Logical Design of VLSI Circuit with Extension of Uncertainty (or monotonic functional completeness of Kleene ternary logic)

by

Sun, Yong

LFCS Report Series

ECS-LFCS-89-95
(also published as CSR-313-89)

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

November 1989

Logical Design of VLSI Circuit with Extension of Uncertainty (or monotonic functional completeness of Kleene ternary logic)

Sun, Yong*

briefly revised October 1989†

Brief Overview

We consider that traditional 2-valued boolean algebra is not sufficient in representing VLSI circuit (say at gate-level), although it seems to be the case for combinational circuit. Instead of introducing the common technique of register-and-transfers to represent sequential circuit, we seek an alternative which is to define an *invertor*, a *nor-gate* and a *nand-gate* according to actual circuit. We identify all uncertain voltages as one, which is denoted as $\perp$ (bottom or uncertain voltages). Other voltages with certainty are, as usual, denoted definitely as 0 (low voltage or ground) and 1 (high voltage or power). The resulting logic turns out to coincide with Kleene 3-valued logic with generalized fixed-point operators.

Unfortunately, Kleene 3-valued logic is functionally incomplete. This means that not every gate can be constructed from invertors, nor-gates and nand-gates if gates are viewed to be equivalent to functions from their inputs to outputs. However, by applying cpo domain theory [Plotkin 85] to our derived 3-valued logic system, we discover that this system is functionally monotonic complete. This implies that all constructible gates share the following property : *whenever the certainty of input voltages of a gate increases, so does the certainty of output voltages of the gate.* As by-products, we also obtain two main facts about logical design of VLSI circuit.

(a) For any gate, if its inputs voltages are uncertain, then we can not expect to be certain about its output voltages; and

(b) even if we can always supply voltages to inputs of every gate with great certainty, there exists a constructible gate which output voltages are always with uncertainty.

So far, out results are mainly semantical ones. We are very interested in pursuiting this research further but from a different point of view, i.e. from the point of view of syntactical derivability, since such research would enable us to derive mathematical secure circuit with more reality to actual circuit design. Recent Avron's work can be regarded as a few step toward this direction [Avron 87, Avron 88]. We should also mention that Mukaidono has independently obtained the same result in [Mukaidono 86], although his approach is not as coherent as ours.

*LFCS, Department of Computer Science, University of Edinburgh, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, U.K. E-mail : *sun%ed.lfcs@nsfnet-relay.ac.uk*

†The result of this report was obtained in February 1986. The present report is a revision of previous working note, which was once presented in Edinburgh VLSI group's workshop in Firbush 1986.

Contents

1 Introduction to Uncertainty and General Fixpoint Operators

In tradition, we use 2-valued boolean functions to represent VLSI circuit at gate-level. 0 represents the low voltage (GROUND) and 1 stands for 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
$f_{NAND}(x, y)$	1	1	1	0

3. NOR

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

It was very successful for combinational circuit, i.e. if we know inputs of a combinational circuit, we can easily obtain the corresponding output of the circuit. However, this is not necessary true for sequential circuits. It is simply because that 2-valued semantics model completely rules out the possibility of representing floating voltages or oscillations inside the model. A tricky example will illuminate this point. Suppose that there is an NAND-gate with a feedback which is represented by following equation (see figure 1.1) :

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

where $f_{NAND}(x, y) = \neg(x \wedge y)$ (we say that the function f_{NAND} is the *combinational* function of equation (1. 1).

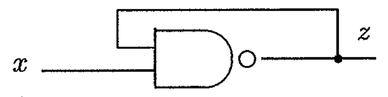

figure 1.1.

Let us assume that the input of x is 1, then regardless what is the value of input z, say either 1 or 0, the value of the output z is always the negation of the value of the input z. Since the input z and the output z are connected to each other, we know that their values should be the same. But they can be neither 1 nor 0, i.e. the value on line z is not possible to be represented in 2-valued semantics models. However, the register-and-transfer technique was previously introduced in literature to deal with this problem (see [Gordon 81] for instance). Nevertheless, we do not think that such a solution is satisfactory and it does not reflect the physics of VLSI circuit.

Another practical example for uncertain voltages is in a flip-flop, see figure. 1. 2.

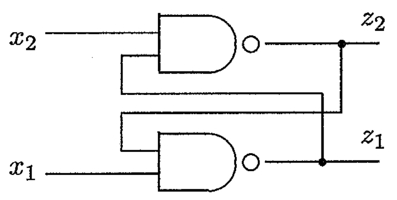

figure 1.2.

Let us assume that $x_1 = x_2 = 1$. Then what are the values on lines z_1 and z_2? As we know, there are two possibilities for stable values :

- $z_1 = 0$ and $z_2 = 1$, or
- $z_1 = 1$ and $z_2 = 0$.

which one is proper is completely depended on the actual circuit (i.e. the implementation or the delay of the individual gates, so-called racing hazard). Pre-assuming any one of the two is a simple assumption but too artificial. Assuming the all possibilities will involves a complicated non-determinism phenomenon. As a compromise, assuming a special uncertain value to designate this seems to be a better choice.

As a conclusion, we introduce new values into 2-valued switching theory with general fixed-point operators to preserve the advantage of digital logic in circuit design and to solve the mentioned problems at the meantime. For our present purpose, i.e. the design of VLSI systems, uncertain inputs would eventually lead to uncertain outputs. In this sense, one extra value $\perp$ besides 0 and 1 is sufficient. Of course, by introducing more than one exyra value into switching theory, we can get some other benefits (see [Epstein 74, Smith 81, Hurst 84, Rich 86] for some references). But this falls out of our main interest for the time being.

We define our semantics domain as a natural extension of $B = \{0, 1\}$, i.e. $B_{\perp} = B \cup \{\perp\}$ with its natural partial order $\sqsubseteq$, and as usual. That is, $x \sqsubseteq y$ means that the certainty of the voltage on y is greater than the one on x. Examples of uncertain voltages are glitches, oscillations, floating voltages and initial (internal) voltages of flip-flops. We can also extend the partial order $\sqsubseteq$ in B to any product of B pointwisely (or componentwisely). The definitions of monotonic and continuous are, as usual, that (a) f is *monotonic* iff $f(x) \sqsubseteq f(y)$ for each pair $x \sqsubseteq y$; and (b) f is *continuous* iff $lim\{f(x_n)\} = f(lim\{x_n\})$ for every sequence $\{x_n\}$. For convenience, we give new definitions for the collection of an INVERTOR (negation), a AND-gate (conjunction) and a OR-gate (disjunction), instead of the one of an INVERTOR, a NAND-gate and a NOR-gate, in the extended domain $B_{\perp}$ as follows :

1. negation

x	$\perp$
$f_{\neg}(x)$	$\perp$

2. conjunction

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

3. disjunction

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

Informally, we understand the following points.

- For INVERTOR (negation), if we are not sure of input, we would not be sure of output.
- For AND-gate (conjunction), if one of its input is 0, and then whatever the another input is, the output is controlled by the 0 input and its output is 0; if one of its input is 1, then the output can not be controlled by this input, we have to know what is the another input.
- For OR-gate (disjunction), if one of its input is 1, then whatever the another input is, the output is controlled by the 1 input and its output is 1; if one of its input is 0, and then the ouput can not be controlled by this input, we have to know the another input.

Note that the above defined conjunction and disjunction are not strict (but monotonic and continuous), and they coincide with the 3-valued logic defined in [Kleene 52].

Now, we turn our attention to introduce a general fixpoint operator, rather than the least fixpoint operator (which is commonly used in computation theory). Let us use the previous example of (1.1) (see figure 1.1) to bring in the idea of general fixedpoint operators. Suppose the external input $x = 1$ and the initial internal input z_0 (isolated voltages) is 0. Then our question is : what is the possible output z? To obtain an reasonable answer, we have the following analysis :

- $z_{2n+1} = f_{NAND}(1, z_{2n}) = \neg z_{2n} = 1$, and
- $z_{2n+2} = f_{NAND}(1, z_{2n+1}) = \neg z_{2n+1} = 0$.

We understand that the sequence of $\{z_n\}$ is *divergent* mathematically, written as $\lim_{n\to\infty} z_n = \perp$. In the physics of VLSI circuit, it means that the output voltages of the gate is oscillated. However, if $x = z_0 = 0$, then $z_n = 1$ for all $n > 0$, 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 convergent points of equation (1.1), and (ii) 1 is the fixedpoint of equation (1.1).

Therefore, we know that (a) the output of the first case should be the uncertain value $\perp$, and (b) the output of the second case should be 1.

Formally speaking, given a (combinational) function $\overline{f} : \overline{B} \to \overline{B'}$ and initial value $\overline{b}_0 \in \overline{B}$, where $\overline{B}$ is B^m for some $m \geq 1$ (or $\underbrace{B \times B \times ... \times B}_{m\ times}$) and $\overline{B'}$ is B^n for some $n \geq 1$ (or $\underbrace{B \times B \times ... \times B}_{n\ times}$) $(m \geq n)$, the output of $\overline{f}$ can be defined by the following :

1. the equation

$$< z_1, z_2, ..., z_n > = \overline{f}(x_1, x_2, ..., x_{m-n}, z_1, z_2, ..., z_n) \quad (1.2)$$

   where x_i is the ith external input and z_j is the jth internal input (feedback). Or

2. the general fixpoint operator $fix_{\overline{B'}}(curry(\overline{f})(x_1, x_2, ..., x_{m-n}))(\overline{z}_0)$ where

   - $curry : (B^m \to B^n) \times B^{m-n} \to B^n$ such that

$$curry(\overline{f})(x_1, x_2, ..., x_{m-n})(z_1, z_2, ..., z_n) =_{df} f(x_1, x_2, ..., x_{m-n}, z_1, z_2, ..., z_n)$$

   - $fix_{B^n} : (\ B^n \to B^n\ ) \times B^n \to B^n$ such that

$$fix_{B^n}(\ f\ )(\ \overline{b}_0\ ) =_{df} \begin{cases} f(\ \lim_{i\to\infty} \overline{b}_i\ ) & if\ \overline{f}\ is\ convergent\ at\ \overline{b}_0 \\ fix_{B^n}(\ f\ )(\ \overline{b^*}_0\ ) & otherwise \end{cases}$$

   where $\overline{b}_{i+1} = f(\ \overline{b}_i\ )$ for $i \geq 0$, and $\overline{b}_0^* = < b_{1,0}^*, b_{2,0}^*, ..., b_{n,0}^* >$, such that

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

By cpo domain theory [Plotkin 85], the least fixpoint theorem and the well-founded property of $\sqsubseteq$ in products of B, we know that fix_{B^n} has its value if f is monotonic.

Technologically, we focus our attention on hardware which can be built by one-input one-output INVERTOR, two-input one-output NAND-gate and two-input one-output NOR-gate; theoretically, we need to show that this collection of basic circuits (or gates) is functionally complete. Unfortunately, this is impossible since that Kleene 3-valued logic is well-know to be functionally incomplete. However, in present paper, we are able to show that our model is functionally monotonic complete. In other words, the Kleene 3-valued logic is functionally monotonic complete. This implies that all contructible gates share the following property : *whenever the certainty of input voltages of a gate increases, so does the certainty of output* *voltages of the gate.* As by-products, we also obtain two main facts about logical design of VLSI circuit.

(i) For any gate, if its input voltages are uncertain, then we can not expect to be certain about its output voltages; and

(ii) even if we can always supply voltages to inputs of every gate with great certainty, there exists a contructible gate which output voltages are always with uncertainty.

We also obtain some results which have theoretical interests, say the canonical (norm) forms of Kleene 3-valued logic. After all, I hope that this paper would provide a more applicable and theoretical sound model for hardware (behaviour). For easy understanding, we choose a semi-formal presentation to provide our results, which should not be regarded as a disadvantage.

At present, we deliberately omit the treatment of shared communication in circuit. For example, we do not consider the case of that two components share one output line. In contrast of non-sharing output lines, we do allow two or more components sharing input lines.

So far, our result are mainly semantical ones. We are very interested in pursuiting this research further but from a different point of view, i.e. from the point of view of syntactical derivability, since such research would enable us to derive mathematically secure circuit with more reality to actual circuit design. Recent Avron's work can be regarded as a few step toward this direction [Avron 87, Avron 88]. Mukaidono independently obtained the same result in [Mukaidono 86] through two stages ("regular" is his terminolgy for "monotonic"), i.e. he uses one method to get a half of the result and then changes completely to a different method to get the second half. Therefore, his approach is not as coherent (or systematic) as ours.

Lastly, we conclude this introduction section by a *Hypothesis*, which is presumed through out this paper.

Hypothesis 1.1 : we assume that for any hardware, all the initial values on its feedback lines are $\perp$'s at the very first start (or at the very beginning).

2 Some Properties of Kleene's 3-valued Logic

To help our future presentation, we briely state some useful properties of Kleene 3-valued logic in this section. We will use infix notation $\neg$, $\wedge$, $\vee$ instead of prefix notation $f_\neg$, $f_\wedge$, $f_\vee$ respectively for readability. e.g. 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 $< f, g >$ functional; (iii) *diag* is the usual diagonalizer; and (iv) π is the usual projection function.

There are some easy-checked laws and some easy-proved facts of Kleene 3-valued logic.

Laws 2.1 :

- (*Negative Negation*)

$$\neg\neg x = x$$

- (*Fixedpoint* of *Negation*)

$$\neg\perp = \perp$$

($\perp$ to be considered as a constant, although it is $unfavourable$ in hardware)

- ($\bigwedge$ *Idempotent*)
$$x \wedge x = x$$
- ($\bigvee$ *Idempotent*)
$$x \vee x = x$$
- ($\bigwedge$ *Commutative*)
$$x \wedge y = y \wedge x$$
- ($\bigvee$ *Commutative*)
$$x \vee y = y \vee x$$
- ($\perp$ *Absorption* of *Excluded Middle*)
$$\perp \wedge (x \vee \neg x) = \perp$$

(we adopt the common order among $\{\neg, \wedge, \vee\}$)

- ($\perp$ *Absorption* of *Contradiction*)
$$\perp \vee (x \wedge \neg x) = \perp$$
- (0 *Absorption* of $\bigwedge$)
$$0 \wedge x = 0$$

(0 being considered as a constant)

- (1 *Absorption* of $\bigwedge$)
$$1 \wedge x = x$$

(1 being considered as a constant)

- (0 *Absorption* of $\bigvee$)
$$0 \vee x = x$$
- (1 *Absorption* of $\bigvee$)
$$1 \vee x = 1$$
- ($\bigwedge$ *Associative*)
$$(x \wedge y) \wedge z = x \wedge (y \wedge z)$$
- ($\bigvee$ *Associative*)
$$(x \vee y) \vee z = x \vee (y \vee z)$$
- ($\bigwedge$ *Distributive*)
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$
- ($\bigvee$ *Distributive*)
$$x \vee y \wedge z = (x \vee y) \wedge (x \vee z)$$

- ($\bigwedge$ *Absorption*)
$$x \wedge (x \vee y) = x$$
- ($\bigvee$ *Absorption*)
$$x \vee (x \wedge y) = x$$
- ($\bigwedge$ *De Morgan*)
$$\neg(x \wedge y) = \neg x \vee \neg y$$
- ($\bigvee$ *De Morgan*)
$$\neg(x \vee y) = \neg x \wedge \neg y$$
- ($\bigwedge$ *Kleene*)
$$(x \wedge \neg x) \wedge (y \vee \neg y) = x \wedge \neg x$$
- ($\bigvee$ *Kleene*)
$$(x \wedge \neg x) \vee (y \vee \neg y) = y \vee \neg y$$

Note that some of the above laws are redundant.
Fact 2.2 :

- $x_1 \wedge x_2 \wedge ... \wedge x_m = 0$ iff for some $x_i = 0$;
- $x_1 \wedge x_2 \wedge ... \wedge x_m = 1$ iff for all $x_i = 1$;
- $x_1 \vee x_2 \vee ... \vee x_m = 0$ iff for all $x_i = 0$;
- $x_1 \vee x_2 \vee ... \vee x_m = 1$ iff for some $x_i = 1$;
- $x_1 \wedge x_2 \wedge ... \wedge x_m = \perp$ iff for all $x_i \neq 0$; and some $x_j = \perp$
- $x_1 \vee x_2 \vee ... \vee x_m = \perp$ iff for all $x_i \neq 1$ and some $x_j = \perp$.

Obviously, we have the following.
Theorem 2.3 : (Unfictitiousness)
Any hardware (with or without feedback) is *unfictitious*; or in other words, for any hardware, whether it is a (combinational) function f only (without feedback) or it is represented by recursive equation like (1. 2) (with feedback), if all its input (including all the initial values of its feedback) are $\perp$'s, then its output are $\perp$'s, ae well.
Proof Structural induction. *Q.E.D.*
This theorem means that, if nothing is the input of a device f, then nothing is also the output of it. This also explains why it is hard (or probably impossible) to implement cycling operation for Post many-valued logic system technologically.
In another point of view, if the input can not be deterministicly identified as 0 or 1, then the output of f can not be, either. So, in order to increase the reliability of circuit (especially, the sequential network), there must be some other technology or theory to reduce the asynchronousness of inputs (or to guarantee the synchronizing inputs) [Chaney 73, Couranz 75, Marino 77, Marino 81].
Theorem 2.4 : (Working Theorem)
For any hardware, whether it has feedback or not, it will have its output at every moment.
Proof

- For the very start, we have **Hypothesis 1.1**, i.e. all initial values of feedback are $\perp$'s at the very beginning. Then any moment after this, the initial values of feedback are provided by the environment (Note that we purposely choose not to present environments in this paper, simply because the existence of environments is irrelevant to the main results of this paper, see section 6 for further comment).
- all basic functions $\{\neg, \wedge, \vee\}$ and construction functionals, like π, *curry*, *diag* and $\circ$, are monotonic or continuous.
- According to the definition of output of feedback in Section 1, all the recursive equations involved have solutions by the well-founded property $\sqsubseteq$ and the Least Fixedpoint Theorem [Plotkin 85, Scott 76].
- Therefore, the hardware has output at every moment.

Q.E.D.

This theorem told us that any built hardware will work but it is not to tell us that the built hardware will work as expected.

3 Some Basic Theorems

For better understanding the future proof in section 4, and also for getting a better understanding of feedback, we provide some necessary or simpler results in this section.

We say that the input(s) of a hardware is (or are) *definite* if it is (or they are) deterministicly identified as 0 or 1 (i.e. an element in $\{0,1\}^*$.) and a *combinational* device is the hardware which has no feedback at all.

Then, we have the fact of Theorem 3.1 below.

Theorem 3.1 : (Combinational Theorem)

For any combinational hardware, i. e. any (combinational) function f, if all its inputs are all definite, then all its output are definite, too.

Proof Structural induction *Q.E.D.*

From the above section (section 2), we knew that the circuit built up by negation, conjunction and disjunction can not prevent the uncertain value $\perp$ propagating. Hence, there is a question,

- " Does a circuit can creat the uncertain value $\perp$ or not, if all its inputs are definite?

Unfortunately, we know the answer already, it is " yes ", see figure 1.1.

Circuits do create the uncertain value. Even worse, there is a circuit whose outputs are the uncertain value only, regardless of its input, i.e. the constant function $f_\perp$, see next theorem (**Theorem 3.2**). We call this kind of functions *indeterminate* functions (or nonsense functions). This tells us that how worse a careless design would be.

Theorem 3.2 : (Indeterminate Lemma)

There is an indeterminate function.

Proof

Let $f_N(\ x,\ y\ ) =_{df} \neg(\ (\ x\ \vee\ \neg x\ ) \wedge y\ )$ and

$$z = f_N(\ x,\ z\ ) \quad (3.1)$$

i. e.

$$z = fix_B(\ curry(\ f_N\ )(\ x\ )\ )(\ z_0\ ) \quad (3.1')$$

where the linearizer *curry* is a construction functional. And whatever the initial value z_0 and the input x are, the output z of the hardware is $\perp$. *Q.E.D.*

The indeterminate function is showed in figure 3.1.

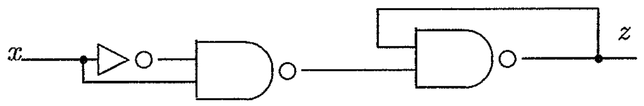

figure 3. 1.

Because the output of (3.1) (or (3.1′)) does not rely on the initial value z_0 and x, we can simply regard it as a constant function $f_\perp(\ x\ )$, or simply $\perp$.

Although indeterminate functions are unpleasant in hardware implementation, they play an important role theoretically.

For the unary functions, we have a strict completeness.

Theorem 3.3 : (Unary Completeness)

All strict unary function (or circuit) can be built by $\{\ \neg,\ \wedge,\ \vee\ \}$.

Proof

We use truth tables to define all unary functions

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

respectively as 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

and build them one by one as follows:

- $f_\perp(\ x\ )$ see **Theorem 3.2**;
- $f_0(\ x\ ) = \neg x\ \vee\ f_\perp(\ x\ )$
- $f_1(\ x\ ) = x \vee f_0(\ x\ )$
- $f_2(\ x\ ) = \neg(\ \neg x\ \vee\ f_\perp(\ x\ )\ )$
- $f_3(\ x\ ) = f_1(\ \neg x\ )$
- $f_4(\ x\ ) = x\ \wedge\ \neg x$
- $f_5(\ x\ ) = x$

- $f_6(x) = \neg x$
- $f_7(x) = x \vee \neg x$

Q.E.D.

Combining unary constant functions 0 and 1 with Theorem 3.3, we would have the monotonic unary functional completeness. In the remaining of the paper, we will see the significant roles of the first five unary functions $f_\perp$, f_0, f_1, f_2, f_3, especially the first three.

Theorem 3.5 : (Definite Completeness or Hardware Completeness)

If restricting input to definite ones, then for any possible (combinational) function f, it can be built up by $\{ \neg, \wedge, \vee \}$.

Proof

- For each of those b_i which satisfy

  $$f(x_1, x_2, ..., x_m) = 1,$$

  we choose

  1. $\#_i = \varepsilon$ (empty or nothing) if $b_i = 1$,
  2. $\#_i = \neg$ if $b_i = 0$

  to form a formula

  $$\#_1 x_1 \wedge \#_2 x_2 \wedge ... \wedge \#_m x_m \quad (3.\ 2)$$

  for m argument (input) x_i.

- For all these (3. 2), we form a formula

  $$(3.\ 2) \vee (3.\ 2) \vee ... \vee (3.\ 2) \qquad (3.\ 3)$$

  We understand that

  1. $f(x_1, x_2, ..., x_m) = 1$ iff $(3.\ 3) = 1$;
  2. if $f(x_1, x_2, ..., x_m) = 0$ then $(3.\ 3) = 0$ (not converse)

  This means that the input which satisfy

  $$(3.\ 2) = 0$$

  are more than the input which satisfy

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

  Therefore, we have to drop some 0's away (which is similar to the above).

- For each of those b_i satisfy

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

  we choose

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

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

to form a formula

$$\#_1 x_1 \wedge \#_2 x_2 \wedge \ldots \wedge \#_m x_m \qquad (3.\ 4)$$

and collect all those (3. 4) together form a formula

$$(3.\ 4) \vee (3.\ 4) \vee \ldots \vee (3.\ 4) \qquad (3.\ 5)$$

Now, we know that

$$f(x_1, x_2, \ldots, x_m) = 0 \ iff \ (3.\ 4) = 1$$

- Hence,

$$f = (3.\ 3) \vee f_0((3.\ 5)).$$

Q.E.D.

From the above theorems, we know that there are two kinds of hardware (combinational vs sequential), one has definite output while another doesn't, when both of them have definite input; i. e. one probably has uncertain outputs $\perp$'s because of feedback.

Because all hardware with feedback can be understood by its combinational functions, it is better to forget the feedback and to concentrate our attention on combinational functions. At this point of view, we extend our basic function set $\{\neg, \wedge, \vee\}$ to include all unary functions; i. e. we use $\{\neg, \wedge, \vee, f_\perp, f_0, f_1, f_2, f_3, \}$ as our basic function set. And we know that, by doing so, we didn't really extend the function space at all (in some sense), but increasing the expressive power. We will assume this and not consider feedback any more from next section (section 4) in this sense.

4 Functional Monotonic Completeness

In this section, we are going to prove the monotonic completeness.

Theorem 4.1 : (Monotonic Completeness)

For any monotonic function, it can be built up by the basic functions $\{\neg, \wedge, \vee\}$, in the sense of combinational functions (see the comment at the end of section 3).

At first, let's introduce some new notations. Be aware of that all functions mentioned in this and next section (section 5) are monotonic.

We will give the proof on one-output devices (f) only. Interested readers should not have major difficulty in extending the proof to multi-output devices.

Given an f, let

- $V(f) = \{ < b_1, b_2, \ldots, b_m > \mid f(b_1, b_2, \ldots, b_m) \neq U \}$, the set of inputs from which f has definite ouputs.
- $V_0(f) = \{ < b_1, b_2, \ldots, b_m > \mid f(b_1, b_2, \ldots, b_m) = 0 \}$, the set of inputs from which f has o output.
- $V_1(f) = \{ < b_1, b_2, \ldots, b_m > \mid f(b_1, b_2, \ldots, b_m) = 1 \}$, the set of inputs from which f has 1 output.

It is obvious that we have

- $V(\, f\,) = V_0(\, f\,) \bigcup V_1(\, f\,)$;
- $V_0(\, f\,) \bigcap V_1(\, f\,) = \emptyset$, where $\emptyset$ means the empty set.

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

- $V_0(\, f\,)$ is up-closed;
- $V_1(\, f\,)$ is up-closed;
- $V(\, f\,)$ is up-closed.

Given $\overline{b}$, for all $b_{i_j} \neq \perp$, we define

$$\begin{aligned}&V_{10}(f)(<\perp,...,\perp,b_{i_1},\perp,...,\perp,b_{i_2},\perp,...,\perp,b_{i_k},\perp,...,\perp>)\\&=\{b^* =< b_1^*, b_2^*, ..., b_m^* > \mid\\&f(\perp,...,\perp,b_{i_1},\perp,...,\perp,b_{i_2},\perp,...,\perp,b_{i_k},\perp,...,\perp) = 1\\&and\ \ b^*\ \ satisfies\ \ \#_1 x_{i_1} \wedge \#_2 x_{i_2} \wedge ... \wedge \#_k x_{i_k} = 0\}\end{aligned}$$

where $\#_j = \neg$ or ϵ (for all $j = 1, ...k$,) is according to b_{i_j}. i. e. if $b_{i_j} = 1$ then $\#_j = \epsilon$; if $b_{i_j} = 0$ then $\#_j = \neg$.

Then, we have $V_{10}(\, f\,)(<\ \perp,\ ...,\ \perp,\ b_{i_1},\ \perp,\ ...,\ \perp,\ b_{i_k},\ \perp,\ ...,\ \perp\ >)$ is up-closed; The proof are based on monotonicity.

Also by monotonicity, we have that if $F(\ \perp,\ ...,\ b_{i_1},\ \perp,\ ...,\ \perp,\ b_{i_k},\ \perp,\ ...,\ \perp\ ) = 1$, then

$$V_0(\, f\,) \subseteq V_{10}(\, f\,)(<\ \perp,\ ...,\ \perp,\ b_{i_1},\ \perp,\ ...,\ \perp,\ b_{i_k},\ \perp,\ ...,\ \perp\ >)$$

It shows us that the corresponding phrase $\#_1 x_{i_1} \wedge ... \wedge \#_k x_{i_k}$ of

$$< U, ..., U, b_{i_1}, U, ..., U, b_{i_k}, U, ..., U >$$

might collect more $0'$s in $V_{10}(f)$ than in $V_0(f)$.

Now, we are ready to prove the completeness.

Proof (of Monotonic Completeness) For one output monotonic f,

- if all output of f are $\perp$, then

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

- if at least one of output of f is not $\perp$,

  1. for each of those b_i of $\overline{b}$ which satisfies

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

  we choose

  (a) x_i and $\#_i = \neg$ if $b_i = 1$,
  (b) x_i and $\#_i = \epsilon$ if $b_i = 0$,
  (c) *nothing* if $x_i = \perp$

to form a formula

$$\#_{i_1}x_{i_1} \wedge \#_{i_2}x_{i_2} \wedge \ldots \wedge \#_{i_k}x_{i_k} \quad (4.\ 0)$$

(where all "*nothing*" are missing already) and we simplify it as

$$\#_1 x_{i_1} \wedge \#_2 x_{i_2} \wedge \ldots \wedge \#_k x_{i_k} \quad (4.\ 1)$$

(later, we will do the simplifying naturally and not mention it again) And we collect all these (4. 1) together to form a formula

$$(4.\ 1) \vee (4.\ 1) \vee \ldots \vee (4.\ 1) \qquad (4.\ 2)$$

2. for each of those b_i of $\overline{b}$ which satisfies

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

we choose

(a) x_i and $\#_i = \neg$ if $b_i = 1$,

(b) x_i and $\#_i = \epsilon$ if $b_i = 0$,

(c) *nothing* if $x_i = U$

to form a formula

$$\#_1 x_{i_1} \wedge \#_2 x_{i_2} \wedge \ldots \wedge \#_k x_{i_k} \quad (4.\ 3)$$

And we collect all these (4. 3) together to form a formula

$$(4.\ 3) \vee (4.\ 3) \vee \ldots \vee (4.\ 3) \qquad (4.\ 4)$$

- We discuss the all cases in above.
  - if there is no case 1. then

$$f(x_1,\ x_2,\ \ldots,\ x_m) = f_0(\,(4.\ 4)\,);$$

  - if there is no case 2. then

$$f(x_1,\ x_2,\ \ldots,\ x_m) = f_1(\,(4.\ 2)\,);$$

  - if there are both cases 1. and 2., we have

$$V_0(f) \subseteq V_{10}(f)(\,(4.\ 1)')$$

where (4. 1)' is the corresponding input which is one of the (4. 1)'s chosen from. Therefore,

$$V_0(f) \subseteq \textit{the intersection of those } V_{10}(f)(\,(4.\ 1)')$$

(Later, we simply refer to this kind of the intersection as $V_{10}(\ f\ )$, ... etc.) i.e.

1. $f(x_1,\ x_2,\ \ldots,\ x_m) = 1$ iff $(4.\ 2) = 1$;

2. if $f(x_1, x_2, ..., x_m) = 0$ then $(4.\ 2) = 0$.

In other words, (4. 2) probably has 0's more than needed, we should drop some away by

$$(4.\ 2) \vee f_0((4.\ 4)) \quad (4.\ 5)$$

So,

$$f(x_1, x_2, ..., x_m) = (4.\ 5)$$

Q.E.D.

This theorem says that any monotonic combinational function can be built by $\{\perp, \neg, \wedge, \vee\}$.

Through this proof, we can get *canonical forms* of Kleene 3-valued logic, which is the subject of next section (section 5).

5 Canonical (Norm) Forms

By the proof of monotonic completeness in the last section, we find out that there is a possible way to get canonical (norm) forms of Kleene system. In last section, we know that

5. 1. :

$$\begin{cases} (4.\ 2) = 1 & iff \quad f(x_1, x_2, ..., x_m) = 1 \\ if \ f(x_1, x_2, ..., x_m) = 0 & then \quad (4.\ 2) = 0 \end{cases}$$

and

5. 2. :

$$\begin{cases} (4.\ 4) = 1 & iff \quad f(x_1, x_2, ..., x_m) = 0 \\ if \ f(x_1, x_2, ..., x_m) = 1 & then \quad (4.\ 4) = 0 \end{cases}$$

So,

$$f(x_1, x_2, ..., x_m) = (4.\ 2) \vee f_0((4.\ 4)).$$

Let's use another notation in this section, i. e.

5. 3. :

$$f(x_1, x_2, ..., x_m) = (C.\ 1) \vee f_0((C.\ 2))$$

where $(C.\ 1) = (4.\ 2)$ and $(C.\ 2) = (4.\ 4)$. Similar to the procedure getting $(C.\ 1)$ and $(C.\ 2)$ in last section, we can get

5. 4. :

$$\begin{cases} (C.\ 3) = 0 & iff \quad f(x_1, x_2, ..., x_m) = 0 \\ if \ f(x_1, x_2, ..., x_m) = 1 & then \quad (C.\ 3) = 1 \end{cases}$$

and

5. 5. :

$$\begin{cases} (C.\ 4) = 0 & iff \quad f(x_1, x_2, ..., x_m) = 1 \\ if \ f(x_1, x_2, ..., x_m) = 0 & then \quad (C.\ 4) = 1 \end{cases}$$

Then,

5. 6. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 3 \,) \wedge f_1(\, (\, C.\ 1 \,)\,)$$

5. 7. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 1 \,) \vee f_2(\, (\, C.\ 3 \,)\,)$$

5. 8. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 3 \,) \wedge f_3(\, (\, C.\ 4 \,)\,)$$

(5. 3.), (5. 6.), (5. 7.) and (5. 8.) are four possibilities to get canonical forms of the system. The disadvantage of them is that they require the function f has both 0 and 1 as its definite output, i. e. each at least has one. Although a hardware which only has one definite 1 or 0 as its output is possibly useful in *ROM* designing, it is reasonable to restrict our attention on the functions which have both 1 and 0 as their definite output. But it is no reason to exclude that kind of functions from canonical forms. Therefore, similar to the procedure getting (5. 1.), (5. 2.), (5. 4.) and (5. 5.), we can get

1.

$$(\, C. 5 \,) = 1 \quad iff \quad f(\, x_1,\ x_2,\ ...,\ x_m \,) = 1 \ \ or \ \ 0;$$

2.

$$(\, C.\ 6 \,) = 0 \quad iff \quad f(\, x_1,\ x_2,\ ...,\ x_m \,) = 1 \ \ or \ \ 0.$$

Then,

5. 9. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 1 \,) \vee f_0(\, (\, C.\ 5 \,)\,)$$

5. 10. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 3 \,) \wedge f_1(\, (\, C.\ 5 \,)\,)$$

5. 11. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 1 \,) \vee f_2(\, (\, C.\ 6 \,)\,)$$

5. 12. :

$$f(\, x_1,\ x_2,\ ...,\ x_m \,) = (\, C.\ 3 \,) \wedge f_3(\, (\, C.\ 6 \,)\,)$$

If we use (5. 9.), (5. 10.), (5. 11.) and (5. 12.) as canonical forms, they do not care about that, for any f, f has both 0 or 1 as its definite output or not; but they do care about that f has definite output or not, that is, if f has no definite output (i. e. nonsense function $f_\perp$), then f can not be represented in the above four forms. It is also no reason to exclude $f_\perp$ from canonical forms, either. Now, we reach

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

and

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

where $f_\perp$ is treated as a constant; i. e. whatever its arguments are, it doesn't matter. For example, we could let it be $f_\perp(\ x_1 \vee x_2 \vee \dots \vee x_m\ )$ or $f_\perp(\ x_1 \wedge x_2 \wedge \dots \wedge x_m\ )$. The (C. 7) and (C. 8) are really what we want for our canonical forms. Before we accept that, we must be sure that it is unique in some sense. Luckily, by monotonicity, we can choose minimals in $V(\ f\ ), V_{10}(\ f\ ), V_{01}(\ f\ )$ and get corresponding (C. 6) (or (C. 5)), (C. 1), (C. 3) (where the minimal element of V' is an element of V', in which you can not find an another distinctive element $\sqsubseteq$ the minimal element), which can also be reached by the laws of $\wedge$ *absorption* and $\vee$ *absorption*. Hence, (C. 7) and (C. 8) are unique in the sense of monotonicity, they are *disjunctive* and *conjunctive* canonical (norm) forms, respectively.

Theorem 5.1 : (Norm Theorem)

For any combinational (one-output) function, it has unique conjunctive and disjunctive canonical (norm) forms in the sense of monotonicity.

6 Future Development

Upto now, our work is model-oriented. It will be very nice if we can develop a syntax-oriented version. Recent work of Avron can be regarded as a few step toward this direction [Avron 87, Avron 88]. There are some related issues which we would like to touch upon briely. We know that the following are two possible ways to incorporate the idea of environments into a semantic model.

- In the denotational approach, we can naturally introduce *Env* to represent environments, say *Env* : *Line* $\rightarrow$ *Voltage*, then a hardware h can be a function from *Input* $\times$ *Env* to *Output* $\times$ *Env*, where $B = Voltage$ and $Input = Output = B^*$. But in this way we will lose the vigorous of the digital logic.

  Going along this line, people can consult [Gordon 81].

- I am much more in favour of the equational presentation with certain extra information of environments. How to incorporate these two along this line is remained for future research.

Since our approach is based on cpo (complete partial order), there is a prospective that the formal methods used in program verification can be used in hardware verification in our approach.

7 Acknowledgement

The author would like to thank Gordon Plotkin for drawing his attention to canonical (norm) form of the Kleene system, to Robin Milner for his comment on an early draft of this paper. And he sincerely thanks to Rod Burstall for his encouragement while he was working on this paper; to Geng-bao Feng for his patiently answering questions which involve in this work in VLSI circuits; and to S. Mohseni for beneficial talking. He also like to thank Chinese Academy of Sciences for its financial support of his research.

8 References

Avron 87 : A. Avron, "Simple Consequence Relations" in LFCS report series No.30, 1987.

Avron 88 : A. Avron, "Foundations and Proof Theory of 3-valued Logics" in LFCS report series No.48, 1988.

Chaney 73 : T. J. Chaney, and E. M. Charles, " Anomalous Behaviour of Synchronizer and Abiter Circuits ", *IEEE Trans. Comput. vol. C-22, No. 4, 1973.*

Couranz 75 : G. R. Couranz and D. F. Wann, " Theoretical and Experimental Behaviour of Synchronizers Operating in the Metastable Region ", *IEEE Trans. Comput. vol. C-24, No. 6, 1975.*

Epstein 74 : G. Epstein, G. Frieder, and D. C. Rine, " The Development of Multiple-valued Logic as Related to Computer Science ", Computer, vol. 7, No. 9, 1974.

Gordon 81 : M. Gordon, "A Model of Register Transfer Systems with Application to Microcode and VLSI Correctness." Technical Report CSR-82-81, University of Edinburgh, 1981.

Istrăţescu 81: V. I. Istrăţescu, " Fixed Point Theory, an Introduction ", *Mathematics and Applications 7, D. Reidel, 1981.*

Hurst 84 : S. L. Hurst, " Multiple-valued Logic – Its Status and Future ", *IEEE Trans. Comput. vol. C-33, No. 12, 1984.*

Kleene 52 : S. C. Kleene, " Introduction to Metamathematics ", *Amsterdam, 1952.*

Marino 77 : L. R. Marino, " the Effect of Asynchronous Inputs on Sequential Network Reliability ", *IEEE Trans. Comput. vol. C-30, No. 11, 1977.*

Marino 81 L. R. Marino, " General Theory of Metastable Operation ", *IEEE Trans. Comput. vol. C-30, No. 2, 1981.*

Mukaidono 86 : M. Mukaidono, " Regular Ternary Logic Functions – Ternary Logic Functions Suitable for Treating Ambiguity ", *IEEE, Trans. Comput. vol. C-35, No. 2, 1986.*

Plotkin 85 : G. Plotkin, " Domains", *Lecture Notes in Department of Computer Science, Edinburgh University, 1985/6.*

Rich 86 : P. A. Rich, " A Survey of Multi-valued memories ", *IEEE Trans. Comput. vol. C-35, No. 2, 1986.*

Scott 76 : D. Scott, " Data Type as Lattices ", *SIAM Journal Of Computing, vol. 5, No. 3, 1976.*

Smith 81 : C. K. Smith, " The Prospects for Multi-valued Logic : a Technology and Applications View ", *IEEE Transactions on Computers, vol. C-30, No. 9, 1981.*