Proof Theory and a Validation Condition Generator for VHDL[*]

Peter T. Breuer   Luis Sánchez Fernández   Carlos Delgado Kloos

Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid, ETSI Telecomunicación
<{ptb,lsanchez,cdk}@dit.upm.es>

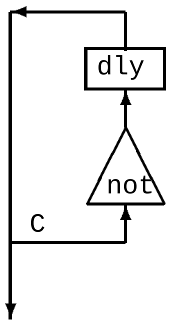
Abstract

We present a Hoare-style programming logic for VHDL *together with a succinct* PROLOG *implementation which acts as a* validation condition *generator. The logic is based on a particularly simple formalization of the language as a pure side-effect on an infinite time-sequence of states. The* PROLOG *program transforms logical assertions to their prerequisite hypotheses, thus reducing formal verification to a proof that certain initial conditions hold.*

1 Introduction

VHDL is a standardized hardware description language, originally developed in the 1980s with the support of the US Department of Defense, and it has gained wide acceptance. Nevertheless, no formal semantics underpins the standard [4], and recently there has been great interest shown in developing formal semantics for the *simulation model* informally defined by it. In previous work [2] we have set out a synchronous semantics based on the idea of a clocked sequence of global states. Every unit time interval appears in the model. This design lends itself to a particularly simple formalization of VHDL as a pure side-effect on an infinite time-sequence of states, and makes Hoare programming logic applicable to the language. We present the logic for the first time here together with a succinct PROLOG implementation which acts as a *validation condition* generator for VHDL descriptions.

The PROLOG program transforms logical assertions to their prerequisite hypotheses. Given a formal assertion about a VHDL simulation at a given time, the generator will express the necessary logical preconditions at a given earlier time which force it to be true. At time zero, these are the startup conditions. So formal verification can reduce to a proof that certain initial conditions obtain.



```
PROCESS
BEGIN
  C <= TRANSPORT (NOT C)
          AFTER dly ;
  WAIT ON C ;
END
```

Figure 1: Process '`a`' hardware diagram and VHDL.

The paper has the following layout. In Section 2 a short description of VHDL is given. Section 3 gives a denotational presentation of the semantics. Section 4 introduces the axiomatic semantics, including the PROLOG implementation as a validation condition generator. Section 5 discusses other models of VHDL semantics that have appeared in the literature.

2 Brief description of VHDL

VHDL is a procedural language in the imperative style of Pascal and C. A program script consists of a set of *process* definitions. The concept is that processes represent hardware modules executing independently and asynchronously and communicating through *signal wires*. The VHDL description and hardware diagram in Figure 1 is of a process `a` that will be alluded to again in this paper. It is a self-oscillator with half-period `dly` outputting 1/0 on signal wire `C`.

The code in the process body is procedural. It may contain `while` loops, `if` branches, and so on. The process

[*]The work reported in this paper has been partially supported by the INTEGRAL project (TIC93–0515–C02–01) sponsored by the Spanish "Comisión Interministerial de Ciencia y Tecnología" and the EuroForm network (2/ERB4050PL921826) sponsored by the Human Capital and Mobility programme of the CEC.

is conceptualized as looping continuously. There are two special kinds of atomic statement:

i] *Signal assignment*, an example of which is the first statement in the body of process `a` in Figure 1.

This statement launches a request for a future assignment on an output signal, prescribing the delay. If the delay is zero, the assignment is called *delta-delayed*.

ii] The explicit `wait` statement, of the general form

`wait on` *signals* `until` *test* `for` *time*

Some parts are optional. The *signals* list those that the `wait` statement is sensitive to. Each time a change occurs in one of these *test* is evaluated. If it is `true`, the process will resume, else it will remain suspended. Entry into a `wait` causes *time* to be evaluated, determining the imeout interval. The process will resume when the timeout expires if no listed signal changes.

The standard VHDL simulation cycle runs each process in a VHDL description until it becomes blocked in a `wait` statement. Zero logical time has elapsed to this point. Logical time is then advanced to the next pending signal assignment, and the assignment is executed. The conditions of each `wait` statement are then checked, and if any are satisfied the appropriate process bodies are executed (in zero logical time) until all become blocked again. Then logical time is advanced again, and so on.

2.1 VHDL pseudo code

The following statement and process constructors and combinators will be needed in a VHDL pseudo-code that we use for convenience: $\fatsemi$ (sequence), $||$ (parallel), $\Leftarrow$ (signal assignment) and `wait`, `if` (conditional branch), `while` (loop) and `process` (encapsulation).

The complete Backus-Naur Form (BNF) for the concrete syntax of the pseudo-code language is given in Figure 2. Note that each identifier in a `process` header declares a signal over which the process has (sole) output rights. The syntax is the same as that in [7], apart from the extra inclusion of the `wait` statement here, and the naming of outputs in a `process`.

3 Denotational semantics

The underlying semantics of VHDL statements proposed here is a side-effect on a schedule of partially defined scheduled future states called a *driver set* (Δ), yielding as result the sequence of states that existed during the time interval that the statement blocked for. This is called an *episode* (ϵ).

```
Statement ::= Id ⇐ Expression after Delay
            | wait on Ids
            | wait for Delay
            | Statement ⨟ Statement
            | if Expr then Statement
                      else Statement
            | while Expr do Statement

Process   ::= process
Ids begin Statement end
            | Process || Process
```

Figure 2: The BNF for the VHDL pseudo-code.

To present the semantics as a pure side-effect (i.e., returning no result), we embed it in a slightly larger space, one which permits non-causal and non-deterministic effects. We group all episodes from previously executed statements together into a *history* $\mathtt{H} = \ldots + \!\!+ \, \epsilon_{\mathtt{i}} + \!\!+ \, \epsilon_{\mathtt{i+1}} + \!\!+ \ldots$. Then a VHDL statement can be seen as side-effecting on the history and driver set together, which we call a *world line* $\mathtt{W} = \mathtt{H} + \!\!+$, plus a current *time point* $\mathtt{T}$. The time point marks the boundary between past and future; between history and driver set. That is, writing 'seq $State$' for the domain of finite or infinite sequences of states, and $\mathtt{W!t}$ for the $\mathtt{t}$'th state in the sequence $\mathtt{W}$:

$$\mathtt{H} :: \mathit{History} \quad \mathtt{where} \quad \mathit{History} = \mathrm{seq}\ \mathit{State}$$
$$\mathtt{H} = [\mathtt{W!t} | \mathtt{t} = \mathtt{0}, \ldots, \mathtt{T} - \mathtt{1}]$$

$$\Delta :: \mathit{DriverSet} \quad \mathtt{where} \quad \mathit{DriverSet} = \mathrm{seq}\ \mathit{State}$$
$$\Delta = [\mathtt{W!t} | \mathtt{t} = \mathtt{T}, \mathtt{T} + \mathtt{1}, \ldots]$$

$$\mathtt{W} :: \mathit{WorldLine} \quad \mathtt{where} \quad \mathit{WorldLine} = \mathrm{seq}\ \mathit{State}$$
$$\mathtt{W} = \mathtt{H} + \!\!+$$

In all realizable circumstances the time point $\mathtt{T}$ is finite, the driver set Δ is an infinite sequence of states, one for each unit of time to come, and the history $\mathtt{H}$ is finite, composed of a finite number of finite episodes.

The semantics $\mathcal{S}[\![\mathtt{R}]\!]\rho$ for a statement $\mathtt{R}$ (with signals ρ available for output) is formally a relation (here $A \leftrightarrow B$ is the domain of relations with source A and target B) between a world line and an altered world line, with a shift of time point:

$$\mathit{StatementSemantics} = (\mathit{WorldLine}, \mathit{TimePoint}) \leftrightarrow (\mathit{WorldLine}, \mathit{TimePoint})$$

The semantics we are interested in are *causal* (and usually *deterministic*). Any causal statement semantics affects a world line by moving the time point forward, and it leaves untouched the existing history. A deterministic semantics is a functional relation – on its given domain. Although statements determine the values of signals to which they have the (sole) right to output, they do not bind other variables and signals, and the semantics is fully non-deterministic with respect to these.

Our treatment will be simplified by the following measures: (1) we ban local variables. We replace them by expressions in the signals where necessary. Static variables can be emulated by signalling appropriate values for future reception. We (2) do not treat signal assignments with zero delay. These are arguably without physical meaning, and the effects on the semantics are bizarre. [7] does not treat delay delays either. Note that we do not track a set of events (γ in [7]). In the sequel, we assume that the nanosecond is the standard unit, and may write 1 ns for a delay of one unit time.

A process `P` has a semantics $\mathcal{P}[\![\mathtt{P}]\!]$ which falls in the same formal domain as statement semantics.

$$\begin{aligned}&\mathit{ProcessSemantics} = \\ &(\mathit{WorldLine}, \mathit{TimePoint}) \leftrightarrow (\mathit{WorldLine}, \mathit{TimePoint})\end{aligned}$$

However, it encodes the transient relationship between an initial driver set and the sequence of resulting states. The meaning is detailed further below.

A *state* binds signal wires and variable identifiers with values. It is a partial function:

$$\mathit{State} = \mathit{Id} \nrightarrow \mathit{Val}$$

To be definite, we use strings as identifiers for the signal wires and integers as the values assigned to them. Booleans will be emulated by 1 and 0 integer values.

$$\mathit{Id} \;=\; \mathit{String} \qquad\qquad \mathit{Val} \;=\; \mathit{Int}$$

We can look up the value of a bound identifier in a state: given the state `s`, `s!x` returns the value bound to `x` in `s`. Lookup is extended to expressions too. We assign a value `v` to an identifier `x` in a state `s` by writing `(x!=v)s`, which returns the overwritten state.

`W!t!x` is the value of `x` in the `t`'th state of world line `W`. We use the notation the other way round too: `W!x` is the sequence of values taken by `x` in `W` as `t` varies, and $\mathtt{W!x!t} = \mathtt{W!t!x}$.

3.1 VHDL statement semantics

Statements are sequenced using relational composition ((1) in Figure 3). But what we are really interested in is signal assignment and `wait` statements. Assignment is non-blocking, so it delivers an empty episode. It overwrites at the specified future state, now+`dly` ns. It also pre-empts later signals to `x`, and that is expressed here by making it write at *all* future times too, using the operator $\mathtt{x} \Leftarrow^{\infty}_{\mathtt{t}} \mathtt{v}$, which overwrites `x` in the world line at all times beyond `t` . The semantics is shown in (2) of Figure 3. Note that the current state is used in the calculation of the expression to assign.

A `wait on` statement blocks during a nonempty episode. If the current state of the signals `xs` differs from the immediately future state, then it releases after 1 ns, with the state in which the transition has just happened now current. Otherwise it blocks again. The semantics is shown in (3) of Figure 3. Recall that we are forbidding any succeeding delta-delayed assignments to the signal, which might otherwise undo the transition. A `wait for` blocks for a specified time (4). An extra `skip` statement has been intriduced for convenience (7). It does nothing and takes zero time to do it.

We define a modified statement semantics $\mathcal{S}'[\![\mathtt{R}]\!]\rho$ for later use in the process semantics. It expresses the idea of an execution which finishes in a *suspension* (in a `wait` statement). It is described in Figure 4.

3.2 VHDL process semantics

The `process` constructor makes a process out of a statement. What is executed is essentially `R;R;R;`..., but only the resulting history is of interest. It is infinitely long if `R` is a legal VHDL process body (which must have a `wait` statement in every execution path, and therefore take non-zero logical time per cycle). We run the loop from some zero time `T0` until a time `T1`, when it is blocked in a `wait` statement, then let it run until time `T2`, when it is again blocked. If the world line `W1` at time `T1` necessarily entails `W2` at `T2`, then (`W1`, `T1`) and (`W2`, `T2`) are said to be related by the $\mathcal{P}$ semantics (with origin `T0`) (13):

$$\begin{aligned}&(\mathtt{W1},\mathtt{T1})\mathcal{P}[\![\mathtt{process}\ \rho\ \mathtt{begin\ R\ end}]\!](\mathtt{W2},\mathtt{T2}) \qquad (13)\\ \Leftrightarrow\ \ &(\_,\mathtt{T0})\mathcal{S}'[\![\mathtt{while\ 1\ do\ R}]\!]\rho(\mathtt{W1},\mathtt{T1})\\ \subseteq\ \ &(\_,\mathtt{T0})\mathcal{S}'[\![\mathtt{while\ 1\ do\ R}]\!]\rho(\mathtt{W2},\mathtt{T2})\end{aligned}$$

`T0` is usually zero. In comparison with the above, concurrency is just a little zip on the reality vector (14).

$$\mathcal{P}[\![\mathtt{P1} \,||\, \mathtt{P2}]\!] = \mathcal{P}[\![\mathtt{P1}]\!] \cap \mathcal{P}[\![\mathtt{P2}]\!] \qquad (14)$$

4 A logical semantics

We give a logical semantics to VHDL statements, following a Hoare logic pattern:

$$\mathcal{S}\rho:\ \ \{\mathtt{Pre},\mathtt{T1}\}\ \mathtt{R}\ \{\mathtt{Post},\mathtt{T2}\}$$

A statement `R` is interpreted as a (meta) relation between propositions `Pre` and `Post` describing the state at times `T1`

$$
\begin{array}{rcll}
\mathcal{S}\llbracket \mathtt{R1} \fatsemi \mathtt{R2}\rrbracket\rho & = & \mathcal{S}\llbracket \mathtt{R2}\rrbracket\rho \ \mathrm{o}\ \mathcal{S}\llbracket \mathtt{R1}\rrbracket\rho & (1)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket \mathtt{x} \Leftarrow \mathtt{ex\ after\ dly}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \Leftrightarrow & \mathtt{T1}=\mathtt{T2}\ \ \&\ \ \mathtt{W2}!\rho = (\mathtt{x} \Leftarrow^{\infty}_{\mathtt{T1+dly}} \mathtt{W1!T1!ex})\ \mathtt{W1}!\rho & (2)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket \mathtt{wait\ on\ xs}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \Leftrightarrow & \mathtt{W1}!\rho=\mathtt{W2}!\rho\ \&\ \mathtt{W1!T1!xs} = \ldots = \mathtt{W1}!(\mathtt{T2}-1)!\mathtt{xs} \neq \mathtt{W1!T2!xs} & (3)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket \mathtt{wait\ for\ n}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \Leftrightarrow & \mathtt{W1}!\rho=\mathtt{W2}!\rho\ \&\ \mathtt{T1}+\mathtt{n}=\mathtt{T2} & (4)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket \mathtt{if\ ex\ then\ R1\ else\ R0}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \Leftrightarrow & \mathit{if}\ \mathtt{W1!T1!ex}\neq 0\ \ \mathit{then}\ \ (\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket\mathtt{R1}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & (5)\\
 & & \qquad\qquad\qquad\quad \mathit{else}\ \ (\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket\mathtt{R0}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \\
\mathcal{S}\llbracket \mathtt{while\ ex\ do\ R}\rrbracket\rho & \Leftrightarrow & \mathcal{S}\llbracket \mathtt{if\ ex\ then\ R} \fatsemi \mathtt{while\ ex\ do\ R\ else\ skip}\rrbracket\rho & (6)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket \mathtt{skip}\rrbracket(\mathtt{W2},\mathtt{T2})\rho & \Leftrightarrow & \mathtt{W1}!\rho=\mathtt{W2}!\rho\ \&\ \mathtt{T1}=\mathtt{T2} & (7)
\end{array}
$$

Figure 3: Denotational semantics of execution until termination for VHDL statements.

$$
\begin{array}{rcll}
\mathcal{S}'\llbracket \mathtt{R1} \fatsemi \mathtt{R2}\rrbracket\rho & = & \mathcal{S}'\llbracket \mathtt{R1}\rrbracket\rho\ \cup\ \mathcal{S}'\llbracket \mathtt{R2}\rrbracket\rho \ \mathrm{o}\ \mathcal{S}\llbracket \mathtt{R1}\rrbracket\rho & (8)\\
\mathcal{S}'\llbracket \mathtt{x} \Leftarrow \mathtt{ex\ after\ dly}\rrbracket & = & \{\ \} & (9)\\
\mathcal{S}'\llbracket \mathtt{while\ ex\ do\ R}\rrbracket\rho & \Leftrightarrow & \mathcal{S}'\llbracket \mathtt{if\ ex\ then\ R} \fatsemi \mathtt{while\ ex\ do\ R\ else\ skip}\rrbracket\rho & (10)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}'\llbracket \mathtt{wait\ on\ xs}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \Leftrightarrow & \mathtt{W1}\rho=\mathtt{W2}!\rho\ \&\ \mathtt{W1!T1!xs} = \ldots = \mathtt{W1!T2!xs} & (11)\\
(\mathtt{W1},\mathtt{T1})\mathcal{S}'\llbracket \mathtt{if\ ex\ then\ R1\ else\ R0}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & \Leftrightarrow & \mathit{if}\ \mathtt{W1!T1!ex}\neq 0\ \ \mathit{then}\ (\mathtt{W1},\mathtt{T1})\mathcal{S}'\llbracket\mathtt{R1}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) & (12)\\
 & & \qquad\qquad\qquad\quad \mathit{else}\ \ (\mathtt{W1},\mathtt{T1})\mathcal{S}'\llbracket\mathtt{R0}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) &
\end{array}
$$

Figure 4: Denotational statement semantics for execution until suspension.

and T2, respectively, immediately prior to control passing to R, and immediately after control leaves it, assuming it does. The set of signals that R can affect is ρ.

The *process semantics* of a (possibly compound) process P is taken between points T1 and T2 where all component processes are properly blocked.

$$\mathcal{P}:\ \ \{\mathtt{Pre},\mathtt{T1}\}\ \mathtt{P}\ \{\mathtt{Post},\mathtt{T2}\}$$

What has happened at times T1, T2 is that each process has completed a sequence of zero-logical time statements, and has entered but not left a `wait`.

To make Hoare logic expressible, we add a time index to program variables, so that we may write of the variable x at time now+2ns, for example, within a logical proposition. We write it as $\odot^2\mathtt{x}$.

$$Id' = \odot^{Delay} Id$$

We write $\odot^0\mathtt{x}$ as just x. The syntax of timed propositions now includes arithmetic operators and relations, plus logical operators, but no quantification over time.

Definition 1 *A world line* W *models a timed proposition* P *at a time point* T*, that is:*

$$(\mathtt{W},\mathtt{T}) \models \mathtt{P}$$

if when we substitute timed references $\odot^t\mathtt{x}$ *in* P *with the values in the* $\mathtt{T}+t$*'th state in* W, $\mathtt{W}!(\mathtt{T}+t)!\mathtt{x}$*, then the proposition becomes* `true`.

Definition 2 $\mathcal{S}\rho:\{\mathtt{P1},\mathtt{T1}\}\ \mathtt{R}\ \{\mathtt{P2},\mathtt{T2}\}$ *iff* $(\forall\, \mathtt{W1},\mathtt{W2}.)$

$$(\mathtt{W1},\mathtt{T1})\models\mathtt{P1}\ \&\ (\mathtt{W1},\mathtt{T1})\mathcal{S}\llbracket\mathtt{R}\rrbracket\rho(\mathtt{W2},\mathtt{T2}) \Rightarrow (\mathtt{W2},\mathtt{T2})\models\mathtt{P2}$$

I.e., the relation is forced by the model. For example:

$$\mathcal{S}[\mathtt{x}]:\ \ \{\mathtt{x}\neq\odot\mathtt{x},2\}\ \ \mathtt{wait\ on\ x}\ \ \{\mathtt{true},3\}$$

holds. If x is about to change at time 2 in world line W1, and control is about to pass into a `wait on x` statement, then control will pass away from it 1 ns later in world line W2, at time 3, according to the $\mathcal{S}$ semantics.

Note that the Hoare logic relation does not stipulate that a statement *must* execute in T2-T1 ns, but that when it does, then the relation must hold.

Definition 3 $\mathcal{P}:\ \{\mathtt{P1},\mathtt{T1}\}\ \mathtt{P}\ \{\mathtt{P2},\mathtt{T2}\}$ *iff* $(\forall\,\mathtt{W1},\mathtt{W2}.)$

$(\mathtt{W1},\mathtt{T1})\models\mathtt{P1}\ \&\ (\mathtt{W1},\mathtt{T1})\mathcal{P}[\![\mathtt{P}]\!](\mathtt{W2},\mathtt{T2})\Rightarrow(\mathtt{W2},\mathtt{T2})\models\mathtt{P2}$

This means reaching `W1` at time `T1` under $\mathcal{S}'$ semantics entails reaching `W2` at time `T2`, later. `T1` need not be a time at which the process starts an execution cycle.

4.1 Hoare logic for VHDL statements

We can represent part of the Hoare logic for the `wait on x` statement as follows:

$$\frac{\mathcal{S}\rho:\{\mathtt{P},\mathtt{T}-1\}\ \mathtt{skip}\ \{\odot\mathtt{Q},\mathtt{T}-1\}}{\mathcal{S}\rho:\{\mathtt{Xs}\neq\odot\mathtt{Xs}\ \&\ \mathtt{P},\mathtt{T}-1\}\ \mathtt{wait\ on\ Xs}\ \{\mathtt{Q},\mathtt{T}\}}\qquad(15)$$

corresponding to (3) in Figure 3, where:

Definition 4 $\odot\mathtt{Q}$ *is* `Q` *with all timed references to variables and signal values* $\odot^{t}\mathtt{x}$ *replaced by references to* $\odot^{t+1}\mathtt{x}$.

If nothing happens for 1 ns, then if $\odot\mathtt{Q}$ was true at absolute time t, `Q` will be true at absolute time $t+1$, and so on. The proposition $\odot\mathtt{Q}$ can therefore be read as "`Q` *will hold in 1 ns from now*".

Writing $[\odot^{t}\mathtt{x}{:=}\mathtt{y}]\mathtt{Q}$ for substitution of the timed variable $\odot^{t}\mathtt{x}$ by the value `y` within predicate `Q`, the semantics of assignment are described along with the rest of the $\mathcal{S}$ logic in (19) of Figure 5.

The 'run until suspend' logic $\mathcal{S}'\rho$ is expressed in Figure 6. It states that the only way of approaching the cut-off time `T2` is through a `wait` statement, and it must not have released yet at the cut-off time.

4.2 VHDL process logic

A literal translation of Definition 3 renders the assertion $\mathcal{P}:\{\mathtt{Pre},\mathtt{T1}\}\ \mathtt{p}\ \{\mathtt{Post},\mathtt{T2}\}$ as:

$$\frac{\begin{array}{rl}\forall\,\mathtt{R},\mathtt{T0}<\mathtt{T1}. & \mathcal{S}'\rho{:}\{\mathtt{R},\mathtt{T0}\}\mathtt{while\ 1\ a}\{\mathtt{P},\mathtt{T1}\}\\ \Rightarrow & \mathcal{S}'\rho{:}\{\mathtt{R},\mathtt{T0}\}\mathtt{while\ 1\ a}\{\mathtt{Q},\mathtt{T2}\}\end{array}}{\mathcal{P}:\{\mathtt{P},\mathtt{T1}\}\ \mathtt{process}\ \rho\ \mathtt{a}\ \{\mathtt{Q},\mathtt{T2}\}}\qquad(26)$$

But we can substitute the first `while 1 a` by `a` in the above, and formulate the deduction in a way that removes the quantification over predicates:

$$\frac{\mathcal{S}'\rho{:}\{\mathtt{R},\mathtt{T0}\}\mathtt{a}\{\mathtt{P},\mathtt{T1}\}\quad\mathcal{S}'\rho{:}\{\mathtt{S},\mathtt{T0}\}\mathtt{while\ 1\ a}\{\mathtt{Q},\mathtt{T2}\}}{\mathcal{P}:\{\odot^{\mathtt{T0}-\mathtt{T1}}(\mathtt{R\&S})\&\mathtt{P},\mathtt{T1}\}\ \mathtt{process}\ \rho\ \mathtt{a}\ \{\mathtt{Q},\mathtt{T2}\}}\qquad(27)$$

The logic for parallel composition is standard:

$$\frac{\mathcal{P}:\{\mathtt{P},\mathtt{T1}\}\ \mathtt{a}\ \{\mathtt{Q},\mathtt{T2}\}\quad\mathcal{P}:\{\mathtt{P},\mathtt{T1}\}\ \mathtt{b}\ \{\mathtt{Q},\mathtt{T2}\}}{\mathcal{P}:\{\mathtt{P},\mathtt{T1}\}\ \mathtt{a}\ ||\ \mathtt{b}\ \{\mathtt{Q},\mathtt{T2}\}}\qquad(28)$$

5 A validation condition generator

The logical rules of the previous section (in weakest precondition form) have been incorporated in a PROLOG program. We express the semantics of `wait`, for example, as a PROLOG relation

```
(s,R):{H1⊢P,T1}: wait X :{H2⊢Q,T2}
```

between weakest precondition `P` at time `T1` under hypotheses `H1`, and postcondition `Q` at time `T2` under hypotheses `H2`.

The $\vdash$ symbol denotes the relation of logical derivation between a proposition and hypotheses. The hypotheses are used as a sink for unproved lemmas. The modality has postcondition `Q` and `T1` and `T2` as inputs (for the *process* semantics $\mathcal{P}$), and precondition `P` as an output. `H2` serves as a working accumulator of necessary hypotheses, initialized to the empty list. `H1` is an output – the list of necessary precondition hypotheses. The *statement* semantics $\mathcal{S}\rho$ can have `T1` either in input or output mode.

This is the self-oscillator `a` of earlier sections:

```
L:X : a : Y :- L:X : process([x],a1⨾a2) : Y .
L:X : a1: Y :- L:X : x ⇐ 1- x@0 after 1 : Y .
L:X : a2: Y :- L:X : wait on x : Y .
```

and in response to a query about the preconditions at time `0` required to force consecutive unequal outputs between times `2` and `3`, for example (and a whole number of cycles completed by time `3`):

```
|?-  p:{H⊢P,0} : a : {[]⊢ x@−1≠ x@0,3}.

H = [ x@0≠1- x@0, 1- x@0≠1-(1- x@0) ]
P = 1-(1- x@0)≠1-(1-(1- x@0))
```

which shows the preconditions as being a proof of a tautology `P` from a set of tautologous hypotheses `H` (extra reduction rules can easily be incorporated to make this plainer), provided that $\mathtt{x@0}\neq 0.5$. Other alternatives are also offered, and to each a proof can be easily supplied, so we can verify that $\mathtt{x@2}\neq\mathtt{x@3}$.

6 Conclusion

Denotational and axiomatic semantics for VHDL have been given and related to each other. The logic has been implemented in PROLOG as a precondition generator for use in validation. Our denotational model sees VHDL statements as side effects on an infinite schedule of past and future states called a *world line*.

$$\frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T}-\mathtt{n}\}\ \mathtt{skip}\ \{\odot^{\mathtt{n}}\mathtt{Q}, \mathtt{T}-\mathtt{n}\}}{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T}-\mathtt{n}\}\ \mathtt{wait\ for\ n}\ \{\mathtt{Q}, \mathtt{T}\}} \qquad \frac{}{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{skip}\ \{\mathtt{Q}, \mathtt{T}\}}[\mathtt{P} \Rightarrow \mathtt{Q}] \tag{16}$$

$$\frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T}-1\}\ \mathtt{wait\ for\ 1}\ \{\mathtt{Q}, \mathtt{T}\}}{\mathcal{S}\rho : \{\mathtt{P}\ \&\ \mathtt{Xs} \neq \odot\mathtt{Xs}, \mathtt{T}-1\}\ \mathtt{wait\ on\ Xs}\ \{\mathtt{Q}, \mathtt{T}\}} \qquad \frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{wait\ for\ 1} \fatsemi \mathtt{wait\ on\ Xs}\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}\rho : \{\mathtt{P}\ \&\ \mathtt{Xs} = \odot\mathtt{Xs}, \mathtt{T1}\}\ \mathtt{wait\ on\ Xs}\ \{\mathtt{Q}, \mathtt{T2}\}} \tag{17}$$

$$\frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{a}\ \{\mathtt{Q}, \mathtt{T}\} \quad \mathcal{S}\rho : \{\mathtt{Q}, \mathtt{T}\}\ \mathtt{b}\ \{\mathtt{R}, \mathtt{T2}\}}{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{a} \fatsemi \mathtt{b}\ \{\mathtt{R}, \mathtt{T2}\}} \tag{18}$$

$$\frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{skip}\ \{\ldots[\odot^{\mathtt{dly}+1}\mathtt{x}{:=}\mathtt{y}][\odot^{\mathtt{dly}}\mathtt{x}{:=}\mathtt{y}]\mathtt{Q}, \mathtt{T}\}}{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{send\ dly\ x\ y}\ \{\mathtt{Q}, \mathtt{T}\}} \tag{19}$$

$$\frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{b}_1\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}\rho : \{\mathtt{P}\ \&\ \mathtt{x} \neq 0, \mathtt{T1}\}\ \mathtt{if\ x\ then\ b}_1\ \mathtt{else\ b}_0\ \{\mathtt{Q}, \mathtt{T2}\}} \qquad \frac{\mathcal{S}\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{b}_0\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}\rho : \{\mathtt{P}\ \&\ \mathtt{x} = 0, \mathtt{T1}\}\ \mathtt{if\ x\ then\ b}_1\ \mathtt{else\ b}_0\ \{\mathtt{Q}, \mathtt{T2}\}} \tag{20}$$

Figure 5: Hoare logic for terminating execution of VHDL statements.

$$\frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{a}\ \{\mathtt{Q}, \mathtt{T}\} \quad \mathcal{S}'\rho : \{\mathtt{Q}, \mathtt{T}\}\ \mathtt{b}\ \{\mathtt{R}, \mathtt{T2}\}}{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{a} \fatsemi \mathtt{b}\ \{\mathtt{R}, \mathtt{T2}\}} \qquad \frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{a}\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{a} \fatsemi \mathtt{b}\ \{\mathtt{Q}, \mathtt{T2}\}} \tag{21}$$

$$\frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T}-1\}\ \mathtt{wait\ for\ 1}\ \{\mathtt{Q}, \mathtt{T}\}}{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T}-1\}\ \mathtt{wait\ on\ x}\ \{\mathtt{Q}, \mathtt{T}\}} \qquad \frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{skip}\ \{\mathtt{Q}, \mathtt{T}\}}{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{wait\ on\ x}\ \{\mathtt{Q}, \mathtt{T}\}} \tag{22}$$

$$\frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{skip}\ \{\mathtt{Q}, \mathtt{T}\}}{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T}\}\ \mathtt{wait\ for\ 1}\ \{\mathtt{Q}, \mathtt{T}\}} \qquad \frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{wait\ for\ 1;\ wait\ for\ n}\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{wait}\ \mathtt{n}+1\ \mathtt{x}\ \{\mathtt{Q}, \mathtt{T2}\}} \tag{23}$$

$$\frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{b}_1\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}'\rho : \{\mathtt{P}\ \&\ \mathtt{x} \neq 0, \mathtt{T1}\}\mathtt{if\ x\ then\ b}_1\ \mathtt{else\ b}_0\{\mathtt{Q}, \mathtt{T2}\}} \qquad \frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{b}_0\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}'\rho : \{\mathtt{P}\ \&\ \mathtt{x} = 0, \mathtt{T1}\}\mathtt{if\ x\ then\ b}_1\ \mathtt{else\ b}_0\{\mathtt{Q}, \mathtt{T2}\}} \tag{24}$$

$$\frac{\mathcal{S}'\rho : \{\mathtt{P}, \mathtt{T1}\}\ \mathtt{b;\ while\ x\ do\ b}\ \{\mathtt{Q}, \mathtt{T2}\}}{\mathcal{S}'\rho : \{\mathtt{P}\ \&\ \mathtt{x} \neq 0, \mathtt{T1}\}\mathtt{while\ x\ do\ b}\{\mathtt{Q}, \mathtt{T2}\}} \tag{25}$$

Figure 6: Hoare logic for execution of VHDL statements until suspension.

Many approaches to semantics for VHDL are now appearing, based on various formalisms: Petri nets [3, 5], Boyer-Moore logic [1], process algebras, etc. A pioneer work was [7], giving an operational semantics. But, in comparison with these, our approach is uniquely declarative and direct.

References

[1] D.D. Borrione, L.V. Pierre, and A.M. Salem. Formal verification of VHDL descriptions in Boyer-Moore. In J. Mermet, editor, *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*. Kluwer, 1992.

[2] P.T. Breuer, L. Sánchez, and C. Delgado Kloos. A clean formal semantics for VHDL. In *European Design and Automation Conference '94*, 1994.

[3] W. Damm et al. A formal semantics for VHDL based on interpreted Petri Nets. Technical report, University of Oldenburg, Germany, 1992.

[4] Institute of Electrical and Electronics Engineers, 345 E. 47th St., New York, *IEEE Standard VHDL Language Reference Manual*, 1988. Std 1076-1987.

[5] S. Olcoz and J.M. Colom. Petri net based analysis of VHDL descriptions. In *2nd International Conference EuroVHDL 91*, Sweden, September 1991.

[6] L. Sánchez. Una semántica formal para VHDL mediante streams. Technical report, ETSI Telecomunicación, Universidad Politécnica de Madrid, Ciudad Universitaria, Madrid, Spain, December 1992.

[7] J.P. van Tassel. A formalization of the VHDL simulation cycle. Technical Report 249, University of Cambridge, Computer Laboratory, UK, 1992.