COE 405
Basic Concepts in VHDL

Dr. Aiman H. El-Maleh
Computer Engineering Department
King Fahd University of Petroleum & Minerals

Outline

- VHDL Objects
- Variables vs. Signals
- Signal Assignment
- Signal Transaction & Event
- Delta Delay
- Transport and Inertial Delay
- Sequential Placement of Transactions
- Signal Attributes

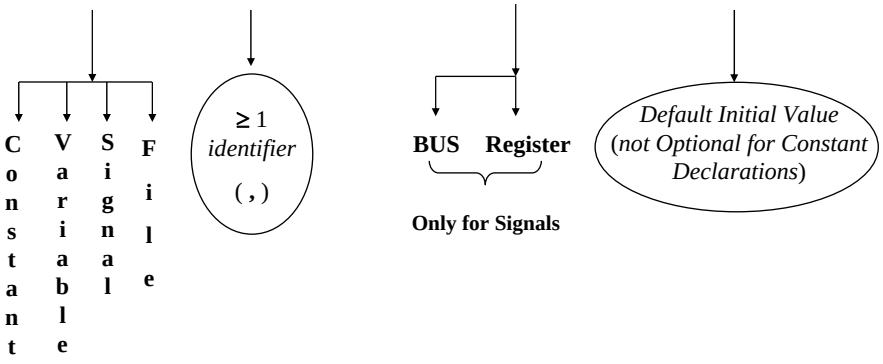
VHDL Objects …

- **VHDL *OBJECT* : Something that can hold a value of a given** *Data Type***.**
- **VHDL has 3 classes of objects**
  - CONSTANTS
  - VARIABLES
  - SIGNALS
- **Every object & expression must unambiguously belong to one *named*** *Data Type*
- **Every object must be** *Declared.*

… VHDL Object …

Syntax

Obj_Class **<*id_list*> : Type/SubType [*signal_kind*] [:= *expression*];**



… VHDL Object …

- **Value of Constants *must* be specified when declared**
- ***Initial* values of Variables or Signals *may* be specified when declared**
- **If not explicitly specified*, Initial* values of Variables or Signals *default* to the value of the Left Element in the type range specified in the declaration.**
- **Examples:**
  - **Constant** Rom_Size **:** Integer := 2**16**;**
  - **Constant** Address_Field **:** Integer := 7**;**
  - **Constant** Ovfl_Msg **:** String (1 **To** 20) := ``Accumulator OverFlow``**;**
  - **Variable** Busy, Active **:** Boolean := False**;**
  - **Variable** Address **:** Bit_Vector (0 **To** Address_Field) := ``00000000``**;**
  - **Signal** Reset**:** Bit := `0`**;**

2-5

Variables vs. Signals

Variables & Signals

VARIABLES	SIGNALS
• Variables are only *Local* and May Only Appear within the Body of a Process or a SubProgram • Variable Declarations Are Not Allowed in Declarative Parts of Architecture Bodies or Blocks.	• Signals May be Local or Global. • Signals May not be Declared within Process or Subprogram Bodies. • All Port Declarations Are for Signals.
A Variable Has No HardWare Correspondence	A Signal Represents a Wire or a Group of Wires (BUS)
Variables Have No ***Time*** Dimension Associated With Them. (*Variable Assignment occurs instantaneously*)	Signals Have ***Time*** Dimension (*A Signal Assignment is Never Instantaneous* (Minimum Delay = δ Delay)
Variable Assignment Statement is always ***SEQUENTIAL***	Signal Assignment Statement is Mostly ***CONCURRENT*** (*Within Architectural* Body). It Can Be **SEQUENTIAL** (*Within Process Body*)
Variable Assignment Operator is :=	Signal Assignment Operator is <=

Variables Within Process Bodies are STATIC, i.e. a Variable Keeps its Value from One Process Call to Another. Variables Within Subprogram Bodies Are Dynamic, i.e. Variable Values are Not held from one Call to Another.

2-6

Signal Assignments …

- **Syntax:**

  Target Signal <= [Transport] ***Waveform*** **;**

  Waveform **:=** ***Waveform_element*** **{,** ***Waveform_element*** **}**

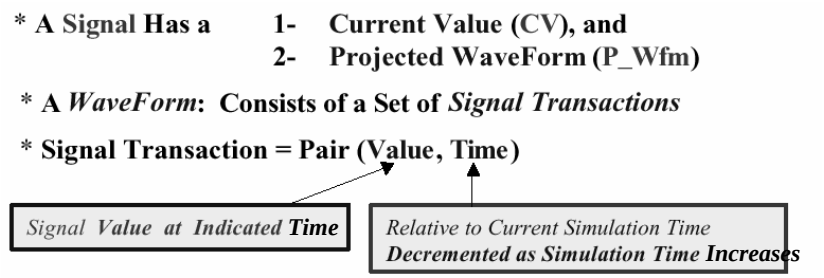
  Waveform_element **:=** ***Value_Expression*** **[** ***After Time_Expression*** **]**

- **Examples:**
  - X <= '0' ; -- Assignment executed After δ delay
  - S <= '1' After 10 ns;
  - Q <= Transport '1' After 10 ns;
  - S <= '1' After 5 ns, '0' After 10 ns, '1' After 15 ns;
- **Signal assignment statement**
  - mostly **concurrent** (within architecture bodies)
  - can be **sequential** (within process body)

2-7

… Signal Assignments

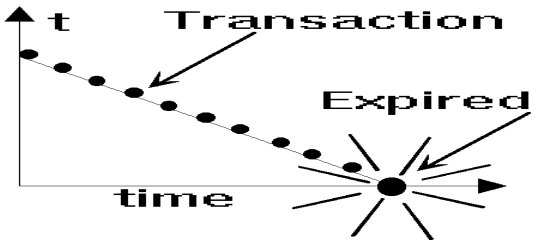
- **Concurrent signal assignments are order independent**
- **Sequential signal assignments are order dependent**
- **Concurrent signal assignments are executed**
  - Once *at the beginning* of simulation
  - Any time a signal on the right hand side changes



2-8

Signal Transaction

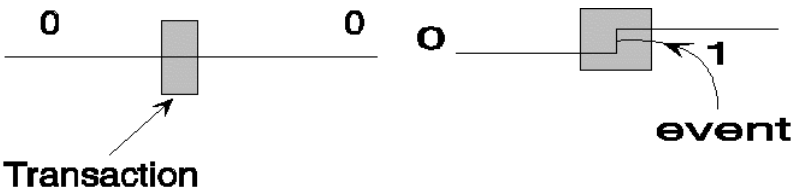
- **When the time element of a signal transaction expires (t=0)**
  - Its associated value is made the current value (CV) of a signal
  - The transaction is deleted from the list of transactions forming the Projected Waveform (P_Wfm) of the signal



2-9

Signal Transaction & Event …

- **When a new value is assigned to a signal, it is said that**
  - a Transaction has been *Scheduled* for this signal
  - or a Transaction has been placed on this *Signal Driver*
- **A Transaction which does not cause a signal transition (Event) is still a Transaction**
- **A Transaction *May/May not* cause a signal transition (Event) on the target signal**



2-10

… Signal Transaction & Event …

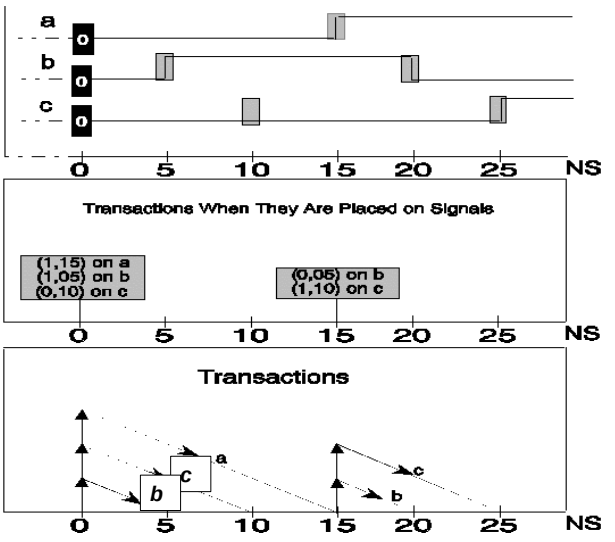
- A <= ‘1’ After 10 ns, ‘0’ After 20 ns, ‘1’ After 30 ns;

	t=0	t=5 ns	t=10 ns	t=20 ns	t=30 ns
A (CV)	‘0’	‘0’	‘1’	‘0’	‘1’
A (P_Wfm)	(‘1’, 10ns) (‘0’, 20ns) (‘1’, 30ns)	(‘1’, 5ns) (‘0’, 15ns) (‘1’, 25ns)	(‘0’, 10ns) (‘1’, 20ns)	(‘1’, 10ns)	

2-11

… Signal Transaction & Event

```
ARCHITECTURE
demo OF example IS
SIGNAL a, b, c : BIT
:= '0';
BEGIN
a <= '1' AFTER 15
NS;
 b <= NOT a AFTER
5 NS;
c <= a AFTER 10 NS;
END demo;
```



2-12

Scope of Signal Declarations

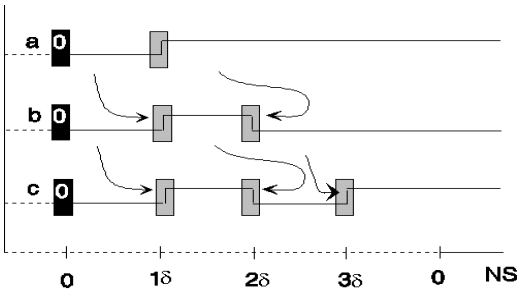
- **Signals declared within a Package are Global usable by all Entities using this package**
- **Signals declared within an Entity are usable by all architectural bodies of this entity**
- **Signals declared within an Architectural body are only usable within this Architectural body**

2-13

Delta Delay …

- **If no Time Delay is *explicitly specified*, Signal assignment is executed after an infinitesimally small δ-delay**
  - Delta is a simulation cycle , and not a real time
  - Delta is used for scheduling
  - A million deltas do not add to a femto second

```
ARCHITECTURE concurrent
OF timing_demo IS
SIGNAL a, b, c : BIT := '0';
BEGIN
a <= '1';
b <= NOT a;
c <= NOT b;
END concurrent;
```

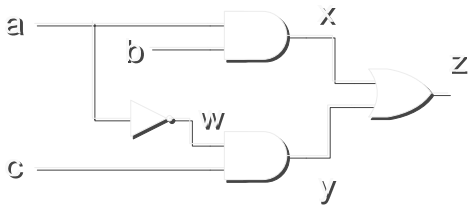


2-14

7

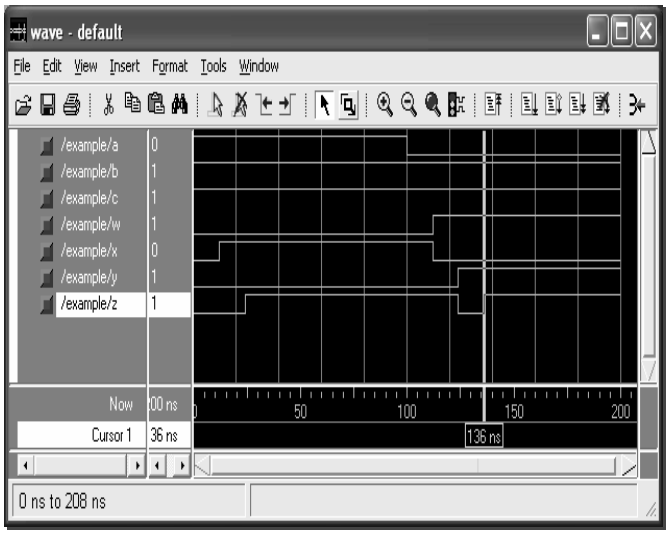
… Delta Delay …

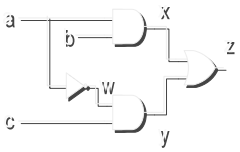
```
Entity example is
Port (a, b, c: IN bit; Z: OUT Bit);
End;
ARCHITECTURE
properly_timed OF example IS
SIGNAL w, x, y : BIT := '0';
BEGIN
y <= c AND w AFTER 12 ns;
w <= NOT a AFTER 12 ns;
x <= a AND b AFTER 12 ns;
z <= x OR y AFTER 12 NS;
END properly_timed;
```



2-15

… Delta Delay …



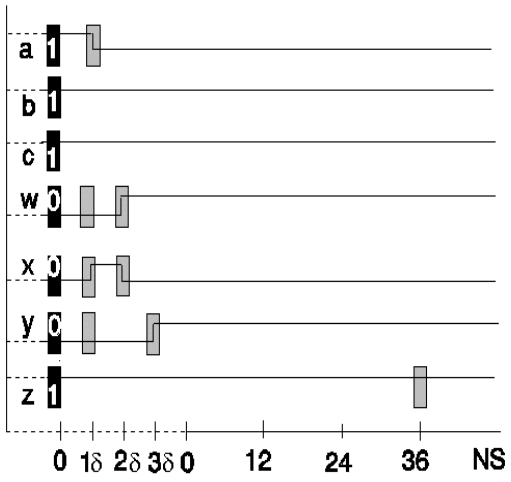


2-16

… Delta Delay …

```
ARCHITECTURE
not_properly_timed OF
example IS
SIGNAL w, x, y : BIT := '0';
BEGIN
y <= c AND w;
w <= NOT a;
x <= a AND b;
z <= x OR y AFTER 36 NS;
END not_properly_timed;
```

a changes from 1 to 0 at time 0.



2-17

Delta Delay in Sequential Signal Assignments …

- **Effect of δ-delay should be carefully considered when signal assignments are embedded within a process**

```
Entity BUFF2 IS
Port (X: IN BIT;
      Z: OUT BIT);
END BUFF2;
```

```
Architecture Wrong of BUFF2 IS
Signal y: BIT;
Begin
  Process(x)
  Begin
    y <= x;
    z <= y;
  End Process;
End Wrong;
```

- **Process activated on x-events only**
  - **y ← x(δ)**
  - **z ← y(0)**
- **z gets OLD value of y and not new value of x**

2-18

… Delta Delay in Sequential Signal Assignments

```
Architecture OK of BUFF2 IS
Signal y: BIT;
Begin
  Process(x,y)
 Begin
    y <= x;
    z <= y;
  End Process;
End OK;
```

- Process activated on both x and y events
- x changes and process activated
  - y ← x; -- y gets x value after δ
  - z ← y; -- z gets y(0) value after δ
- Process terminated
- After δ, y changes and process activated
- z gets new y (=x) after δ
- Process terminated

2-19

Oscillation in Zero Real Time

```
Architecture forever of
oscillating IS
Signal x: BIT :='0';
Signal y: BIT :='1';
Begin
     x <= y;
     y <= NOT x;
End forever;
```

Delta	x	y
+0	0	1
+1	1	1
+2	1	0
+3	0	0
+4	0	1
+5	1	1
+6	1	0
+7	0	0
+8	0	1

2-20

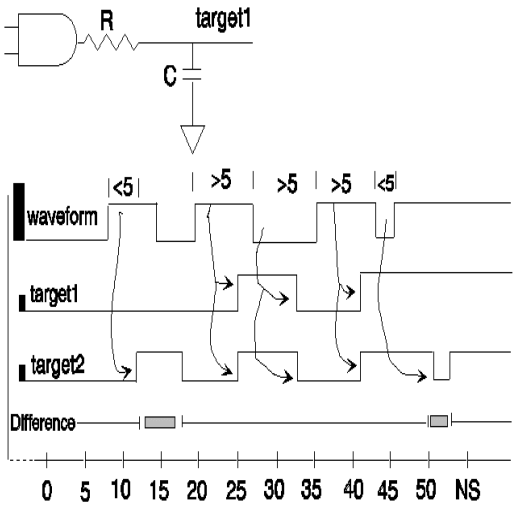
Signal Assignment & Delay Types

- **Two types of signal delay: Inertial and Transport**
- **Transport Delay**
  - Exact delayed version of input signal no matter how short the input stimulus
  - **Transport** keyword must be used
    - Example: S<= TRANSPORT waveform after 5 ns;
  - Models delays through transmission lines and networks with virtually infinite frequency response
- **Inertial Delay**
  - A delayed version of the input waveform
  - Signal changes (Glitches) that do not persist for a specified duration are missed
  - Default delay type
    - Example: S<= waveform after 5 ns;
  - Models capacitive networks and represents hardware inertial delay
  - Can have additional Reject specification
    - Example: S<= REJECT 3 ns INERTIAL waveform after 5 ns;

2-21

Transport & Inertial Delay

```
ARCHITECTURE delay OF
example IS
SIGNAL target1, target2,
waveform : BIT;
-- this is a comment
BEGIN
-- the following illustrates
inertial delay
target1 <= waveform
AFTER 5 NS;
-- the following illustrates
transport delay
target2 <= TRANSPORT
waveform AFTER 5 NS;
END delay;
```



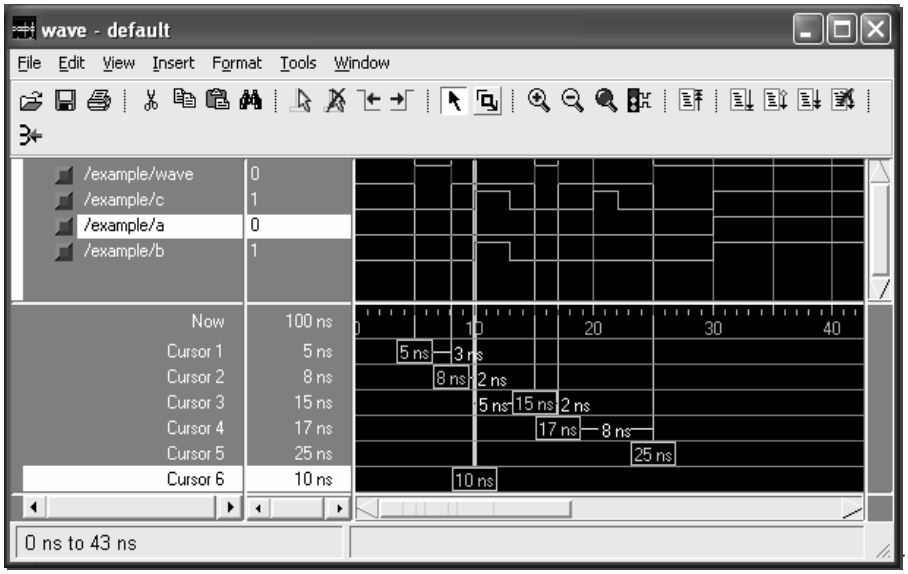
2-22

…Transport & Inertial Delay…

```
Entity example Is
End example;
Architecture ex1 of example is
SIGNAL a, b, c, wave : BIT;
BEGIN
a <= wave after 5 ns;
b <= REJECT 2 ns INERTIAL   wave after 5 ns;
c <= transport wave after 5 ns;
wave  <= '1' after 5 ns, '0' after 8 ns, '1' after 15
ns, '0' after 17 ns, '1' after 25 ns;
END ex1;
```

2-23

…Transport & Inertial Delay



12

Sequential Placement of Transactions

Scheduling of Signal Transactions

- The Scheduling Mechanism Should Reflect The Difference in Nature Between *Inertial* & *Transport* Delays.
- Let The Projected Waveform of A Signal Consist of the Transactions $(V_1, T_1), (V_2, T_2) \ldots (V_i, T_i) \ldots\ldots (V_n, T_n)$, *where:*
  $T_n > T_{n-1} > .. > T_i > .. T_2 > T_1$
- IF a new Transaction (V , T) is Scheduled on this Signal, Then:

Transport Delay

(a) $T > T_n$

Append New Transaction to P_Wfm $\{(V_1, T_1), (V_2, T_2), (V_n, T_n), (V, T)\}$

(b) $T_i < T \leq T_{i+1}$

- Later Transactions Are Discarded $(V_{i+1}, T_{i+1}) \ldots (V_n, T_n)$
- All Earlier Transactions Are Maintained
- The Resulting P_Wfm is $\{(V_1, T_1), (V_2, T_2), .., (V, T)\}$

Inertial Delay

$T_i < T \leq T_{i+1}$

- Later Transactions Are Discarded $(V_{i+1}, T_{i+1}) \ldots (V_n, T_n)$
- Earlier Transactions with $Vj \neq V$ Are Discarded ($\forall j = 1,2, .., i$)
- Append New Transaction

Lemma

The Semantics of Inertial-Delay Signal Assignment is Such That if a Number of Elements are Placed on the Assignment Statement as in (*S <= 1 After 1 NS, 3 After 3 NS, 5 After 5 NS*;), *Only the First Element is Considered to Have Inertial Delay, i.e. Elements After the First One Are Considered to Have Transport Delay*

Note: Wfm Elements Must Have Ascending Time Delays

2-25

Sequential Placement of Transactions

	Transport	Inertial
New Transaction is Before Already Existing	Overwrite existing transaction	Overwrite existing transaction
New Transaction is After Already Existing	Append new transaction	

$V_{new} \neq V_{old}$ Overwrite existing transaction

$V_{new} = V_{old}$ Append new transaction

$V_{new} \neq V_{old}$ $T_{new}-T_{old}$> Reject Append new transaction

$V_{new} \neq V_{old}$ $T_{new}-T_{old}$<= Reject Overwrite existing transaction

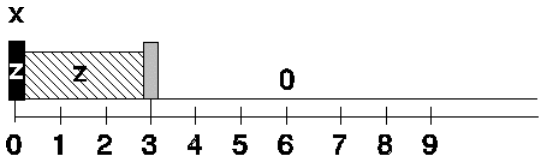
2-26

13

Example: Transport, New Transaction Before Already Existing

```
ARCHITECTURE sequential OF
overwriting_old IS
Type tit is (‘0’, ‘1’, ‘Z’);
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= '1' AFTER 5 NS;
        x <= Transport '0' AFTER 3 NS;
   WAIT;
  END PROCESS;
END sequential;
```

Discards previous transaction

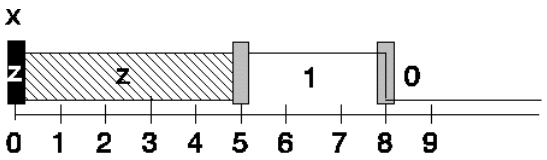


2-27

Example: Transport, New Transaction After Already Existing

```
ARCHITECTURE sequential OF
saving_all IS
Type tit is (‘0’, ‘1’, ‘Z’);
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= '1' AFTER 5 NS;
        x <= Transport '0' AFTER 8 NS;
   WAIT;
  END PROCESS;
END sequential;
```

Appends transaction



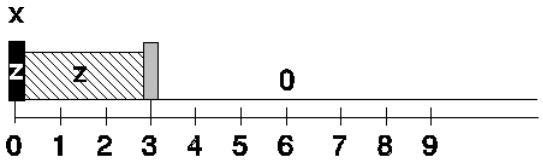
2-28

14

Example: Inertial, New Transaction Before Already Existing

```
ARCHITECTURE sequential OF
overwriting_old IS
Type tit is (‘0’, ‘1’, ‘Z’);
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= '1' AFTER 5 NS;
        x <= '0' AFTER 3 NS;
   WAIT;
  END PROCESS;
END sequential;
```

Discards previous transaction

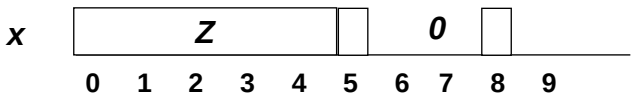


2-29

Example: Inertial, New Transaction After Already Existing (Same Value)

```
ARCHITECTURE sequential OF
saving_all IS
Type tit is (‘0’, ‘1’, ‘Z’);
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= ‘0' AFTER 5 NS;
        x <= '0' AFTER 8 NS;
   WAIT;
  END PROCESS;
END sequential;
```

Saves previous Transaction with Same value

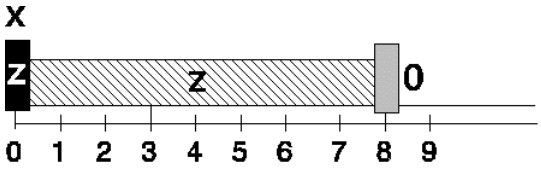


2-30

Example: Inertial, New Transaction After Already Existing (Different Value)

```
ARCHITECTURE sequential OF
discarding_old IS
Type tit is (‘0’, ‘1’, ‘Z’);
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= ‘1' AFTER 5 NS;
        x <= ‘0' AFTER 8 NS;
   WAIT;
  END PROCESS;
END sequential;
```

Discards old value

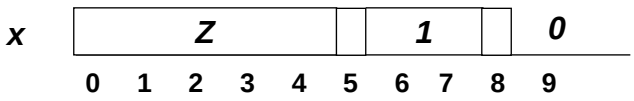


2-31

Example: Inertial, New Transaction After Already Existing (Diff. Value with Reject)

```
ARCHITECTURE sequential OF
appending IS
Type tit is (‘0’, ‘1’, ‘Z’);
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= ‘1' AFTER 5 NS;
        x <= Reject 2 ns Inertial ‘0' AFTER 8 NS;
   WAIT;
  END PROCESS;
END sequential;
```

- *Time difference between new and existing transaction is greater than reject value*
- *Appends transaction*

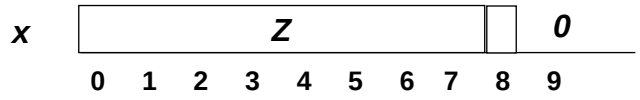


2-32

Example: Inertial, New Transaction After Already Existing (Diff. Value with Reject)

```
ARCHITECTURE sequential OF
appending IS
Type tit is ('0', '1', 'Z');
SIGNAL x : tit := 'Z';
BEGIN
  PROCESS
  BEGIN
        x <= '1' AFTER 5 NS;
        x <= Reject 4 ns Inertial '0' AFTER 8 NS;
   WAIT;
  END PROCESS;
END sequential;
```

- *Time difference between new and existing transaction is less than reject value*
- *Discards old value*



2-33

Sequential Placement of Transactions

Examples

Process Begin A <= Transport 1 After 5 NS; A <= Transport 2 After 10 NS; Wait; **End** Process;	**Process** Begin B <= Transport 2 After 10 NS; B <= Transport 1 After 5 NS; Wait; **End** Process;

Time	**P_WFM(A)**	**Time**	**P_WFM(B)**
t0	(1, 5NS)	t0	(2, 10 NS)
t1	(1, 5NS) (2, 10 NS)	t1	(1, 5 NS) {*Previous Transaction Discarded*}

t0 = Time After Executing the First Statement
t1 = Time After Executing the Second Statement

Process Begin A <= 1 After 5 NS; A <= 2 After 10 NS; Wait; **End** Process;	**Process** Begin B <= 2 After 10 NS; B <= 1 After 5 NS; Wait; **End** Process;

Time	**P_WFM(A)**	**Time**	**P_WFM(B)**
t0	(1, 5NS)	t0	(2, 10 NS)
t1	(2, 10 NS)) { *Discard Previous Transaction Since 1 ≠ 2*}	t1	(1, 5 NS) {*Later Transaction Discarded*}

t0 = Time After Executing the First Statement
t1 = Time After Executing the Second Statement

2-34

17

Sequential Placement of Transactions

Examples

```
Process
Begin
S1:     A <= 1 After 1 NS, 3 After 3 NS, 5 After 5 NS;
S2:     A <= 3 After 4 NS, 4 After 5 NS;
    Wait;
End Process;
```

Statement	P_WFM(A)
S1	(1, 1NS) (3, 3NS) (5, 5NS)
S2	Upon Adding (3, 4 NS)) **(Inertial)** • Discard (5, 5 NS) Since it is later than (3, 4 NS) • Discard (1, 1 NS) Since 1 ≠ 3 (3, 3 NS) (3, 4 NS) Upon Adding (4, 5 NS) **(Transport)** • Maintain All Earlier Transactions (3NS & 4NS) (3, 3 NS) (3, 4 NS) (4, 5 NS)

t0 = Time After Executing the First Statement
t1 = Time After Executing the Second Statement

Example

```
A <= 1 After 1 NS, 3 After 3 NS, 5 After 5 NS;
```

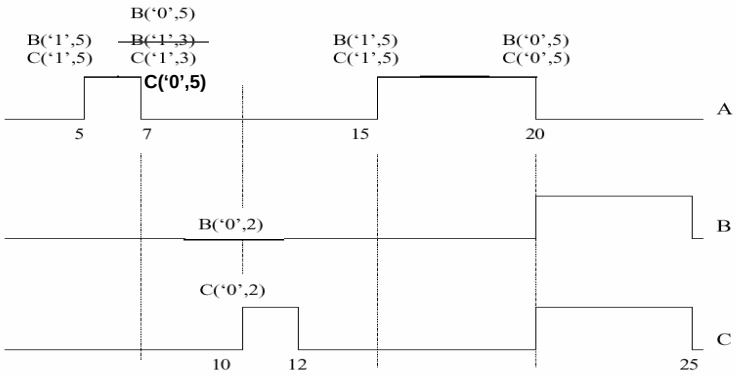
Is NOT Equivalent To:	Is Equivalent To:
A <= 1 After 1 NS; A <= 3 After 3 NS; A <= 5 After 5 NS;	A <= 1 After 1 NS; A <= Transport 3 After 3 NS; A <= Transport 5 After 5 NS;

2-35

Sequential Placement of Transactions

Example

```
Architecture Ex OF Delay IS
Begin
A <= '1' After 5 NS, '0' After 7 NS, '1' After 15 NS, '0' After 20 NS;
B <= A After 5 NS;
C <= Transport A After 5 NS;
End;
```



2-36

18

Sequential Placement of Transactions

	t=0	t=5 ns	t=7 ns	t=15 ns	t=20 ns
A (CV)	' 0 '	' 1 '	' 0 '	' 1 '	' 0 '
A(P_ Wfm)	('1 ', 5 ns) ('0 ', 7 ns) ('1 ', 15 ns) ('0 ', 20 ns)	('0 ', 2 ns) ('1 ',10 ns) ('0 ',15 ns)	('1 ', 8 ns) ('0 ',13 ns)	('0 ', 5 ns)	--

	t=0	t=5 ns	t=7 ns	t=12 ns	t=15 ns	t=20 ns
B(CV)	' 0 '	' 0 '	' 0 '	' 0 '	' 0 '	' 1 '
B(P_ Wfm)	--	('1 ', 5ns)	('0 ', 5 ns) x('1 ',3 ns)	--	('1 ', 5 ns)	('0 ', 5 ns)

	t=0	5 ns	7 ns	10 ns	12 ns	15 ns	20 ns
C(CV)	' 0 '	' 0 '	' 0 '	' 1 '	'0'	' 0 '	' 1 '
C(P_ Wfm)	--	('1 ',5 ns)	('0 ',5 ns) ('1 ',3 ns)	('0 ',2 ns)	--	('1 ', 5 ns)	('0 ', 5 ns)

2-37

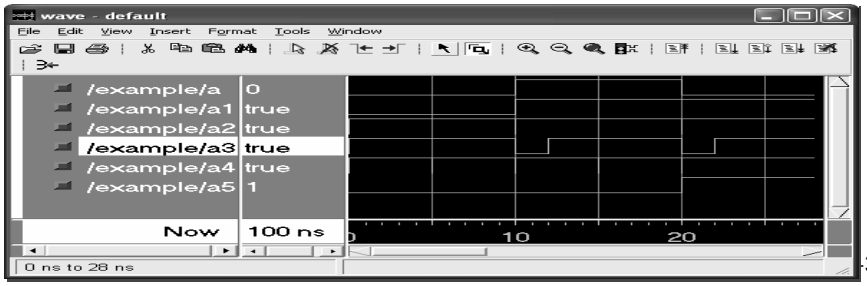
Signal Attributes…

- **Attributes are named characteristics of an Object (or Type) which has a value that can be referenced.**
- **Signal Attributes**
  - S`Event **--** Is TRUE if Signal S has changed.
  - S`Stable(t) **--** Is TRUE if Signal S has not changed for the last ``**t**`` period. If t=0; it is written as S`Stable
  - S`Last_Value **--** Returns the previous value of S before the last change.
  - S`Active **-- --** Is TRUE if Signal S has had a transaction in the current simulation cycle.
  - S`Quiet(t) **-- --** Is TRUE if no transaction has been placed on Signal S for the last ``**t**`` period. If t=0; it is written as S`Quiet
  - S`Last_Event **--** Returns the amount of time since the last value change on S**.**

2-38

…Signal Attributes

```
architecture ex of example is
signal a, a5: bit; signal a1, a2, a3, a4: Boolean;
begin
a <= '0' after 5 ns, '1' after 10 ns, '1' after 15 ns, '0' after 20 ns;
a1 <= a'event;
a2 <= a'stable;
a3 <= a'stable(2 ns);
a4 <= a'quiet;
a5 <= a'last_value;
end ex;
```



39