Signal Attributes

We introduced the idea of attributes of types, which give information about allowed values for the type.

We can also refer to attributes of signals to find information about their history of transactions and events of signals. Given a signal S, and a value T of type time, VHDL defines the following attributes:

<table>
<thead>
<tr>
<th>Signal's Attribute</th>
<th>Meaning</th>
<th>Kind</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S'delayed(T)</td>
<td>a signal with same values as S but delayed by time T</td>
<td>Signal</td>
<td>As S</td>
</tr>
<tr>
<td>S'stable(T)</td>
<td>a boolean signal that is true if there has been no event on S in the time interval T up to current time</td>
<td>Signal</td>
<td>boolean</td>
</tr>
<tr>
<td>S'quiet(T)</td>
<td>boolean signal that is true if there has been no transaction on S in the time interval T up to current time</td>
<td>Signal</td>
<td>boolean</td>
</tr>
</tbody>
</table>
In order to understand the table you have to know the difference between Event and Transaction.

We call the change in signal from 0 to 1 or 1 to 0 an event. While a transaction is when signals get assigned a new value, even without change: 0 to 0, 1 to 1, 1 to 0 or 0 to 1 is considered a transaction.

**Z<=’1’ after 5ns, ’0’ after 10 ns**

---

**Event and transaction**

**Z<=’1’ after 5ns, ’1’ after 10 ns**

---

Not an event but considered a transaction
Example

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ev is
end ev;

architecture ev of ev is
  signal z, z_transaction, z_delayed: bit;
  signal z_stable, z_quiet: boolean;
begin
  z <= '1' after 5 ns, '0' after 10 ns, '0' after 15 ns, '1' after 20 ns;
  z_transaction <= z'transaction;
  z_delayed <= z'delayed(7 ns);
  z_stable <= z'stable(5 ns);
  z_quiet <= z'quiet(5 ns);
end ev;
```

Waveform

<table>
<thead>
<tr>
<th>Signal name</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z_transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z_delayed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z_stable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z_quiet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find the following values

z'last_value, z'last_active, and z'last_event: at time 20 ns
z'last_value = '0'
z'last_active = 5 ns
z'last_event = 10 ns
Wait statements

A wait statement is a sequential statement that specifies when processes respond to changes in signal values.

- **wait on signal(s):**

  A sensitivity clause, it results in the process being suspended until an event takes place on any of the signals (sensitivity list).

- **Wait for timeout:**

  A timeout clause results in the process being suspended for the time specified in the construct.

- **wait until condition:**

  A condition clause results in the process being suspended until the condition is true. The condition expression is tested while the process is suspended to determine whether to resume the process. So if the condition is true during the execution of the wait statement, the process will still suspended until the appropriate signals change and cause the condition to be true again. This happens because the process is sensitive to signals events not transactions. If no sensitivity list exists, the condition is tested whenever an event occurs on any of the signals mentioned in the condition.

Any combination of the wait statements can be used, or none of them.

```vhls
wait on clk until reset = '0';
```

Here the condition is tested on each change on clk (sensitivity clause) regardless of any change on reset (condition)

```vhls
wait until trigger = '1' for 1 ms;
```

causes the executing process to suspend until trigger changes to '1', or until 1 ms of simulation time has elapsed.
wait;

It causes the executing process to suspend for the remainder of the simulation, `suspend forever`, it is useful in a process whose purpose is to generate a stimuli for a simulation.

---

**Keep in Mind That...**

- A process may contain either a sensitivity list or a wait statement, but not both.

---

**Example**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity clock_gen is
  port(clk : out bit);
end clock_gen;

architecture behav of clock_gen is
Constant TP_W : time := 10 ns ;
begin
clk_gen: process is
begin
  clk <= '1' after TP_W,'0' after 2*TP_W;
  wait for 2*TP_W;
  -- you can replace the above wait statement with
  -- wait until clk = '0'; since it will be '0' every 2*TP &
end process;
end behav;
```

**Waveform**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Stimulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Transport and Inertial Delay Mechanisms

The following two rules are for both transport and inertial delay:

1. All old transactions that are projected to occur at or after the time at which the earliest new transaction is projected to occur are deleted from the output move form.

2. The new transactions are then appended to the output in the order of their projected occurrence.

Example:

```plaintext
Signal s: integer:=0;
Process
Begin
S <= transport 1 after 1 ns;
S <= transport 2 after 2 ns;
Wait;
End process;
```

Example:

```plaintext
S <= transport 1 after 1 ns, 3 after 3ns, 5 after 5 ns;
S <= transport 4 after 4 ns;
Wait;
```

In the example, the first green is the oldest and the blue is the newest. The other colors are between the two.

Example:

```plaintext
1
2
S
1ns
2ns
```
Example:

S <= transport 1 after 1 ns, 3 after 3ns, 5 after 5 ns;
S <= transport 4 after 4 ns;

Wait for 5 ns;

S <= transport 4 after 4 ns;
S <= transport 4 after 4 ns, 6 after 6 ns;

S <= transport 4 after 4 ns +5 ns;
S <= transport 4 after 4 +5 ns, 6 after 6+5 ns;

S <= transport 4 after 9 ns;
S <= transport 4 after 9 ns, 6 after 11 ns;

ومن نضيف للشعل القديم:

تحذفناها لأنو أول وحدة في الأزرق بتساويها في القيمة واكنا قلنا بنحذف أي شي بيساوي أو أكبر من أول وحدة بنضيفها

في الجملة الثانية
Rules on inertial delay only (Default mechanism)

3. All new transactions are marked.
4. An old transaction is marked if the time at which it is projected to occur is less than the time at which the first new transaction is projected to occur minus the pulse rejection limit.
5. If \( \text{time old} < (\text{time first new transaction} - \text{time reject limit}) \) then mark the old transaction.
6. For each remaining unmarked old transactions. The old transaction is marked if it immediately proceeds (marked transaction) and its value is the same as that of the marked transaction.
7. The transactions that determined the current value is marked.
8. All unmarked transactions are deleted from the output waveform.

Example:

\[
S \leq 1 \text{ after } 1 \text{ ns}; \\
S \leq 2 \text{ after } 2 \text{ ns};
\]

\[
S \leq 1 \text{ after } 1 \text{ ns}, 3 \text{ after } 3 \text{ ns}, 5 \text{ after } 5 \text{ ns}; \\
S \leq 3 \text{ after } 4 \text{ ns}, 4 \text{ after } 5 \text{ ns};
\]
Homework Exercises

1)  \[ \begin{array}{cccc}
5 & 15 & 17 & 30 \\
\hline
A & \_ & \_ & \_ \\
\end{array} \]

Given the above signal draw the signal using simulation and then find the output of the following signal assignments:

- \( B \leq A \) after 5 ns;
- \( C \leq \text{transport } A \) after 5 ns;

2) find the output of ADDR_BUS after executing the following code (manually).

a) process
begin
ADDR_BUS <= 1 after 5 ns, 6 after 10 ns, 12 after 19 ns;
ADDR_BUS <= 6 after 12 ns, 20 after 19 ns;
wait;
end process;
b)

$S \leftarrow \text{transport 1 after 1 ns, 6 after 3ns, 5 after 7 ns;}
S \leftarrow \text{transport 3 after 3 ns;}

\text{Wait for 7 ns;}

S \leftarrow \text{transport 13 after 1 ns;}
S \leftarrow \text{transport 4 after 4 ns, 7 after 7 ns;}

\text{And check both part using simulation}