ISSN1000-9825 Journal of Software 软件学报 2000,11(5):569~583

An Application of Domain Theory to Logical Design of VLSI Circuits*

SUN Yong[1] HU Yi[2]

[1](School of Computer Science The Queen's University of Belfast Northern Ireland)

[2](Integrated Silicon Systems Limited Northern Ireland)

E-mail: y.sun@qub.ac.uk/yhu@iss-dsp.com

Abstract This paper is to suggest that traditional 2-valued Boolean algebra is not sufficient for representation of VLSI circuits at logic gate level, although it is the case for combinational circuits. Instead of using the Register-and-Transfer technique at RT level to represent sequential circuits, an alternative is sought and found. That is, all uncertain voltages such as oscillations and floating voltages are identified by the same value, denoted as $\perp$ (called *bottom*). The two certain voltages are, as usual, *ground* and *power*; they are denoted by 0 and 1 respectively. An *invertor*, a *nor-gate* and a *nand-gate* are defined according to the physics of VLSI circuits instead of the Boolean algebra. As a result, a logic is obtained which coincides with Kleene 3-valued logic provided that Kleene's $u=\perp$. As it is well known, Kleene 3-valued logic is functionally *incomplete*. This means that not every function (or gate) can be constructed from invertors, nor-gates and nand-gates. However, by introducing a partial order $\sqsubseteq$ into the logic, by using general fixed-point operators instead of the least fixpoint operator to deal with feedbacks in VLSI circuits, and by applying CPO (complete partial order) domain theory to the derived 3-valued logic system, the result obtained means that this system is functionally *monotonic complete*. Also, the canonical normal forms for this Kleene 3-valued logic are obtained. Although the present results are mainly semantic, it is very interesting in pursuing the research further by investigating the syntactical derivability. Such research would derive more secure circuits close to reality, which is to make the work compatible with VHDL, Verilog HDL and/or EDIF. Incidentally, Mukaidono has obtained similar results, although his approach is not as coherent as the one presented in this paper.

Key words VLSI circuit, logic gate, Kleene 3-valued logic, Complete Partial Order (CPO), general fixpoint operator, monotonicity.

* **SUN Yong** is a lecturer in the School of Computer Science, the Queen's University of Belfast (Northern Ireland). He received a Ph.D. degree from the University of Edinburgh (Scotland). His current research areas include intelligent tutoring systems, artificial intelligence, formal methods, theoretical computer science, computer science education, and foundations of computer science. **HU Yi** is the Chief Technologist with the Integrated Silicon Systems Ltd (Northern Ireland). He received a Ph.D. from the Queen's University of Belfast (Northern Ireland) in 1992. In China he was a Research Engineer involved in the development of VLSI CAD tools and the design of integrated circuits. In Belfast he has developed an ASIC design automation tool, with special application to DSP chip design. In ISS he has implemented a library of parameterisable and synthesisable VHDL functions for DSP ASICs, as well as a range of complex DSP cores including MPEG, JPEG, FFT and ADPCM. He has wide ranging design experience including all aspects of complex chip design from algorithm development through to ASIC layout level.

Manuscript received 1998-09-01, accepted 1998-12-24.

1 Introduction and Overview

Traditionally, 2-valued Boolean functions are used to represent VLSI circuits at gate-level (or logical-level). 0 represents the low voltage (*GROUND*) and 1 represents the high voltage (*POWER*). For instance, an INVERTOR, a NAND-gate, and a NOR-gate are defined as follows:

1. INVERTOR

x	0	1
$f_{\neg}(x)$	1	0

2. NAND

x	0	0	1	1
y	0	1	0	1
$f_{\mathrm{NAND}}(x,y)$	1	1	1	0

3. NOR

x	0	0	1	1
y	0	1	0	1
$f_{\mathrm{NOR}}(x,y)$	1	0	0	0

The application of 2-valued Boolean algebra to combinational circuits is very successful. However, this is not necessary true for sequential circuits. It is simply because that 2-valued model excludes the possibility of representing floating voltages or oscillations within the model. The following example illustrates this point.

Consider a NAND-gate with a feedback represented by the equation (see Fig. 1):

$$z=f_{\mathrm{NAND}}(x,z) \tag{1.1}$$

where $f_{\mathrm{NAND}}(x,y)=\neg(x\wedge y)$ (the function f_{NAND} is called the *combinational* function of Eq. (1.1)).

Fig. 1

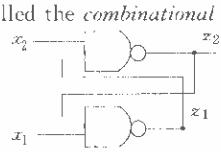

Fig. 2

Assume that the input on line x is 1; then, regardless of the input on line z (either 1 or 0), the output value on line z is always the complement of the input value. Since the input z and the output z are connected, their values should be identical. However, they can be neither 0 nor 1, i.e. the value on line z cannot be represented in a 2-valued model.

To overcome this problem, the Register-and-Transfer technique has been used. However, we suggest that this solution is unsatisfactory and does not correspond to the physics of VLSI circuits.

Another example in uncertain voltages occurs in a flip-flop, see Fig. 2.

Assume that $x_1=x_2=1$. The question arises: what are the values on lines z_1 and z_2?

There are two possibilities for their stable values:

(i) $z_1=0$ and $z_2=1$, or

(ii) $z_1=1$ and $z_2=0$.

Which of these two occurs is completely dependent on the details of the particular circuit (i.e. the implementation or the delay of the individual gates, so-called *racing* hazard). Assuming one or other of the two is too arbitrary. Allowing both possibilities requires the modelling of a complicated *non-determinism* phenomenon. As a compromise, one value to designate uncertainty seems to be a better choice.

Thus, we introduce new values into 2-valued switching theory to preserve the advantage of digital logic in

circuit design and to solve the problem illustrated earlier at the same time.

For our present purpose, the design of VLSI systems, uncertain inputs would eventually lead to uncertain outputs anyway. In this sense, a single extra value $\perp$ (besides 0 and 1) is sufficient. In other words, examples of uncertain voltages such as glitches, oscillations, floating voltages and initial (internal) voltages of flip-flops are considered to be the single value $\perp$. Of course, by introducing more values into switching theory, we can reap some other benefits (see Refs. [1~4] for some references). However, this is outside the main interest of this paper.

We define our semantics domain as a natural extenion of $B=\{0,1\}$, i.e. $B_\perp=B\cup\{\perp\}$ with its usual natural partial order $\sqsubseteq$ (i.e. $\perp\sqsubseteq 0$ and $\perp\sqsubseteq 1$). That is, $x\sqsubseteq y$ means that the certainty of the voltage on y is greater than that on x.

We can extend the partial order $\sqsubseteq$ pointwise to any product of $B_\perp$. The definitions of *monotonic* and *continuous* are as usual:

(a) f is *monotonic* iff $f(x)\sqsubseteq f(y)$ for each pair $x\sqsubseteq y$;

(b) f is *continuous* iff $\lim_n\{f(x_n)\}=f(\lim_n\{x_n\})$ for every (monotonic) sequence $\{x_n\}$.

We now give definitions for an INVERTOR (negation), an AND-gate (conjunction) and an OR gate (disjunction), in the extended $B_\perp$ as follows:

1. negation

x	$\perp$	0	1
$f_\neg(x)$	$\perp$	1	0

2. conjunction

x	$\perp$	$\perp$	$\perp$	0	1	0	0	1	1
y	$\perp$	0	1	$\perp$	$\perp$	0	1	0	1
$f_\wedge(x,y)$	$\perp$	0	$\perp$	0	$\perp$	0	0	0	1

3. disjunction

x	$\perp$	$\perp$	$\perp$	0	1	0	0	1	1
y	$\perp$	0	1	$\perp$	$\perp$	0	1	0	1
$f_\vee(x,y)$	$\perp$	$\perp$	1	$\perp$	1	0	1	1	1

The movitation for these definitions is as follows:

(1′) If for an INVERTOR (negation) its input is uncertain, so is its output.

(2′) If for an AND-gate (conjunction) one of its inputs is 0, and then whatever value its other input has, its output is determined by the input 0, i.e. its output is 0; if one of its two inputs is 1, then its output cannot be determined by this input; we have to know the value of its other input before we know its output with certainty.

(3′) If for an OR-gate (disjunction) one of its two inputs is 1, then whatever value its other input is, its output is determined by the input 1, i.e. its output is 1; if one of its two inputs is 0, and then its output cannot be determined by this input; we have to know the value of its other input before we know its output with certainty.

Note that the conjunction and disjunction operations defined above are not strict (although they are monotonic and continuous) and they together with the above defined negation coincide with the 3-valued logic defined by Kleene[5].

Now, we introduce a general fixpoint operator (see Ref. [6] for further details of general fixpoint operators), instead of the least fixpoint operator (which is commonly used in computation theory[7~9]).

Consider the example (see Fig. 1) as a way of introducing general fixpoint operators. Suppose the external

input $x=1$ and the initial internal input z_0(i.e. the isolated voltage) is 0. Then, the question is: *what is the possible output on line z?*

To obtain a reasonable answer, we have the following analysis:

$$\begin{cases} z_{2n+1}=f_{NAND}(1,z_{2n})=\neg z_{2n}=1, & \text{and} \\ z_{2n+2}=f_{NAND}(1,z_{2n+1})=\neg z_{2n+1}=0. \end{cases}$$

We understand that the sequence of $\{z_n\}$ is *divergent*, written as $\lim_{n\to\infty}\{z_n\}=\perp$. In the physics of VLSI circuits, this divergence indicates that the output voltage of the gate is *oscillating*. However, if $x=z_0=0$, then $z_n=1$ for all $n\geqslant 0$; i.e. the sequence $\{z_n\}$ is *convergent*, written as $\lim_{n\to\infty}\{z_n\}=1$, and we say that when $x=0$,

(i) both 0 and 1 are convergence points of Eq. (1.1); and

(ii) 1 is the fixpoint of Eq. (1.1) under the condition that either $z_0=0$ or $z_0=1$.

Thus,

(a) the output in the first case (i.e. $x=1$) should be the uncertain value, $\perp$, and

(b) the output in the second case (i.e. $x=0$) should be 1.

However, on the other hand, when $x=1$, the least fixpoint of Eq. (1.1) is $\perp$; i.e. the result of the least fixpoint of Eq. (1.1) under $x=1$ is not the result of the actual output on line z in the circuit of Fig. 1. Therefore, we have no choice but to use the general fixpoint operators defined as follows instead of the least fixpoint operator.

Formally, given a (combinational) function $f:\vec{B}_\perp\to\vec{B}'_\perp$ and the initial values $\vec{b}_0\in\vec{B}_\perp$, where $\vec{B}_\perp$ is $B_\perp^m$ for some $m\geqslant 1$ (or $\underbrace{B_\perp\times B_\perp\times\cdots\times B_\perp}_{m\text{ times}}$) and $\vec{B}'_\perp$ is $B_\perp^n$ for some $n\geqslant 1$ (or $\underbrace{B_\perp\times B_\perp\times\cdots\times B_\perp}_{n\text{ times}}$) ($m\geqslant n$), the output of f can be defined:

1. either by the equation

$$\vec{z}=f(x_1,x_2,\ldots,x_m,\vec{z}),$$

where $\vec{z}=(z_1,z_2,\ldots,z_n)$, x_i is the ith external input of f, and z_j is the jth internal input (feedback) of f;

2. or by using the general fixpoint operator $fix_{B_\perp^n}$, i.e. $fix_{B_\perp^n}(curry(f)(x_1,x_2,\ldots,x_{m-n}))(\vec{z}_0)$, where

(a) $curry:(B_\perp^m\to B_\perp^n)\times B_\perp^{m-n}\to B_\perp^n$ is such that

$$curry(f)(x_1,x_2,\ldots,x_{m-n})(z_1,z_2,\ldots,z_n)=f(x_1,x_2,\ldots,x_{m-n},z_1,z_2,\ldots,z_n),$$

(b) $fix_{B_\perp^n}:(B_\perp^n\to B_\perp^n)\times B_\perp^n\to B_\perp^n$ is such that

$$fix_{B_\perp^n}(f)(\vec{b}_0)=\begin{cases} f(\lim_{i\to\infty}\vec{b}_i) & \text{if } f \text{ is convergent at } \vec{b}_0 \\ fix_{B_\perp^n}(f)(\vec{b}_0^*) & \text{otherwise} \end{cases},$$

where $\vec{b}_{i+1}=f(\vec{b}_i)$ for $i\geqslant 0$ and $\vec{b}_0^*=(b_{1,0}^*,b_{2,0}^*,\ldots,b_{n,0}^*)$ with

$$b_{i,0}^*=\begin{cases} \lim_{k\to\infty}b_{i,k} & \text{if } \{b_{i,k}\} \text{ is convergent} \\ \perp & \text{otherwise} \end{cases}.$$

By the well-founded property of the partial order $\sqsubseteq$ in products of $B_\perp$, and by the least fixpoint theorem in CPO (complete partial order) domain theory[7~9], we know that $fix_{B_\perp^n}$ has its value if f is monotonic.

Technologically, we focus our attention on hardware which can be built by one-input and one-ouput INVERTORs, two input and one-output NAND-gates, two-input and one-output NOR-gates; theoretically, we need to show that this collection of basic circuits (or gates) is functionally complete. Unfortunately, this is impossible since Kleene 3-valued logic is well-known to be functionally incomplete. However, in this paper, we show that our model is functionally monotonic complete. In other words, the Kleene 3-valued logic is functionally monotonic complete. This implies that all gates have the following property: *whenever the certainty of input voltages increases, so does the certainty of the output voltages.*

We also present some results which are of theoretical interest, for example, canonical normal forms of Kleene 3-valued logic.

We hope that the results in this paper provide a more realistic and theoretically sound model for hardware (device behaviour) than traditional 2-valued Boolean algebra. The reason is that our 3 valued logic is a sub-logic of both 9-valued logic used in VHDL (when considering $\{X,0,1\}$ out of $\{U,X,0,1,Z,W,L,H,-\}$ provided that $\perp=X$)[10] and 4 valued logic used in Verilog HDL (when restricted to $\{0,1,x\}$ from $\{0,1,x,z\}$ provided that $\perp=x$)[11].

For ease of understanding, we choose a semi-formal presentation in giving our results.

For the moment, we deliberately omit the treatment of shared communication in circuits. For example, we do not consider the case in which two components share an output line. However, we allow two or more components to share one input line.

So far, our results are mainly semantic. We are, however, interested in pursuing this research further but from a different point of view, from the viewpoint of syntactical derivability, since this would enable us to derive secure circuits for actual circuit design.

Mukaidono[12] independently obtained the similar results. He uses the term "*regular*" instead of "*monotonic*". He defines his domain as $V=\{0,\frac{1}{2},1\}$ (i.e. $\frac{1}{2}$ is the third value) and his partial order as $\propto$ (i.e. $0\propto\frac{1}{2}$ and $1\propto\frac{1}{2}$). The partial order $\propto$ is naturally extended pointwisely to any product of V. Mukaidono's *regular* function f is the function satisfying that if $f(A)\in B=\{0,1\}$ then $f(A')=f(A)$ for every A' such as $A'\propto A$. He also defines an *A-ternary logic function* as a *monotonic* function; i.e., his monotonicity is the opposite monotonicity to ours (since $x\propto y$ iff $y\sqsubseteq x$ provided that $\perp=\frac{1}{2}$). He observed that his regularity was equivalent to his A-ternary function (i.e. his monotonicity) at the end of Section 3 of Ref. [12] but he never went on to formally prove this observation. He obtained his regular functional completeness and his canonical forms in Sections 3 and 4 of Ref. [12], respectively. His completeness is obtained by a semantical method in contrast with his syntactical method for obtaining his canonical forms. However, his approach is not as coherent (or systematic) as the one presented in this paper. By the way, Yamamoto[13] and Hata *et al.*[14] extended Mukaidono's work further.

Lastly, we conclude this section by an *assumption* or a hypothesis, which is presumed throughout this paper.

Assumption 1.1. We assume that for any hardware (device or circuit), all of the initial values on its feedback lines are $\perp$s.

2 Some Properties of Kleene 3-Valued Logic

In this section, we briefly state some useful properties of Kleene 3-valued logic, which will be used later.

We shall use infix notations $\neg$ (negation), $\wedge$ (conjunction), and $\vee$ (disjunction) instead of prefix notations $f_\neg$, $f_\wedge$ and $f_\vee$ respectively. For example, we use

$$f_t(x)=x\vee\neg x$$

instead of

$$f_t(x)=\circ(f_\vee,\circ(\otimes(\pi,f_\neg),diag))(x),$$

where

(i) $\circ$ is the usual compositional functional (function);

(ii) $\otimes$ is the prefix notation for the usual $\langle f,g\rangle$ functional;

(iii) *diag* is the usual diagonalizer; and

(iv) π is the usual projection function.

Below are some easy-verified laws and facts of Kleene 3-valued logic.

Law 2.1.

1. Negative negation:

$$\neg\neg x=x.$$

2. $\wedge$ Kleene:

$$(x\wedge\neg x)\wedge(y\vee\neg y)=x\wedge\neg x.$$

3. $\vee$ Kleene:

$$(x\wedge\neg x)\vee(y\vee\neg y)=y\vee\neg y.$$

Fact 2.2.

1. $x_1\wedge x_2\wedge\ldots\wedge x_m=0$ iff for some i $x_i=0$;
2. $x_1\wedge x_2\wedge\ldots\wedge x_m=1$ iff for all i $x_i=1$;
3. $x_1\vee x_2\vee\ldots\vee x_m=0$ iff for all i $x_i=0$;
4. $x_1\vee x_2\vee\ldots\vee x_m=1$ iff for some i $x_i=1$;
5. $x_1\wedge x_2\wedge\ldots\wedge x_m=\perp$ iff for all i $x_i\neq 0$, and for some j $x_j=\perp$;
6. $x_1\vee x_2\vee\ldots\vee x_m=\perp$ iff for all i $x_i\neq 1$, and for some j $x_j=\perp$.

Obviously, we have the following.

Theorem 2.3. Any hardware with or without feedbacks is said to be unfictitious; or in other words, for any hardware (circuit), whether it is a (combinational) function f (i.e. without feedbacks) or it is represented by recursive equations like Eq. (1.2) in Section 1 (i.e. with feedbacks), if all of its inputs (including all the initial values of its feedbacks) are $\perp$, then its outputs are $\perp$ also.

Proof. By structural induction. □

This theorem states that, if the inputs of a device f are undefined, then its outputs are. In other words, if the input cannot be deterministicly identified as either 0 or 1, then the output of it cannot be, either. So, in order to increase the reliability of circuits (especially of sequential networks), a technology or theory must be provided to reduce the asynchronousness of inputs (or to guarantee their synchronization)[15~18].

3 Preliminary Theorems

In order to understand better the proof for the main theorem (Theorem 4.1), and also to understand better the nature of hardware feedbacks, we provide some preliminary results in this section.

We say that the inputs of a hardware are *definite* if they are deterministically identified as either 0 or 1.

Theorem 3.1 (combinational theorem). For any combinational hardware (circuit), i.e. for any (combinational) function f, if all of its inputs are definite, then all of its outputs are definite, also.

Proof. By structural induction □

From Section 1, we know that circuits using negation, conjunction and disjunction cannot prevent the uncertain value $\perp$ from being propagated in it. Hence, a question arises:

"Can a circuit generate the uncertain value $\perp$ at its output if all of its inputs are always definite?"

We have already indicated in Section 1 that the answer to the question is positive (see Fig. 1). Circuits do create the uncertain value. Even worse, there is a circuit whose outputs are the uncertain value regardless of its inputs. This is exemplified by the constant function f_1 in the proof of Theorem 3.2 below (see Fig. 3). We refer to this type of function as an *indeterminate* function (or *nonsense* function). This indicates how bad a careless

design may be.

Theorem 3.2 (indeterminate lemma). There is an indeterminate function.

Proof.

Let $f_N(x,y)=\neg((x\vee\neg x)\wedge y)$ and

$$z=f_N(x,z), \tag{3.1}$$

i.e.

$$z=fix_{B_\perp}(curry(f_N)(x))(z_0), \tag{3.1'}$$

where the linearizer *curry* is a construction functional and z_0 is the initial value of line z. Whatever the initial value z_0 and the input x are, the ouput on line z is always $\perp$. □

The logical circuit of the indeterminate device (or function) given in the proof of Theorem 3.2 is shown in Fig. 3.

Fig. 3

Since the output on line z specified in Eq. (3.1) does not rely on either the input x or the initial value z_0, we can simply regard the device as a constant function $f_\perp(x)$, or simply $\perp$.

Although indeterminate devices are undesirable in hardware implementation, they play an important role in their theory. This is demonstrated in Theorem 3.3 below, i.e. for the unary functions, we have *strict* completeness.

Theorem 3.3 (unary strict completeness). All strict unary functions (or devices or circuits) can be built from $\{\neg,\wedge,\vee\}$.

Proof. We use truth-tables to define all strict unary functions (totally 9)

$$f_\perp,f_0,f_1,f_2,f_3,f_4,f_5,f_6,f_7$$

as shown below:

x	$f_\perp$	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	$\perp$	0	1	0	0	1	1
1	$\perp$	0	1	$\perp$	$\perp$	0	1	0	1

We can implement these functions as follows:

1. $f_\perp(x)$ is the same as in the proof of Theorem 3.2 (see Fig. 3)
2. $f_0(x)=\neg x\vee f_\perp(x)$
3. $f_1(x)=x\vee f_0(x)$
4. $f_2(x)=\neg(\neg x\vee f_1(x))$
5. $f_3(x)=f_1(\neg x)$
6. $f_4(x)=x\wedge\neg x$
7. $f_5(x)=x$
8. $f_6(x)=\neg x$
9. $f_7(x)=x\vee\neg x$. □

Combining the unary constant functions 0 and 1 with the functions listed in the proof of Theorem 3.3, we have the unary functional completeness in monotonic sense. In the remainder of the paper, we will see the sig-

nificant roles played by the first five unary functions $\{f_1, f_0, f_1, f_2, f_3\}$, especially the first three.

Theorem 3.4 (definite completeness or hardware completeness). If the inputs of a hardware device are restricted to definite values, then for any (combinational) function f it can be built from $\{\neg, \wedge, \vee\}$.

Proof. For a given assignment $\langle b_1, b_2, \ldots, b_m\rangle$ to $\langle x_1, x_2, \ldots, x_m\rangle$, which satisfies

$$f(x_1, x_2, \ldots, x_m) = 1,$$

we choose

1. $\#_i = \varepsilon$ (empty list) if $b_i = 1$, and

2. $\#_i = \neg$ if $b_i = 0$

and construct a formula

$$\#_1 x_1 \wedge \#_2 x_2 \wedge \ldots \wedge \#_m x_m, \tag{3.2}$$

corresponding to the m argument inputs x_is.

For all (3.2)s, we construct a formula

$$(3.2)_1 \vee (3.2)_2 \vee \ldots \vee (3.2)_t. \tag{3.3}$$

We understand that

1. $f(x_1, x_2, \ldots, x_m) = 1$ iff (3.3) $= 1$;

2. if $f(x_1, x_2, \ldots, x_m) = 0$ then (3.3) $= 0$ (not the converse); hence, all (3.2)s are 0.

This means that the set of inputs which satisfy

$$(3.2) = 0$$

is a superset of the inputs which satisfy

$$f(x_1, x_2, \ldots, x_m) = 0.$$

Therefore, we have to drop some of the latter inputs off. The technique for doing so is similar to the above.

For a given assignment $\langle b_1, b_2, \ldots, b_m\rangle$ to $\langle x_1, x_2, \ldots, x_m\rangle$, which satisfies

$$f(x_1, x_2, \ldots, x_m) = 0,$$

we choose

1. $\#_i = \neg$ if $b_i = 0$, and

2. $\#_i = \varepsilon$ if $b_i = 1$

and construct a formula

$$\#_1 x_1 \wedge \#_2 x_2 \wedge \ldots \wedge \#_m x_m, \tag{3.4}$$

and then construct a formula (3.5) from all these (3.4)s

$$(3.4)_1 \vee (3.4)_2 \vee \ldots \vee (3.4)_k. \tag{3.5}$$

Now, we know that

$$f(x_1, x_2, \ldots, x_m) = 0 \text{ iff } (3.5) = 1.$$

Hence, we have

$$f = (3.3) \vee f_0((3.5)). \qquad \square$$

From the above theorems, it follows that there are two types of hardware: *combinational* vs *sequential*. One of the types always has definite output whereas the other doesn't, even when the inputs are definite. The devices in the latter type probably have uncertain output value, $\perp$, because of feedbacks.

Since we can understand any sequential circuit (i.e. hardware with feedbacks) by its combinational function, it is better to ignore the feedbacks and concentrate our attention on combinational functions. For this purpose, we only need to extend the basic function set from $\{\neg, \wedge, \vee\}$ to $\{\neg, \wedge, \vee, f_\perp, f_0, f_1, f_2, f_3\}$. By this means, we do not increase the function space available but we increase the expressive power. From now on, we

will assume this and the will not consider feedbacks further in this sense.

4 Monotonic Functional Completeness

We present now the main theorem (Theorem 4.1) of this paper together with its proof.

***Theorem 4.1* (monotonic functional completeness).** Any monotonic function can be constructed from the basic function set $\{\neg, \wedge, \vee\}$.

From the comment at the end of Section 3, we know that the basic building blocks are in $\{\neg, \wedge, \vee, f_{\perp}, f_0, f_1, f_2, f_3\}$.

First, let us introduce some notations to facilitate our presentation. Note that we limit our attention to monotonic functions only in this and the following sections (i.e. Sections 4 and 5). Also, we will give a proof based on one-output devices. There is no difficulty in extending the proof to more general case of multi-output devices.

Given $f(x_1, x_2, \ldots, x_m)$ and $\vec{b}=(b_1, b_2, \ldots, b_m) \in B_1^m$, let

1. $V(f)=\{\vec{b} \mid f(\vec{b}) \neq \perp\}$, the set of inputs for which f yields definite outputs;
2. $V_0(f)=\{\vec{b} \mid f(\vec{b})=0\}$, the set of inputs for which f yields the output 0; and
3. $V_1(f)=\{\vec{b} \mid f(\vec{b})=1\}$, the set of inputs for which f yields the output 1.

Clearly,

(a) $V(f)=V_0(f) \bigcup V_1(f)$;

(b) $V_0(f) \bigcap V_1(f)=\varnothing$, where $\varnothing$ means an empty set.

For a sub-domain V' of V, V' is said to be *up-closed* if $a \in V'$, $b \in V$ and $a \sqsubseteq b$, then $b \in V'$. Naturally, by monotonicity, we have

(i) $V_0(f)$ is up-closed;

(ii) $V_1(f)$ is up-closed; and

(iii) $V(f)$ is up-closed.

Given $\vec{b}$ where $b_i \neq \perp$ (i.e. $\vec{b}=(\perp, \ldots, \perp, b_1, \perp, \ldots, \perp, b_2, \perp, \ldots, \perp, b_k, \perp, \ldots, \perp)$), let $b^*=(b_1^*, b_2^*, \ldots, b_m^*)$, and we define

1.

$$V_{10}(f)(\vec{b})=\left\{b^* \;\middle|\; \begin{array}{l} f(\vec{b})=1 \text{ and } b^* \text{ satisfies} \\ \#_{i_1} x_{i_1} \wedge \#_{i_2} x_{i_2} \wedge \ldots \wedge \#_{i_k} x_{i_k}=0 \end{array}\right\},$$

where $\#_{i_j}$ is either $\neg$ or ε (for all $j=1, 2, \ldots, k$) according to whether $b_{i_j}=0$ or $b_{i_j}=1$ respectively; and the formula $\#_{i_1} x_{i_1} \wedge \#_{i_2} x_{i_2} \wedge \ldots \wedge \#_{i_k} x_{i_k}$ is obtained as explained in (3.2) (and (3.4));

2.

$$V_{00}(f)(\vec{b})=\left\{b^* \;\middle|\; \begin{array}{l} f(\vec{b})=0 \text{ and } b^* \text{ satisfies} \\ \#'_{i_1} y_{i_1} \wedge \#'_{i_2} y_{i_2} \wedge \ldots \wedge \#'_{i_k} y_{i_k}=0 \end{array}\right\},$$

where $\#'_{i_j}$ is either $\neg$ or ε (for all $j=1, 2, \ldots, k$) according to whether $b_{i_j}=0$ or $b_{i_j}=1$ respectively; and the formula $\#'_{i_1} y_{i_1} \wedge \#'_{i_2} y_{i_2} \wedge \ldots \wedge \#'_{i_k} y_{i_k}$ is obtained as explained in (3.4) (and (3.2)).

Then, we have that $V_{10}(f)(\vec{b})$ is up-closed (the proof is based on monotonicity of f). Similarly, $V_{00}(f)(\vec{b})$ is up-closed.

Lemma 4.2. For a given $\vec{b}$, we have that

1. if $f(\vec{b})=1$, then

$$V_0(f) \subseteq V_{10}(f)(\vec{b});$$

2. if $f(\vec{b})=0$, then

$$V_1(f)\subseteq V_{00}(f)(\vec{b}).$$

Note that the above lemma shows that the corresponding formula $\#_{i_1}x_{i_1}\wedge\#_{i_2}x_{i_2}\wedge\ldots\wedge\#_{i_k}x_{i_k}$ of $\vec{b}$ may collect more 0s in $V_{10}(f)$s than in $V_0(f)$. Also, it shows that the corresponding formula $\#'_{i_1}y_{i_1}\wedge\#'_{i_2}y_{i_2}\wedge\ldots\wedge\#'_{i_k}y_{i_k}$ of $\vec{b}$ may collect more 1s in $V_{00}(f)$s than in $V_1(f)$.

Proof of Lemma 4.2. We will prove the first half of Lemma 4.2. The second half can be similarly derived.

Suppose $b_{i_j}\neq\perp$ in $\vec{b}$ (i.e. $\vec{b}=(\perp,\ldots,\perp,b_{i_1},\perp,\ldots,\perp,b_{i_2},\perp,\ldots,\perp,b_{i_k},\perp,\ldots,\perp)$). From $f(\vec{b})=1$, we know that $\vec{b}'\in V_0(f)$ means $\vec{b}'$ satisfies

$$\#_{i_1}x_{i_1}\wedge\#_{i_2}x_{i_2}\wedge\ldots\wedge\#_{i_k}x_{i_k}=\begin{cases}0\\ \perp\end{cases}.$$

We need to show that it is impossible for $\vec{b}'$ to satisfy $\#_{i_1}x_{i_1}\wedge\#_{i_2}x_{i_2}\wedge\ldots\wedge\#_{i_k}x_{i_k}=\perp$.

Let us assume to the contrary, i.e. $\vec{b}'$ satisfies $\#_{i_1}x_{i_1}\wedge\#_{i_2}x_{i_2}\wedge\ldots\wedge\#_{i_k}x_{i_k}=\perp$. By item 5 in Fact 2.2, we know that all $\#_{i_j}b'_{i_j}=1$ except for some (at least one) $\#_{i_j}b'_{i_j}=\perp$. That is,

$$(b'_{i_1},b'_{i_2},\ldots,b'_{i_k})\subseteq(b_{i_1},b_{i_2},\ldots,b_{i_k}).$$

On the other hand, let $\vec{b}'\uparrow_{i_1,i_2,\ldots,i_k}$ be a sub-part of $\vec{b}'$ by taking off elements in $i_1,i_2,\ldots,i_k$ positions, i.e.

$$\vec{b}'\uparrow_{i_1,i_2,\ldots,i_k}=\begin{pmatrix}b'_1,\ldots,b'_{i_1-1},\\ b'_{i_1+1},\ldots,b'_{i_2-1},\\ b'_{i_2+1},\ldots,b'_{i_k-1},\\ b'_{i_k+1},\ldots,b'_m\end{pmatrix}$$

and similarly $\vec{b}\uparrow_{i_1,i_2,\ldots,i_k}$ be a sub-part of $\vec{b}$ by taking off elements in $i_1,i_2,\ldots,i_k$ positions, i.e.

$$\vec{b}\uparrow_{i_1,i_2,\ldots,i_k}=\begin{pmatrix}b_1,\ldots,b_{i_1-1},\\ b_{i_1+1},\ldots,b_{i_2-1},\\ b_{i_2+1},\ldots,b_{i_k-1},\\ b_{i_k+1},\ldots,b_m\end{pmatrix},$$

then we obviously have $(\perp,\perp,\ldots,\perp)=\begin{pmatrix}b_1,\ldots,b_{i_1-1},\\ b_{i_1+1},\ldots,b_{i_2-1},\\ b_{i_2+1},\ldots,b_{i_k-1},\\ b_{i_k+1},\ldots,b_m\end{pmatrix}\subseteq\begin{pmatrix}b'_1,\ldots,b'_{i_1-1},\\ b'_{i_1+1},\ldots,b'_{i_2-1},\\ b'_{i_2+1},\ldots,b'_{i_k-1},\\ b'_{i_k+1},\ldots,b'_m\end{pmatrix}$.

Therefore, let

$$\vec{b}^*=\begin{pmatrix}b'_1,\ldots,b'_{i_1-1},b_{i_1},\\ b'_{i_1+1},\ldots,b'_{i_2-1},b_{i_2},\\ b'_{i_2+1},\ldots,b'_{i_k-1},b_{i_k},\\ b'_{i_k+1},\ldots,b'_m\end{pmatrix},$$

and we have both $\vec{b}'\subseteq\vec{b}^*$ and $\vec{b}\subseteq\vec{b}^*$.

Now, by $\vec{b}'\subseteq\vec{b}^*$, we have $f(\vec{b}^*)=f(\vec{b}')=0$ because of function f's monotonicity. On the other hand, by $\vec{b}\subseteq\vec{b}^*$, we have $f(\vec{b}^*)=f(\vec{b})=1$ because of function f's monotonicity. Thus, function f has both values of 0 and 1 at $\vec{b}^*$. This is impossible. □

Now, we are ready to prove the main theorem (Theorem 4.1).

Proof of Theorem 4.1. For a one-output monotonic f:

1. If all output values of f are $\perp$s, then

$$f(x_1,x_2,\ldots,x_m)=f_\perp(x_1\vee x_2\vee\ldots\vee x_m).$$

2. If at least one of the outputs of f is not $\perp$, then there are two cases.

(a) For each b_i in $\vec{b}$ which satisfies

$$f(x_1,x_2,\ldots,x_m)=1,$$

we choose

i. x_i and $\#_i=\neg$ if $b_i=0$,

ii. x_i and $\#_i=\varepsilon$ if $b_i=1$, and

iii. ε (i.e. nothing) if $b_i=\perp$

and construct a formula

$$\#_{i_1}x_{i_1}\wedge\#_{i_2}x_{i_2}\wedge\ldots\wedge\#_{i_k}x_{i_k}. \tag{4.0}$$

This may be simplified to

$$\#_1x_{i_1}\wedge\#_2x_{i_2}\wedge\ldots\wedge\#_kx_{i_k}, \tag{4.1}$$

Later, we will assume this type of simplification without comment.

We construct a formula (4.2) from the instances of (4.1)s

$$(4.1)_1\vee(4.1)_2\vee\ldots\vee(4.1)_r. \tag{4.2}$$

(b) For each b_i in $\vec{b}$ which satisfies

$$f(x_1,x_2,\ldots,x_m)=0,$$

we similarly choose

i. x_i and $\#_i=\neg$ if $b_i=0$,

ii. x_i and $\#_i=\varepsilon$ if $b_i=1$, and

iii. *nothing* if $b_i=\perp$

and construct a formula

$$\#_1x_{i_1}\wedge\#_2x_{i_2}\wedge\ldots\wedge\#_kx_{i_k}. \tag{4.3}$$

We construct a formula (4.4) from the instances of (4.3)s

$$(4.3)_1\vee(4.3)_2\vee\ldots\vee(4.3)_k. \tag{4.4}$$

3. We discuss the cases above as follows:

(a) if there is no case for which $f(x_1,x_2,\ldots,x_m)=1$, then

$$f(x_1,x_2,\ldots,x_m)=f_0((4.4));$$

(b) if there is no case for which $f(x_1,x_2,\ldots,x_m)=0$, then

$$f(x_1,x_2,\ldots,x_m)=f_1((4.2));$$

(c) if both cases occur, we know that

$$V_0(f)\subseteq V_{10}(f)((4.1)'),$$

where $(4.1)'$ is the input value corresponding to a choice from among the (4.1)s.

Therefore, we have

$$V_0(f)\subseteq\bigcap_j V_{10}(f)((4.1)'_j).$$

Later, we simplify this intersection as $V_{10}(f)$.

From the above containment, we have

i. $f(x_1,x_2,\ldots,x_m)=1$ iff $(4.2)=1$;

ii. if $f(x_1,x_2,\ldots,x_m)=0$ then $(4.2)=0$.

In other words, (4.2) probably has more 0s as its output values than necessary. So, these 0s are dropped off as follows:

$$(4.2) \vee f_0((4.4)). \tag{4.5}$$

From this, we have

$$f(x_1, x_2, \ldots, x_m) = (4.5). \qquad \square$$

This theorem asserts that any monotonic combinational function (or device) can be built from $\{\neg, \wedge, \vee\}$. From the proof, we can obtain the *canonical forms* of Kleene 3-valued logic.

5 Canonical (Normal) Forms

From the proof of Theorem 4.1, we observe a possible way of obtaining the canonical normal forms of the Kleene system. From Section 4, let $\vec{x} = (x_1, x_2, \ldots, x_m)$, we have the following:

5.1

$$\begin{cases} (4.2) = 1 & \text{iff } f(\vec{x}) = 1 \\ \text{if } f(\vec{x}) = 0 & \text{then } (4.2) = 0 \end{cases}.$$

and

5.2

$$\begin{cases} (4.4) = 1 & \text{iff } f(\vec{x}) = 0 \\ \text{if } f(\vec{x}) = 1 & \text{then } (4.4) = 0 \end{cases}.$$

So, we can have

$$f(\vec{x}) = (4.2) \vee f_0((4.4)).$$

Choosing a different notation, we write

5.3　　$$f(\vec{x}) = (C.1) \vee f_0((C.2)),$$

where (C.1) and (C.2) are (4.2) and (4.4) respectively. Using a similar renaming, we can have

5.4

$$\begin{cases} (C.3) = 0 & \text{iff } f(\vec{x}) = 0 \\ \text{if } f(\vec{x}) = 1 & \text{then } (C.3) = 1 \end{cases}.$$

and

5.5

$$\begin{cases} (C.4) = 0 & \text{iff } f(\vec{x}) = 1 \\ \text{if } f(\vec{x}) = 0 & \text{then } (C.4) = 1 \end{cases}.$$

Then, we have the following:

5.6　　$$f(\vec{x}) = (C.3) \vee f_1((C.1)).$$

5.7　　$$f(\vec{x}) = (C.1) \vee f_2((C.3)).$$

5.8　　$$f(\vec{x}) = (C.3) \vee f_3((C.4)).$$

That is, (5.3), (5.6), (5.7) and (5.8) can be our candidates for the canonical form(s). However, if we choose them as the canonical forms, there is a disadvantage, i.e. all of them require the function f to have both 0 and 1 as its output values (each value at least appears once). This disadvantage forces us to exclude certain functions from the above choices of the canonical forms. This is not acceptable and so we seek an alternative. Similar to the procedure used to obtain (5.1), (5.2), (5.4) and (5.5), we can obtain the following:

1.　　$(C.5) = 1$　iff $f(\vec{x})$ = either 1 or 0

2.　　$(C.6) = 0$ iff $f(\vec{x})$ = either 1 or 0.

Then, we have the following:

5.9　　$$f(\vec{x}) = (C.1) \vee f_0((C.5)).$$

5.10　　$$f(\vec{x}) = (C.3) \wedge f_1((C.5)).$$

5.11 $$f(\vec{x}) = (C.1) \vee f_2((C.6)).$$

5.12 $$f(\vec{x}) = (C.3) \wedge f_3((C.6)).$$

Now, it appears that we can choose (5.9), (5.10), (5.11) and (5.12) as the candidates for the canonical forms. However, although they have improved the situation, these choices require the function f to have a definite output value (at least once). So, by choosing them, we have to exclude certain functions (say $f_\perp$) from the canonical forms. There is no reason for us to do so. Therefore, we need to improve the situation further as follows:

$$(C.7) = (f_\perp \wedge (C.6)) \vee (C.1),$$

and

$$(C.8) = (f_\perp \vee (C.5)) \wedge (C.3),$$

where $f_\perp$ is treated as a constant. Whatever the arguments of $f_\perp$ are, they would not make a difference. For example, we can let it be either $f_\perp(x_1 \vee x_2 \vee \ldots \vee x_m)$ or $f_\perp(x_1 \wedge x_2 \wedge \ldots \wedge x_m)$. Now, we have arrived at the ideal canonical forms; viz. (C.7) and (C.8). Before we accept them, we must make sure that they are unique according to certain criteria. We proceed as follows.

This uniqueness is assured by choosing the minimals in $V(f)$, $V_0(f)$, $V_1(f)$, $V_{10}(f)$ and $V_{00}(f)$, i.e. we can obtain the corresponding (C.6), (C.5), (C.1) and (C.3) from the minimals by monotonicity of f, where a minimal element in V' is the element in V' such that there is no other distinctive element $\sqsubseteq$ the minimal element.

We refer to (C.7) and (C.8) as *disjunctive* and *conjunctive* canonical (normal) forms of Kleene 3-valued logic, respectively.

In summary, we have Theorem 5.1.

Theorem 5.1 (normal forms). Any combinational (one-output) device (function) has unique conjunctive and disjunctive canonical normal forms.

6 Conclusion

In this paper, we have applied the CPO domain theory[7~9] from denotational semantics of programming languages to the logical design of VLSI circuits. Our approach is model-oriented. It will also be useful if we can develop a syntax-oriented version, i.e. going from the formula of Kleene 3-valued logic to VLSI circuits at gate-level (via EDIF[19] or VHDL[20] or Verilog HDL[11]).

There are some issues which we would like to touch upon briefly. There are a number of ways to introduce environments into the semantic model. In the denotational approach, we can introduce *Env* to represent environments, say *Env*: *Line*→*Voltage*. Thus, a hardware device h can be a function from *Input*×*Env* to *Output*×*Env*. But by doing so, we would lose the vigorousness of the digital logic. An alternative is to seek a way to represent environments in the equational style of this paper so that we can preserve the vigorousness of the digital logic in logical design of VLSI circuits.

Since the work presented here is based on CPO domain theory[7~9], there may be a perspective in our present approach to use formal methods in cross-fertilization between software design and hardware design (or co-design, hybrid system design) such as various research lines presented in Refs. [21~24].

Acknowledgement

We would like to express our gratefulness to M. Clint for his constructive comments on an early version of this paper. Also, the first author would like to gratefully acknowledge that G D Plotkin brought the issue of the canonical forms into his attention when he had an opportunity to discuss the results with G D Plotkin.

References

1 Epstein G, Frieder G, Rine D C. The development of mutiple-valued logic as related to computer science. Computer, 1974,7(9)

2 Smith C K. The prospects for multi-valued logic: a technology and applications view. IEEE Transactions on Computers, 1981,C-30(9)

3 Hurst S L. Multiple-valued logic——its status and future. IEEE Transactions on Computers, 1984,C-33(12)

4 Rich P A. A survey of multi-valued memories. IEEE Transactions on Computers, 1986,C-35(2)

5 Kleene S C. Introduction to metamathematics. Amsterdam, Groningen: North-Holland Publisher, 1952

6 Istrătescu V I. Fixed point theory: an introduction. In: Reidel D ed. Mathematics and Application, 1981.7

7 Plotkin G D. A powerdomain construction. SIAM Journal of Computing, 1976,5:452～487

8 Scott D. Data types as lattices. SIAM Journal of Computing, 1976,5:522～587

9 Plotkin G D. T^{ω} as a universal domain. Journal of Computer and System Sciences, 1978,17:209～236

10 IEEE. IEEE standard multivalue logic system for VHDL model interoperability. IEEE Standard 1164～1993, 1993

11 IEEE, Verilog hardware description language reference manual (LRM). IEEE Draft Standard 1364, April 1995

12 Mukaidono M. Regular Ternary Logic Functions——Ternary Logic Functions Suitable for Treating Ambiguity. IEEE Transactions on Computers, 1986,C-35(2):179～183

13 Yamamoto Y, Mukaidono M. Meaningful special classes of ternary logic functions——regular ternary logic functions and ternary majority functions. IEEE Transactions on Computers, 1988,37(7):799～806

14 Hata Y, Nakashima K, Yamato K. Some fundamental properties of multiple-valued Kleenean functions and determination of their logic formulas. IEEE Transactions on Computers, 1993,42(8):950～961

15 Chaney T J, Charles E M. Anomalous behaviour of synchronizer and abiter circuits. IEEE Transactions on Computers, 1973,C-22(4)

16 Couranz G R, Wann D F. Theoretical and experimental behaviour of synchronizers operating in the metastable region. IEEE Transactions on Computers, 1975,C-24(6)

17 Marino L R. The effect of asynchronous inputs on sequential network reliability. IEEE Transactions on Computers, 1977, C-26(11)

18 Marino L R. General theory of metastable operation. IEEE Transactions on Computers, 1981,C-30(2)

19 EIA (Electronic Industries Association). Electronic design interchange format version 3 0 0, Electronic Industries Association, December 1993

20 Lipsett R. VHDL: hardware description and design. Kluwer Academic Publishers, 1989

21 Bochman G. Hardware specification with temporal logic. IEEE Transactions on Computers, 1982,C-31(3):223～231

22 Hoare C A R, Gordon M J C *et al*. Mechanized reasoning and hardware design. New York, London: Prentice-Hall, 1992

23 Gordon M. The semantic challenge of Verilog HDL (an invited paper). in the proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science (LICS'95). IEEE Computer Society Press, San Diego, California, USA, 1995

24 Kloos C D, Breuer P T *et al*. Formal semantics for VHDL. Kluwer Academic Publishers, March 1995

论域理论在超大规模集成电路逻辑设计上的运用

孙踊[1]　胡易[2]

[1](贝尔法斯特女王大学计算机科学系　英国)

[2](硅集成化系统公司　英国)

摘要 认为传统的二值布尔不利于大规模集成电路的设计,尤其是在逻辑门电路上.为此引入了三值逻辑.此三值逻辑是基于集成电路的物理性质,且碰巧等同于 Kleene 的三值逻辑.鉴于 Kleene 三值逻辑的不完备性,文章将论域理论以及普通不动点算子运用于此,使三值逻辑获得此逻辑系统的单调完备性定理.文章认为这个结果有利于集成电路设计的可靠性,具有广阔的应用前景.

关键词 大规模集成电路,逻辑门,Kleene 三值逻辑,全半序,普通不动点算子,单调性.

中图法分类号 TP302

中国计算机学会(CCF)2000年部分学术活动计划

会议名称及内容	地点	时间	主办单位	联系人及地址
2000年全国体系结构学术年会	哈尔滨	8月15～16日	体系结构专委	李振洲,哈工大计算机系,150001;Tel.:(0451)6416614
全国人工智能联合学术会议	北京	8月15～17日	人工智能与模式识别专委	怀进鹏,北京航空航天大学计算机系,100083
2000年全国计算机新科技与继续教育学术会议	乌鲁木齐	8月21～23日	教育专委	张凤祥,武汉市东湖路160号,430077;Tel.:(027)86772638;E-mail:cceic@public.wuhan.cngb.com
全国第3届Java技术及应用学术交流会	北京	8月22～23日	计算机应用专委	贾志梅 张兆新,北京市927信箱,100083;Tel.:(010)62327331－115;E-mail:jiazm@infopo.lshs.ac.cn
全国信息保密学术年会	大连	9月11～15日	信息保密专委	杜虹,北京513信箱,100031;Tel.:(010)83086093
第11届全国CAD与CG学术年会	成都	9月25～27日	CAD&CG专委	刘锦德 刘乃琦,成都电子科技大学计算机学院,610054
第17届全国数据库学术会议	保定	10月10～12日	数据库专委	李建中,哈尔滨工业大学计算机系,150001;Tel.:(0451)6683459
2000年全国理论计算机科学学术年会	重庆	10月12～15日	理论计算机科学专委	张为群,西南师大计算机学院,400715
2000年全国测试学术会议	北京	10月15～18日	容错计算专委	李晓维,Tel.:(010)62751792;E-mail:lxw@cs.pku.edu.cn
职业教育专委第3届年会	青岛	10月17～20日	职教专委	吴清萍,北京宝钞胡同21号财经学校,100009;Tel.:(010)64075279
第9届全国多媒体技术学术会议	北京	10月18～20日	多媒体专委	钟玉琢,清华大学计算机系,100084;Tel.:(010)62784141
第11届全国信息存储技术学术会议	北京	10月25～27日	信息存储专委	唐志敏,中科院计算所,100080;Tel.:(010)62559641;E-mail:tang@chpc.ict.ac.cn
第9届全国多值逻辑与模糊逻辑学术会议	成都	12月10～14日	多值逻辑专委	徐扬,Tel.:(028)7600760
第4届亚洲高性能计算机会议	北京	5月14～17日	CCF	徐志伟,北京2704信箱中科院计算所,100080;Tel.:(010)62617948;E-mail:zxu@gatekeeper.ncic.ac.cn
第15届世界信息安全会议	北京	8月21--25日	CCF	卿斯汉,北京8718信箱中科院软件所,100080;Tel.:(010)62635150;E-mail:qsh@sun.ihep.ac.cn