Formal Verification using Don't-care and Vanishing Polynomials

Cunxi Yu, Maciej Ciesielski
Department of Electrical Computer Engineering
University of Massachusetts, Amherst, MA, USA
Email: ycunxi@umass.edu, ciesiel@ecs.umass.edu

***Abstract*—The paper describes a method of verifying sequential arithmetic circuits by adding a special type of redundancy, called "Vanishing Polynomials" and "Don't Care Polynomials". The proof of functional correctness consists in transforming the polynomial expression at the primary outputs into a unique polynomial in the primary inputs and comparing the computed expression against the expected specification. Experimental results show that the technique is efficient and scalable; for example, a 512-bit serial squarer requiring over 2000 clock cycles was verified in just 330 seconds. The runtime complexity is linear and memory complexity is quadratic in the number of gates.**

I. Introduction

Increased use of arithmetic modules in data-intensive applications makes arithmetic circuit verification is important. Even if arithmetic components generated by synthesis tool can be considered "correct by construction" , the problem of verifying non-standard, highly bit-optimized, embedded arithmetic circuits remains open. It becomes particularly important in sequential, bit/word-serial arithmetic circuits, where the result is computed over a fixed number of cycles. In such circuits, the input is provided serially and the result is accumulated over a number of cycles to produce an n-bit (or word-level) result. The goal is to prove that the circuit computes the required arithmetic function collected sequentially at the primary outputs.

Even though functional verification of such arithmetic circuits can be cast as a combinational bounded model problem, it is still challenging due to a large number of bits in practical arithmetic circuits. Boolean logic techniques, based on binary decision diagrams (BDDs) and satisfiability (SAT) solvers, have limited application to arithmetic circuits as they require flattening of the design into bit-level netlists, known as "bit-blasting". To cope with this issue, this paper addresses the verification problem by modeling the circuit as an *algebraic system* and proving that the polynomial word computed by the circuit matches the design specification, expressed in terms of the primary inputs.

An example of the type of circuits considered here is shown in Figure 1. It is an n-bit serial adder built out of a single-bit adder, which operates for n clock cycles to produce an (n+1)-bit result. An equivalent combinational model is obtained by unrolling the adder n times. The proof of functional correctness consists in transforming the polynomial associated with the result $Z = z_o + 2^1 z_1 + \cdots 2^n z_n$ into a polynomial expressed in primary inputs, $\{a_i\}, \{b_i\}$, applied to the circuit serially; and checking if this polynomial indeed represents the addition of two input operands: $Z = A + B = (a_o + 2^1 a_1 + \cdots 2^{n-1} a_{n-1}) + (b_o + 2^1 b_1 + \cdots 2^{n-1} b_{n-1})$. However, as demonstrated in the paper, such a straightforward unrolling may be inefficient from the verification point of view, and special techniques are needed to make it effective and scalable. Those techniques are the main focus of this paper.

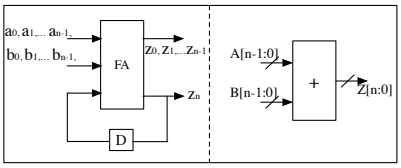


Fig. 1: Sequential n-bit adder, $Z = A + B$.

II. Previous Work

Several approaches have been proposed for formal verification of *logic* circuits. Most of them are based on a canonical, graph-based representation, such as Binary Decision Diagrams (BDDs) [1] and their derivatives, and other hybrid diagrams [2]. These diagrams helped improve the scalability of formal methods, but as the size of modern designs grows they lead to unavoidable memory explosion [3], [4]. To deal with these issues, word-level canonical representations have been developed that provide a higher abstraction of the circuit, including Binary Moment Diagrams (BMDs) [5], Taylor Expansion Diagrams (TED) [6], and others. These representations offer a better space complexity but require word-level information of the design, which is often not available or is hard to extract from bit-level netlists; for this reason they are not directly applicable to sequential circuits.

A comprehensive review of algebraic approach to verification of *combinational* arithmetic circuits, applicable to our work, has been provided in [7]. The sequential aspect of the arithmetic circuit adds another dimension to the verification problem since such circuits must be verified not only for the entire range of input data but also for potentially long computational sequences or iterations.

A lot of research has been done in sequential equivalence checking, reachability analysis, state traversal, etc., applied to control logic, but relatively little has been published on *functional* verification of *arithmetic* circuits. Boolean satisfiability (SAT), which is an effective platform for encoding

many CAD problems [4], [8], [9], has been used in verification of both control logic and arithmetic designs. SAT models for sequential designs typically rely on an Iterative Logic Array (ILA) representation by unrolling the combinational circuit component over a bounded number of cycles. Unfortunately, this technique when applied to modern industrial designs over a large number of cycles often exceeds the available memory resources [10].

A method for reducing sequential equivalence checking (SEC) of sequential logic into an equivalent combinational equivalence checking (CEC) is presented in [11]. The paper theoretically investigates when SEC can be reduced to CEC. It addresses the control part of large industrial designs, including pipelines but does not discuss the sequential arithmetic circuit verification. The work of [12] compares ATPG and SAT for checking safety properties and shows that, for relatively small circuits, two approaches are equally viable. Other analysis shows that sequential ATPG-based bounded model checkers outperform traditional SAT-based techniques, particularly for large designs [13].

Some attempts have been made to model the arithmetic circuit verification problem using satisfiability modulo theories (SMT) techniques. In [14] the circuit is modeled as a bit-level arithmetic (ABL) network composed of half-adders, represented as a set of polynomials B. The verification problem, i.e., proving that the implementation satisfies the specification, is posed as the reduction of the specification polynomial F modulo B. The method is limited to combinational arithmetic networks composed of half adders. In contrast to SAT, SMT techniques depart from treating the problem in a strictly Boolean domain and integrate different theories (Boolean logic, bit vectors, integer arithmetic, etc.) into a DPLL-style SAT decision procedure [15]. However, SMT solvers still model the problem as a decision process and are not efficient at verifying large arithmetic circuits.

Another group of verification methods is Theorem Provers, deductive systems for proving that an implementation satisfies the specification, using mathematical reasoning. The proof system is based on a large and problem-specific set of axioms and inference rules, such as simplification, term rewriting, induction, etc. The success of verification using theorem prover depends on the set of available axioms and rewrite rules, and on the order in which the rules are applied during the proof process, with no guarantee for a conclusive answer [16]. While theorem provers may be successful in well-structured circuits, they are ineffective in handling synthesized designs, especially the gate-level circuits considered in our work.

III. Preliminaries

The functional verification method described in this paper extends the combinational verification technique proposed in [7] to sequential arithmetic circuits. It computes a unique bit-level polynomial function implemented by the circuit directly from its gate-level implementation. This is done by rewriting the polynomial representing encoding of the primary outputs (the *output signature*) into a polynomial expressed in terms of the primary inputs (the *input signature*), using algebraic model of the internal gates. The method uses an algebraic model of the circuit, with logic gates represented by algebraic expressions, while treating signals as strictly *Boolean* variables. The following algebraic model is used to represent basic Boolean gates.

$$\begin{aligned} \neg a &= 1 - a \\ a \wedge b &= a \cdot b \\ a \vee b &= a + b - a \cdot b \\ a \oplus b &= a + b - 2a \cdot b \end{aligned} \tag{1}$$

Input signature, denoted by Sig_{in}, is a pseudo-Boolean polynomial in primary input (PI) variables that uniquely represents an integer function computed by the circuit, i.e., its specification. For example, input signature for an n-bit binary adder is $Sig_{in} = \sum_{i=0}^{n-1} 2^i a_i + \sum_{i=0}^{n-1} 2^i b_i$.

Output signature, Sig_{out}, of the circuit is similarly defined as a pseudo-Boolean polynomial in the primary output (PO) signals. Such a polynomial is uniquely determined by the binary encoding of the output. For example, the output signature of an unsigned arithmetic circuit with output bits z_i is $Sig_{out} = \sum_{i=0}^{n-1} 2^i z_i$.

The proof of functional correctness is based on successively rewriting the output signature Sig_{out} into a signature in the primary inputs and comparing it with the expected input signature Sig_{in}. At each step of the procedure, an intermediate polynomial generated by the rewriting corresponds to some *cut* in the circuit, a set of signals separating primary inputs from primary outputs. The rewriting process recursively applies Eq. (1), followed by an algebraic simplification of polynomial terms to arrive at a unique algebraic expression. It also applies a *Boolean reduction* by reducing any occurrence of a nonlinear term x^k, to a single variable x. During rewriting of nonlinear terms, the size of an intermediate polynomial representing a cut may increase exponentially, which seriously impacts the efficiency of the procedure. The size of the peak polynomial, commonly called the "fat belly", is a bottleneck for both the performance (CPU time) and memory used by the procedure.

The choice of the cuts (or, equivalently, the order in which the variables are eliminated by substitution) has big influence on the size of the fat belly and the efficiency of the rewriting process. A number of heuristics can be used to improve the efficiency, including: efficient data structure in the search of substituted variables; fast elimination of redundant terms; and other heuristics to minimize the size of the fat belly [7]. These techniques alone are not sufficient to avoid potential polynomial size explosion and additional techniques are needed. Some of them, specific to sequential arithmetic circuits, are discussed in the remainder of the paper.

A. Vanishing Polynomials

Definition 1 *Vanishing Polynomial (VP): Starting with a Boolean signal v, the rewriting process generates a set of pseudo-Boolean polynomials* $\mathcal{P} = \{p_1, p_2, ..., p_i\}$ *(p_i is the polynomial when rewriting reaches PIs). If there is a subset* $\mathcal{P}' = \{p_1, p_2, ..., p_i\}$ *($2 < i < n$) such that each p is non-zero polynomial, and* $p_{i+1} = p_{i+2} = ... = p_n = 0$, $\mathcal{P}'$ *are vanishing polynomials.*

Vanishing polynomials have been used to test if two fixed-size datapaths F_1, F_2 are equivalent by testing whether or

not a difference polynomial, F_1-F_2, reduces to 0 over Z_2^m for the given bit-width [17]. Vanishing polynomials over Z_2^m have also been used as an optimization technique in high-level synthesis [18] by adding redundancy to the fixed bit-width polynomial computation in order to minimize the implementation. The vanishing polynomials in our work are similar to those used in high-level synthesis, but serve a different purpose. They are pseudo-Boolean expressions that always evaluate to 0 and insertion of such polynomials into the design will reduce the complexity of the verification process.

The use of vanishing polynomial in our work is illustrated with a 2-bit squarer circuit in Figure 2(a), with mathematical computation done by the circuit shown in Fig. 2(b).

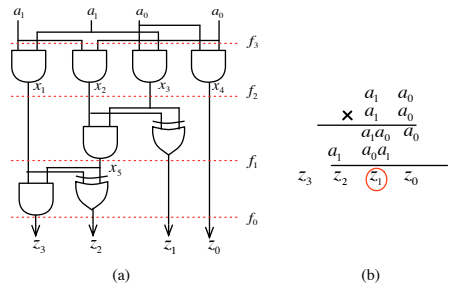


Fig. 2: A 2-bit combinational squarer circuit.

Close examination of the result shows that bit $z_1 = 0$, as it is a sum bit of two identical terms, a_1a_0 and a_0a_1. The resulting carry out bit (if any) is shifted one bit to the left and added to a_1 to produce z_2. Hence, with $z_1 = 0$, the initial starting point is $Sig_{out} = f_0 = 8z_3 + 4z_2 + z_0$ (without z_1). It is then transformed into f_1 using substitutions $z_3 = x_1x_5$ and $z_2 = x_1 + x_5 - 2x_1x_5$ ($c.f.$ Eq. 1). Subsequent rewriting, using equation for $x_5 = x_2x_3$, results in $f_2 = 4x_1 + 4x_2x_3 + z_0$. It is then transformed (using equations for the AND gates), to produce $Sig_{in} = f_3 = 4a_1 + 4a_1a_0 + a_0$. Comparing this result with the expected specification, $F_{spec} = (2a_1 + a_0)^2 = 4a_1 + 4a_0a_1 + a_0$, shows that the circuit correctly computes the square function.

Now let us include the vanishing polynomial for z_1, which in terms of the immediate signal variables can be written as $z_1 = x_2 + x_3 - 2x_2x_3$. In the first step, f_0 is transformed into f_1 as before. Then, f_2 is obtained from f_1 using equations for x_5 and z_1, resulting in $f_2 = 4x_1 + 2(x_2 + x_3) + z_0$. Finally f_3 is obtained by substituting variables x_1, x_2, x_3, x_4 with the corresponding equations for AND gates in terms of the primary inputs, a_0, a_1. The result is the input signature $Sig_{in} = f_3 = 4a_1 + 4a_1a_0 + a_0$. It also demonstrates that this is a correct arithmetic function. However, note that the intermediate form, $f_2 = 4x_1 + 2(x_2 + x_3) + z_0$, is simpler than the one computed without z_1, as it does not contain any nonlinear terms.

In general, vanishing polynomials contain terms that cancel other monomials during the cut rewriting and keep them smaller, especially for sequential arithmetic verification. This is because that many internal signals evaluate to zero iff the rewriting process reaches the PIs. These internal signals exit in the cascade arithmetic functions which are created by unrolling process. We demonstrate this using a Multiply-Accumulator in Section 4.1.

B. Don't-care Polynomials

Definition 2 *Don't-care Polynomial (DCP): Assuming d is a Boolean signal and $\mathcal{P}$=$\{p_1, p_2, ..., p_n\}$ is a set of polynomials of d which are generated by rewriting, if d is included in the arithmetic algorithm but excluded in the design, d and $\mathcal{P}$ are Don't-care Polynomials.*

Don't-cares are critical in verification because they can reduce the complexity of the netlist to be analyzed by equivalence checking. For example, in [19],[20] it is demonstrated that generating observability don't-cares for node merging, followed by SAT-based verification, greatly improves scalability and performance over other solutions. The don't-cares in our work are similar to those in [19], [20]. In our case, however, the role of don't-cares is to minimize the size of polynomials in each substitution step, rather than minimizing the computational complexity of SAT solving.

For example, let's assume that there is a 3-bit 2's complement adder, which the three LSBs $x[2:0]$ are used in the design. The MSB x_3 is the sign bit of this addition. Based on Definition 2, x_3 can be used as a *don't care* polynomial. According to [7], the *output signature* should be $x_0 + 2x_1 + 4x_2$. However, we observe that the internal expressions explore in the rewriting process without *don't care* signal, x_3. We compare the internal expressions with/without x_3 in Figure 3. We can see that the peak size (i.e. the number of monomials) of the internal expressions without x_3 is twice larger for a 3-bit adder. For larger designs, without the *don't cares*, the rewriting process will contain a 100x larger peak internal expressions which causes memory explosion problem. Including this bit as part of the arithmetic algorithm can simplify the verification process because it contains potentially cancelable monomials. Similarly, we are able to verify a n-bit comparator by including the output bits $[n-2, 0]$ of a n-bit subtractor.

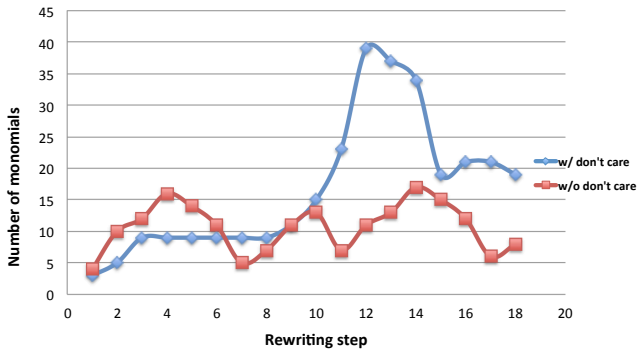


Fig. 3: Compare the size of internal expressions with, without *don't care* polynomial x_3.

Another type of don't-care polynomials can be generated in unbounded sequential circuits; they can be obtained by expanding the reachable states to those are not actually reached during the computation within the given range of input vectors [11] (in our case, within the given number of serial bits). A bit-serial squarer circuit, described in Section 4.2, will demonstrate this type of don't-care polynomial.

Including the *vanishing* and *don't-care* polynomials amounts to introducing redundancy into the original design, with the goal to improve the verification performance and scalability. This technique can be applied verbatim to formal verification of hardware for cryptography applications, e.g., extension fields arithmetic circuits, used in *Advanced Encryption Standard* (AES).

IV. Sequential Verification

In this section we show the application of VP and DCP to several types of sequential arithmetic circuits.

A. Multiply-Accumulator (MAC)

Consider an n-bit unsigned MAC circuit shown in Figure 4. The circuit should compute the result $R = \sum_{i=1}^{k} A_i B_i + C_0$ in k cycles, for k sets of n-bit inputs, $A_i[0...n-1], B_i[0...n-1]$, where $i = 1, ..., k$, and with some carry input vector C_0.

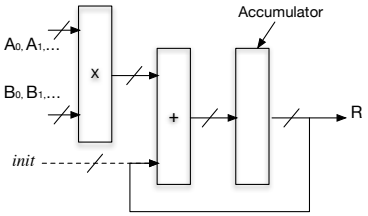


Fig. 4: Original MAC circuit: $R = \sum_i A_i \cdot B_i + C_0$.

Figure 5 shows the unrolled version of the circuit for $n = 4$ bits and $k = 2$ cycles. The proof of functional correctness is obtained by transforming $Sig_{out} = \sum_{i=0}^{9} 2^i r_i$ using algebraic equations of internal gates of the circuit into an input signature, using the rewriting technique discussed earlier. The resulting input signature Sig_{in} is a function of the 4-bit primary inputs A_0, B_0, A_1, B_1 and C_0.

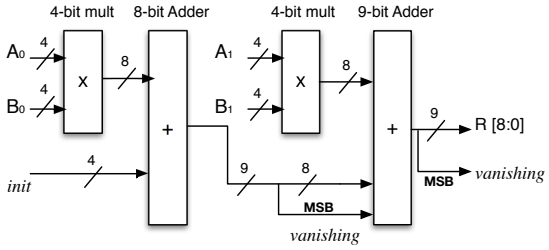


Fig. 5: MAC circuit unrolled over two cycles.

Note that the bit-widths of the two inputs to the first adder circuit in the unrolled model are different: they are 4-bit and 8-bit wide. Similarly, the bit-widths of the second adder are different (8 and 9 bits). This bit-width mismatch can be adjusted by the sign extension applied at the shorter inputs. Since the extension bits are all "0", the most significant bit, i.e., the carry-out of the first adder, always evaluates to 0 (it is a *Vanishing Polynomial*). This, combined with the mismatch between the inputs to the second adder, cause the two MSBs of the output to evaluate to 0 as well. Therefore, it may seems logical to exclude the two most significant bits from the computation to simplify the model.

However, such a straightforward application of the signature rewriting scheme to this circuit is not efficient, as it may result in a large number of product terms of the intermediate signature before reaching the PIs. As a result, the CPU time and memory consumption can be prohibitive. For example, it takes more than 190 seconds of the CPU time and requires 2 GB memory to verify a 4-bit MAC circuit on our computing platform (see the Results section).

To simplify and speed up the verification procedure, we take advantage of the structure of the unrolled model by identifying the *Vanishing Polynomials* associated with the 0-function bits and adding them to the output signature Sig_{out}. As mentioned before, adding such a redundancy can simplify the elimination and substitution procedure during signature rewriting. Specifically, many cancellations will occur between the terms of the vanishing polynomial and other terms of the computed signature during the elimination and substitution process. After adding the vanishing polynomials, we could verify a 64-bit MAC in just 4 seconds, with only 142 MB memory – compared to 190 seconds and 2 GB memory for the 4-bit version of the circuit (see Table II).

B. Serial Squarer

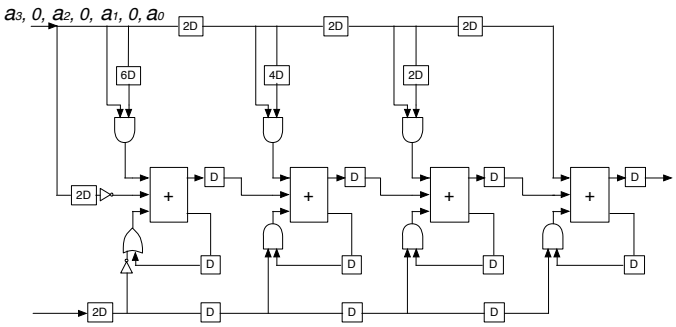
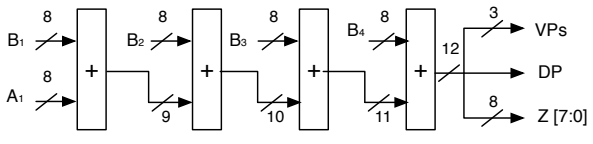


Fig. 6: A 4-bit Serial Squarer.

Another type of redundancy encountered in bit-serial arithmetic circuits appears in a *serial squarer* circuit [21] which computes a square value of an n-bit integer input. An example of a 4-bit serial squarer is shown in Figure 6. The input bits of an integer number are provided serially over a single line, interleaved with 3 zeros, and outputs bits of the result are collected serially at the single output line. The proof of functional correctness is obtained by transforming the Sig_{out} using algebraic equations of internal gates of the circuit into an input signature. The delay elements D are modeled by unrolling each module $2n$ times. The fully unrolled model is too large to be shown in the paper; a simplified model is shown in Figure 7.

To make the verification efficient, we extend bit-widths of the adder in the serial squarer circuit. Each of the four 1-bit adders is expanded over eight cycles, using standard techniques, into a combinational 8-bit adder. The resulting 8-bit adders as shown in Figure 7. The 8-bit input, B_2, to the



DP=*Don't-care polynomial* , VPs=*Vanishing polynomials*

Fig. 7: Unrolled 4-bit Serial Squarer.

second stage adder requires sign extension, while the other input is already 9-bit wide, as generated by the previous adder. Similarly, the inputs to the remaining two adders, B_3 and B_4 are sign-extended by 2 and 3 bits, respectively. As a result, the 12-bit output is composed of extra four bits. The most significant three of those bits are always 0 because of the sign-extension; they can be represented by vanishing polynomials. The other bit however, is not a 0-functions, but a *Don't Care*, providing the two's complement corrector to the remaining 8-bit adder. This represents a reachable state that is unreached in the design.

By including both types of polynomials (vanishing and don't cares), the computation of the input signature can be greatly simplified. Specifically, it is used to simplify the algebraic equations during the rewriting process in order to minimize the size of the "fat belly".

V. Experimental Results

The sequential verification method described in this paper has been implemented as a C++ program and tested on a number of sequential, serial arithmetic circuits, taken from [21] and [22]. The experiments were run on a PC with Intel Processor Core i5-3470 CPU 3.20GHz $\times$4 and 15.6 GB memory. Table I includes the CPU time and memory results for integer multiply-accumulator and add-shift multiplier circuits (an instance of the $s344/s349$ circuit from ISCAS-89 benchmarks, without the counter), extended to 256 bits.

Table II summarizes the benefit of using vanishing polynomials and don't care polynomials in computing the input signature, and its effect on solving the sequential arithmetic verification problem. The table shows that using both, don't care and vanishing polynomials, gives best solution in CPU time and memory usage. The reason why using the don't care polynomials only is better than using the vanishing polynomials only is that the don't care bit, which is the first unreached state in the serial squarer design, contains more information and can produce more cancellation of intermediate terms.

Circuit	256-bit		
	# Gates (Unrolled)	CPU [sec]	Mem
$GF(2^{256})$ Adder	3.5 K	0.21	1.1 MB
Add-Shift-Mult	587K	31.81	1.2 GB
2-Cycle MAC	1,049K	66.71	2.3 GB
6-Cycle MAC	3,148K	203.63	6.9 GB

TABLE I: Verification results for GF(2^{256}) Adder, MAC, and Add-shift Multipliers

To further analyze the effect of vanishing and don't care polynomials, we monitored the largest-size polynomial during rewriting (the fat-belly). The results are shown in Figure 8. The horizontal axis represents the time-line of the rewriting process as percentage of the complete run; the vertical axis represents the size of the expression at each point of the computation. We can see that the worst case corresponds to the case when output signature contains only PO signals (without vanishing or don't care polynomials). Including both types of redundant polynomials significantly reduces the size of fat belly and of the largest monomial. In summary, including don't-care and vanishing polynomials can improve efficiency and scalability of arithmetic verification.

Our verification method was also compared to SAT and SMT techniques for a bit-serial squarer circuit, ranging from 4 to 512 bits. The results, showing CPU runtime and memory usage, are given in Table III.

SAT comparison: The functional verification problem was modeled as Boolean satisfiability (SAT) on a product circuit, generated with a miter, and solved using the ABC system [23]. A miter was created between the unrolled serial squarer circuit and the reference design (a multiplier with two inputs tied together), and the miter's output was tested for unSAT. Several SAT tools were tested, including ABC (cec command) [23], miniSAT [24] , $lingeling$ [25], and $minisat\_blbd$, the winners of 2014 SAT competition [26].

For SMT comparison, we tested Boolector 2.0.0 (first place in SMT Competition 2014) [27], as well as Z3, and CVC4 tools. We tried two models: 1) We directly translated the algebraic equations of the unrolled serial squarer into SMT2 format and modeled the specification (Sig_{out}-Sig_{in}) as a Pseudo-Boolean polynomial using Boolean vector operations. Among the SMT solvers, Boolector produced the best result for the serial squarer circuit, but it was only able to solve up to an 8-bit version of the circuit in 3,000 seconds of CPU time. 2) The product circuit (miter) was translated directly into SAT by converting the CNF model into SMT2 format. This approach showed better performance; it is the one shown in Table III. As shown in the tables, our method has approximately linear CPU time complexity for all the tested circuits, and wins with the best SAT and SMT tools by several orders of magnitude of CPU times for designs above 16 bits.

VI. Conclusions

We introduced an efficient method for functional verification of gate-level sequential arithmetic circuits. This method goes beyond simple unrolling to a sequential circuit into a combinational one, as it applies *vanishing (VP)* and *don't-care (DP)* polynomials that are specific to sequential circuit operation. In addition to arithmetic verification, the proposed procedure can also be used to derive (i.e., extract) an arithmetic function implemented by the circuit by computing its input signature from the known output signature. We demonstrated that inserting a useful type of redundancy, the *VP* and *DP*, can greatly improve the verification time and scalability. To automatically generate *VP* or *DP* during unrolling, it requires certain account of domain knowledge of the circuit under verification, which is the current limitation.

ACKNOWLEDGMENT: This work was supported by a grant from the National Science Foundation under award CCF-1319496.

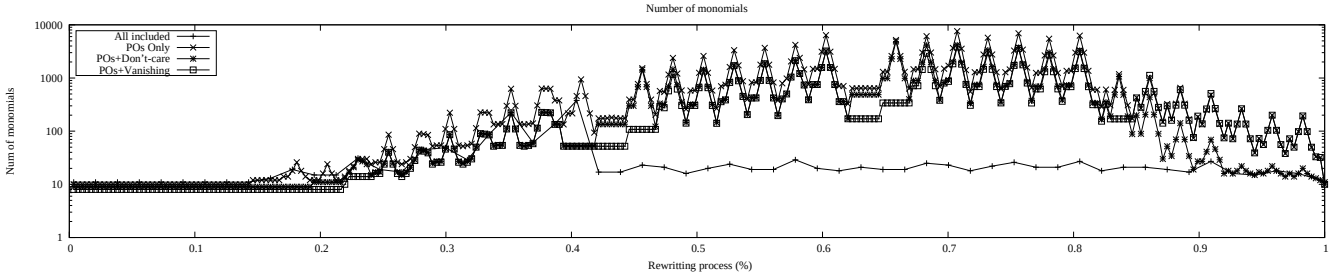


Fig. 8: Evaluation of Don't Care and Vanishing polynomials on a 4-bit serial squarer.

Size	2-Cycle Integer MAC				Serial Squarer							
	Vanishing Poly		*POs Only*		*Vanishing+Don't Cares*		*Don't Cares Only*		*Vanishing Only*		*POs Only*	
	CPU [sec]	Mem	CPU [sec]	Mem	CPU [sec]	Mem	CPU [sec]	Mem	CPU	Mem	CPU	Mem
4	0.01	2.2 MB	190.86	2.1 GB	0.01	2.3 MB	0.13	11.3 MB	-	MO	-	MO
6	0.02	3.0 MB	-	MO	0.04	3.1 MB	3.94	205.2 MB	-	MO	-	MO
8	0.06	3.9 MB	-	MO	0.06	4.2 MB	-	MO	-	MO	-	MO
64	4.24	142.1 MB	-	MO	4.71	161.8 MB	-	MO	-	MO	-	MO

TABLE II: Effect of Vanishing and Don't Care Polynomials for MAC and Serial Squarer (MO = Memory_out 8 GB)

Serial Squarer			Our		SAT [sec]				SMT [sec]		
Size	Clk Cycles	#.Gates	CPU [sec]	Mem	*minisat*	*ABC*	*lingeling*	*minisat_blbd*	*Boolector*	*Z3*	*CVC4*
4	11	255	0.01	2.32 MB	0.00	0.01	0.00	0.00	0.00	0.00	0.01
16	59	4.33K	0.26	11.7 MB	35.96	61.99	18.34	12.45	19.87	20.86	92.56
20	75	6.87K	0.42	17.3 MB	1698.53	TO	720.33	549.41	533.59	1045.18	TO
24	91	9.92K	0.62	24.1 MB	TO	TO	TO	TO	TO	TO	TO
64	251	71.2K	4.71	161.8 MB	TO	TO	TO	TO	TO	TO	TO
128	507	285K	18.63	667.3 MB	TO	TO	TO	TO	TO	TO	TO
256	1019	1.14M	77.12	2.63 GB	TO	TO	TO	TO	TO	TO	TO
512	2043	4.58M	331.64	10.5 GB	TO	TO	TO	TO	TO	TO	TO

TABLE III: Sequential Squarer results: comparison with SAT and SMT (TO = Time_out after 3600 sec)

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[2] D. Pradhan and e. I.G. Harris, *Practical Design Verification*. Cambridge University Press, 2009.

[3] M. Ganai and A. Gupta, *SAT-based scalable formal verification solutions*. Springer, 2007.

[4] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, *Symbolic model checking without BDDs*. Springer, 1999.

[5] R. E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *32nd DAC*. ACM, 1995, pp. 535–541.

[6] M. Ciesielski, P. Kalla, and S. Askar, "Taylor Expansion Diagrams: A Canonical Representation for Verification of Data Flow Designs," *IEEE Trans. on Computers*, vol. 55, no. 9, pp. 1188–1201, Sept. 2006.

[7] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of gate-level arithmetic circuits by function extraction," in *ACM Design Automation Conference (DAC-2015)*, 2015.

[8] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan, "An analysis of sat-based model checking techniques in an industrial environment," in *Correct hardware design and verification methods*. Springer, 2005, pp. 254–268.

[9] D. Kaiss, M. Skaba, Z. Hanna, and Z. Khasidashvili, "Industrial strength sat-based alignability algorithm for hardware equivalence verification," in *FMCAD*. IEEE, 2007, pp. 20–26.

[10] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in computers*, vol. 58, pp. 117–148, 2003.

[11] H. Savoj, A. Mishchenko, and R. Brayton, "Sequential equivalence checking for clock-gated circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 2, pp. 305–317, 2014.

[12] G. Parthasarathy, C.-Y. Huang, and K.-T. Cheng, "An Analysis of ATPG and SAT Algorithms for Formal Verification," in *HLDVT. Proceedings. Sixth IEEE International*. IEEE, 2001, pp. 177–182.

[13] D. G. Saab, J. A. Abraham, and V. M. Vedula, "Formal Verification using Bounded Model Checking: SAT versus Sequential ATPG Engines," in *VLSI Design, 2003. Proceedings. 16th International Conference on*. IEEE, 2003, pp. 243–248.

[14] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, A. Dreyer, F. Seelisch, and G. Greuel, "Stable: A new qf-bv smt solver for hard verification problems combining boolean reasoning with computer algebra," in *DATE*, 2011, pp. 155–160.

[15] A. Biere, M. Heule, and H. van Maaren, "Handbook of satisfiability," vol. 185, 2009.

[16] S. Vasudevan, V. Viswanath, R. W. Sumners, and J. A. Abraham, "Automatic Verification of Arithmetic Circuits in RTL using Stepwise Refinement of Term Rewriting Systems," *IEEE Trans. on Computers*, vol. 56, no. 10, pp. 1401–1414, 2007.

[17] N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan, "Exploiting vanishing polynomials for equivalence verification of fixed-size arithmetic datapaths," in *ICCD*. IEEE, 2005, pp. 215–220.

[18] S. Ghandali, B. Alizadeh, M. Fujita, and Z. Navabi, "Automatic high-level data-flow synthesis and optimization of polynomial datapaths using functional decomposition," *Computers, IEEE Transactions on*, 2014.

[19] Q. Zhu, N. B. Kitchen, and A. Kuehlmann, "Sat sweeping with local observability don′t-cares," in *43rd annual DAC*, 2006, pp. 229–234.

[20] S. M. Plaza, K.-h. Chang, I. L. Markov, and V. Bertacco, "Node mergers in the presence of don't cares," in *ASP-DAC'07*. IEEE, 2007, pp. 414–419.

[21] A. E. Cohen and K. K. Parhi, "Architecture optimizations for the rsa public key cryptosystem: a tutorial," *Circuits and Systems Magazine, IEEE*, vol. 11, no. 4, pp. 24–34, 2011.

[22] I. Koren, *Computer arithmetic algorithms*. Universities Press, 2002.

[23] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www. eecs. berkeley. edu/˜ alanmi/abc*, 2007.

[24] N. Sörensson and N. Eén, "Minisat 2.1 and minisat++ 1.0 - sat race 2008 editions," *SAT*, p. 31, 2009.

[25] A. Biere, "Lingeling, plingeling, picosat and precosat at sat race 2010," *FMV Report Series Technical Report*, vol. 10, no. 1, 2010.

[26] "Sat competition 2014," *URL http://www.satcompetition.org/2014/results.shtml*, July 14-17 in Vienna, Austria.

[27] R. Brummayer and A. Biere, "Boolector 2.0.0, winner of smt competition 2014," 2014.