JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 9, NO. 12, DECEMBER 2006(pp. 1607-1616)

SOC Verification Based on WGL

Zhen-jun Du[†], Min Li[††]

ABSTRACT

The growing market of multimedia and digital signal processing requires significant data-path portions of SoCs. However, the common models for verification are not suitable for SoCs. A novel model -- WGL (Weighted Generalized List) is proposed, which is based on the general-list decomposition of polynomials, with three different weights and manipulation rules introduced to effect node sharing and the canonicity. Timing parameters and operations on them are also considered. Examples show the word-level WGL is the only model to linearly represent the common word-level functions and the bit-level WGL is especially suitable for arithmetic intensive circuits. The model is proved to be a uniform and efficient model for both bit-level and word-level functions. Then Based on the WGL model, a backward-construction logic-verification approach is presented, which reduces time and space complexity for multipliers to polynomial complexity (time complexity is less than $O(n^{3.6})$ and space complexity is less than $O(n^{1.5})$) without hierarchical partitioning. Finally, a construction methodology of word-level polynomials is also presented in order to implement complex high-level verification, which combines order computation and coefficient solving, and adopts an efficient backward approach. The construction complexity is much less than the existing ones, e.g. the construction time for multipliers grows at the power of less than 1.6 in the size of the input word without increasing the maximal space required. The WGL model and the verification methods based on WGL show their theoretical and applicable significance in SoC design.

Keywords: SOC, Verification, WGL, Polynomial, Data path

1. INTRODUCTION

SoC (System-on-Chip) design is the trend for the development of embedded systems. The growing market of multimedia and digital signal processing requires more and more significant data-path portions of SoCs. Verification for such SoCs is a hard task baffling designers. The common used OBDDs (Ordered Binary Decision Diagrams) [1-4] to formally verify circuits are only bit-level diagrams, unable to describe functions above bit level and not suitable for high-level verification. However, data paths have regular structures, such as ALU, that can be described easily on a higher level of abstraction. Recently, EVBDDs [5], MTBDDs [6], *BMDs [6], (*P)HDDs [7-8] and ZBDDs [9] are used to overcome the shortcomings of OBDDs. Among them, *BMDs have been successfully developed to verify functions of linear circuits with word-level representations, which enable us to easily verify integer circuits such as multipliers and dividers. However, *BMDs are unsuitable for use in non-linear functions due to the resulting exponential complexity [6]. And other tools such as (*P)HDDs and ZBDDs also have their complexity problems or applicable limitations [7-9]. Furthermore, these above diagrams are not suitable for quantitatively comparing similarities or differences between two descriptions.

※ Corresponding Author : Zhen-jun Du, Address : School of Computer Science & Technology, Dalian Maritime University, Dalian China 116026, TEL : 086-0411-88354290, FAX : 086-0411-84726912, E-mail : duzhenju@newmail.dlmu.edu.cn

Receipt date : Oct. 24, 2006, Approval date : Dec. 18, 2006

[†] School of Computer Science & Technology, Dalian Maritime University

[††] School of Computer Science & Technology, Dalian Maritime University (liyuanjiangong12@163.com)

※ This work was supported by China National Natural Science Foundation Grant (60273081)

In recent years, polynomial algebra began to arouse interests in chip design. Some scholars initially used word-level polynomials to match and verify components in 1998 [10]. Data-path behavior was then described with polynomials and the allocation of data paths was realized in 1999 [11]. Algorithmic level synthesis of data intensive circuits was presented in 2001 [12]. And a methodology that maps algorithmic constructs of the software specification to a library of complex software elements was proposed in 2002 [13]. Their work shows polynomial representations can be used to compactly, canonically describe data-path behaviors and are efficient for functional verification and component matching and reuse, showing applicable significance of polynomial symbolic algebra in SoC design. However, in their work, the computation of word-level polynomial coefficients and the determination of the minimal polynomial orders are separate, and the computation of the minimal order is very complex. Also, it is difficult to consider timing factors or other parameters in their method, difficult to effect exact delay matching. Hence, In order to overcome the deficiencies, we propose a model that is uniform for bit-level and word-level functions and present a method of verification based on the model.

2. WGL MODEL

Based on polynomial manipulations, a data structure WGL (Weighted Generalized List) from the generalized list is presented in this section, which represents and manipulates both bit-level and word-level polynomials efficiently.

2.1 Function Decompositions

OBDDs and MTBDDs are based on a decomposition of Boolean functions called the *Shannon expansion*. FDDs (Functional Decision Diagrams) and BMDs (Binary Moment Diagrams) use the positive Davio decomposition. In HDDs each variable can use one of three different decompositions.

To represent polynomials with low complexity, we adopt the decomposition of generalized list as follows:

```
typedef enum {ATOM, LIST} Elemtag; //ATOM=0; terminal, LIST=1;
non-terminal (sub list)
typedef struct MPNode{
  Elemtag tag;              //to identify terminal and non-terminal
  int   exp;                //the exponent field
  union{
     float  coef;           //the coefficient field
     struct  MPNode * hp;   //the coefficient pointer
};
  struct MPNode * tp;       // pointing to next node in the same level
}  * MPList;
```

Additive and multiplicative operations for each node follow manipulation rules for one-variable polynomials. The addition rule is simple: for all terms with the same exponents, add coefficients in counterparts. If the addition result is not zero, it forms one term in the addition polynomial. For all terms with different exponents, copy these terms to the addition polynomials. Multiplicative operations are manipulated as the addition operations.

2.2 WGL Model

Based on the above idea, we propose a data structure for Boolean polynomials as follows. Substitute logic operations with addition, subtraction and multiplication, one Boolean function can be transformed to a Boolean polynomial. A Boolean polynomial is a multi-variable polynomial with Boolean variables. The exponent field for each node can only be 0 or 1, so we can simplify each node by combining nodes having linking relations in the same level into one node. Thus, there are four types of combined nodes as 00,01,10,11 with tag=0,1,2,3, where tag=1,2,3 denotes non-terminal nodes and tag=0 denotes terminal nodes.

Let coef0 and coef1 are coefficients for 0-order and 1-order terms. Let hp0 and hp1 be coefficient pointers for 0-order and 1-order terms respectively. The data structure for nodes is as follows:

```
typedef enum {0, 1, 2, 3} Elemtag;
typedef struct MPNode{
  Elemtag   tag;              // to identify terminal and non-terminal nodes
  union {
       float coef0;           // coefficients of zero-order terms for terminal nodes
       struct MPNode *hp0;    //coefficient pointers of zero-order terms non-terminal
};
union {
       float  coef1;          // coefficients of one-order terms for terminal nodes
       struct  MPNode * hp1;  // coefficient pointers of one-order terms non-terminal
       };
} * MBList;
```

To acquire maximal node sharing, we firstly propose two types of edge weights (right weights and left weights), which are introduced into the generalized lists. Thus the following definition is presented.

Definition 1 The data structure of a WGL (Weighted Generalized List) is that of a normal generalized list with two edge weights (right and left weights).

2.2.1 Right Weights

If there is only one-factor difference between two nodes, the concept of right weights can be introduced for the two nodes to share the same one. Put a number on the right of the directed edge to denote the right weight. Representations of A and $2A$ are shown in Fig. 1.

The usage of right weights will affect the canonicity of the WGL to some extent. So the rule for manipulating right weights should be specified to guarantee the resulting WGL canonical.

Rule 1: It's required that the right weights for the two branches leaving a vertex be relatively prime, i.e., the greatest common divisor (GCD) from both branches is extracted and the edge weights combine multiplicatively.

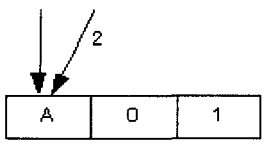


Fig. 1. The right weight.

2.2.2 Left weights

As for two nodes with only one constant different, we put a weight on the left of the directed edge to effect node sharing. The left weight means that the constant term of the node pointed by the directed edge should add this left weight. Fig. 2 shows A and $A+1$ share the same node by left weights.

To guarantee the canonicity, we add the following rules.

Rule 2: In constructing a WGL, constants are extracted to be at the level as high as possible to guarantee the canonicity and maximal node sharing.

The introduction of left weights will perhaps affect canonicity of right weights, so *rule* 1 needs some modifications.

Rule 3: It is required that all the four weights (two left weights and two right weights) for the two branches leaving a vertex be relatively prime.

The above rules are maintained by the way in which the WGLs are generated, in a manner analogous to BDD generation.

Additive and multiplicative operations of Boolean polynomials are similar to that of real-valued polynomials, only a constraint is required, i.e. the Boolean variable $x \cdot x = x$. Addition and multiplication are basic operations for WGLs. Other operations needed such as the *Concatenation* operation that concatenates smaller words (including bits) into one bigger word can be implemented directly via addition and multiplication.

Consider two WGLs (with identical variable ordering and decomposition types per variable) representing the integer-valued functions f_s and f_t for the words s and t, respectively. We define the

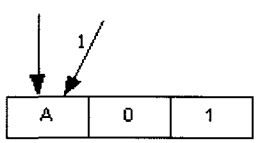


Fig. 2. The left weight.

Concatenation-Operation by setting

$$f_{st} = \text{Concat}(f_s, f_t) = (2^m f_s + f_t) \quad (1)$$

i.e. f_{st} is the function realized on the concatenated word s·t. Let $f_{s,0}$ ($f_{s,1}$) denotes the 0-order (1-order) term of f_s respectively, and so is for f_t, using the equation

$$\begin{aligned}\text{Concat}(f_s, f_t) &= (2^m f_s + f_t) \\ &= 2^m (f_{s,0} + x f_{s,1}) + (f_{t,0} + x f_{t,1}) \\ &= (2^m f_{s,0} + f_{t,0}) + x(2^m f_{s,1} + f_{t,1}) \\ &= \text{Concat}(f_{s,0} + f_{t,0}) + x\text{Concat}(f_{s,1} + f_{t,1}) \quad (2)\end{aligned}$$

The *Concatenation* operation can be computed on WGLs recursively.

If time parameters are put into a Boolean polynomial, it will become a WFP (Wave Form Polynomial) [14]. We propose a mechanism of delay extraction to further share nodes for WFPs.

2.2.3 Delay Extraction (Middle weights)

Delay extraction is to extract time parameters from WFP variables, shown in the middle of the directed edge as the form of weight. For instance, to represent $A_r + B_r$, extract delay τ and put it in the middle of the directed edge, as shown in Fig. 3. The coefficient sub-list pointed by one directed edge after delay extraction is called the *base list*. Polynomials those have the same base list can share it, which greatly improves node sharing.

Pure delay operation is introduced for WFPs, which only needs to manipulate middle weights

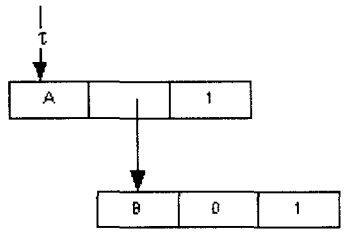


Fig. 3. Delay extraction.

(extracted delays) on directed edges.

Examples of Representations of Boolean Functions (Boolean Polynomials) are shown in Fig. 4. Representations as WGLs for the examples are comparable in size to BDDs.

Because of the flexibility of *delay extraction*, it is easy to represent combinatorial circuits with delays and sequential circuits.

2.3 WGL Representations of Word-level Functions

The WGL model proposed in this paper can not only represent Boolean functions, but also efficiently represent word-level functions with respect to both word-level variables and bit-level variables. Fig. 5 illustrates WGLs for representing the word-level functions with respect to bit-level variables such as X, $X+Y$, $X \times Y$, X^2, 2^X. The WGLs for them all grow linearly in word size, which are really good results. Table 1 compares WGLs with other common-used models for representing the word-level functions such as X, $X+Y$, $X \times Y$, X^2, 2^X with respect to bit-level variables.

Fig. 6 illustrates WGLs for representing the

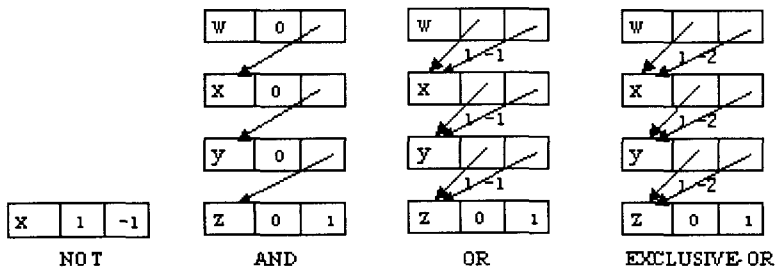


Fig. 4. WGLs for Boolean Functions.

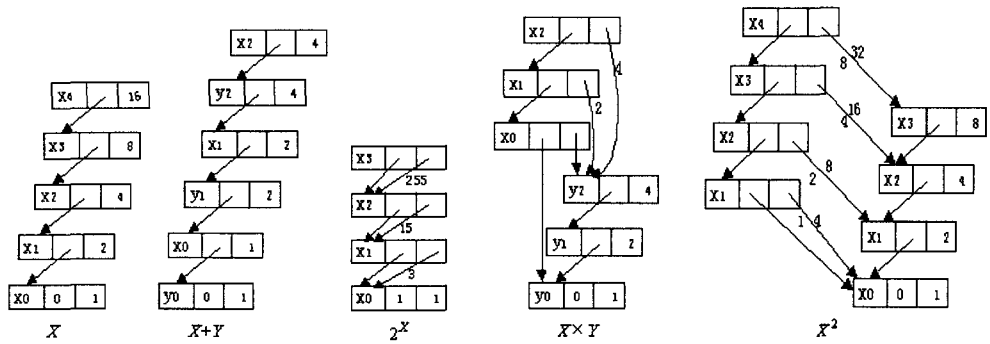


Fig. 5. WGLs for word-level functions (bit inputs).

Table 1. Word-level WGL (bit inputs) complexity

Models	X	X+Y	$X \times Y$	X^2	2^X
MTBDD	exponential	exponential	exponential	exponential	exponential
EVBDD	linear	linear	exponential	exponential	exponential
BMD	linear	linear	quadratic	quadratic	exponential
HDD	linear	linear	quadratic	quadratic	exponential
*BMD	linear	linear	linear	quadratic	linear
WGL	linear	linear	linear	linear	linear

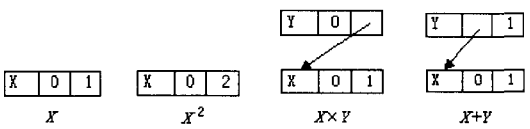


Fig. 6. WGLs for word-level functions (word inputs).

word-level functions with respect to word-level variables such as X, X^2, $X+Y$, $X \times Y$. It really deserves mention that our WGL model can not only represent word-level functions with respect to word-level variables linearly, but also represent them linearly with respect to bit-level variables.

It's also easy to consider timing parameters for word-level polynomials. In general, if the case is with time parameters, since every word-level variable corresponds to a vector composed of bit-level primary input variables, delays of word-level variables are the maximal delays from these corresponding primary inputs to outputs.

Bit-level WGL is suitable for datapaths, especially for arithmetic intensive circuits. And word-level WGLs with respect to word-level variables are efficient for all kinds of circuits. More importantly, WGLs can be used to transform bit-level descriptions to word-level polynomials with respect to word-level variables. Using word-level WGLs as to word-level variables we can effect across-level verification.

3. VERIFICATION

Assume we are given an encoding function ENC_0 defining a word-level interpretation of the output. Finally, we are given as specification an arithmetic function $F(X_1, \cdots, X_k)$, where $X_i = ENC_i(\vec{x}^i)$. The task of verification is then to prove the equivalence:

$$ENC_0(\vec{f}(\vec{x}^1, \ldots, \vec{x}^k)) = F(ENC_1(\vec{x}^1), \ldots, ENC_k(\vec{x}^k)) \quad (3)$$

The WGL provides a suitable data structure for this form of verification, because they can efficiently represent both bit-level and word-level

polynomials as to bit or word-level variables. We propose a verification methodology based on WGLs that can overcome the defects as above. Given a Boolean implementation and the word-level specification $F(X_1,\cdots, X_k)$, an intuitively straightforward approach to verifying circuits with WGLs is to construct $\vec{f}(\vec{x}^1,...,\vec{x}^k)$ using WGLs, and encode it to $ENC_0(\vec{f}(\vec{x}^1,...,\vec{x}^k))$. Testing equation (3) for equivalence then completes the verification. This straightforward approach works well for many circuits, however to avoid partitioning the circuit into blocks or partitioning input words into smaller ones, we adopt the backward approach to verify circuits.

3.1 Backward Construction Verification

We develop a verification approach—a backward construction approach with the WGL that does not require a hierarchical partitioning in this section.

Verification Algorithm ( )

{ /* *The input is the net-list or other gate-level description of a circuit. The output is the WGL of* $ENC_0(\vec{f}(\vec{x}^1,...,\vec{x}^k))$. */

1. Start with a WGL representation of the function $F=ENC_0(\vec{z})$, *where* $\vec{z}=(z_0, z_1, \cdots, z_m)$ *represents the circuit output. We construct a WGL of the word-level representation encoded from the bit-level representation.*

2. Regard the circuit as a DAG, topologically sort the circuit (using breadth-first sorting. Then get the reversed topological ordering of gates as the ordering to choose gates.

3. *while (not all the inputs are covered)*

{*Choose the next gate G according to the ordering in 2. Reconstruct F using the Boolean function corresponding to the gate G.*

}

The derived WGL represents $ENC_0(\vec{f}(\vec{x}^1,...,\vec{x}^k))$.

}

Example 1: Consider a logic circuit in Fig. 7. The backward construction for the example in Fig. 7 is as follows.

(1). Firstly we get $F = y_0+2y_1 = c+2y_1$ and its WGL is shown in (a) in Fig. 8.

(2). Topologicaly sort the gates in the circuit, get the gate ordering 1-2-3-4. Reverse the ordering to 4-3-2-1 as the order for gates to choose.

(3). Choose gate 4 as the first backward step, have

$y_1 = c\, x_0x_1$. Then we have $F= c+2c\, x_0x_1$. Its WGL becomes (b) in Fig. 8.

(4). Choose gate 3 as the second backward step, have

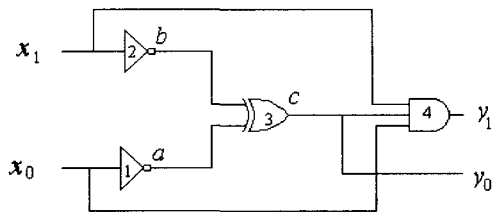


Fig. 7. A circuit example.

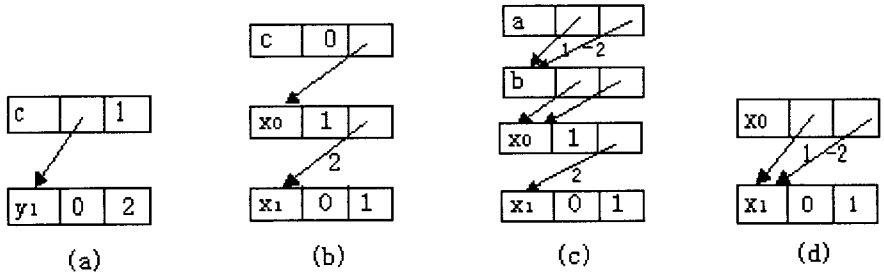


Fig. 8. WGLs for F.

$c = a+b-2ab$. Then get $F= (a+b-2ab)(1+2\ x_0x_1)$, whose WGL becomes (c) in Fig. 8.

(5). Then continue to choose gate 2 and gate 1, get $b=1-\ x_1$, $a=1-x_0$, have $F = x_0+x_1-2x_0x_1$, whose WGL becomes (d) in Fig. 8. The WGL for F derived till now represents $ENC_0(\vec{f}(\vec{x}^1,...,\vec{x}^k))$.

3.2 Construction of word-level polynomials

The above backward construction approach constructs $ENC_0(\vec{f}(\vec{x}^1,...,\vec{x}^k))$ from a Boolean description. Next we present a new efficient way to transform $ENC_0(\vec{f}(\vec{x}^1,...,\vec{x}^k))$ to $F(X_1,\cdots, X_k)$ in order for us to directly use $F(X_1,\cdots, X_k)$ as the model to implement complex high-level verification.

Given a Boolean polynomial vector $\vec{f}(x_0,...,x_{n-1})$, using backward construction, we can get its output word-level polynomial with respect to bit vector $(x_0, \cdots, x_i, \cdots, x_{n-1})$ is

$$P(x_0, \cdots, x_{n-1})= ENC_0(\vec{f}(x_0,...,x_{n-1}))$$
$$=2^{n-1}f_{n-1}(x_0,...,x_{n-1})+2^{n-2}f_{n-2}(x_0,...,x_{n-1})+...+2^0f_0(x_0,...,x_{n-1}) \quad (4)$$

Because word-level variable (assume only one word variable)

$$X = 2^{n-1}x_n + 2^{n-2}x_{n-1} + ... + 2^0x_1 \quad (5)$$

Suppose k is the order of this polynomial, output word-level polynomial with respect to the word-level variable is

$$P(X) = c_kX^k + c_{k-1}X^{k-1} + ... + c_0 \quad (6)$$

The formal description of the verification is:

Determination algorithm for word-level polynomial ( )

{

1. Let k=the order of $P(x_0, \cdots, x_{n-1})$; */* Let the initial value of k be the order of expression (4) */*

2. Substitute (5) for (6), then compare coefficients with expression (4), build a linear equation set with respect to coefficients c_i ;

3. while (no solutions for the equation set)

{ $k=k+1$;

Substitute (5) for (6), then compare coefficients with expression (4), build a linear equation set with respect to the coefficients;

}

Output the polynomial order and coefficients;

/ if there is a solution, it is also unique */*

}

The above method combines order computation and coefficient solving, and adopts an efficient backward approach to derive $ENC_0(\vec{f}(x_0,...,x_{n-1}))$, more efficient than the way in [13] and more easily to be realized. This method for one-variable polynomial can be easily extended to multi-variable cases, only needing to follow multi-variable polynomial operations.

Example 2: Consider a circuit (2 inputs and 5 outputs, i.e. input word length is 2 and output word length is 5). Its Boolean polynomial vector is

$$f_0 = x_0\ ;\ f_1 = x_0x_1;\ f_2 = 0\ ;\ f_3 = x_1\ ;\ f_4 = x_0x_1.$$

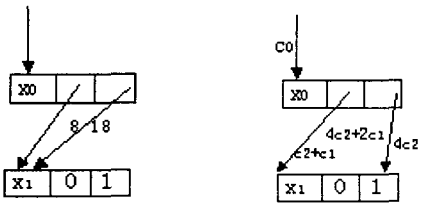
We form the one-variable word-level polynomial for this Boolean polynomial vector as follows.

Using backward construction, when the variables of the output word-level polynomial are bit variables (x_0,x_1), we get

$$P(x_0, x_1) = 2^4f_4 + 2^3f_3 + 2^2f_2 + 2f_1 + 2^0f_0$$
$$= (2^4+2)x_0x_1 + 2^3x_1 + x_0.$$

Suppose word-level variable $X=2x_1+x_0$. When the output word-level polynomial is with respect to word-level variable X, let the order of $P(X)$ be 2 (because the order of $P(x_0, x_1)$ is 2). Then have

$$P(X) = c_2X^2 + c_1X + c_0$$
$$= 4c_2x_1x_0 + (4c_2+2c_1)x_1 + (c_2+c_1)x_0 + c_0.$$



WGL for $P(x_0, x_1)$ WGL for $P(X) = c_2X^2 + c_1X + c_0$

Fig. 9. WGLs for Example 2.

Compare all coefficients of WGLs for $P(x_0, x_1)$ and $P(X)$, there is no solution. Add 1 to the order of $P(X)$, and have

$$P(X) = c_3X^3 + c_2X^2 + c_1X + c_0$$
$$= (18c_3 + 4c_2)x_1x_0 + (8c_3 + 4c_2 + 2c_1)x_1 + (c_3 + c_2 + c_1)x_0 + c_0.$$

Compare coefficients and get $c_3 = 1$, $c_2 = c_1 = c_0 = 0$. So $P(X) = X^3$.

4. EXPERIMENTAL RESULTS

Experiments are made on a 64bit PC with 512M memories for multipliers to test our backward construction methodology without partitioning. Multipliers used are combinational n-bit ones constructed out of combinational 4bit multipliers. See table 2 and table 3 for the results.

Table 2. Verification time for multipliers (backward construction without partitioning)

Multipliers	Win	Pw	Tp	Tw
M4	4	1	1	1
M8	8	2	10.1	3.33
M16	16	4	123.6	3.47
M32	32	8	1611.3	3.55
M64	64	16	23746.2.2	3.63

Where

Win : input word length

Pw: the ratio of input word length (devided by word length 4)

Tp: verification time ratio (divided by that of M4)

Tw = log_{Pw} Tp

Table 3. Verification size for multipliers (backward construction without partitioning)

Multipliers	Win	Pw	Sp	Sw
M4	4	1	1	1
M8	8	2	2.5	1.32
M16	16	4	6.8	1.38
M32	32	8	18.2	1.39
M64	64	16	51.2	1.42

Where

Win: input word length

Pw: the ratio of input word length (devided by word length 4)

Sp: ratio of maximum WGL size (divided by that of the M4)

Sw = log_{Pw} Sp

If without word partitioning, the complexity of the method in [10-13] is exponential. Ref. [10-13] cannot implement verification for 16bit multipliers due to its blown-up OBDD node-numbers. But using our backward construction method, we effect verification for multipliers in polynomial complexity, with time complexity less than $O(n^{3.6})$ and space complexity less than $O(n^{1.5})$. This result, we think, is promising in actual chip design.

Experiments on the construction method of word-level polynomials for the multipliers are also made. See table 4 for the results. Construction time grows at the power of less than 1.6 in the size of the input word, better than the existing results.

Table 4. Construction results of multipliers (with partitioning)

Multipliers	Win	Pw	Tp	Tw
M4	4	1	1	1
M8	8	2	2.72	1.44
M16	16	4	7.75	1.48
M32	32	8	23.02	1.51
M64	64	16	71.14	1.54

Where

Win : input word length

Pw: the ratio of input word length (devided by word length 4)

Tp: construction time ratio (divided by the construction time of the 4 bit adder)

$Tw = log_{Pw} Tp$

5. CONCLUSIONS

A new model -- WGL for SoC verification based on polynomial symbolic manipulations is proposed in this paper. The model is proved to be a uniform and canonical model for bit-level and word-level polynomials. Examples show that the word-level WGL is the only model to linearly represent common word-level functions. Based on WGL, a backward-construction verification approach is proposed, which effects polynomial verification for multipliers without hierarchical partitioning. And a construction methodology of word-level polynomials as to word-level variables is also proposed in this paper in order to implement complex high-level verification, whose construction complexity is proved to be much less than the existing ones. Both the model and the verification methods show their theoretical and applicable significance in chip design.

6. REFERENCES

[1] R. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, Vol. C-35, No. 8, pp. 677-691, 1986.

[2] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, Vol. 24, No. 3, pp. 293-318, 1992.

[3] J. Bern, C. Meinel, and A. Slobodova, "Efficient OBDD-based Boolean Manipulations in CAD beyond Current Limits," *Int'l Proc. 32nd DAC*, pp. 408-413, 1995.

[4] K. Ravi, K. McMillan, and T. Shiple, "Approximation and Decomposition of Binary Decision Diagrams," *Int'l Proc. 35th DAC*, pp. 445-450, 1998.

[5] C. Kern and M. Greenstreet, "Formal Verification in Hardware Design: a Survey," *ACM Trans. DA of Elec. & Sys.*, Vol. 4, No. 4, pp. 123-142, 1999.

[6] R. Bryant and Y. A. Chen, "Verification of Arithmetic Circuits With Binary Moment Diagrams," *Int'l Proc. 32nd DAC*, pp. 535-541, 1995.

[7] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid Decision Diagrams," *Int'l Proc. ICCAD*, pp. 159-163, 1995.

[8] Y. A. Chen and R. Bryant, "*PHDD: an Efficient Graph Representation for Floating Point Circuit Verification," *Int'l Proc. ICCAD*, pp. 2-7, 1997.

[9] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *Int'l Proc. DAC*, pp. 272-277, 1993.

[10] J. Smith and G. Micheli, "Polynomial Methods for Component Matching and Verification," *Int'l Proc. ICCAD*, pp. 678-685, 1998.

[11] J. Smith and G. Micheli, "Polynomial Methods for Allocating Complex Components," *Int'l Proc. DATE*, pp. 217-222, 1999.

[12] A. Peymandoust and G. Micheli, "Using Symbolic Algebra in Algorithmic Level DSP Synthesis," *Int'l Proc. 38th DAC*, pp. 277-282, 2001.

[13] A. Peymandoust, T. Simunic, and G. Micheli, "Complex Library Mapping for Embedded Software Using Symbolic Algebra," *Int'l Proc. 39th DAC*, pp. 325-330, 2002.

[14] Y. Min and Z. Li, "An Analytical Delay Model," *Journal of Computer Science & Technology*, Vol. 14, No. 2, pp. 97-115,1999.

Zhen-jun Du

Zhen jun Du, who was born in 1975, now is an associate professor in school of computer science and technology, Dalian Maritime University in China. He got his PhD degree of Computer science in Harbin Engineering University in 2003, having two years' post doctorate research experience in Harbin Institute of Technology. His research interests include embedded system design, multimedia theories and algorithms, chip design, high performance computing and etc.

Min Li

Min Li, who was born in 1982, now is pursuing her post-graduate degree in school of computer science and technology, Dalian Maritime University in China. She got her Bachelor degree of Computer science and technology in Shenyang Jianzhu University in 2005. Her research interests include embedded system design, multimedia algorithms and etc.