Deriving Operational Semantics from Denotational Semantics for Verilog

Zhu Huibiao Jonathan P. Bowen
South Bank University
Centre for Applied Formal Methods
SCISM, Borough Road, London SE1 0AA, UK
Email: {*huibiaz,bowenjp*}*@sbu.ac.uk*
URL: *http://www.cafm.sbu.ac.uk/*

He Jifeng
The United Nations University
International Institute for Software Technology
UNU/IIST, P.O. Box 3058, Macau, China
Email: *jifeng@iist.unu.edu*
URL: *http://www.iist.unu.edu/*

Abstract

This paper presents the derivation of an operational semantics from a denotational semantics for a subset of the widely used hardware description language Verilog. Our aim is to build equivalence between the operational and denotational semantics. We propose a discrete denotational semantic model for Verilog. A phase semantics is provided for each type of transition in order to derive the operational semantics.

1. Introduction

Modern hardware design typically uses a hardware description language (HDL) to express designs at various levels of abstraction. An HDL is a high level programming language, with usual programming constructs such as assignments, conditionals and iterations and appropriate extensions for real-time, concurrency and data structures suitable for modelling hardware. Verilog is an HDL that has been standardized and widely used in industry [6]. Verilog programs can exhibit a rich variety of behaviours, including event-driven computation and shared-variable concurrency.

The semantics for Verilog is very important. At UNU/IIST, the operational semantics has been explored in [1, 3, 4, 7]. Verilog's denotational semantics [9] has also been explored based on the operational semantics using Duration Calculus [8]. The two semantics can be considered equivalent informally. The question is how the two semantics can be proved equivalent formally. We have already investigated the derivation of denotational semantics from operational semantics for Verilog [10]. Therefore we have a method to guarantee the two semantics are equivalent.

This paper considers the inverse approach of the equivalence of the two semantics. The aim is to derive the operational semantics for Verilog from its denotational semantics. The similar problem was also investigated in [5] for Dijkstra's sequential language and Hoare's CSP. In our paper we define a transitional condition and a phase semantics for each type of transition. A program is said to execute a certain transition if the sequential composition of the phase semantics and the denotational semantics of the process in the transition's right *configuration* (see section 3.1) implies the denotational semantics of the process in the transition's left configuration.

This paper is organized as follows. Section 2 introduces the language and presents a discrete denotational semantic model. We also design a refinement calculus for this discrete model. Section 3 is devoted to deriving the operational semantics from its denotational semantics. We introduce transition types for Verilog and define a phase semantics of each type of transition. The denotational derivation of Verilog's operational semantics is investigated in section 3.2 based on the phase semantics. We derive the operational semantics for Verilog's statements based on our derivation strategy in section 4. Therefore, the operational semantics is considered equivalent with its denotational semantics.

2. The Discrete Denotational Model

2.1. The Syntax for Verilog

The language discussed in this paper is a subset of Verilog. It contains the following categories of syntactic elements introduced in [2].

1. Sequential Process (Thread):

$$S ::= PC \mid S \,;\, S \mid \mathbf{if}\; b\; \mathbf{then}\; S\; \mathbf{else}\; S \mid \mathbf{while}\; b\; \mathbf{do}\; S \mid c\; S$$

where PC ranges over primitive commands.

$$PC ::= (x := e) \mid \mathbf{SKIP} \mid \mathbf{Chaos} \mid \mathbf{STOP}$$

and $c\ S$ denotes timing controlled statement, and c is a time control used for scheduling. It can be either time delay $\#(\Delta)$ or event control $@(\eta)$.

$c ::= \#(\Delta) \mid @(\eta)$, where $\quad \eta ::= v \mid\uparrow v \mid\downarrow v$

2. Parallel Process (Module):
$P ::= S \quad | \quad P \parallel P$

To accommodate the expansion laws of parallel construct, the language is equipped with a hybrid control event hc:
$hc ::= @(x := e) \mid @(g) \mid \#(\Delta)$
$g ::= \eta \mid g\ or\ g \mid g\ and\ g \mid g\ and\ \neg g$
and the guarded choice $(hc_1\ P_1)\| \ldots \|(hc_n\ P_n)$

2.2. Denotational Semantic Model

Verilog processes are allowed to share program variables. In order to deal with this shared-variable feature, we describe the behaviour of a process in terms of a trace of *snapshots*, which records the sequence of atomic actions in which that process has engaged to some moment in time. Our semantic model contains a variable tr to denote that trace.

Function "$last$" yields the last snapshot of a trace. $tr_1{}^\frown tr_2$ denotes the concatenation of trace tr_1 and tr_2. Suppose tr_1 is a prefix of tr_2, $tr_2 - tr_1$ denotes the result of subtracting those snapshots in tr_1 from tr_2. The notation tr_1 **in** tr_2 indicates that tr_1 is contained in tr_2.

A snapshot is used to specify the behaviour of an atomic action, and expressed by a triple (t, σ, μ) where:
(1) t indicates the time when the atomic action happens;
(2) σ denotes the final values of program variables at the termination of an atomic action;
(3) μ is the control flag indicating which process is in control: $\mu = 1$ states the atomic action is engaged by the process, whereas $\mu = 0$ implies it is performed by the environment.

We select the components of a snapshot using the projections: $\pi_1((t, \sigma, \mu)) =_{df} t$,
$\pi_2((t, \sigma, \mu)) =_{df} \sigma, \quad \pi_3((t, \sigma, \mu)) =_{df} \mu$

Once a Verilog process is activated, it continues its execution until the completion of an atomic action; namely either it encounters a timing controlled statement, or it terminates successfully. An atomic action usually consists of a sequence of assignments as shown below.

Example 2.1: Consider the parallel program $P\|Q$ where $P =_{df} (x := 1; y := x+1; z := x+2)$ and $Q =_{df} x := 2$. Three assignments of P form an atomic action, and their execution is uninterrupted. The process Q can only be started at the beginning or at the end of the execution of P. $\square$

To trace the accumulated change made by individual assignment within an atomic action we introduce a pair of global variables $ttr =_{df} (ttr1, ttr2)$, and identify an assignment as a binary relation over the variable ttr. On the completion of an atomic action, its result will be added to the trace.

Assignment guard $@(x := e)$ is introduced in Verilog to support parallel expansion laws. We regard $@(x := e)$ as an atomic action. But its result is also stored in ttr. In order to distinguish an assignment guard from an assignment, we assign a control $flag$ with 0 to identify this case. The result of the assignment guard will be added when its sequential statement is encountered (not only time controls). $\square$

We are now ready to represent the observation by a tuple
$$(\overleftarrow{time}, \overrightarrow{time}, \overleftarrow{tr}, \overrightarrow{tr}, ttr, ttr', flag, flag')$$
where:
• $\overleftarrow{time}$ and $\overrightarrow{time}$ are the start point and the end point of a time interval over which the observation is recorded. We use $\delta(time)$ to represent the length of the time interval.
$\delta(time) =_{df} (\overrightarrow{time} - \overleftarrow{time})$
• $\overleftarrow{tr}$ stands for the initial trace of a program over the interval which is passed by its predecessor. $\overrightarrow{tr}$ stands for the final trace of a program over the interval.
$\overrightarrow{tr} - \overleftarrow{tr}$ stands for the sequence of snapshots contributed by the program itself and its environment during the interval.
• ttr and ttr' stand for the initial and final value of the variable ttr which are used to store the contribution of an atomic action over the interval.
• $flag$ and $flag'$ stand for the initial and final value of the control flag. There are two cases to indicate the end of its prior atomic action ("$ttr = null$" or "$ttr \neq null \wedge flag = 0$").

We introduce a binary "chop" operator to describe the composite behaviour of sequential composition.

Definition 2.2
$$P \frown Q =_{df}$$
$$\exists t, s, tt, f \bullet \quad P[s/\overrightarrow{tr}, t/\overrightarrow{time}, tt/ttr', f/flag'] \wedge Q[s/\overleftarrow{tr}, t/\overleftarrow{time}, tt/ttr, f/flag] \quad \square$$

The "chop" operator is associative, and distributes over disjunction. It has I has its unit and **false** as its zero, where
$$I =_{df} \delta(time) = 0 \wedge \overrightarrow{tr} = \overleftarrow{tr} \wedge ttr' = ttr \wedge flag' = flag$$

Execution of a Verilog thread can never undo an atomic action performed already. A formula P which satisfies a program must therefore imply this fact, i.e., it has to meet the *healthiness condition*.

$(H1) \qquad P = P \wedge R1$, where $R1 =_{df} \overleftarrow{tr} \preceq \overrightarrow{tr}$

A Verilog process may perform an infinite computation and enter a *divergent state*. To distinguish its chaotic behaviour from the stable ones we introduce the variables $ok, ok' : Bool$ into the semantic model, where $ok = true$ indicates the process has been started, and $ok' = true$ states the process has become *stable*.

A timing controlled statement cannot start its execution before its guard is triggered. To distinguish its waiting behaviour from terminating one, we introduce another pair of variables $wait, wait' : Bool$. $wait = true$ indicates that the process starts in an intermediate state, and $wait' = true$

means the process is waiting. The introduction of intermediate waiting state has implications for sequential composition "$P;Q$": if Q is asked to start in a waiting state of P, it leaves the state unchanged, i.e., it satisfies the *healthiness condition*.

$(H2)\ Q\ =\ II \lhd wait \rhd Q,$

where $II\ =_{df}\ \mathbf{true} \vdash (\delta(time) = 0) \wedge (\overrightarrow{tr} = \overleftarrow{tr}) \wedge$
$(\bigwedge_{s\in\{wait,ttr,flag\}} s' = s)$

$$P \lhd Q \rhd R\ =_{df}\ (P \wedge Q) \vee (\neg Q \wedge R)$$
$$P \vdash R\ =_{df}\ (ok \wedge P) \Rightarrow (ok' \wedge R) \qquad \Box$$

Definition 2.3: Let P and Q be formulae. Define

$P\ ;\ Q =_{df}$
$\exists w, o \bullet (\ P[w/wait', o/ok']^\frown Q[w/wait, o/ok]\ ) \qquad \Box$

Definition 2.4: A formula is called a *healthy formula* if it has the following form.

$$\mathbf{H}(Q \vdash W \lhd wait' \rhd T)$$

where $\mathbf{H}(X)\ =\ II \lhd wait \rhd (X \wedge R1)$ $\Box$

Theorem 2.5: $\mathbf{H}(P)$ satisfies healthiness condition $(H1)$ and $(H2)$. $\Box$

Theorem 2.6: If D_1, D_2 are healthy formulae, so are $D_1 \vee D_2$, $D_1 \lhd b \rhd D_2$ and $D_1\ ;\ D_2$,

where

if $\neg Q_1 = \neg Q_1 \wedge R1$ and $\neg Q_2 = \neg Q_2 \wedge R1$, then

$\mathbf{H}(Q_1 \vdash W_1 \lhd wait' \rhd T_1)\ ;\ \mathbf{H}(Q_2 \vdash W_2 \lhd wait' \rhd T_2)$
$=\ \mathbf{H}(\neg(\neg Q_1\ ;\ R1) \wedge \neg(T_1\ ;\ \neg Q_2) \vdash$
$(W_1 \vee (T_1\ ;\ W_2)) \lhd wait' \rhd (T_1\ ;\ T_2)) \qquad \Box$

Tthe denotational semantics of a process P is described as:

$$\mathbf{H}(\neg P_{div} \vdash P_{wait} \lhd wait' \rhd P_{ter})$$

where, P_{div}, P_{wait} and P_{ter} are the divergent, waiting and terminating behaviour of P respectively.

3. Denotational Derivation for Operational Semantics

3.1. Operational Structure, Transitional Condition and Phase Semantics

There are six types of transition for Verilog based on configurations. In order to derive Verilog's operational semantics from its denotational semantics we define a transitional condition and a phase semantics for each type of transition. A *configuration* usually consists of four components (or five in some cases):

(1) a program text P representing the rest of the program that remains to be executed;
(2) a data state σ (the second element of a configuration) denoting the initial data state of an atomic action;
(3) another data state σ' (the third element) representing the current data state during the execution of an atomic action ($\sigma' = \emptyset$ represents the previous atomic action ends and the new atomic action has not been scheduled);
(4) a control flag k (the fourth element) indicating whether or not the program P is activated: $k = 1$ says that P is scheduled to execute, whereas $k = 0$ implies that P is waiting to be activated;
(5) a thread number i (in some configurations) denoting the i-th thread of process P is being executed (i.e., this thread obtains the control flag).

The relationship between a transition and the variables in the denotational model can be described by the following diagram of an example transition.

$$\begin{array}{ccccccccc}
 & \pi_2(last(\overleftarrow{tr})) & ttr2 & flag & & & \pi_2(last(\overrightarrow{tr})) & ttr2' & flag' \\
 & \uparrow & \uparrow & \uparrow & & & \uparrow & \uparrow & \uparrow \\
<P, & \sigma, & \sigma', & 0> & \xrightarrow{<\sigma,\sigma'>}_c & <P', & \sigma', & \emptyset, & 0>
\end{array}$$

Let $O(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ stands for the observation of ttr and $flag$.

$O(\alpha_1, \alpha_2, \alpha_3, \alpha_4) =_{df} ttr = \alpha_1 \wedge ttr' = \alpha_2 \wedge$
$flag = \alpha_3 \wedge flag' = \alpha_4$

We use "$ttr = notnull$" to indicate "$ttr \neq null$".

The transition rules can be grouped into the following types [7]. We define a transitional condition $\mathbf{Cond}_{i,j}$ and its corresponding phase semantics for each type of transition. Our map from denotational semantics to operational semantics is based on the phase semantics. Here, $\mathbf{Cond}_{i,j}$ stands for the transitional condition of the **j**-th transition of type $\mathbf{T}_i$.

• Instantaneous transition

$\mathbf{T}_1$: The i-th thread of process P can perform an instantaneous action, and P enters the instantaneous section by its i-th thread being activated.

$<P, \sigma, \emptyset, 0>\ \longrightarrow\ <P, \sigma, \sigma, 1, i>,\ \ i \in \{1, 2\}$
$\mathbf{Cond}_{1,1} =_{df}$
$\overrightarrow{tr} = \overleftarrow{tr} \wedge O(null, (\pi_2(last(\overleftarrow{tr})), \pi_2(last(\overleftarrow{tr}))), 0, 1)$

$<P, \sigma, \sigma', 1>\ \longrightarrow\ <P, \sigma, \sigma', 1, i>,\ \ i \in \{1, 2\}$
$\mathbf{Cond}_{1,2} =_{df} \overrightarrow{tr} = \overleftarrow{tr} \wedge O(notnull, ttr, 1, 1)$

$\mathbf{T}_2$: Within the instantaneous section, the i-th thread of the process P performs a transition, and remains in the section or terminates. This transition assigns the successor of P an active status.

$<P, \sigma_0, \sigma, 1, i>\ \longrightarrow\ <P', \sigma_0, \sigma', 1, i>,\ \ i \in \{1, 2\}$
$<P, \sigma_0, \sigma, 1, i>\ \longrightarrow\ <P', \sigma_0, \sigma', 1>,\ \ i \in \{1, 2\}$

For a specific program P, σ' should be of the form $f(\sigma)$. The two transitional conditions are the same.

$\mathbf{Cond}_{2,1} =_{df} \overrightarrow{tr} = \overleftarrow{tr} \wedge O(notnull, (ttr1, f(ttr2)), 1, 1)$

$\mathbf{T}_3$: Within the instantaneous section, the i-th thread of a process may leave the instantaneous section. If the process is breakable, it can also leave the instantaneous section.

$< P, \sigma_0, \sigma', 1, i > \ \longrightarrow \ < P, \sigma_0, \sigma', 0 >, \ \ i \in \{1, 2\}$
$< P, \sigma_0, \sigma', 1 > \ \longrightarrow \ < P, \sigma_0, \sigma', 0 >$

The two transitional conditions are the same.

$\mathbf{Cond}_{3,1} =_{df} \overrightarrow{tr} = \overleftarrow{tr} \wedge O(notnull, ttr, 1, 0)$

$\mathbf{T}_4$: A transition represents that the program executes an assignment guard (i.e., assignment guard is regarded as an atomic action).

$< P, \sigma, \emptyset, 0 > \ \longrightarrow \ < P', \sigma, \sigma', 0 >$

For a specific process P, σ' should be of the form $f(\sigma)$.

$\mathbf{Cond}_{4,1} =_{df}$
$\overrightarrow{tr} = \overleftarrow{tr} \wedge O(null, (\pi_2(last(\overleftarrow{tr})), f(\pi_2(last(\overleftarrow{tr})))), 0, 0)$

- Triggered transition

$\mathbf{T}_5$: (1) A transition can be triggered by its sequential predecessor. This kind of transition is called the self-triggered transition.

$$< P, \sigma, \sigma', 0 > \ \overset{<\sigma,\sigma'>}{\longrightarrow}_c \ < P', \sigma', \emptyset, 0 >$$

Here, c in notation $\overset{<\sigma,\sigma'>}{\longrightarrow}_c$ represents the condition which triggers the transition. It has the form $c(\sigma, \sigma')$ based on a pair of states $< \sigma, \sigma' >$. If there is no this kind of condition, it can be understood as $\mathbf{true}$. If σ and σ' are the same, σ' will not be attached to the end of the trace.

$$\begin{aligned}\mathbf{Cond}_{5,1} =_{df} \ & c(ttr1, ttr2) \wedge O(notnull, null, 0, 0) \wedge \\ & (\overrightarrow{tr} = \overleftarrow{tr} \lhd \pi_2(last(\overleftarrow{tr})) = ttr2 \rhd \\ & \qquad \overrightarrow{tr} = \overleftarrow{tr}^\frown < (\overleftarrow{time}, ttr2, 1) >)\end{aligned}$$

(2) A transition can be triggered by its parallel partner.

$$< P, \sigma, \emptyset, 0 > \ \overset{<\sigma,\sigma'>}{\longrightarrow}_c \ < P', \sigma', \emptyset, 0 >$$

A process can also records the contribution of its environment's atomic action. But the control flag μ in the snapshot is 0. If σ and σ' are the same, the environment will not attach σ' to the end of the trace. Therefore, the process's trace remains unchanged (i.e., $\overrightarrow{tr} = \overleftarrow{tr}$) in this case.

$$\begin{aligned}&\mathbf{Cond}_{5,2} =_{df} \\ &O(null, null, 0, 0) \wedge c(\pi_2(last(\overleftarrow{tr})), \pi_2(last(\overrightarrow{tr}))) \\ &\wedge \left(\overrightarrow{tr} = \overleftarrow{tr} \vee \begin{pmatrix} \pi_1(\overrightarrow{tr} - \overleftarrow{tr}) = \overleftarrow{time} \wedge \\ \pi_3(\overrightarrow{tr} - \overleftarrow{tr}) = 0 \end{pmatrix} \right)\end{aligned}$$

The above five types of transitions have the instantaneous feature (the program itself or its environment). The corresponding phase semantics of each transition can be expressed as $Inst(\mathbf{Cond}_{i,j})$ where $\mathbf{Cond}_{i,j}$ can be the above nine transitional conditions.

$Inst(\mathbf{X}) =_{df} \mathbf{H}(\mathbf{true} \vdash \neg wait' \wedge \delta(time) = 0 \wedge \mathbf{X})$

"$\delta(time) = 0$" indicates those transitions consume zero time.

- Timing advancing transition

$\mathbf{T}_6$: $< P, \sigma, \emptyset, 0 > \ \overset{1}{\longrightarrow} \ < P', \sigma, \emptyset, 0 >$

$\mathbf{Cond}_{6,1} =_{df} \overrightarrow{tr} = \overleftarrow{tr} \wedge O(null, null, 0, 0)$

If process P cannot do any other transitions at the moment, time will advance. We regard the unit of time advancing is 1. During this period, there are no atomic actions contributed by the process P itself and its environment. Hence, time advancing keeps the trace unchanged. Its phase semantics is:

$$\begin{aligned}\mathbf{phase6} =_{df} \mathbf{H}(\mathbf{true} \vdash \mathbf{Cond}_{6,1} \wedge (\delta(time) < 1 \\ \lhd wait' \rhd \delta(time) = 1)\ )\end{aligned}$$

3.2. From Denotational Semantics to Operational Semantics

It is the purpose of an operational semantics to define the relationship between a program and its allowed execution. For this we need a clear definition of transition for Verilog. Furthermore it is the major aim of this paper to show it is possible to derive the operational semantics for Verilog in such a way as to guarantee its equivalence with the denotational semantics.

In order to derive the operational semantics from the denotational semantics, the notation of a *configuration condition* is introduced. It links the configuration state with a denotational condition.

For notational simplicity, we will use $< P, \alpha >$ to represent a configuration in later discussion and $\rho_i(< P, \alpha >)$ to denote the i-th component of $< P, \alpha >$.

Definition 3.1 (Configuration Condition)
$Condition(< P, \alpha >) =_{df}$
$\quad (ttr = null \lhd \rho_3(< P, \alpha >) = null \rhd ttr \neq null)$
$\wedge flag = \rho_4(< P, \alpha >)$ □

Let $C(\alpha_1, \alpha_2) =_{df} ttr = \alpha_1 \wedge flag = \alpha_2$. We again use "$ttr = notnull$" to indicate "$ttr \neq null$". $C(\alpha_1, \alpha_2)$ can be used to stand for the configuration condition. For example, $Condition(< P, \sigma, \sigma', 0 >) = C(notnull, 0)$

Example 3.2: Assignment $x := e$ under state $< x := e, \sigma, \emptyset, 0 >$ can be scheduled at once and then takes an instantaneous transition. The environment can also be scheduled to execute first. The order in which $x := e$ and its environment is selected is arbitrary, i.e.,

$$< x := e, \sigma, \emptyset, 0 > \ \longrightarrow \ < x := e, \sigma, \sigma, 1, 1 > \quad (1)$$

$$< x := e, \sigma, \emptyset, 0 > \ \overset{<\sigma,\sigma'>}{\longrightarrow} \ < x := e, \sigma', \emptyset, 0 > \quad (2)$$

This means $< x := e, \sigma, \sigma, 1, 1 >$ and $< x := e, \sigma', \emptyset, 0 >$ are the two execution branches of the configuration $< x := e, \sigma, \emptyset, 0 >$. On the other hand, from the denotational view, we can prove:

$$Inst(\mathbf{Cond}_{1,1}) \ ; \ x := e \ \Rightarrow \ C(null, 0) \wedge x := e \quad (3)$$

$$Inst(\mathbf{Cond}_{5,2}) \ ; \ x := e \ \Rightarrow \ C(null, 0) \wedge x := e \quad (4)$$

Here $Inst(\mathbf{Cond}_{1,1})$ and $Inst(\mathbf{Cond}_{5,2})$ are the phase semantics of the above two transitions. $C(null, 0)$ indicates the denotational semantics $x := e$ is under the configuration condition $ttr = null \wedge flag = 0$. Therefore logical formulae (3) and (4) are consistent with transitions (1) and (2) respectively. This leads to the definition of our transition strategy. $\Box$

Definition 3.3 (From Denotational Semantics to Operational Semantics)

$$< P, \alpha > \xrightarrow{\beta} < P', \alpha' >$$
$$=_{df} \quad sem \, ; \, P' \;\Rightarrow\; Condition(< P, \alpha >) \wedge P$$

where, sem is the phase semantics of transition $< P, \alpha > \xrightarrow{\beta} < P', \alpha >$. $\xrightarrow{\beta}$ can be of the transition form $\longrightarrow$, $\xrightarrow{<\sigma, \sigma'>}_c$ or $\xrightarrow{1}$. $\Box$

Here, "$\Rightarrow$" represents logical implication. P and P' in the first line of the definition stand for the syntax, whereas P and P' in the second line stand for the denotational semantics. We regard the denotational semantics of the empty process ε as II.

This definition allows the transition system of Verilog to be derived as theorems, rather than being presented as postulates; they can be strictly proved from the denotational semantics. Therefore the derived operational semantics is equivalent to or consistent with the denotational semantics.

Our main goal is to derive the operational semantics in [7]. In this sense the operational semantics of Verilog in [7] is consistent with our denotational semantics. On the other hand there may be more derived transition rules than the rules in [7]. In order to let the derived transition rules work properly, we add the following restrictions:

- Transition type $\mathbf{T_1}$ and $\mathbf{T_2}$ cannot be used for $\mathbf{STOP}$, $@(g)$, $\#n$, $@(x = e)$ and guarded choice.
- The first rule of transition type $\mathbf{T_2}$ (or $\mathbf{T_3}$) is only for those parallel processes (except $\mathbf{Chaos}$), whereas the second rule of $\mathbf{T_2}$ is only for those processes that have no parallel structure outside.
- Transition type $\mathbf{T_6}$and the second rule of $\mathbf{T_2}$ cannot be used for $\mathbf{Chaos}$.

4. Deriving Operational Semantics for Verilog Statements by Proof

In this section we will derive the operational semantics for Verilog statements by strict proof. Therefore, our derived operational semantics is equivalent to or consistent with its denotational semantics [9].

4.1. Primitive Statements

$\mathbf{SKIP}$ first adds the result of its previous atomic action if the result has not been added and then behaves in two different ways according to its role in atomic action:

(1) When it is the first statement of an atomic action, its activation can be held by the environment for a while (in fact zero time units!), and afterwards it assigns the last snapshot of the trace to ttr.
(2) Otherwise, it terminates immediately.

Its denotational semantics was defined in [9]

$$\mathbf{SKIP} \;=\; flash \lhd (ttr \neq null \wedge flag = 0) \rhd II$$
$$; \; (hold(0) \, ; \, init) \lhd ttr = null \rhd II$$

where:

$flash =_{df}$
$$Inst\begin{pmatrix} ttr' = null \wedge flag' = 0 \wedge (\overrightarrow{tr} = \overleftarrow{tr} \\ \lhd (ttr = null \;\vee\; \pi_2(last(\overleftarrow{tr})) = ttr2) \rhd \\ \overrightarrow{tr} = \overleftarrow{tr}^\frown < (\overleftarrow{time}, ttr2, 1) > \;) \end{pmatrix}$$

$init =_{df} Inst(\overrightarrow{tr} = \overleftarrow{tr} \wedge$
$\quad O(null, (\pi_2(last(\overleftarrow{tr})), \pi_2(last(\overleftarrow{tr}))), 0, 1) \,)$

$hold(n) =_{df}$
$\mathbf{H}(\mathbf{true} \vdash idle \wedge ttr' = ttr \wedge flag' = flag \wedge$
$\quad (\delta < n \lhd wait' \rhd \delta = n) \,),$

$idle =_{df} \pi_3(\overrightarrow{tr} - \overleftarrow{tr}) \in 0^* \wedge incr(\pi_1(\overrightarrow{tr} - \overleftarrow{tr})),$

$incr(s) =_{df} \forall < t_1, t_2 > \textbf{ in } s \bullet (t_2 - t_1) \in Nat$

Nat is the set containing all the non-negative integers.
δ is the abbreviation of $\delta(time)$, which is $\overrightarrow{time} - \overleftarrow{time}$. $\Box$

Theorem 4.1

$\mathbf{T}_1$: $< \mathbf{SKIP}, \sigma, \emptyset, 0 > \;\longrightarrow\; < \mathbf{SKIP}, \sigma, \sigma, 1, 1 >$
$\quad < \mathbf{SKIP}, \sigma, \sigma', 1 > \;\longrightarrow\; < \mathbf{SKIP}, \sigma, \sigma', 1, 1 >$
$\mathbf{T}_2$: $< \mathbf{SKIP}, \sigma, \sigma', 1, 1 > \;\longrightarrow\; < \varepsilon, \sigma, \sigma', 1 >$
$\mathbf{T}_5$: $< \mathbf{SKIP}, \sigma, \sigma', 0 > \;\xrightarrow{<\sigma,\sigma'>}\; < \mathbf{SKIP}, \sigma', \emptyset, 0 >$
$\quad < \mathbf{SKIP}, \sigma, \emptyset, 0 > \;\xrightarrow{<\sigma,\sigma'>}\; < \mathbf{SKIP}, \sigma', \emptyset, 0 >$

Proof: Here $\mathbf{T_{i,j}}$ indicates the **j**-th transition of Transition $\mathbf{T_i}$. We only give the proof of $\mathbf{T_{5,1}}$. Others are similar.

$Inst(\mathbf{Cond}_{5,1}) \; ; \; \mathbf{SKIP}$
$\qquad \{$Def of $\mathbf{SKIP}$ and $\mathbf{Cond}_{5,1}\}$
$= \; Inst(\mathbf{Cond}_{5,1}) \; ; \; hold(0) \; ; \; init$
$\qquad \{$Def of flash$\}$
$\Rightarrow \; C(notnull, 0) \wedge (flash \; ; \; hold(0) \; ; \; init)$
$\qquad \{$Def of $\mathbf{SKIP}\}$
$= \; C(notnull, 0) \wedge \mathbf{SKIP}$

We can also prove that $\mathbf{SKIP}$ cannot do transitions of type $\mathbf{T_3}$, $\mathbf{T_4}$ and $\mathbf{T_6}$. $\Box$

The execution of $x := e$ assigns the value of e to x. Like the treatment of $\mathbf{SKIP}$, we distinguish the case of $x := e$ is the first statement of atomic action from the other cases.

$$x := e \;=_{df}\; \mathbf{SKIP} \; ; \; assign(x, e)$$

where

$assign(x, e) =_{df}$

$Inst(\overrightarrow{tr} = \overleftarrow{tr} \wedge ttr1' = ttr1 \wedge ttr2' = ttr2[e/x]$
$\quad \wedge\, flag' = flag\ )$

$ttr2[e/x]$ is the same as $ttr2$ except mapping x to e.

Theorem 4.2

$\mathbf{T}_1$: $< x := e, \sigma, \emptyset, 0 > \ \longrightarrow\ < x := e, \sigma, \sigma, 1, 1 >$
$\quad < x := e, \sigma, \sigma', 1 > \ \longrightarrow\ < x := e, \sigma, \sigma', 1, 1 >$

$\mathbf{T}_2$: $< x := e, \sigma, \sigma', 1, 1 > \ \longrightarrow\ < \varepsilon, \sigma, \sigma'[e(\sigma')/x], 1 >$

$\mathbf{T}_5$: $< x := e, \sigma, \sigma', 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}\ < x := e, \sigma', \emptyset, 0 >$
$\quad < x := e, \sigma, \emptyset, 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}\ < x := e, \sigma', \emptyset, 0 >$

Proof

We first prove $\mathbf{T_1}$ and $\mathbf{T_5}$ based on the result of $\mathbf{SKIP}$.

$$\begin{aligned}
& Inst(\mathbf{Cond}_{i,j})\ ;\ x := e \\
& \hfill \{\text{Def of x:=e}\} \\
=\ & Inst(\mathbf{Cond}_{i,j})\ ;\ \mathbf{SKIP}\ ;\ assign(x, e) \\
& \hfill \{\text{Transition of } \mathbf{SKIP}\} \\
\Rightarrow\ & (Condition(left) \wedge \mathbf{SKIP})\ ;\ assign(x, e) \\
& \hfill \{\text{PL}\} \\
=\ & Condition(left) \wedge (\mathbf{SKIP}\ ;\ assign(x, e)) \\
& \hfill \{\text{Def of x:=e}\} \\
=\ & Condition(left) \wedge x := e
\end{aligned}$$

The prooof of $\mathbf{T_2}$ is similar to the proof of $\mathbf{T_2}$ for $\mathbf{SKIP}$ [11]. □

4.2. Timing Control

In Verilog the timing control events are introduced to synchronize the execution of parallel process. There are two kinds of events. One is the guard event; the other is the time delay. The guard event is denoted by $@(g)$. A primitive guard g can be of the following forms:

- $\uparrow v$ waits for an increase of the value of v.
- $\downarrow v$ waits for a decrease of the value of v.
- v waits for a change of v.

We introduce a predicate $fire(g)(\sigma, \sigma')$ to indicate the transition from state σ to state σ' can awake the guard $@(g)$.

$fire(\uparrow v)(\sigma, \sigma') =_{df} \sigma(v) < \sigma'(v)$
$fire(\downarrow v)(\sigma, \sigma') =_{df} \sigma(v) > \sigma'(v)$
$fire(v)(\sigma, \sigma') =_{df} \sigma(v) \neq \sigma'(v)$

The event guard $@(g)$ can be immediately fired after it is scheduled to executed. In this case, it is actually triggered by the execution of its priori atomic action, and can be specified by $seltrig(g)$.

Another case is the guard $@(g)$ waits to be fired by its environment, its idle behaviour is described by $await(g)$. When the guard is eventually triggered, its behaviour is modelled by $trig(g)$.

$$@(g)\ =_{df}\ selftrig(g)\ \vee\ (await(g)\,;\, trig(g)\ )$$

where

$selftrig(g) =_{df} \mathbf{H}(\mathbf{true} \vdash ttr \neq null\ \wedge$
$\quad fire(g)(ttr1, ttr2))\ \wedge\ II\ ;\ flash$

The definition of $await(g)$ and $trig(g)$ can be found in [11].

Theorem 4.3

$\mathbf{T}_3$: $< @(g), \sigma, \sigma', 1 > \ \longrightarrow\ < @(g), \sigma, \sigma', 0 >$

$\mathbf{T}_5$: $< @(g), \sigma, \sigma', 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}_{fire(g)}\ < \varepsilon, \sigma', \emptyset, 0 >$
$\quad < @(g), \sigma, \emptyset, 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}_{fire(g)}\ < \varepsilon, \sigma', \emptyset, 0 >$
$\quad < @(g), \sigma, \sigma', 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}_{\neg fire(g)}\ < @(g), \sigma', \emptyset, 0 >$
$\quad < @(g), \sigma, \emptyset, 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}_{\neg fire(g)}\ < @(g), \sigma', \emptyset, 0 >$

$\mathbf{T}_6$: $< @(g), \sigma, \emptyset, 0 > \ \stackrel{1}{\longrightarrow}\ < @(g), \sigma, \emptyset, 0 >$

Proof: Here we give the proof of the first rule of transition type $\mathbf{T_5}$. Other proofs can be found in [11]. Let

$\mathbf{attach1} =_{df} \overrightarrow{tr} = \overleftarrow{tr} \lhd \pi_2(last(\overleftarrow{tr})) = ttr2 \rhd$
$\quad \overrightarrow{tr} = \overleftarrow{tr}^\frown < (\overleftarrow{time}, ttr1, 1) >$

$$\begin{aligned}
& Inst(\mathbf{Cond}_{5,1})\ ;\ II \\
& \hfill \{\text{Def of } \mathbf{Cond}_{5,1}, \text{Th 2.6}\} \\
=\ & Inst(\ fire(g)(tt1, ttr2) \wedge \\
& \quad O(notnull, null, 0, 0) \wedge \mathbf{attach1}\ ) \\
& \hfill \{\text{PL}\} \\
=\ & C(notnull, 0) \wedge (seltrig(g)) \\
& \hfill \{\text{Def of @(g)}\} \\
\Rightarrow\ & C(notnull, 0) \wedge (@(g))
\end{aligned}$$

□

4.3. Iteration

The denotational semantics of Verilog iteration construct is defined in the same way as its counterpart in the conventional programming languages.

$$\mathbf{while}\ b\ \mathbf{do}\ P\ =_{df}\ \mu_{HF} X \bullet \phi(X),$$

where:

$\mu_{HF} X \bullet \phi(X)\ =_{df}\ \sqcap \{X \mid X \Rightarrow \phi(X),\ X \in HF\}$,
$\phi(X) =_{df} \mathbf{SKIP}\ ;\ ((P; X) \lhd b(ttr2) \rhd II)$,
HF is the set of all healthy formulae.

Let $b * P$ stand for $\mathbf{while}\ b\ \mathbf{do}\ P$.

Theorem 4.4

T1 $< b * P, \sigma, \emptyset, 0 > \ \longrightarrow\ < b * P, \sigma, \sigma, 1, 1 >$
$\quad < b * P, \sigma, \sigma', 1 > \ \longrightarrow\ < b * P, \sigma, \sigma', 1, 1 >$

T2 $< b * P, \sigma, \sigma', 1, 1 > \ \longrightarrow\ < P\ ;\ b * P, \sigma, \sigma', 1 >$ if $b(\sigma')$
$\quad < b * P, \sigma, \sigma', 1, 1 > \ \longrightarrow\ < \varepsilon, \sigma, \sigma', 1 >$ if $\neg b(\sigma')$

T5 $< b * P, \sigma, \sigma', 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}\ < b * P, \sigma', \emptyset, 0 >$
$\quad < b * P, \sigma, \emptyset, 0 > \ \stackrel{<\sigma,\sigma'>}{\longrightarrow}\ < b * P, \emptyset, 0 >$

Proof: Below is the proof of transition $\mathbf{T_1}$ and $\mathbf{T_5}$. The proof for $\mathbf{T_2}$ can be found in [11]. Let sem stand for the phase semantics for transition $\mathbf{T_1}$ or $\mathbf{T_5}$.

The following two laws about $\mu_{HF}X \bullet \phi(X)$ will be employed later.

$$\phi(\mu_{HF}X \bullet \phi(X)) = \mu_{HF}X \bullet \phi(X) \quad (1)$$

$$\text{If } F \Rightarrow \phi(F), \text{ then } F \Rightarrow \mu_{HF}X \bullet \phi(X) \quad (2)$$

For any healthy formula X_0 which satisfies $X_0 \Rightarrow \phi(X_0)$,

$$\begin{array}{ll}
 & sem\ ;\ X_0 \qquad \{X_0 \Rightarrow \phi(X_0)\} \\
\Rightarrow & sem\ ;\ \mathbf{SKIP}\ ;\ (P\ ;\ X_0) \lhd b(ttr2) \rhd \mathit{II} \\
 & \qquad \{\text{Transition of } \mathbf{SKIP}\} \\
\Rightarrow & (Condition(left) \wedge \mathbf{SKIP})\ ;\ (P\ ;\ X_0) \lhd b(ttr2) \rhd \mathit{II} \\
 & \qquad \{X_0 \Rightarrow \mu_{HF}X \bullet \phi(X)\} \\
\Rightarrow & Condition(left) \wedge \\
 & \quad (\mathbf{SKIP}\ ;\ (P\ ;\ \mu_{HF}X \bullet \phi(X)) \lhd b(ttr2) \rhd \mathit{II}\ ) \\
 & \qquad \{\text{Formula (1)}\} \\
\Rightarrow & Condition(left) \wedge (\mu_{HF}X \bullet \phi(X))
\end{array}$$

$$\begin{array}{ll}
 & sem\ ;\ \mu_{HF}X \bullet \phi(X) \qquad \{\text{Def of } \mu_{HF}X \bullet \phi(X)\} \\
= & sem\ ;\ \sqcap\{X \mid X \Rightarrow \phi(X), X \in HF\} \\
 & \qquad \{\text{PL}\} \\
= & \sqcap\{sem\ ;\ X \mid X \Rightarrow \phi(X), X \in HF\} \\
 & \qquad \{\text{Above result, PL}\} \\
\Rightarrow & Condition(left) \wedge (\mu_{HF}X \bullet \phi(X))
\end{array}$$

□

4.4. Parallel

In order to derive the transition rules for parallel, we first give an overview of the denotational semantics for parallel.

The trace of parallel construct is formed by interleaving of atomic actions performed by its components. Let F and G be formulae of variables $\overleftarrow{tr}$, $\overrightarrow{tr}$, ttr, ttr', $flag$ and $flag'$, which do not contain ok, ok', $wait$ and $wait'$. The merge of F and G can be expressed by $F \otimes G$ [9, 11].

The following lemmas about $\otimes$ will be employed in the later proof.

Lemma 4.5: If $P_1 \Rightarrow (ttr = null) \wedge (ttr' = null)$ and $P_2 \Rightarrow (ttr = null) \wedge (ttr' = null)$,

$$\begin{array}{ll}
\text{then} & (\ (P_1 \wedge len(\overrightarrow{tr} - \overleftarrow{tr}) = n \wedge \delta = m)\ ; Q_1\ ) \otimes \\
 & (\ (P_2 \wedge len(\overrightarrow{tr} - \overleftarrow{tr}) = n \wedge \delta = m)\ ; Q_2\ ) \\
= & (\ (P_1 \wedge len(\overrightarrow{tr} - \overleftarrow{tr}) = n \wedge \delta = m) \otimes \\
 & \ (P_2 \wedge len(\overrightarrow{tr} - \overleftarrow{tr}) = n \wedge \delta = m)\ ) \\
 & ;\ (Q_1 \otimes Q_2)
\end{array}$$

□

Lemma 4.6

If $P \Rightarrow \pi_3(\overrightarrow{tr} - \overleftarrow{tr}) \in 0^* \wedge (ttr = null) \wedge (ttr' = null)$, then $(P\ ;\ Q_1) \otimes (P\ ;\ Q_2)\ =\ P\ ;\ (Q_1 \otimes Q_2)$ □

Lemma 4.7: If $P_1 \Rightarrow P_2$ and $Q_1 \Rightarrow Q_2$, then $(P_1 \otimes Q_1) \Rightarrow (P_2 \otimes Q_2)$ □

The parallel construct $P \parallel Q$ runs P and Q in parallel.

$$P \parallel Q =_{df} attach\ ;\ \mathbf{par}(P; flash,\ Q; flash)$$

where $attach$ behaves the same as $flash$ except it keeps the value of ttr' unchanged [9, 11].

$\mathbf{par}(P, Q)$ is defined in terms of $\otimes$ in [9, 11], and its behaviour is determined by that of its components in the following ways:

- It stays at a waiting state if either component does so;
- It terminates when both components complete their execution;
- It behaves chaotically when either component is divergent.

Next we discuss the transition rules for parallel.

Theorem 4.8 (Program Refinement)

$$P\ \Rightarrow Q \ \textbf{ iff } (P_{div} \Rightarrow Q_{div})\ \wedge\ (P_{wait} \Rightarrow (Q_{div} \vee Q_{wait}))\ \wedge\ (P_{ter} \Rightarrow (Q_{div} \vee Q_{ter}))$$ □

This theorem is useful in deriving the transition rules for parallel.

Definition 4.9 (Consecutive instantaneous action)

Let P be a program, and $\alpha =< \sigma,\ \emptyset,\ 0 >$ or $< \sigma_0,\ \sigma,\ 1 >$. $< P, \alpha > \ \xrightarrow{a}\ < P', \alpha' >$ if there exists a finite sequence of configurations $\{D_i \mid 0 \leq i \leq n\}$ such that

(1) $D_0 =< P,\ \alpha >$, (2) $D_i \longrightarrow D_{i+1}$ for $0 \leq i < n$, (3) $\rho_4(D_i) = 1$ for $1 \leq i < n$, (4) $D_n =< Q, \alpha' >$ □

Next we introduce $\Longrightarrow_c$ to specify an atomic action.

Definition 4.10 (Atomic action)

Let $D =< P, \alpha >$ where $\alpha =< \sigma,\ \emptyset,\ 0 >$ or $< \sigma,\ \sigma',\ 1 >$.

$$\begin{array}{ll}
 & D\ \ \Longrightarrow_c\ < P', \sigma', \emptyset, 0 > \\
=_{df} & \exists P', \sigma, \sigma' \bullet D\ \ \xrightarrow{a}\ < P', \sigma,\ \sigma',\ 0 > \wedge \\
 & < P', \sigma,\ \sigma',\ 0 >\ \xrightarrow{<\sigma,\sigma'>}_c\ < Q', \sigma', \emptyset, 0 >
\end{array}$$ □

We can also generalize the transitional condition for an atomic action.

Theorem 4.11 If $< P, \sigma, \emptyset, 0 > \ \Longrightarrow_c\ < P', \sigma', \emptyset, 0 >$, where $\sigma' = f(\sigma)$

then $\quad Inst(atomic'(c))\ ;\ P' \Rightarrow C(null, 0) \wedge P$

where $atomic'(c) =_{df}$
$\overrightarrow{tr} = \overleftarrow{tr} \wedge O(null, (\pi_2(last(\overleftarrow{tr})),\ f(\pi_2(last(\overleftarrow{tr})))),\ 0, 0)$
$;\ \ \mathbf{Cond}_{5,1}$

$atomic(c) =_{df} atomic'(c) \wedge \delta(time) = 0$

$\mathbf{Cond}_{5,1}$ contains the condition c in its definition. □

Theorem 4.12

If $\ < P, \sigma, \emptyset, 0 > \ \Longrightarrow_{c1}\ < P', \sigma', \emptyset, 0 >$,

and $\ < Q, \sigma, \emptyset, 0 >\ \xrightarrow{<\sigma,\ \sigma'>}_{c2}\ < Q', \sigma', \emptyset, 0 >$

then (1) $atomic(c1 \wedge c2)\ ;\ (P' \parallel Q')_{div} \Rightarrow (P \parallel Q)_{div}$

(2) $atomic(c1 \wedge c2)\ ;\ (P' \parallel Q')_{wait} \Rightarrow (\ (P \parallel Q)_{div}\ \vee\ (P \parallel Q)_{wait}\ )$

(3) $atomic(c1 \wedge c2)\ ;\ (P' \parallel Q')_{ter} \Rightarrow (\ (P \parallel Q)_{div}\ \vee\ (P \parallel Q)_{ter}\ )$ □

The detailed proof can be found in [11].

Theorem 4.13

If $< P, \sigma, \emptyset, 0 > \Longrightarrow_{c1} < P', \sigma', \emptyset, 0 >$ and
$< Q, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c2} < Q', \sigma', \emptyset, 0 >$
then $Inst(atomic'(c1 \wedge c2))$; $(P' \parallel Q') \Rightarrow (P \parallel Q)$

Proof from theorem 4.12, 4.8 and 2.6. □

Theorem 4.14

If $< P, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c1} < P', \sigma', \emptyset, 0 >$ and
$< Q, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c2} < Q', \sigma', \emptyset, 0 >$
then $Inst(\mathbf{Cond}_{5,2})$; $(P' \parallel Q') \Rightarrow (P \parallel Q)$
Here $\mathbf{Cond}_{5,2}$ contains the condition $c1 \wedge c2$. □

Theorem 4.15

If $< P, \sigma, \emptyset, 0 > \stackrel{1}{\longrightarrow} < P', \sigma, \emptyset, 0 >$ and
$< Q, \sigma, \emptyset, 0 > \stackrel{1}{\longrightarrow} < Q', \sigma', \emptyset, 0 >$
then $\mathbf{phase6}$; $(P' \parallel Q') \Rightarrow (P \parallel Q)$ □

Theorem 4.16 (Transition system for parallel)

(1) If $< P, \sigma, \emptyset, 0 > \Longrightarrow_{c1} < P', \sigma', \emptyset, 0 >$ and
$< Q, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c2} < Q', \sigma', \emptyset, 0 >$,
then $< P \parallel Q, \sigma, \emptyset, 0 > \Longrightarrow_{c1 \wedge c2} < P' \parallel Q', \sigma', \emptyset, 0 >$

(2) If $< P, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c1} < P', \sigma', \emptyset, 0 >$ and
$< Q, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c2} < Q', \sigma', \emptyset, 0 >$
then $< P \parallel Q, \sigma, \emptyset, 0 > \stackrel{<\sigma, \sigma'>}{\longrightarrow}_{c1 \wedge c2} < P' \parallel Q', \sigma', \emptyset, 0 >$

(3) If $< P, \sigma, \emptyset, 0 > \stackrel{1}{\longrightarrow} < P', \sigma', \emptyset, 0 >$ and
$< Q, \sigma, \emptyset, 0 > \stackrel{1}{\longrightarrow} < Q', \sigma', \emptyset, 0 >$
then $< P \parallel Q, \sigma, \emptyset, 0 > \stackrel{1}{\longrightarrow} < P' \parallel Q', \sigma', \emptyset, 0 >$

Proof directly from theorem 4.13, 4.14 and 4.15. □

Transition rules of Theorem 4.16(2)(3) are consistent with the parallel rules $\mathbf{T_5}$, $\mathbf{T_6}$ in [7]. Our proved rule of Theorem 4.16(1) is the universal rule of $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, $\mathbf{T_4}$. We can extend this general rule to the detailed rules of $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, $\mathbf{T_4}$ in [7] according to the simulation-based scheduler. Then our whole transition system can work properly.

For other statements of Verilog, the derived transition rules and their proofs are presented in [11].

5. Conclusion

The main contribution of our work is to derive the operational semantics for a subset of Verilog from its denotational semantics. Thus, our operational semantics presented here is equivalent to its denotational semantics. We provide a discrete denotational model and design a refinement calculus for it. Our approach is new. We define a transitional condition and phase semantics for each type transition. A transition can be derived if the sequential composition of the phase semantics and the denotational semantics of the process in the transition's right configuration implies the denotational semantics of the process in the transition's left configuration.

For the future, we are continuing to explore unifying theories of Verilog. The completeness of the derived operational semantics for Verilog is another interesting topic for study.

References

[1] J. P. Bowen, He Jifeng and Xu Qiwen. An Animatable Operational Semantics of the VERILOG Hardware Description Language. *Proc. ICFEM2000: 3rd IEEE International Conference on Formal Engineering Methods*, IEEE Computer Society Press, pp. 199–207, York, UK, September 2000.

[2] M. J. C. Gordon. The Semantic Challenge of Verilog HDL. *Proc. Tenth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, pp. 136–145, June 1995.

[3] He Jifeng and Xu Qiwen. An Operational Semantics of a Simulator Algorithm. Technical Report 204, UNU/IIST, P.O. Box 3058, Macau, 2000.

[4] He Jifeng and Zhu Huibiao. Formalising Verilog. *Proc. IEEE International Conference on Electronics, Circuits and Systems*, IEEE Computer Society Press, pp. 412–415, Lebanon, December 2000.

[5] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming.* Prentice Hall International Series in Computer Science, 1998.

[6] IEEE *Standard Hardware Description Language based on the Verilog Hardware Description Language*. IEEE Standard 1364-1995, 1995.

[7] Li Yongjian and He Jifeng. Formalising VERILOG: Operational Semantics and Bisimulation. Technical Report 217, UNU/IIST, P.O. Box 3058, Macau, November 2000.

[8] Zhou Chaochen, C. A. R. Hoare and A. P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.

[9] Zhu Huibiao and He Jifeng. A Semantics of Verilog using Duration Calculus. *Proc. International Conference on Software: Theory and Practice*, pp. 421–432, Beijing, China, August 2000.

[10] Zhu Huibiao, J. P. Bowen and He Jifeng. From Operational Semantics to Denotational Semantics for Verilog. *Proc. CHARME 2001: 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Livingston, Scotland, 4–7 September 2001. Springer-Verlag, LNCS 2144, 2001.

[11] Zhu Huibiao, J. P. Bowen and He Jifeng. Deriving Operational Semantics from Denotational Semantics for Verilog. Technical Report SBU-CISM-01-16, South Bank University, London, UK, June 2001.