4. VHDL Describes Behaviour

In Section 1.2 we stated that the behaviour of a digital system could be described in terms of programming language notation. The familiar sequential programming language aspects of VHDL were covered in detail in Chapter 2. In this chapter, we describe how these are extended to include statements for modifying values on signals, and means of responding to the changing signal values.

4.1. Signal Assignment

A signal assignment schedules one or more transactions to a signal (or port). The syntax of a signal assignment is:

\[
\text{signal_assignment_statement} ::= \text{target} <= [\text{transport}] \text{waveform} ;
\]
\[
\text{target} ::= \text{name} | \text{aggregate}
\]
\[
\text{waveform} ::= \text{waveform_element} [ , \text{waveform_element} ]
\]
\[
\text{waveform_element} ::= \text{value_expression} [ \text{after time_expression} ]
\]
\[
| \text{null} [ \text{after time_expression} ]
\]

The target must represent a signal, or be an aggregate of signals (see also variable assignments, Section 2.4.1). If the time expression for the delay is omitted, it defaults to 0 fs. This means that the transaction will be scheduled for the same time as the assignment is executed, but during the next simulation cycle.

Each signal has associated with it a projected output waveform, which is a list of transactions giving future values for the signal. A signal assignment adds transactions to this waveform. So, for example, the signal assignment:

\[
s <= '0' \text{ after } 10 \text{ ns};
\]

will cause the signal enable to assume the value true 10 ns after the assignment is executed. We can represent the projected output waveform graphically by showing the transactions along a time axis. So if the above assignment were executed at time 5 ns, the projected waveform would be:

```
\[\begin{array}{c}
15\text{ns} \\
10\text{ns} \\
0
\end{array}\]
```

When simulation time reaches 15 ns, this transaction will be processed and the signal updated.

Suppose then at time 16 ns, the assignment:

\[
s <= '1' \text{ after } 4 \text{ ns}, '0' \text{ after } 20 \text{ ns};
\]

were executed. The two new transactions are added to the projected output waveform:
Note that when multiple transactions are listed in a signal assignment, the delay times specified must be in ascending order.

If a signal assignment is executed, and there are already old transactions from a previous assignment on the projected output waveform, then some of the old transactions may be deleted. The way this is done depends on whether the word transport is included in the new assignment. If it is included, the assignment is said to use transport delay. In this case, all old transactions scheduled to occur after the first new transaction are deleted before the new transactions are added. It is as though the new transactions supercede the old ones. So given the projected output waveform shown immediately above, if the assignment:

\[ s \triangleq \text{transport 'Z' after 10 ns}; \]

were executed at time 18 ns, then the transaction scheduled for 36 ns would be deleted, and the projected output waveform would become:

\[ 20\text{ns} \quad 28\text{ns} \]

\[ '1' \quad 'Z' \]

The second kind of delay, inertial delay, is used to model devices which do not respond to input pulses shorter than their output delay. An inertial delay is specified by omitting the word transport from the signal assignment. When an inertial delay transaction is added to a projected output waveform, firstly all old transactions scheduled to occur after the new transaction are deleted, and the new transaction is added, as in the case of transport delay. Next, all old transactions scheduled to occur before the new transaction are examined. If there are any with a different value from the new transaction, then all transactions up to the last one with a different value are deleted. The remaining transactions with the same value are left.

To illustrate this, suppose the projected output waveform at time 0 ns is:

\[ 10\text{ns} \quad 15\text{ns} \quad 20\text{ns} \quad 30\text{ns} \]

\[ '1' \quad '0' \quad '1' \quad 'Z' \]

and the assignment:

\[ s \triangleq '1' \text{ after 25 ns}; \]

is executed also at 0 ns. Then the new projected output waveform is:

\[ 20\text{ns} \quad 25\text{ns} \]

\[ '1' \quad '1' \]

When a signal assignment with multiple waveform elements is specified with inertial delay, only the first transaction uses inertial delay; the rest are treated as being transport delay transactions.

4.2. Processes and the Wait Statement

The primary unit of behavioural description in VHDL is the process. A process is a sequential body of code which can be activated in response to changes in state. When more than one process is activated at the same
time, they execute concurrently. A process is specified in a process statement, with the syntax:

```vhdl
process_statement ::= [ process_label : ]
  process [ ( sensitivity_list ) ]
  process_declarative_part
  begin
  process_statement_part
  end process [ process_label ];
```

```vhdl
process_declarative_part ::= [ process_declarative_item ]
```

```vhdl
process_declarative_item ::= subprogram_declaration
| subprogram_body
| type_declaration
| subtype_declaration
| constant_declaration
| variable_declaration
| alias_declaration
| use_clause
```

```vhdl
process_statement_part ::= [ sequential_statement ]
```

```vhdl
sequential_statement ::= wait_statement
| assertion_statement
| signal_assignment_statement
| variable_assignment_statement
| procedure_call_statement
| if_statement
| case_statement
| loop_statement
| next_statement
| exit_statement
| return_statement
| null_statement
```

A process statement is a concurrent statement which can be used in an architecture body or block. The declarations define items which can be used locally within the process. Note that variables may be defined here and used to store state in a model.

A process may contain a number of signal assignment statements for a given signal, which together form a driver for the signal. Normally there may only be one driver for a signal, and so the code which determines a signals value is confined to one process.

A process is activated initially during the initialisation phase of simulation. It executes all of the sequential statements, and then repeats, starting again with the first statement. A process may suspended itself by executing a wait statement. This is of the form:

```vhdl
wait_statement ::= wait [ sensitivity_clause ] [ condition_clause ] [ timeout_clause ];
```

```vhdl
sensitivity_clause ::= on sensitivity_list
```

```vhdl
sensitivity_list ::= signal_name [ , signal_name ]
```

```vhdl
condition_clause ::= until condition
```

```vhdl
timeout_clause ::= for time_expression
```

The sensitivity list of the wait statement specifies a set of signals to which the process is sensitive while it is suspended. When an event occurs
on any of these signals (that is, the value of the signal changes), the process resumes and evaluates the condition. If it is true or if the condition is omitted, execution proceeds with the next statement, otherwise the process resuspends. If the sensitivity clause is omitted, then the process is sensitive to all of the signals mentioned in the condition expression. The timeout expression must evaluate to a positive duration, and indicates the maximum time for which the process will wait. If it is omitted, the process may wait indefinitely.

If a sensitivity list is included in the header of a process statement, then the process is assumed to have an implicit wait statement at the end of its statement part. The sensitivity list of this implicit wait statement is the same as that in the process header. In this case the process may not contain any explicit wait statements.

An example of a process statements with a sensitivity list:

```vhdl
process (reset, clock)
  variable state : bit := false;
begin
  if reset then
    state := false;
  elsif clock = true then
    state := not state;
  end if;
  q <= state after prop_delay;
  -- implicit wait on reset, clock
end process;
```

During the initialization phase of simulation, the process is activated and assigns the initial value of state to the signal q. It then suspends at the implicit wait statement indicated in the comment. When either reset or clock change value, the process is resumed, and execution repeats from the beginning.

The next example describes the behaviour of a synchronization device called a Muller-C element used to construct asynchronous logic. The output of the device starts at the value ‘0’, and stays at this value until both inputs are ‘1’, at which time the output changes to ‘1’. The output then stays ‘1’ until both inputs are ‘0’, at which time the output changes back to ‘0’.

```vhdl
muller_c_2 : process
begin
  wait until a = '1' and b = '1';
  q <= '1';
  wait until a = '0' and b = '0';
  q <= '0';
end process muller_c_2;
```

This process does not include a sensitivity list, so explicit wait statements are used to control the suspension and activation of the process. In both wait statements, the sensitivity list is the set of signals a and b, determined from the condition expression.

### 4.3. Concurrent Signal Assignment Statements

Often a process describing a driver for a signal contains only one signal assignment statement. VHDL provides a convenient short-hand notation, called a concurrent signal assignment statement, for expressing such processes. The syntax is:
4. VHDL Describes Behaviour

concurrent_signal_assignment_statement ::= 
[ label : ] conditional_signal_assignment 
| [ label : ] selected_signal_assignment

For each kind of concurrent signal assignment, there is a 
corresponding process statement with the same meaning.

4.3.1. Conditional Signal Assignment

A conditional signal assignment statement is a shorthand for a process 
containing signal assignments in an if statement. The syntax is:

conditional_signal_assignment ::= target <= options conditional_waveforms ;
options ::= [ guarded ] [ transport ]
conditional_waveforms ::= 
( waveform when condition else )
waveform

Use of the word guarded is not covered in this booklet. If the word transport 
is included, then the signal assignments in the equivalent process use 
transport delay.

Suppose we have a conditional signal assignment:

s <= waveform_1 when condition_1 else 
   waveform_2 when condition_2 else 
   ... 
   waveform_n;

Then the equivalent process is:

process
  if condition_1 then
   s <= waveform_1;
  elsif condition_2 then
   s <= waveform_2;
  elsif ...
  else
   s <= waveform_n;
  wait [ sensitivity_clause ];
end process;

If none of the waveform value expressions or conditions contains a 
reference to a signal, then the wait statement at the end of the equivalent 
process has no sensitivity clause. This means that after the assignment is 
made, the process suspends indefinitely. For example, the conditional 
assignment:

reset <= '1', '0' after 10 ns when short_pulse_required else 
   '1', '0' after 50 ns;

schedules two transactions on the signal reset, then suspends for the rest of 
the simulation.

On the other hand, if there are references to signals in the waveform 
value expressions or conditions, then the wait statement has a sensitivity 
list consisting of all of the signals referenced. So the conditional 
assignment:

mux_out <= 'Z' after Tpd when en = '0' else 
in_0 after Tpd when sel = '0' else 
in_1 after Tpd;

is sensitive to the signals en and sel. The process is activated during the 
initialization phase, and thereafter whenever either of en or sel changes 
value.
The degenerate case of a conditional signal assignment, containing no conditional parts, is equivalent to a process containing just a signal assignment statement. So:

\[ s \leftarrow \text{waveform}; \]
is equivalent to:

\[
\begin{align*}
\text{process} \\
s \leftarrow \text{waveform}; \\
\text{wait} \{ \text{sensitivity_clause} \}; \\
\text{end process};
\end{align*}
\]

4.3.2. Selected Signal Assignment

A selected signal assignment statement is a shorthand for a process containing signal assignments in a case statement. The syntax is:

\[
\text{selected_signal_assignment} ::= \\
\quad \text{with expression select} \\
\quad \quad \text{target} \leftarrow \text{options selected_waveforms}; \\
\text{selected_waveforms} ::= \\
\quad \{ \text{waveform when choices}, \} \\
\quad \text{waveform when choices} \\
\quad \text{choices} ::= \text{choice} \{ | \text{choice} \}
\]

The options part is the same as for a conditional signal assignment. So if the word transport is included, then the signal assignments in the equivalent process use transport delay.

Suppose we have a selected signal assignment:

\[
\text{with expression select} \\
\quad s \leftarrow \text{waveform}_1 \text{ when choice_list}_1, \\
\quad \quad \text{waveform}_2 \text{ when choice_list}_2, \\
\quad \quad \ldots \\
\quad \quad \text{waveform}_n \text{ when choice_list}_n;
\]

Then the equivalent process is:

\[
\begin{align*}
\text{process} \\
\quad \text{case expression is} \\
\quad \quad \text{when choice_list}_1=> \\
\quad \quad \quad s \leftarrow \text{waveform}_1; \\
\quad \quad \text{when choice_list}_2=> \\
\quad \quad \quad s \leftarrow \text{waveform}_2; \\
\quad \quad \ldots \\
\quad \quad \text{when choice_list}_n=> \\
\quad \quad \quad s \leftarrow \text{waveform}_n; \\
\quad \text{end case}; \\
\quad \text{wait} \{ \text{sensitivity_clause} \}; \\
\quad \text{end process};
\end{align*}
\]

The sensitivity list for the wait statement is determined in the same way as for a conditional signal assignment. That is, if no signals are referenced in the selected signal assignment expression or waveforms, the wait statement has no sensitivity clause. Otherwise the sensitivity clause contains all the signals referenced in the expression and waveforms.

An example of a selected signal assignment statement:

\[
\begin{align*}
\text{with alu_function select} \\
\quad \text{alu_result} \leftarrow \text{op1} + \text{op2} \text{ when alu_add | alu_incr,} \\
\quad \quad \text{op1} - \text{op2} \text{ when alu_subtract,} \\
\quad \quad \text{op1 and op2 when alu_and,} \\
\quad \quad \text{op1 or op2 when alu_or,} \\
\quad \quad \text{op1 and not op2 when alu_mask;}
\end{align*}
\]
In this example, the value of the signal `alu_function` is used to select which signal assignment to `alu_result` to execute. The statement is sensitive to the signals `alu_function`, `op1` and `op2`, so whenever any of these change value, the selected signal assignment is resumed.