Filname="ch1.doc"

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). Known as System 1076 in Mentor Graphics environment.
- 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, GcAs) 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.

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

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:

  PORT(namelist:mode type;...;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:

```
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:
  target_signal <= waveform;

Example:

```
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:
  target_signal <= waveform WHEN condition ELSE waveform

- Must end with a waveform.
- May contain multiple "waveform WHEN condition" clauses.
- Condition must produce boolean value.

Example:

```
signal_a <= `0' WHEN In1='1' 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:

```
new_signal <= signal_expression AFTER time;
```

Example:

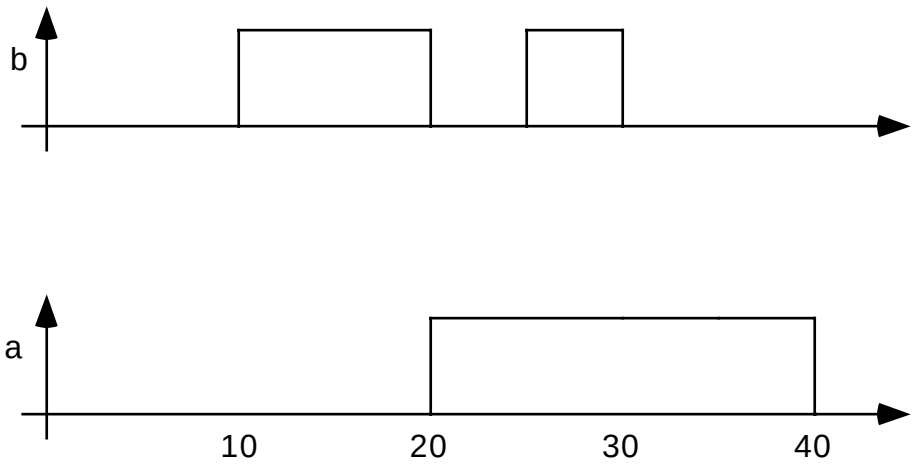
```
In1 <=  `1',
        `0' AFTER 250 ns,
        `1' AFTER 280 ns;
```

Signals whose duration is less than delay specified in AFTER clause are ignored.

Example:

```
-- signal assignment statement
    a <= b AFTER 10 ns;
```

```
-- stimulus set assignment
                    b<= `0',
                    `1' AFTER 10ns,
                    `0' AFTER 20ns,
                    `1' AFTER 25ns,
                    `0' AFTER 30ns;
```
* 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;          SIGNAL S:Integer:=0;
P1:                            P2:
PROCESS                       PROCESS
 BEGIN                        BEGIN
  S<= 1 AFTER 1 ns;            S<= 2 AFTER 2 ns;
  S<= 2 AFTER 2 ns;            S<= 1 AFTER 1 ns;
  WAIT;                        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<= 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:

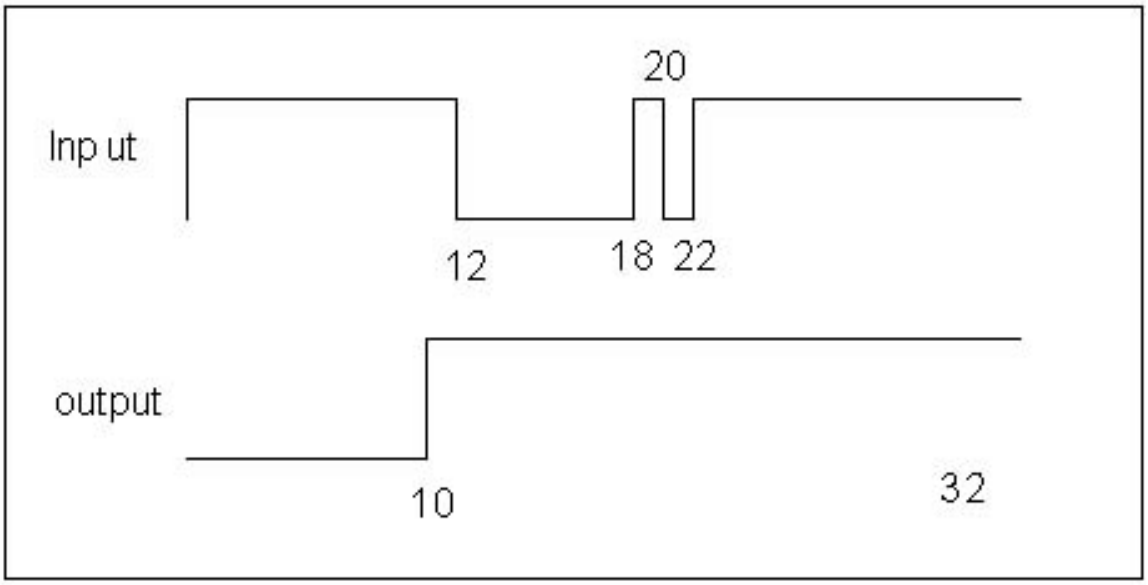
Describe a buffer having a rise time of 10 ns and a fall time of 14 ns.

```
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;
```

Input Event	Input Signal	Input Time(ns)	Output Signal	Output Time(ns)	Comments
1	0=>1	0	0=>1	10	=0+10=10(<12)
2	1=>0	12	1=>0	26	=12+14=26(>18)
3	0=>1	18	0=>1	28	=18+10=28(>20)
4	1=>0	20	1=>0	34	=20+14=34(>22)
5	0=>1	22	0=>1	32	=22+10=32(<∞)

Time (ns)	Driver Transactions	Driver Scheduled Transaction
0	(1,10ns)	(1,10ns)
10	(1,10ns) executed	
12	(0,26ns)	(0,26ns)

18	(0,26ns), (1,28ns)	(1,28ns)
20	(1,28ns),(0,34ns)	(0,34ns)
22	(0,34ns),(1,32ns)	(1,32ns)
32	(1,32ns) executed	



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.

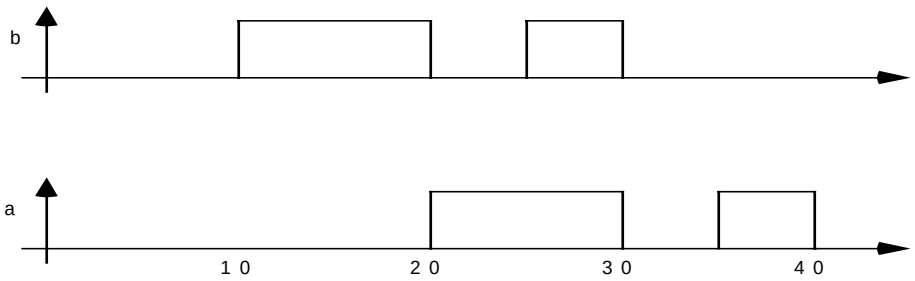
Transport Delay:

- 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:

```
-- signal assignment statement
a <= TRANSPORT b AFTER 10 ns;
-- stimulus set assignment
b<= `0',
     `1' AFTER 10ns,
     `0' AFTER 20ns,
     `1' AFTER 25ns,
     `0' AFTER 30ns;
```

*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

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

Example 2: the effects of overlapping assignement 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.

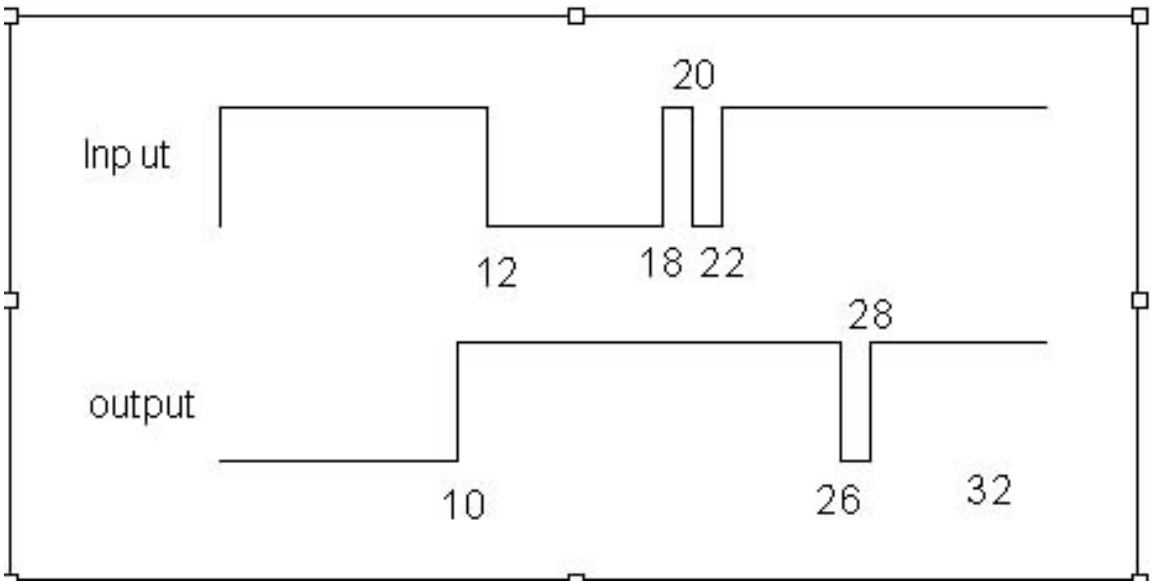
```
ENTITY Buffer IS
        PORT(input: IN BIT;output:OUT BIT);
END Buffer;

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.

Input Event	Input Signal	Input Time(ns)	Output Signal	Output Time(ns)	Comments
1	0=>1	0	0=>1	10	=0+10=10(<12)
2	1=>0	12	1=>0	26	=12+14=26(>18)
3	0=>1	18	0=>1	28	=18+10=28(>20)
4	1=>0	20	1=>0	34	=20+14=34(>22)
5	0=>1	22	0=>1	32	=22+10=32(<∞)

Time (ns)	Driver Transactions	Driver Scheduled Transaction
0	(1,10ns)	(1,10ns)
10	(1,10ns) executed	
12	(0,26ns)	(0,26ns)
18	(0,26ns), (1,28ns)	(0,26ns),(1,28ns)
20	(0,26ns),(1,28ns),(0,34ns)	(0,26ns),(1,28ns),(0,34ns)
22	(0,26ns),(1,28ns),(0,34ns),(1,32ns)	(0,26ns),(1,28ns),(1,32ns)
26	(0,26ns) executed	(1,28ns), (1,32ns)
28	(1,28ns)	(1,32ns)
32	(1,32ns) executed	



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 AFTER 1 ns, 3 AFTER 3 ns, 5 AFTER 5 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:

```
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, whreas 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 Example Testing The Crazy AND GATE

Specification : The AND gate has two inputs X and Y, and one output Z. When the output transition is caused by the changed of input X, the propogation delay for high to low change at output Z is 14 ns, and that for low to high transition is 18 ns. When the output transition is caused by the changed of input Y, the propagation delay is 15 ns for high to low transition at Z, and 20 ns for low to high transition.

X	Y	Z	Delay(ns)
1	1=>0	1=>0	15
1	0=>1	0=>1	20
1=>0	1	1=>0	14
0=>1	1	0=>1	18

```
ENTITY CRAZY_AND IS
```

```
        PORT(X,Y:IN BIT;Z:OUT BIT);
END CRAZY_AND;

ARCHITECTURE Behavior OF CRAZY_AND IS
SIGNAL X1,Y1: BIT;
BEGIN
X1 <= `1' AFTER 18 ns WHEN X='1' ELSE
      `0' AFTER 14 ns;
Y1 <= `1' AFTER 20 ns WHEN Y='1' ELSE
      `0' AFTER 15ns;
Z <= X1 AND Y1;
END Behavior;
```

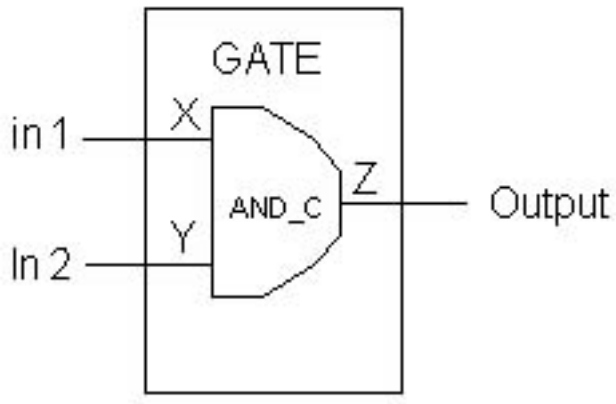
Creating a Test Bench

Now that the body (architecture) of the specification has been created, a test bench should be defined to apply test inputs to the circuit. The results of these tests should be used to check the design. Once the design has been checked, the simulation results can be used as a reference to check the implementation as it is developed.

The test bench of a circuit should include signal receptors for the inputs and outputs of the circuit. It should also include test stimulation for the circuit. This stimulation is applied to the signals used for the inputs to the circuit. In many cases, it is desirable to partition the test stimulator into a separate component which is instantiated from within the test bench. This allows the designer to isolate out the specifics of deriving the test set. In this circuit however, the test stimulation can be very straightforward and therefore is included directly in the test bench.

The test bench defines a component declaration for the GATE ciruit. The component declaration can be thought of as a template for the type of design unit that will be bound to this architecture. In this case, the component contains the same number of ports as the design entity. The component is instantiated and the instantiation maps the signals to be used for input and output to the ports of the component.

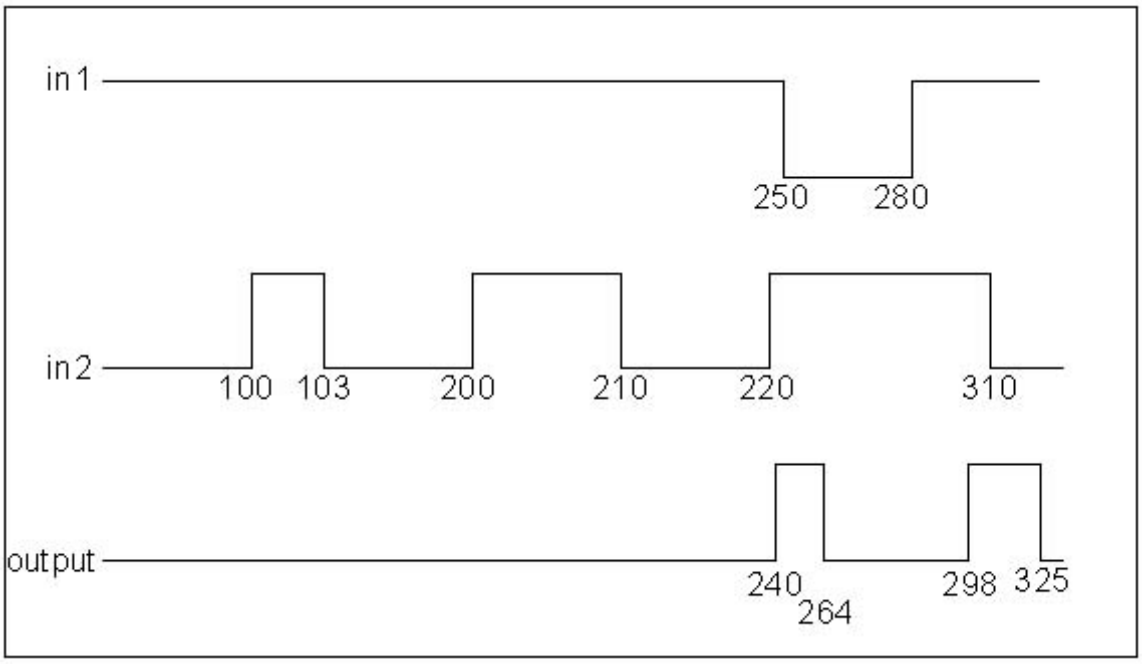
Finally, test stimulation is applied to the inputs of the GATE component. The two input signals In1 and In2 are used to stimulate it.




```
ENTITY TB IS -- no inputs or outputs
END TB;

ARCHITECTURE A OF TB IS
        COMPONENT AND_C
                PORT(X,Y:IN BIT;Z:OUT BIT);
```

```
        END COMPONENT;
        FOR GATE:AND_C USE ENTITY WORK.CRAZY_AND(Behavior);
        SIGNAL In1,In2,Output: BIT;
BEGIN
        GATE: AND_C
        PORT MAP(In1,In2,Output);
        In1 <= `1',
                `0' AFTER 250 ns,
                `1' AFTER 280 ns;
        In2 <= `0',
                `1' AFTER 100 ns,
                `0' AFTER 103 ns,
                `1' AFTER 200 ns,
                `0' AFTER 210 ns,
                `1' AFTER 220 ns,
                `0' AFTER 310 ns;
END A;
```

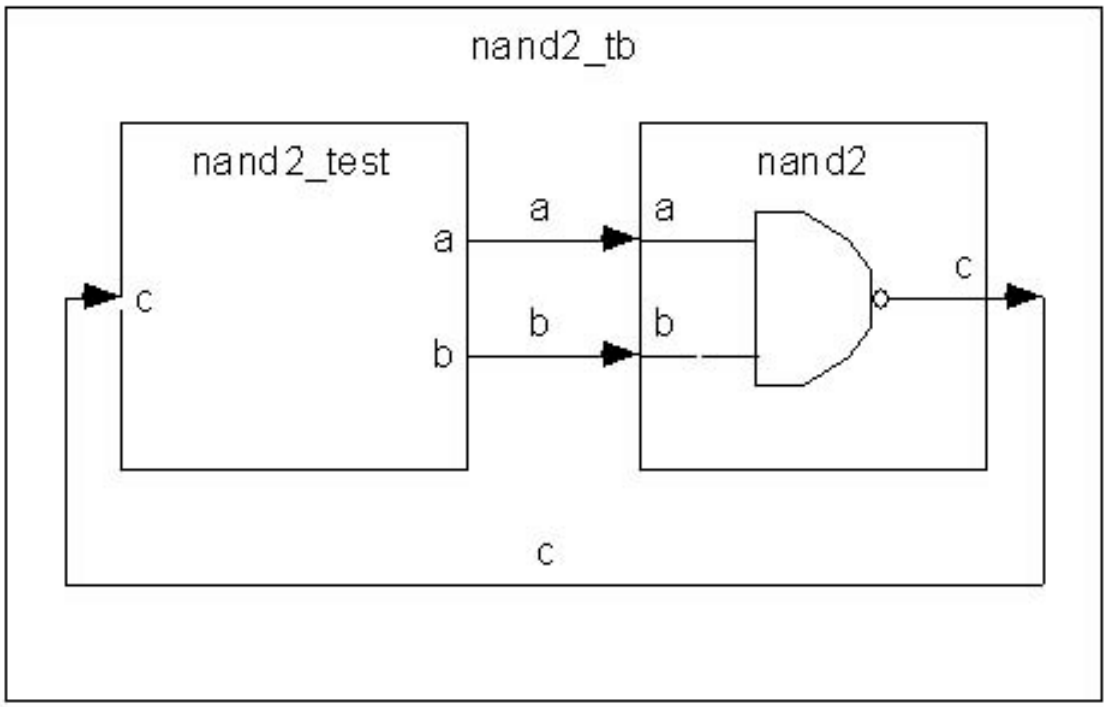
Time (ns)	In1	In2	Z Transactions	Z Scheduled Transaction
100	1	0=>1	(1,120ns)	(1,120ns)
103	1	1=>0	(1,120ns),(0,118ns)	(0,118ns)
118	1	0	(0,118ns) exe	
200	1	0=>1	(1,220ns)	(1,220ns)
210	1	1=>0	(1,220ns),(0,225ns)	(0,225ns)
220	1	0=>1	(0,225ns),(1,240ns)	(1,240ns)
240	1	1	(1,240ns) exe	
250	1=>0	1	(0,264ns)	(0,264ns)
264	0	1	(0,264ns) exe	
280	0=>1	1	(1,298ns)	(1,298ns)
298	1	1	(1,298ns) exe	
310	1	1=>0	(0,325ns)	(0,325ns)
325	1	0	(0,325ns) exe	



Violation message occurs at 221 ns. Examining the inputs immediately prior to the violation, we see that In1 has been high since the start of the simulation, and that In2 changed from `0' to `1' at 200 ns, from `1' to `0' at 210 ns, and from `0' to `1' at 220 ns. In the test bench architecture, In1 is connected to the X input and In2 is connected to the Y input. A change from `0' to `1' on Y should produce a change to `1' on the output after 20 ns, if no other faster change occurs before then. Since the next input change doesn't occur until 210 ns, the output should change to `1' at 220 ns. This doesn't happen. Hence the violation message at 221 ns.

The source of error is in the inertial delay for placing events in the event queue. This means that all events are cleared from the queue when an event is placed in the queue. In this case, the changed on In2 at 210 ns preempts the change scheduled earlier at 200 ns. The results is that the effect of the event at 200 ns is lost. The specification can be satisfied by changing the implementation of signal assignments from inertial to transport delay, in the implementation of CRAZY AND gate.

Creating A Closed Loop Test Bench




```
library ieee;
use ieee.std_logic_1164.all;

ENTITY nand2 IS
	PORT(a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
END nand2;

ARCHITECTURE arch_nand2 OF nand2 IS
BEGIN
	c <= a NAND b;
END arch_nand2;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity nand2_test is
	port (a,b:out std_logic;
		c:in std_logic);
end entity nand2_test;

architecture behav of nand2_test is
begin
	test_process: process is
	begin
		a<='1';
		b<='1';
		wait for 1 ns;
		assert (c='0')
			report "Not Working" severity error;
				wait for 10 ns;
		a<='0';
		b<='1';
		wait for 1 ns;
```

```
                assert (c='1')
                        report "Not Working" severity error;
                                wait for 10 ns;
                a<='1';
                b<='0';
                wait for 1 ns;
                assert (c='1')
                        report "Not Working" severity error;
                                wait for 10 ns;
                a<='0';
                b<='0';
                wait for 1 ns;
                assert (c='1')
                        report "Not Working" severity error;
                        wait for 100 ns;
        end process test_process;
end architecture behav;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity nand2_tb is
end nand2_tb;

architecture behav of nand2_tb is

        component nand2
                port (a,b:in std_logic;
                c:out std_logic);
        end component;

        component nand2_test
                port (a,b:out std_logic;
                c:in std_logic);
        end component;

        for U1:nand2_test use entity work.nand_test(behav);
        for U2:nand2 use entity work.nand2(arch_nand2);
        signal a, b, c:std_logic;
begin
        U1:nand2_test port map(a, b, c);
        U2:nand2 port map(a, b, c);
end behav;
```

1.7 A General Delay Time Model For Circuit

- The total delay time of the circuit F is modeled as two parts:
- The first part is called the Data_Valid time and is modeled using the VHDL inertial delay model. This part characterizes the minimum stable duration ti of the input signal before the operation F can be performed.
- The second part is the pure propagation delay (tp) and is modeled by three transport delay components. The first one (tp1) is the pure propagation delay inherent in the circuit. The other two components model the effects of external capacitive loadings due to the connecting input devices and interconnects.

- The Data_Valid time part of this delay time model can be thought of as a lowpass filter that eliminates glitches of duration shorter than the specified time ti.

Example:

The first pulse of d1 does not make it through the filter since its duration is less than t1. The other pulse pass through the filter. Once through the filter, however, the input changes become valid and activate the process F. The result is the occurrence of a transaction on q0, which may or may not cause a change of value in q0. If a change does occur, then the change, called an event, will propagate to q after tp=tp1+tp2+tp3.

The VHDL coding of the general device F is given below:

```
d01 <= d1 AFTER t1;
…..
d0n <= dn AFTER tn;
q0 <= F(d01,…,d0n);
q <= TRANSPORT q0 AFTER tp;
```