**INTRODUCTION**

**VHDL** - The VHSIC Hardware Description Language  
**VHSIC** - Very High Speed Integrated Circuits

**Characteristics of VHDL:**

- **Public Availability** - VHDL was developed under a Government contract and is now an IEEE standard (IEEE-1076).
- **Design Methodology and Design Technology Support**  
  Support many different methodologies (top-down vs library-based)  
  Support different design technologies (i.e. synchronous vs asynchronous, PLA vs random logic)
- **Technology and Process Independence**  
  Provides abstraction capabilities that facilitate the insertion of new technologies (e.g., CMOS, nMOS, GeAs) into existing designs
- **Wide Range of Descriptive Capability**  
  Supports behavioral description from digital system level to the gate level.
- **Design Exchange**  
  VHDL is a standard (IEEE-1076), VHDL models are guaranteed to run on any system that conforms to that standard.
- **Large Scale Design and Design Re-use**  
  Support design decomposition, design sharing, experimentation and design management.
- **Government Support**  
  The VHSIC Program Office developed VHDL with a clear vision of how DOD wanted to use VHDL.

DOD is currently requiring VHDL descriptions on all contracts that develop ASIC's

**1.1 A Model of Behavior**

1. A digital device is a discrete system, i.e. one which transforms discrete-valued input into discrete valued output. It does this by performing a number of operations or transformations on the input values.
2. In VHDL, each operation is a process, and the pathways in which values are passed through the system as signals.
3. A process can be viewed as a program; it is constructed out of procedural statements and can call subprograms much as a program written in a general purpose procedural language like Pascal or C.
4. In VHDL, all processes in a model are said to be executing concurrently. That is a VHDL model is a collection of independent programs running in parallel.
5. Signal is a mechanism defined to handle communication between processes. Signals define a data pathway between two processes.
6. Processes continue to execute until they are suspended. Once suspended, a process can be reactivated in a number of ways:
   A. After a specified maximum timeout.  
   B. When some change in the state of the system takes place.  
      a) VHDL provides a means for a process to express its sensitivity to the value of a data pathway.  
         These pathways are called sensitivity channels. When the value on a sensitivity channel changes, the process is reactivated.  
      b) The wait statement is used to designate any timeout conditions and sensitivity channels for a process.

```
Xor1: PROCESS(In1,In2)
```
BEGIN
  Out1 <= In1 Xor In2;
END PROCESS;

Xor2: PROCESS
BEGIN
  Out1 <= In1 Xor In2;
  Wait on In1,In2
END PROCESS;

1.2 A Model of Time

Simulation Time - defines when events occur during the course of simulation.

Scheduling transaction - when a process generate a value on a data pathway it may also designate the amount of time before the value is sent over the pathway.

Signal Driver - is a set of time/value pairs which hold the value of each transaction and the time at which the transaction should occur.

Simulation Cycle - A two stage model based on the stimulus and response behavior of digital hardware.

Stage One - values are propagated through the data pathways (signals). This stage is complete when all data pathways which are scheduled to obtain new values at the current simulation time are updated.

Stage Two - those active elements (processes) which receive information on their sensitivity channels are exercised until they suspend. This stage is completed when all active processes are suspended.

At the completion of the simulation cycle, the simulation clock is set to the next simulation time at which a transaction is to occur and the cycle is started again.

There is always some delay between the time a process puts a value on a data pathway and the time at which the data pathway reflects that value. If no delay is given in the assignment of a value to the data pathway a delta delay is used.

Delta delay does not update the time of the simulation clock but does require the passing of a new simulation cycle.

An Event is said to have occurred, if the new value being assigned to a signal is different from previous value of the signal.

• When a process is sensitive to a signal, it is sensitive to events on that signal, not to general transactions.

• When a process needs to become active when there is a transaction (i.e. even if the new value of the signal is the same as the old value) the process should be made sensitive to the signal-valued attribute Transaction.
  
  Wait on Internal1'Transaction, Internal2'Transaction

1.3 A Model of Structure

1. Models often contain conceptual partitions which can be used to decompose the model into functionally related sections. This decomposition is called the Structure of the model.
2. Digital devices are designed by combining a number of sub-devices together and tying (wiring) the sub-devices together.
3. When a discrete system ties together two subsystems it is really connecting a data pathway from one subsystem to the data pathway of the other subsystem. In this way, the two subsystems can communicate with each other.
4. These connections are called ports and they have some special characteristics. Port represents a declaration of a signal and, therefore, of a data pathway.

1.4 Model Development

Specification:

The highest-level specification is usually written expression of an idea. It may be incomplete, poorly expressed, or not even capable of being implemented.

Example:

Create an entity which receives two digital signals and which outputs a single signal. If both input signals are high the output signal is to be high. For any other combination of high or low inputs the output is to be low.

Analysis:

Once given a specification, the modeler begins to provide definition to the problem through analysis of the design. Truth tables, flow charts, and other means of expressing logic constructs can be used.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Design:

- Basic modeling unit is called a Design Entity
- It may represent an entire system, a board, a chip, or a gate.

Design Entity

- Consists of an Entity Declaration and an Architecture Body

Entity Declaration:

- Names entity and defines interface between entity and its environment.
- Format:

```
ENTITY entity_name IS
  PORT(port_list);
END [entity_name];
```

- Example:

```
ENTITY and_gate IS
  PORT(In1,In2: IN BIT;Out1:OUT BIT);
END and_gate;
```
**Port Clause:**
- Identifies ports used by entity to communicate with its environment.
- Format:
  \[
  \text{PORT}(\text{namelist:mode type};...;\text{name_list:mode type});
  \]

**Port Mode:**
- Identifies direction of data flow through port.
- All ports must have an identified mode.
- Allowable modes:
  - **IN** - flow is into entity.
  - **OUT** - flow is out of entity.
  - **INOUT** - flow maybe either in or out.
  - **BUFFER** - in or out, ports only limited to one in source.

**Architecture Body:**
- Establishes relationship between inputs and outputs of design
- Format:

  \[
  \text{ARCHITECTURE body\_name OF entity\_name IS}
  
  -- declarative statements
  
  BEGIN
  
  -- activity statements
  
  END [body\_name];
  \]

Example:

ARCHITECTURE behavior OF and\_gate IS
BEGIN
  Out1 <= '0' WHEN In1='0' AND In2='0' ELSE '0' WHEN In1='0' AND In2='1' ELSE '0' WHEN In1='1' AND In2='0' ELSE '1' WHEN In1='1' AND In2='1' ELSE '0';
END behavior;

Complete Design Entity Model Example : and\_gate Model:

ENTITY and\_gate IS
  PORT(In1,In2: IN BIT;Out1:OUT BIT);
END and\_gate;

ARCHITECTURE behavior OF and\_gate IS
BEGIN
  Out1 <= '0' WHEN In1='0' AND In2='0' ELSE '0' WHEN In1='0' AND In2='1' ELSE '0' WHEN In1='1' AND In2='0' ELSE '1' WHEN In1='1' AND In2='1' ELSE '0';
END behavior;

The next phases in the development cycle are to enter the model into the testing environment and to test it.
Signal Assignment

- Assignment of waveform to signal.
- Format:
  
  \[
  \text{target\_signal} <= \text{waveform};
  \]

Example:

\[
\text{signal\_a} <= '0';
\]

<= only assignment operator which can be used to assign waveforms to signals.

Conditional Signal Assignment:

- Assigns waveforms to signals based on the validity of a condition.
- Format:
  
  \[
  \text{target\_signal} <= \text{waveform WHEN condition ELSE waveform}
  \]
  
- Must end with a waveform.
- May contain multiple "waveform WHEN condition" clauses.
- Condition must produce boolean value.

Example:

\[
\text{signal\_a} <= '0' \text{ WHEN } \text{In1}='1' \text{ ELSE '1'};
\]

1.5 Delay

- Time period between cause and effect.
- Delay selection: Inertial, Transport
- Internal delay: Delta
- 1 femtosecond (fs)=10E-15 second is the smallest unit of macro time measurable in VHDL.
- Macro units are referred to as time points.
- Micro units (delta) - is a "unit delay" time, any number of micro-units may exist between any time points.
- An infinite number of delta times add up to zero.
- All activated statements are evaluated and the results assigned one delta time later.

1.5.1 Inertial Delay:

- Characteristic of any system where there is a time lag between cause and effect.
- Inertial delay is modeled with after clause, and is the default mode.
- Format:

  \[
  \text{new\_signal} <= \text{signal\_expression AFTER time};
  \]

Example:

\[
\text{In1} <= '1',
\]

\[
'0' \text{ AFTER 250 ns},
\]

\[
'1' \text{ AFTER 280 ns};
\]

Signals whose duration is less than delay specified in AFTER clause are ignored.
ENTITY inertial_del IS
       --"inertial" is a reserved word, can’t use as entity name
END ENTITY inertial_del;

ARCHITECTURE behav OF inertial_del IS
    SIGNAL a, b :BIT;
BEGIN
    a <= b AFTER 10 ns;
    b<= '0',
    '1' AFTER 10ns,
    '0' AFTER 20ns,
'1' AFTER 25ns,
'0' AFTER 30ns;
END ARCHITECTURE behav;

* Note: the 5ns high pulse on b at 25ns will not be passed to a.

Rules of assigning transaction on the signal driver: All transactions which are scheduled to occur after the delay given in the inertial delay are discarded. For transaction which are scheduled to occur before the new transaction, there are two cases to consider. Case 1, if the value of a scheduled transaction is different from the value of the new transaction, the old transaction is discarded. Case 2, if the value is the same as the new transaction, the old transaction is left on the projected output waveform.

Example 1: the effect of reversing the order of signal assignment with Inertial delay

```
SIGNAL S:Integer:=0;
P1:
PROCESS
BEGIN
S<= 1 AFTER 1 ns;
S<= 2 AFTER 2 ns;
WAIT;
END PROCESS;
```

For process P1, the effects of execution of each assignment on signal driver for S are:
(1,1 ns) after S<= 1 AFTER 1 ns;
(2,2 ns) after S<= 2 AFTER 2 ns;

The second assignment overrides the first assignment because a new value is introduced to the driver with inertial delay. That is, the old transaction (1,1ns) is scheduled before the new transaction (2,2ns), different values discard the old transaction.

For process P2, we have:
(2,2 ns) after S<= 2 AFTER 2 ns;
(1,1 ns) after S<= 1 AFTER 1 ns;*

* That is, (1,1 ns) overrides the transaction (2, 2ns) which occurs after (1,1 ns).

Example 2: the effects of overlapping assignment statements.

```
SIGNAL S:Integer :=0;
P3:
PROCESS
BEGIN
S <= 1 AFTER 1 ns, 3 AFTER 3 ns, 5 AFTER 5 ns;
S <= 3 AFTER 4 ns, 4 AFTER 5 ns;
WAIT;
END PROCESS;
```

The first assignment places the following transactions on the driver for S:
(1,1 ns), (3,3 ns), (5,5 ns)
The second assignment removes the transaction (1,1 ns) because the new transaction (3,4 ns) has a different value component. The transaction (3,3 ns) is retained on the projected output waveform because the value component is the same as the value component of the new transaction (3,4 ns). The last transaction (5,5 ns) is removed because the time component is greater than the time component of the new transaction (3,4 ns).
Finally, the second transaction from the second assignment is added to the end of the projected output waveform of S. The resulting driver of S is:
(3,3 ns), (3,4 ns), (4,5 ns)

Another Example:

Describe a buffer having a rise time of 10 ns and a fall time of 14 ns.

ENTITY Buffer IS
  PORT(input: IN BIT; output: OUT BIT);
END Buffer;

ARCHITECTURE InertB OF Buffer IS
BEGIN
  output <= '1' AFTER 10 ns WHEN input = '1' ELSE
            '0' AFTER 14 ns;
END InertB;

<table>
<thead>
<tr>
<th>Input Event</th>
<th>Input Signal</th>
<th>Input Time(ns)</th>
<th>Output Signal</th>
<th>Output Time(ns)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0=&gt;1</td>
<td>0</td>
<td>0=&gt;1</td>
<td>10</td>
<td>=0+10=10(&lt;12)</td>
</tr>
<tr>
<td>2</td>
<td>1=&gt;0</td>
<td>12</td>
<td>1=&gt;0</td>
<td>26</td>
<td>=12+14=26(&gt;18)</td>
</tr>
<tr>
<td>3</td>
<td>0=&gt;1</td>
<td>18</td>
<td>0=&gt;1</td>
<td>28</td>
<td>=18+10=28(&gt;20)</td>
</tr>
<tr>
<td>4</td>
<td>1=&gt;0</td>
<td>20</td>
<td>1=&gt;0</td>
<td>34</td>
<td>=20+14=34(&gt;22)</td>
</tr>
<tr>
<td>5</td>
<td>0=&gt;1</td>
<td>22</td>
<td>0=&gt;1</td>
<td>32</td>
<td>=22+10=32(&lt;∞)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (ns)</th>
<th>Driver Transactions</th>
<th>Driver Scheduled Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1,10ns)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>(1,10ns) executed</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(0,26ns)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>(0,26ns), (1,28ns)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>(1,28ns), (0,34ns)</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>(0,34ns), (1,32ns)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>(1,32ns) executed</td>
<td></td>
</tr>
</tbody>
</table>
The output will only pass the input signal that is greater than or equal to the given delay period. That is, in this example any input with high period greater than or equal to 10ns will be passed, and low period greater than or equal to 14ns will be passed. In this example, only the first high period of 12ns will be passed. The others which fluctuate faster than the specified delays will be ignored. That is, the output will remained high.

1.5.2 Transport Delay:

- Passes signals regardless of duration, provided no time reversal in the projected output signal.
- Used when signals are delayed but `filtering effect produced by AFTER clause is unwanted.

Example:

```
transport_del
```

```
transport_del1.b
```
ENTITY transport_del IS
END ENTITY transport_del;

ARCHITECTURE behav OF transport_del IS
  SIGNAL a, b : BIT;
  BEGIN
    a <= TRANSPORT b AFTER 10 ns;
    b <= '0',
       '1' AFTER 10ns,
       '0' AFTER 20ns,
       '1' AFTER 25ns,
       '0' AFTER 30ns;
  END ARCHITECTURE behav;

*Note: both high pulses in b will be passed to a.

**Rules of assigning transaction on the signal driver:** All transactions which are scheduled to occur after the delay given in the transport delay are discarded.

**Example 1:** the effect of reversing the order of signal assignment with transport delay

```vhdl
SIGNAL S: Integer:=0;
BEGIN
  P1:
  PROCESS
  BEGIN
    S <= TRANSPORT 1 AFTER 1 ns;
    S <= TRANSPORT 2 AFTER 2 ns;
    WAIT;
  END PROCESS;
  P2:
  BEGIN
    S <= TRANSPORT 2 AFTER 2 ns;
    S <= TRANSPORT 1 AFTER 1 ns;
    WAIT;
  END PROCESS;
END PROCESS;
```

For process P1, the effects of execution of each assignment on signal driver for S are:
(1,1 ns) after S <= TRANSPORT 1 AFTER 1 ns;
(1,1 ns), (2,2 ns) after S <= TRANSPORT 2 AFTER 2 ns;
For process P2, we have:

(2, 2 ns) after S <= TRANSPORT 2 AFTER 2 ns;
(1, 1 ns) after S <= TRANSPORT 1 AFTER 1 ns;*

* That is, (1, 1 ns) overrides the transaction (2, 2 ns)

**Example 2:** the effects of overlapping assignment statements.

```
SIGNAL S: Integer := 0;
P3:
PROCESS
BEGIN
  S <= TRANSPORT 1 AFTER 1 ns, 3 AFTER 3 ns, 5 AFTER 5 ns;
  S <= TRANSPORT 4 AFTER 4 ns;
  WAIT;
END PROCESS;
```

The first assignment adds three transactions to the driver of signal S: (1, 1 ns), (3, 3 ns), (5, 5 ns).
The second assignment overwrites the last transaction of the previous assignment because the transaction (4, 4 ns) is scheduled before the transaction (5, 5 ns). The resulting driver of S is: (1, 1 ns), (3, 3 ns), (4, 4 ns).

Another Example:

Describe a buffer having a rise time of 10 ns and a fall time of 14 ns.

**ENTITY Buffer IS**
```
  PORT(input: IN BIT; output: OUT BIT);
END Buffer;
```

**ARCHITECTURE TransB OF Buffer IS**
```
BEGIN
  output <= TRANSPORT `1' AFTER 10 ns WHEN input = `1' ELSE
          `0' AFTER 14 ns;
END TransB;
```

In this implementation: When input changes to `1' at time T, an event `1' is projected on output at time T+10 ns. If input changes back to `0' t time units later, then an event `0' is projected on output at time T+t+14 ns. This implies that a positive input pulse becomes longer, while a negative input pulse becomes shorter at the output.

<table>
<thead>
<tr>
<th>Input Event</th>
<th>Input Signal</th>
<th>Input Time(ns)</th>
<th>Output Signal</th>
<th>Output Time(ns)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0=&gt;1</td>
<td>0</td>
<td>0=&gt;1</td>
<td>10</td>
<td>=0+10=10(&lt;12)</td>
</tr>
<tr>
<td>2</td>
<td>1=&gt;0</td>
<td>12</td>
<td>1=&gt;0</td>
<td>26</td>
<td>=12+14=26(&gt;18)</td>
</tr>
<tr>
<td>3</td>
<td>0=&gt;1</td>
<td>18</td>
<td>0=&gt;1</td>
<td>28</td>
<td>=18+10=28(&gt;20)</td>
</tr>
<tr>
<td>4</td>
<td>1=&gt;0</td>
<td>20</td>
<td>1=&gt;0</td>
<td>34</td>
<td>=20+14=34(&gt;22)</td>
</tr>
<tr>
<td>5</td>
<td>0=&gt;1</td>
<td>22</td>
<td>0=&gt;1</td>
<td>32</td>
<td>=22+10=32(&gt;∞)</td>
</tr>
</tbody>
</table>

**Time (ns) | Driver Transactions | Driver Scheduled Transaction**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 10ns)</td>
<td>(1, 10ns)</td>
</tr>
<tr>
<td>10</td>
<td>(1, 10ns) executed</td>
<td></td>
</tr>
</tbody>
</table>
Since the rise time is 4ns (=14-10) faster than the fall time, it has the effect of increasing the high period by 4ns and decreasing the low period by 4ns. With the transport delay, the output waveform is essentially a copy of the input waveform with the stretching of the high period and reduction of the low period. Except when there is a time reversal, which occurs at the input low period of 2ns from 20ns to 22ns, at the output this low period is reduced by 4ns causing a negative period or time reversal. That is, this low period will not be observed at the output.

If the rise and fall time are identical in a transport delay, the output is an exact copy of the input with a delay period given by the rise time (or fall time).

**To facilitate waveform generation**, the semantics of the inertial delay are such that if a number of elements are given on the waveform of an inertial assignment as in:

\[
S <= 1 \text{ AFTER } 1 \text{ ns}, 3 \text{ AFTER } 3 \text{ ns}, 5 \text{ AFTER } 5 \text{ ns};
\]

The elements after the first element are not considered to be inertial assignments. This is because the language requires that the waveform elements in a signal assignment must be ascending. Once the first transaction is on the driver, each subsequent transaction is guaranteed to be scheduled after all other transactions on the driver, which is the same behavior as in the transport case.

That makes the above statement different from the following process:

```vhdl
SIGNAL S:Integer := 0;
PROCESS
```
BEGIN
S <= 1 AFTER 1 ns;
S <= 3 AFTER 3 ns;
S <= 5 AFTER 5 ns;-- This will be the only one in the driver
END PROCESS

But the above statement is equivalent to the following process:

SIGNAL  S:Integer := 0;
PROCESS
BEGIN
S <= 1 AFTER 1 ns;
S <= TRANSPORT 3 AFTER 3 ns;
S <= TRANSPORT 5 AFTER 5 ns;
END PROCESS

In summary:
Each time a new output event is projected, the entire event queue is first cleared. That is, the inertial delay model preempts the event queue in both directions, whereas the transport delay model preempted only future events (it can have multiple output events scheduled, as long as there is no event scheduled to occur after the last output event to be scheduled, no time reversal)

Signal Drivers:
• Containers for projected waveforms of signals.
• Created when a signal assignment is made.

Example:
-- filling a driver
Clock <= '0',
   '1' AFTER 5 ns,
   '0' AFTER 10ns,
   '1' AFTER 15ns;

Filling Drivers:
• Negative time is not allowed.
• Assignments within a single signal assignment must be made in ascending time order.
• Assignments to a single signal by multiple concurrent statements create multiple drivers which must be "resolved".

Example:
signal_out <= signal_in_1 AFTER 10ns;
signal_out <= signal_in_2 AFTER 10ns;

Multiple executions of assignment statement modify projected waveforms in driver.

1.6 What is VHDL-AMS ?

IEEE Std. 1076-1993: VHDL (VHSIC Hardware Description Language) supports the description and simulation of event-driven systems.

IEEE Std. 1076.1-1999: Extension to VHDL to support the description and simulation of analog and mixed-signal circuits and systems.

IEEE Std. 1076.1-1999 together with IEEE Std. 1076-1993 is informally known as VHDL-AMS

VHDL-AMS is a strict superset of IEEE Std. 1076-1993. Any model valid in VHDL 1076 is valid in VHDL-AMS and yields the same simulation results.

### 1.6.1 VHDL-AMS: Add new simulation model supporting continuous behavior

- Continuous models based on differential algebraic equations (DAEs)
- DAEs solved by dedicated simulation kernel: the analog solver
- Handling of initial conditions, piecewise-defined behavior, and discontinuities
- Optimization of the set of DAEs being solved and how the analog solver computes its solution are outside the scope of VHDL-AMS
- Extend structural semantics:
  - Conservative semantic to model physical systems: e.g. Kirchoff’s law for electrical circuits
  - Non-conservative semantics for abstract models: Signal flow description
  - Mixed-signal interfaces: Models can have digital and analog ports
- Mixed-signal semantics:
  - Unified model of time for a consistent synchronization of mixed event-driven/continuous behavior
  - Mixed-signal initialization and simulation cycle
  - Mixed-signal description of behavior
- Frequency domain support
- Small-signal frequency and noise modeling and simulation

### 1.6.2 VHDL-AMS Model Execution

- A VHDL-AMS model is the result of the elaboration of the design hierarchy
  - Digital part => set of processes + digital simulation kernel
  - Analog part => set of equations + analog solver
- Two Phases
- Determination of quiescent state of the model: includes initialization phase and simulation cycles at time 0ns
- Simulation: time domain, small-signal frequency, or noise. For time domain simulation, time $\geq 0$ns

- Reduces to the VHDL 1076 initialization and simulation cycle if the model does not include any quantities
- Only the analog solver is executed after initialization if the model does not include any signals

Example:

Most electronic circuits consist of both analog and digital circuits. In the given circuit, we want to create a digital clock signal for the two D flip-flops connected to generate a four-phase clock source. The digital clock is generated using a sine wave generator (vsine), and passing it to a comparator circuit, which convert an analog waveform to a digital or Boolean signal. The D flip-flop is sensitive to clock of type BIT, and not of type BOOLEAN. This requires a signal converter from Boolean to bit (BL$\Rightarrow$B).

![Circuit Diagram]

D Flip-Flop

ENTITY ffd IS
  GENERIC (T_PROP : REAL := 0.0);
  PORT (CLK : IN BIT := '0';
         DIN : IN BIT := '0';
         Q : OUT BIT := '0';
         QB : OUT BIT := '1');
END ENTITY ffd;

ARCHITECTURE behav of ffd IS
  CONSTANT del : TIME := T_PROP* 1 sec;
BEGIN
  PROCESS (CLK)
  BEGIN
    IF (CLK'EVENT AND CLK = '1') THEN
      Q <= DIN AFTER del;
      QB <= NOT DIN AFTER del;
    END IF;
  END PROCESS;
END behav;

Ideal Comparator

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY Comparator IS
  GENERIC(vthresh: REAL); --threshold
  PORT(TERMINAL ain, ref: ELECTRICAL;
       SIGNAL dout: OUT BOOLEAN);
END ENTITY Comparator;

ARCHITECTURE Ideal OF comparator IS
  QUANTITY vin ACROSS ain TO ref;
BEGIN
  dout<=vin'Above(vthresh); --implicit signal
END ARCHITECTURE Ideal;

Sine Wave Generator

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;
ENTITY Vsine IS
  GENERIC ( ampl, freq : REAL);
  PORT (TERMINAL p, m : ELECTRICAL);
END ENTITY Vsine;

ARCHITECTURE Sine OF Vsine IS
  QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
  v == ampl * sin (math_2_pi*freq*NOW);
END ARCHITECTURE Sine;

LIBRARY TRANSFORMATIONS;
USE TRANSFORMATIONS.OMNICASTER_PACKAGE.ALL;

ENTITY boolean_bit IS
  GENERIC( t_prop : REAL := 0.0);
  PORT( SIGNAL INP : IN BOOLEAN := FALSE;
        SIGNAL VAL : OUT BIT := '0');
END ENTITY boolean_bit;

ARCHITECTURE behav OF boolean_bit IS
  CONSTANT del : TIME := t_prop * 1 sec;
BEGIN
  val <= bl2b(inp) AFTER del;
END ARCHITECTURE behav;
Implicit Signal

- Q'Above(E): A Boolean signal that is TRUE when quantity Q is above threshold E
  - Q must be a scalar quantity, E must be an expression of the same type as Q
  - A event occurs on signal Q'Above(E) at the exact time of the threshold crossing
  - A process can be sensitive to Q'Above(E), since it is a signal

Simulation Cycle[1]

The execution of a model consists of an initialization phase followed by the repetitive execution of process statements in the description of that model. Each such repetition is said to be a simulation cycle. In each cycle, the values of all signals and quantities in the description are computed. If as a result of this computation an event occurs on a given signal, process statements that are sensitive to that signal will resume and will be executed as part of the simulation cycle.

Tn’ is the smaller on Tn and the earliest crossing event.
(a) advance analog solver to $T_n'$

(b) set $T_c = T_n'$

(c), (d) update signals

(e), (f) resume processes

(g) compute next time $T_n$

$T_n = T_c$

no

(h) resume postponed processes

(i) compute next time $T_n$
A simulation cycle consists of the following steps:

(a) The analog solver is executed.
(b) The current time $T_c$ is set to $T_n'$. Simulation is complete when $T_c$ is equal to the universal time corresponding to TIME’HIGH and there are no active drivers or process resumption at $T_c$.
(c) Each active explicit signal in the model is updated. (Events may occur on signals as a result.)
(d) Each implicit signal in the model is updated. (Events may occur on signals as a result.)
(e) For each process $P$, if $P$ is currently sensitive to a signal $S$ and if an event has occurred on $S$ in this simulation cycle, then $P$ resumes.
(f) Each non-postponed process that has resumed in the current simulation cycle is executed until it suspends.
(g) Compute next time $T_n'$ as the smaller of $T_n$ and the earliest crossing event. If $T_n'=T_c$, then the next simulation cycle (if any) will be a delta cycle.
(h) Each postponed process that has resumed but has not been executed since its last resumption is executed until it suspends.
(i) $T_n'$ is computed as in (g), then go to (a).

REFERENCES