78–82.

[7] R. Drechsler, B. Becker, and S. Ruppertz, "The K $*$ BMD: A verification data structure," *IEEE Des. Test Comput.*, vol. 14, no. 2, pp. 51–59, Apr.–Jun. 1997.

[8] M. Ciesielski, P. Kalla, Z. Zeng, and B. Rouzeyre, "Taylor expansion diagrams: A compact, canonical representation with applications to symbolic verification," in *Proc. Design Automation and Test Eur. (DATE)*, Paris, France, Mar. 2002, pp. 285–289.

[9] G. Fey, R. Drechsler, and M. Ciesielski, "Algorithms for Taylor expansion diagrams," in *Proc. 34th Int. Symp. Multiple-Valued Logic*, Toronto, ON, Canada, 2004, pp. 235–240.

[10] J.-H. R. Jiang and R. K. Brayton, "On the verification of sequential equivalence," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 6, pp. 686–697, Jun. 2003.

[11] M. Clark, M. Mulligan, D. Jackson, and D. Linebarger, "Fixed-point modeling in an UWB wireless communication system," in *Matlab Dig.*, 2004, [Online]. Available: www.mathworks.com/company/newsletters/digest/may04/uwb.html

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

[13] D. W. Currie, A. J. Hu, S. Rajan, and M. Fujita, "Automatic formal verification of DSP software," in *Proc. 37th ACM/IEEE Design Automation Conf.*, Los Angeles, CA, 2000, pp. 130–135.

[14] K. Radecka and Z. Zilic, "Using arithmetic transform for verification of datapath circuits via error modeling," in *Proc. VLSI Test Symp. (VTS)*, Montreal, QC, Canada, 2000, pp. 271–277.

[15] ——, "Design verification by test vectors and arithmetic transform universal test set," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 628–640, May 2004.

[16] ——, "Arithmetic transforms for verifying compositions of sequential datapaths," in *Proc. IEEE Int. Symp. Computer Design*, Austin, TX, 2001, pp. 348–358.

[17] M. Huhn, K. Schneider, T. Kropf, and G. Logothetis, "Verifying imprecisely working arithmetic circuits," in *Proc. Design Automation and Test Eur.*, Munich, Germany, 1999, pp. 65–69.

[18] Z. Zilic and K. Radecka, "On feasible multivariate interpolations over arbitrary fields," in *Proc. ACM Int. Symp. Symbolic and Algebraic Computing*, Vancouver, BC, Canada, 1999, pp. 67–74.

[19] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[20] D. Knuth, *The Art of Computer Programming*. Reading, MA: Addison-Wesley, 1998.

[21] D.-U. Lee, W. Luk, J. D. Villasenor, and P. Y. K. Cheung, "A Gaussian noise generator for hardware-based simulations," *IEEE Trans. Comput.*, vol. 53, no. 12, pp. 1523–1534, Dec. 2004.

[22] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.

[23] J. Radecki, J. Konrad, and M. Dubois, "The application of two-dimensional finite precision IIR filters to enhanced NTSC encoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 4, pp. 355–374, Aug. 1992.

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