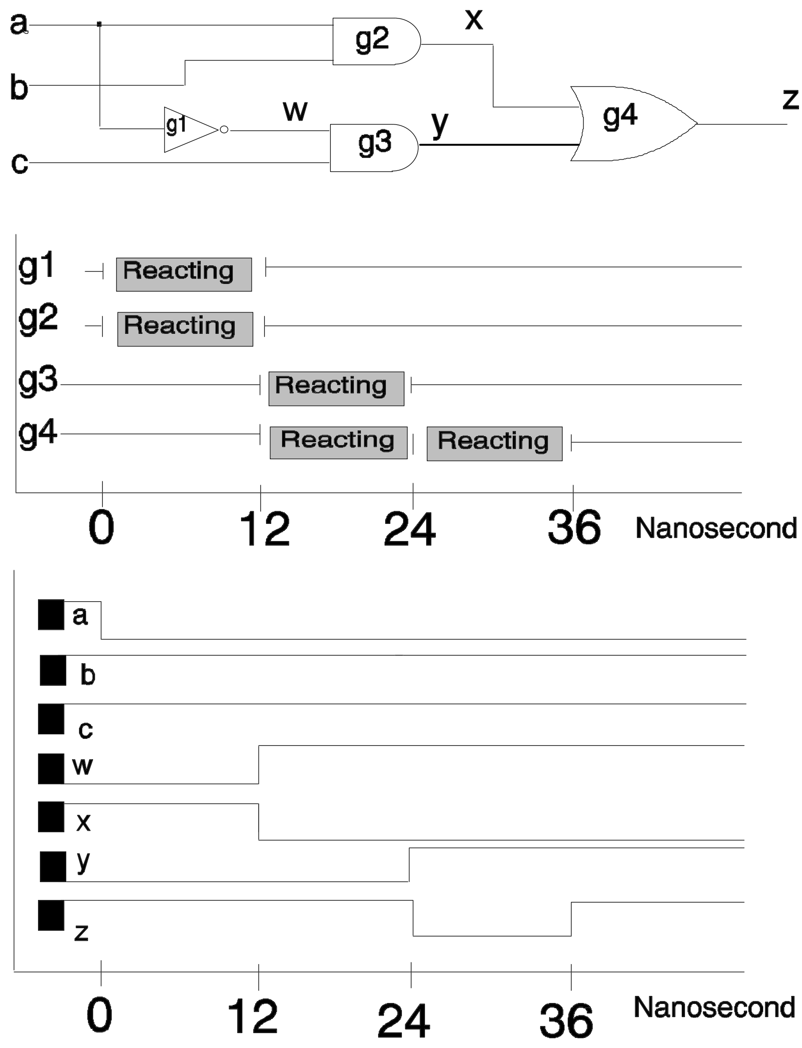
PART 3

VHDL_Timing

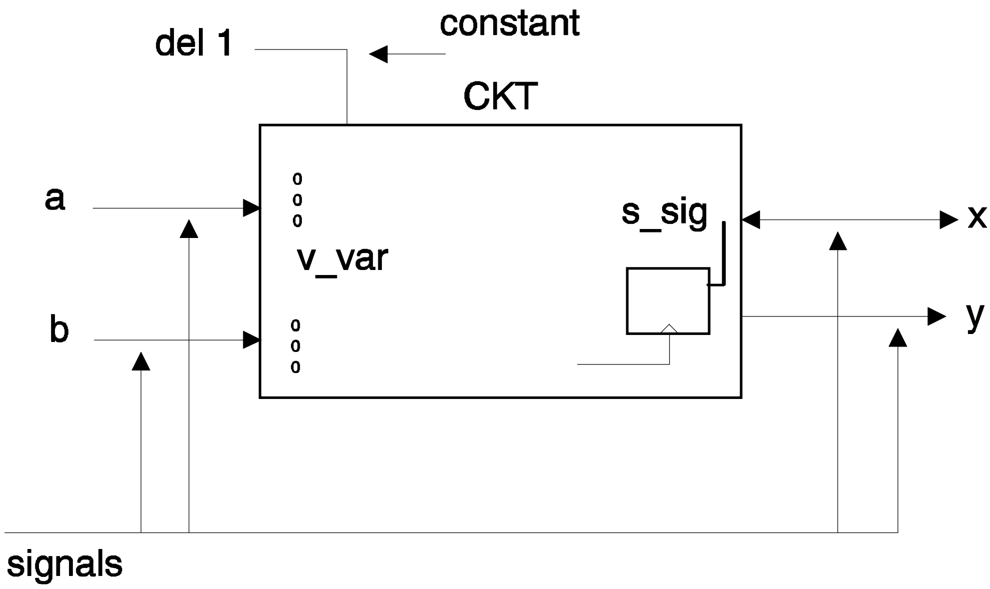
- Hardware modeling requirements
- Objects and classes
- Signals & variables
- Concurrent & sequential assignments
- Events, transactions, delta delays
- Delay modeling
- Sequential placement of transactions
- Conventions & syntax

VHDL_Timing.hardware_modeling_requirements



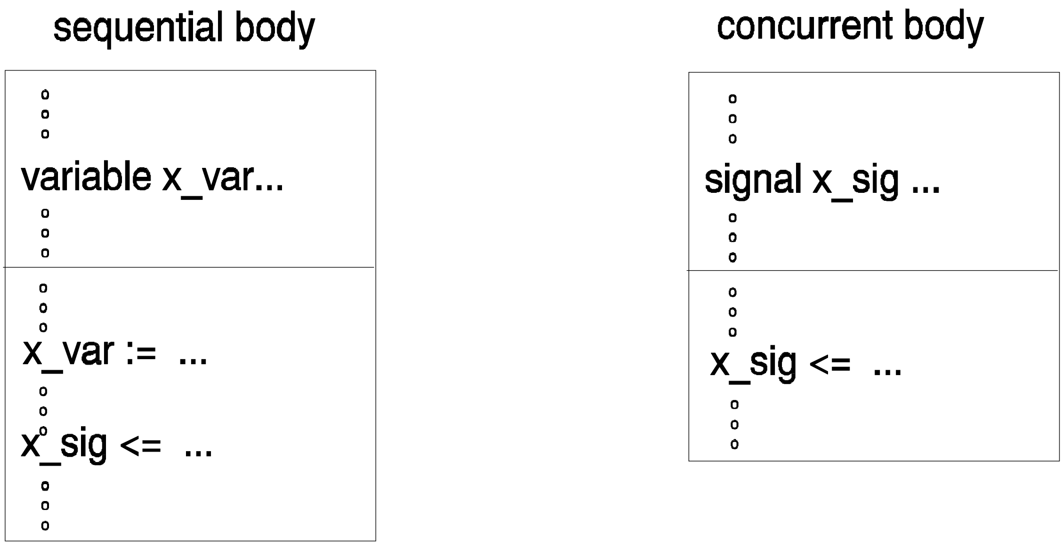
- Proper modeling requires simultaneous processing
- Waveform shows node values

VHDL_Timing.objects_and_classes



- Signals for hardware carriers
- Variables as temporary carriers
- Constants for fixed parameters
- A signal is an object whose class is signal

VHDL_Timing.signals_and_variables



- Signal assignments have a time component
- x_sig <= value AFTER 6 NS;

VHDL_Timing.concurrent/sequential_assignments

sequential body

```
o
o
o
x_sig <= val1 ;
o
o
o
y_sig <= a_sig ;
o
o
o
z_sig <= val2

WAIT FOR 5NS ;
z_sig <= val3 ;
o
o
o
```

concurrent body

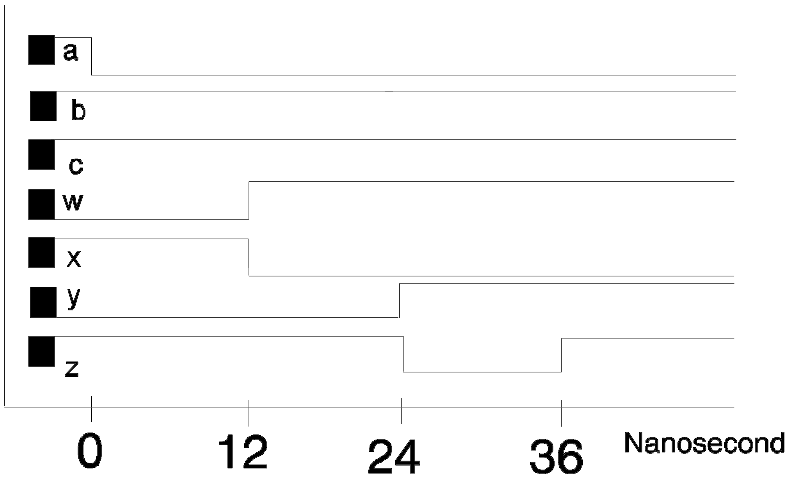
```
o
o
o
x_sig  <= val1 ;
o
o
o
y_sig <= a_sig ;
o
o
o
z_sig <=  val2 ,  val3  AFTER 5 NS;
```

- *val2* and *val3* are sequentially placed on *z_sig*

- Assignment of *a_sig* to *y_sig* is done:
  - in the concurrent body, when event occurs on *a_sig*
  - in the sequential body, when program flow reaches it

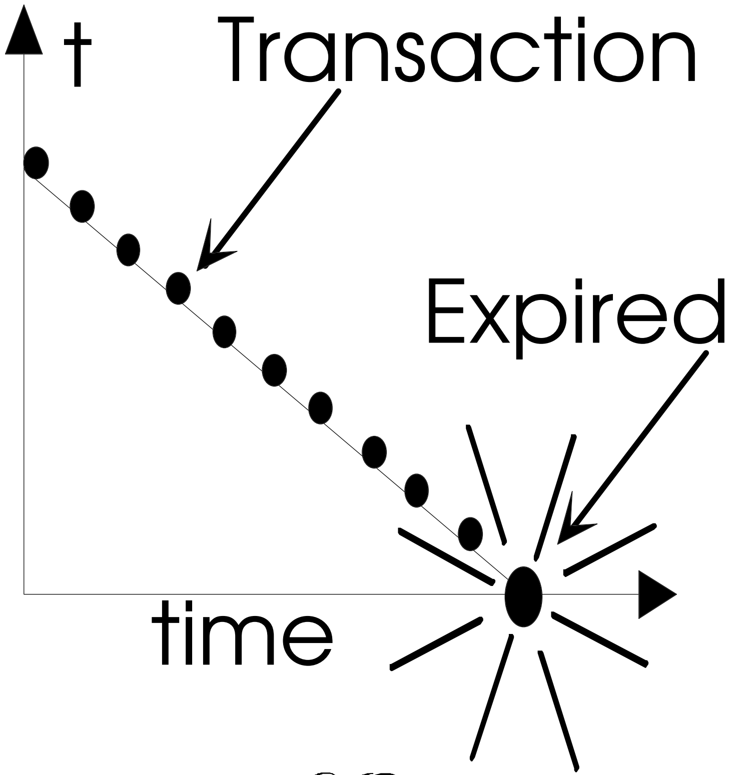
VHDL_Timing.concurrent/sequential_assignments

```
ENTITY example IS
   PORT (a, b, c : IN  BIT ; z : OUT BIT);
END example;
--
ARCHITECTURE concurrent OF example IS
    SIGNAL w, x, y : BIT;
BEGIN
    w <= NOT a AFTER 12 NS;
    x <= a AND b AFTER 12 NS;
    y <= c AND w AFTER 12 NS;
    z <= x OR y AFTER 12 NS;
END concurrent;
```



- A concurrency example
- Properly model gate level circuit by concurrency
- Assume *a* changes from '1' to '0' at time 0

VHDL_Timing.events, transactions, delta_delay

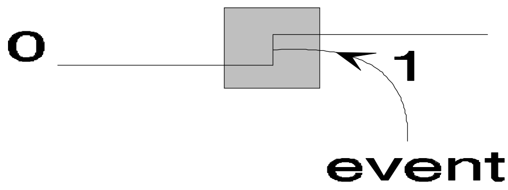


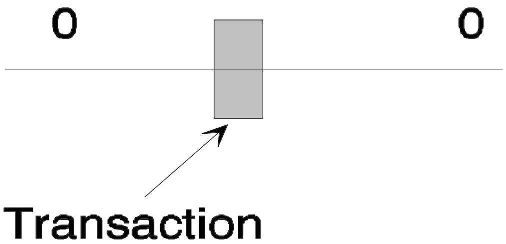
- A signal assignment causes the placement of a transaction on the driver of the left hand side signal

Transaction Representation: (value , time)

- A transaction has a value and a time component
- The time component of a transaction continuously decreases
- Transaction is expired when its time component reaches 0
- A signal receives value when a transaction on its driver expires

VHDL Timing.events, transactions, delta_delay



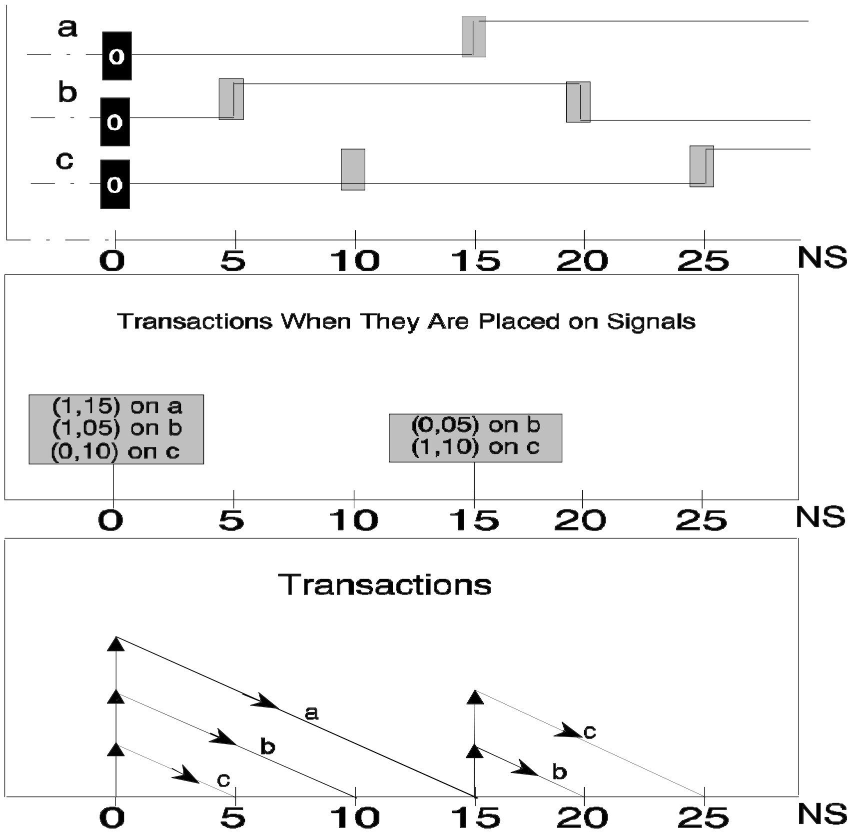


- Assignment to a signal causes an EVENT on the signal if the signal changes value

- An assignment creates a transaction on a signal if the value of the signal does not change

VHDL_Timing.events, transactions, delta_delays

```
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;
```



- Example shows events and transactions
- Concurrent assignment on *a*, *b*, and *c*

VHDL_Timing.events, transactions, delta_delays

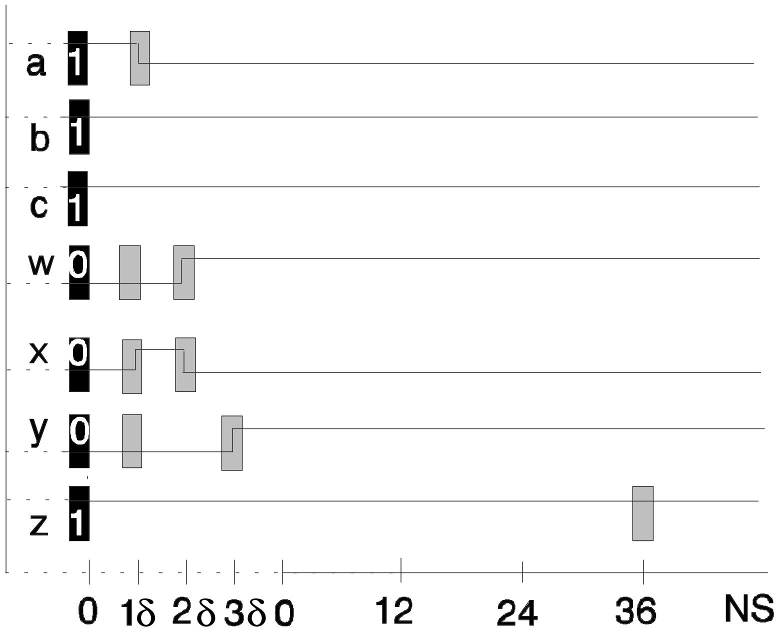
A transaction with zero *time* expires a delta after it is placed on the driver on a signal

delta δ

- 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
- Various forms of delta: Delta d δ +1

VHDL_Timing.events, transactions, delta_delays

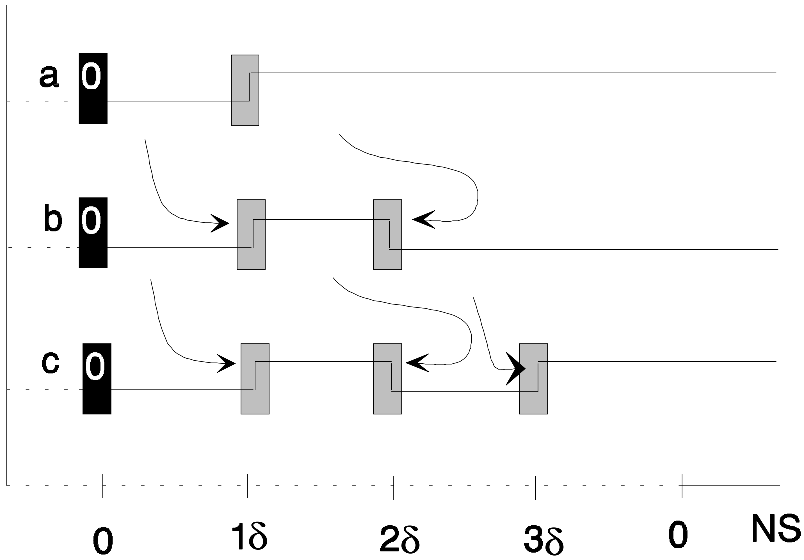
```
ARCHITECTURE not_properly_timed OF example IS
     SIGNAL w, x, y : BIT := '0';
BEGIN
     y <= c AND w;
    w <= NOT a;
     x <= a AND b;
     z <= x OR y AFTER 36 NS;
END not_properly_timed;
```



- A delta delay example

VHDL_Timing.events, transactions, delta_delays

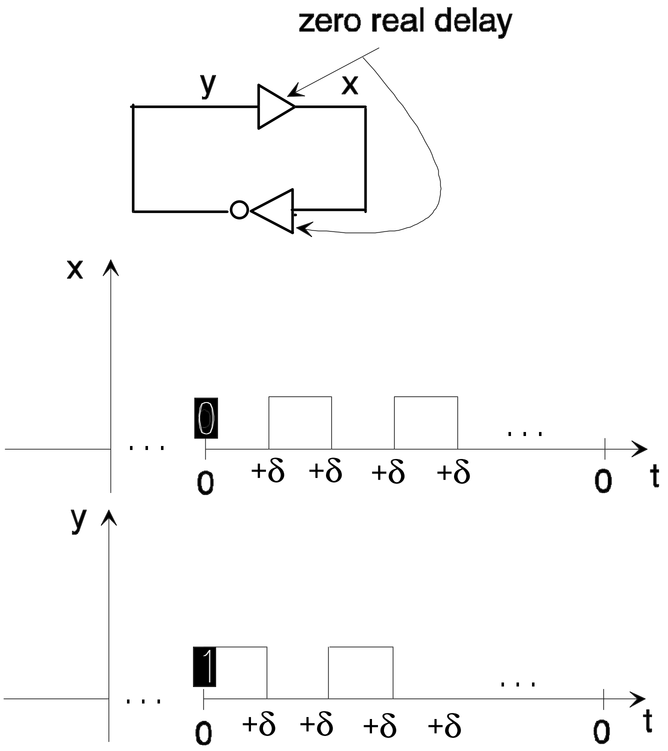
```
ARCHITECTURE concurrent OF timing_demo IS
    SIGNAL a, b, c : BIT := '0';
BEGIN
    a <= '1';
    b <= NOT a;
    c <= NOT b;
END concurrent;
```



- Another delta example
- Zero real time is spent from change on *a* until *c* stabilizes

VHDL_Timing.events, transactions, delta_delays

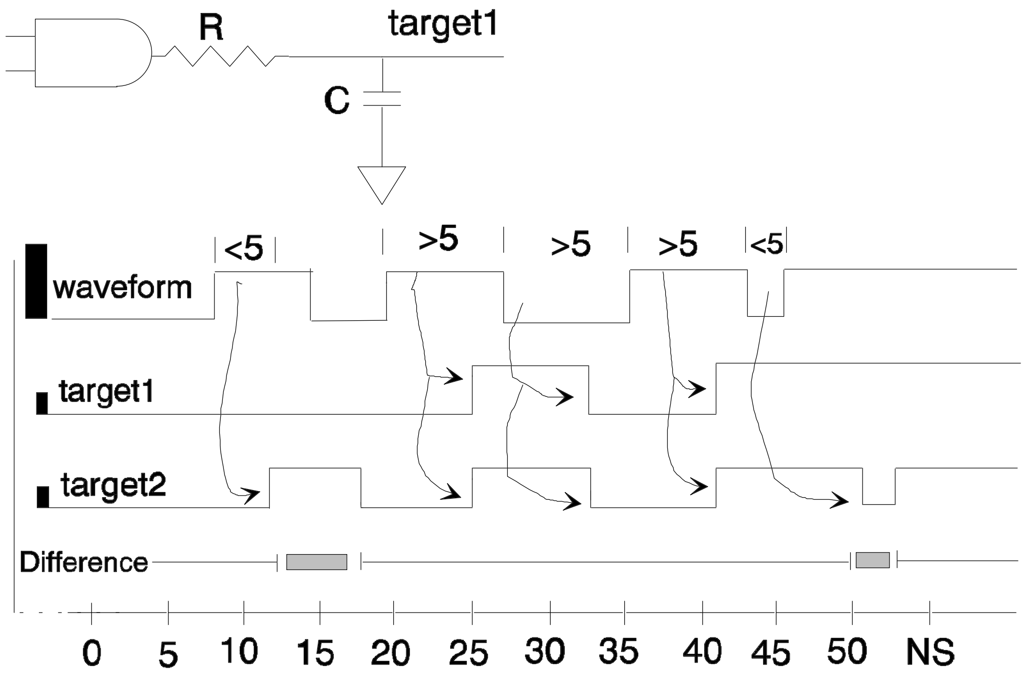
x <= y
y <= NOT X



- An interesting example
- Assignments model zero delay inverter/buffer connection
- Oscillation continues forever in zero real time

VHDL_Timing.delay_modeling

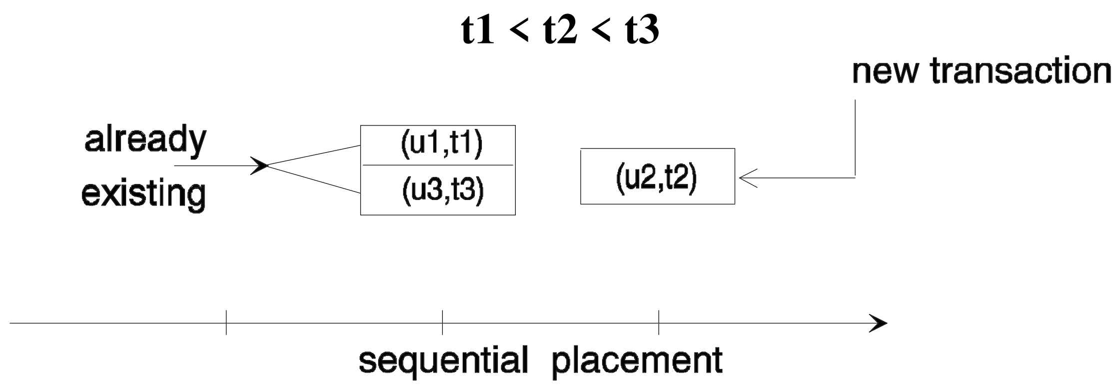
```
ARCHITECTURE delay OF example IS
    SIGNAL target1, target2, waveform : BIT;
    -- this is a comment
BEGIN
    -- the following illustrates inertial delay
    target1 <= waveform AFTER 5 NS;
    -- the following illustrates transport delay
    target2 <= TRANSPORT waveform AFTER 5 NS;
END delay;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



- Inertial delay model represents hardware inertial delay
- Transport delay model is used ingeneration waveforms

VHDL_Timing.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 the new transaction to the driver.	Overwrite existing if different values, otherwise keep both.



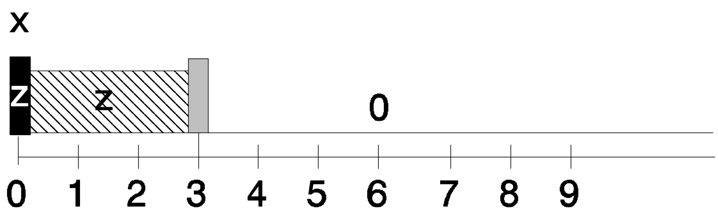
- Sequential placement of transactions justifies delay model
- New transaction AFTER (u1, t1)
- New transaction BEFORE (u3, t3)
- Can use process to place transactions sequentially

VHDL_Timing.sequential_placement_of_transactions

EXAMPLES for:
Inertial / Before
Inertial / After / Different values
Inertial / After / Same values
Transport / Before
Transport / After

Assume x takes '0', '1', and 'Z' values

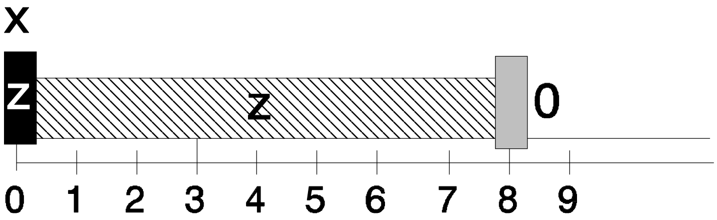
```
ARCHITECTURE sequential OF overwriting_old  IS
      SIGNAL x : tit := 'Z';
BEGIN
 PROCESS  BEGIN
    x <= '1' AFTER 5 NS;     x <= '0' AFTER 3 NS;
    WAIT;
 END PROCESS;
END sequential;
```



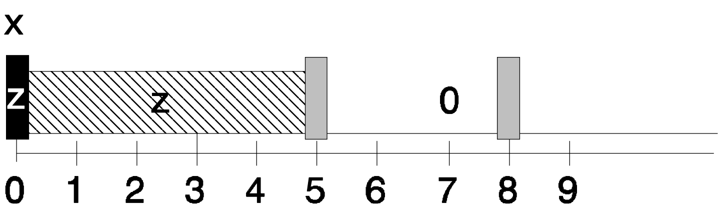
- Inertial, new transaction BEFORE already existing
- Overwrites already existing

VHDL_Timing.sequential_placement_of_transactions

```
ARCHITECTURE sequential OF discarding_old IS
    SIGNAL x : tit := 'Z';
BEGIN
 PROCESS   BEGIN
   x <= '1' AFTER 5 NS;     x <= '0' AFTER 8 NS;
   WAIT;
 END PROCESS;
END sequential;
```



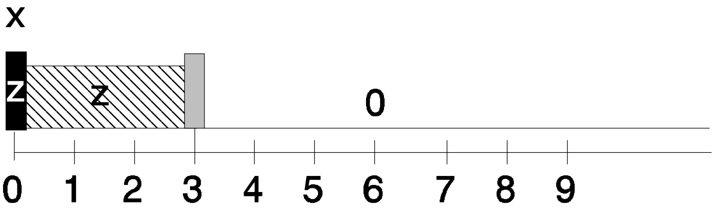

```
ARCHITECTURE sequential OF saving_all IS
    SIGNAL x : tit := 'Z';
BEGIN
 PROCESS  BEGIN
    x <= '0' AFTER 5 NS;   x <= '0' AFTER 8 NS;
    WAIT;
 END PROCESS;
END sequential;
```



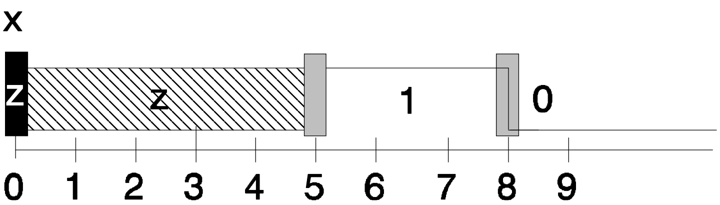
- Inertial, new transaction AFTER existing, different values
- Inertial, new transaction AFTER existing, same values

VHDL_Timing.sequential_placement_of_transactions

```
ARCHITECTURE sequential OF discarding_old IS
    SIGNAL x : tit := 'Z';
BEGIN
 PROCESS  BEGIN
    x <= TRANSPORT '1' AFTER 5 NS;   x <= TRANSPORT '0' AFTER 8 NS;
    WAIT;
 END PROCESS;
END sequential;
```



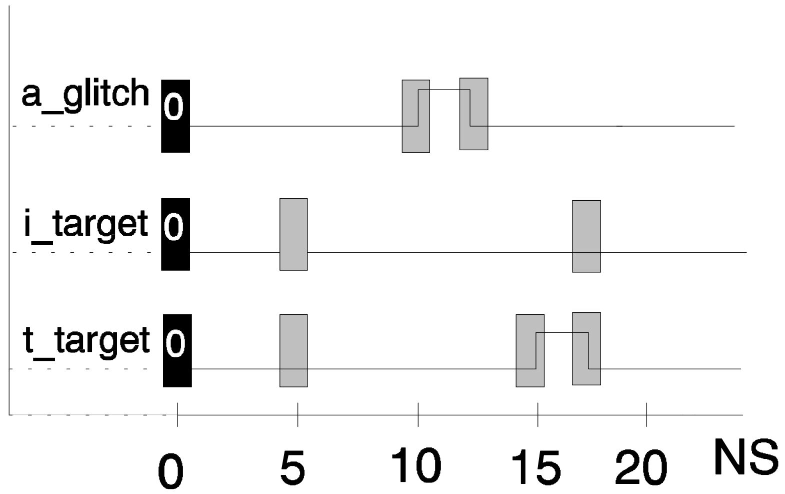

```
ARCHITECTURE sequential OF saving_all IS
    SIGNAL x : tit := 'Z';
BEGIN
 PROCESS   BEGIN
   x <= TRANSPORT '1' AFTER 5 NS;    x <= TRANSPORT '0' AFTER 8 NS;
   WAIT;
 END PROCESS;
END sequential;
```



- Transport, new transaction BEFORE already existing
- Transport, new transaction AFTER already existing

VHDL_Timing.sequential_placement_of_transactions

```
ARCHITECTURE glitch OF inertial_transport_demo IS
     SIGNAL i_target, t_target, a_glitch : BIT := '0';
BEGIN
    a_glitch <= '1' AFTER 10 NS, '0' AFTER 12 NS;
    i_target <= a_glitch AFTER 5 NS;
    t_target <= TRANSPORT a_glitch AFTER 5 NS;
END glitch;
```



- Sequential placement of transactions causes removal of short glitches in the inertial delay

VHDL_Timing.convetions & syntax

```
ARCHITECTURE demo OF ezample 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;
```

architecture declarative_part

architecture statement_part

architecture body

- Syntax details show language constructs
- Use -- (two dashes) for comments
- Use upper case for keywords & predifined words

1. Outline: Introduction, Organization, Outline

2. Review: Behavioral description, Using process statements, Top-down design, Using available components, Wiring predefined components, Wiring from bottom to top, Generation of testbench data, Using procedures

3. VHDL_Timing: Modeling requirements, Objects & classes, Signals & variables, Concurrent & sequential assignments, Events, transactions & delta delays, Delay modeling, Sequential placement of transactions, Conventions

4. Structural_Description_of_Hardware: Wiring parts into larger designs, Start with primitives, Wire gates into general purpose components, Use iterative constructs, Generate testbenches, Show binding alternatives, Use gate-based components for a larger design

5. Design_Organization: Subprograms, Packaging, Parameter specification, Parametrization, Top level configuration, Design libraries, A complete example

6. Utilities_for_high_level_descriptions: Language aspects, Types, Over loading subprograms operators, Predefined Attributes, User defined attributes

7. Dataflow: Constructs for dataflow descriptions, Multiplexing and clocking, Multiple assignments, State machines, Open collector gates, A complete dataflow example, Load dependent timing

8. Behavioral_Descriptions: Constructs for sequential descriptions, Assertion for behavioral checks, Handshaking constructs, Timing control, Formatted I/O, MSI parts, A complete MSI based design

9. STANDARDS: MVL9: logic value system, Logic type, Operators, Conversions; VHDL'93: Operators, Delay model, Instantiation, Binding, Attributes, Signal assignments, Report, Postponed process