Behavioural modelling

Lecture 3

Outline

- 3.1. Processes. Sequential statements. WAIT statement.
- 3.2. Simulation mechanism.
- 3.3. Sequential signal assignment. Transport delay and inertial delay mechanisms.

3.1. Processes. Sequential statements. WAIT statement

Process statement

- The fundamental element in VHDL for sequential (behavioural) modelling is the process
- The process is basically a set of statements that will be executed sequentially and that is activated in response to certain events
- PROCESS statement is a concurrent statement
- Inside an architecture can be several PROCESS statements, that are executed concurrently (in parallel)
- The processes from an architecture are executed in parallel with other processes, from other architectures from the same design (model)
- Processes can communicate with each other **only** by using signals !

PROCESS

Process_statement::=

[label:] [POSTPONED] PROCESS [(sensitivity_list)] [IS]

process_declarative_part

BEGIN

{sequential_statement}

END PROCESS [label];

A process may have a *label* (optional).

A process can be postponed – will be discussed.

After the keyword PROCESS it can be a sensitivity list, which is a list of signals, between parenthesis. The process will be executed each time when there is an event on al least one of the signals from the sensitivity list.

***Event* = modification of the value of a signal.**

PROCESS: declarations

- The process declarative part may contain declarations of:
  - Constants, variables, subprograms, types
- In a process **MAY NOT** be declared:
  - **signals**: because the signals are used for communication between processes, and a signal declared in a process will be visible only inside that process, hence it cannot serve for communication between processes
  - **Components**: a component will be bound to an entity-architecture pair, and an architecture contains processes or equivalent statements, but there cannot exist processes inside other processes !!

PROCESS: statements

- The process body, between BEGIN and END PROCESS, is a loop, in the sense that after the sequential execution by the simulator of all statements, the simulator will continue from BEGIN
- The execution of a process is suspended when a WAIT statement is executed
- The sensitivity list is equivalent with a WAIT ON signals_from_the_sensitivity_list statement placed before END PROCESS
- VHDL syntax does not allow a process to have both sensitivity list and WAIT statement(s)
- It is an error if a process has neither sensitivity list nor WAIT statements (because the process would be never suspended !)
- Inside a process the simulation time advances ONLY at the execution of a WAIT statement (it will be explained at the simulation mechanism)

PROCESS: statements

- In a process can be the following statements:

1. Variable assignments
2. Sequential signal assignments
3. IF statement
4. CASE statement
5. LOOP statement, with EXIT and NEXT
6. Sequential procedure calls – will be discussed in another chapter
7. Sequential ASSERT and REPORT statements
8. NULL statement – doesn’t do anything
9. WAIT statement

PROCESS: statements

- Statements 2, 6 and 7 can be also concurrent, all others can be only sequential (i.e., they can appear only in processes and subprograms)
- In a process may not exist:
  - Other PROCESS statements
  - Component instantiation statements
  - BLOCK statements
  - GENERATE statements
- Another sequential statement is RETURN, which can appear only in subprograms (but not in processes)

Sequential statements

1. Variable_assignment_statement::= Example:

target := expression ; `x := x+3;`

where: *target* – object belonging to class *variable*

- having the same base type as the *expression*

When the variable assignment statement is executed the expression from the right side is evaluated and the value of the expression is given to the variable *immediately*, i.e. the variable updates its value at the current simulation time (we will see that the sequential signal assignment statement works differently)

Sequential statements: IF

3. IF_statement::=

IF condition THEN

sequence_of_statements

{ELSIF condition THEN

sequence_of_statements}

[ELSE

sequence_of_statements]

END IF;

Example:

```
IF x<0 THEN
        cnt_n:=cnt_n+1;
ELSIF x>0 THEN
        cnt_p:=cnt_p+1;
ELSE
        cnt_z:=cnt_z+1;
END IF;
```

Conditions are evaluated one by one, until one is found to be true -> the corresponding sequence of statements will be executed.

If no condition is true, the sequence of statements from the ELSE clause is executed, if ELSE is present. If there is no ELSE, then no sequence of statements is executed (the IF statement is left).

Sequential statements : CASE

4. Case_statement::=

CASE expression IS

WHEN choices => sequence_of_statements

{WHEN choices => sequence_of_statements}

END CASE;

choices ::= choice {|choice}

choice ::=

discrete_range | simple_expression | OTHERS

Example:

```
CASE x IS
        WHEN 0 TO 9 => y:=y+1; -- discrete range
        WHEN 10|20|30|40|50 => z:=z+1; -- simple expressions
        WHEN OTHERS => q:=q+1; --OTHERS
END CASE;
```

CASE

- The *expression* must be of a discrete type (enumeration or integer) or it must be a one-dimension array of a character type (an enumeration type that has at least one element a character literal => BIT is a character type)
- Execution of the CASE statement consists in the evaluation of the expression, followed by the execution of the sequence of statements from the choices that contain the value of the expression
- The choices must be distinct and they must cover all the possible values of the expression, either explicitly or using OTHERS

LOOP

4. LOOP_statement ::=

[label] [iteration_scheme] LOOP

sequence_of_statements

END LOOP [label];

iteration_scheme ::=

WHILE condition

| FOR identifier IN discrete_range

```
--Example:
i:=3;-- i must be declared
WHILE i>0 LOOP
       IF v(i)='1' THEN
            nr_of_1 := nr_of_1 +1;
       END IF;
       i:=i-1;
END LOOP
```

```
--Same example with FOR:
FOR i IN 0 TO 3 LOOP
       IF v(i)='1' THEN
                  nr_of_1 :=
nr_of_1 +1;
       END IF;
END LOOP;
```

-- in this case the variable i (the loop parameter) must not be declared

LOOP

If the iteration scheme is missing, the LOOP will be executed indefinitely. EXIT or, if the loop is inside a subprogram, RETURN statement can be used to leave the loop.

NEXT_statement::=

NEXT [loop_label] [WHEN condition]

EXIT_statement::=

EXIT [loop_label] [WHEN condition]

NEXT statement ends the current iteration and the execution continues with the next iteration, while EXIT statement completes the execution of the enclosing loop.

If a loop label is specified for EXIT or NEXT, they refer to the labelled loop, otherwise EXIT or NEXT refer to the innermost loop.

Example:

```
l1: LOOP
     l2: LOOP
          l3: LOOP
             NEXT l1 WHEN cond1;
             EXIT l2 WHEN cond2;
          END LOOP l3;
     END LOOP l2;
END LOOP l1;
```

ASSERT and REPORT

7. ASSERT_statement ::=

    ASSERT condition

    [REPORT expression]

    [SEVERITY severity_expression];

The statement verifies the condition and displays a message if the condition is **false**.

expression is a string of characters specified by the programmer. Implicitly is “assertion violation”.

Severity_expression is of type SEVERITY_LEVEL, predefined in VHDL:

```
TYPE SEVERITY_LEVEL IS (NOTE, WARNING, ERROR, FAILURE);
```

-- implicit value is ERROR

The severity degree increases from NOTE to FAILURE. Implementation dependent, the simulator can be configured to stop the simulation if the severity level is higher than a threshold.

ASSERT and REPORT

ASSERT statement is used mostly for verifying conditions like set-up and hold time for sequential logic, forbidden input combination (like for R-S latches).

In VHDL'93 ìt was introduced the REPORT statement, similar with ASSERT, but without condition verification:

REPORT_statement ::=

REPORT expression

[SEVERITY severity_expression];

```
--ASSERT example:
--displays a message if both inputs are '1'
ASSERT NOT ((reset='1') AND (set='1'))
REPORT "forbidden input combination" SEVERITY WARNING;
```

WAIT statement

WAIT_statement ::=

WAIT [sensitivity_clause] [condition_clause] [timeout_clause];

sensitivity_clause ::=

ON signal_name { , signal_name}

condition_clause ::=

UNTIL condition

timeout_clause ::=

FOR time_expression

Clauses are optional, but if they appear, then they should be in this order.

WAIT may appear only in processes.

When the WAIT statement is executed, a process is suspended. The process will resume either because one of the signals from the sensitivity list has an event, either because the condition from the condition clause is met, or because the time specified by the timeout clause has expired.

Forms of WAIT

1. WAIT
   - Suspends forever the execution of the process
2. WAIT ON s1,s2,s3;
   - The process resumes if an event takes place on at least one of the signals s1, s2, s3 (one signal changes its value).
3. WAIT UNTIL (b+c < 10);
   - The process rsumes when the condition b+c <10 is meat.
   - In this example either b, or c, or both b and c should be signals (there must be signals in the condition)
   - The condition is evaluated when there is an event on at least one of the signals from the condition
   - If b and c are both signals, then this statement is equivalent with:
     - WAIT ON b,c UNTIL b+c<10;
4. WAIT FOR 2000ns;
   - Process resumes after 2000 ns (i.e. after 2 us)

Forms of WAIT

5. WAIT ON s1,s2, s3 UNTIL b+c <10;
   - When an event takes place on s1, s2 or s3, the condition is evaluated and, if true, the process resumes; if the condition is false, the process remains suspended
   - It is not necessary to have signals in condition when the sensitivity clause is present
   - In this example, if b and c are signals, an event on b or c will NOT determine the evaluation of the condition, hence the statement is NOT equivalent with:
     - WAIT ON s1,s2,s3,b,c UNTIL b+c <10;

Forms of WAIT

6. WAIT ON s1,s2,s3 UNTIL b+c<10 FOR 2000ns;
   - When this statement is executed
     - execution of the process is suspended
     - The process waits for an event on s1, s2 or s3. If the event take place before the 2000 ns timer expires, the condition is evaluated. If it is true, the process resumes and the 2000 ns timer is stopped
     - A 2000 ns timer is started: if the timer expires (which means that the process didn’t resume because of the ON and UNTIL clauses) then the process resumes
   - From a logical point of view, the clauses ON and UNTIL are in parallel with the FOR clause.

Sensitivity list and WAIT

A process must have either sensitivity list, or WAIT statement(s), but not both of them.

The sensitivity list is equivalent with a statement `WAIT ON signals from the sensitivity list` situated before END PROCESS:

```
PROCESS(s1, s2, s3)
BEGIN
        ...
END PROCESS
```

⇔

```
PROCESS
BEGIN
        ...
  WAIT ON s1, s2, s3;
END PROCESS
```

A dynamic sensitivity list can be created with WAIT ON statements:

```
PROCESS
BEGIN
        WAIT ON s1;
        . . .
        WAIT ON s2, s3;
END PROCESS;
```

3.2. Simulation. The simulation mechanism

Simulation

- VHDL is defined in terms of simulation (it has a simulation oriented semantics).
- If the simulation model has a hierarchical structure, then an elaboration phase takes place before the simulation starts. In this phase:
  - Components from structural descriptions are bound to entities
  - The hierarchical structure is *flattened*
  - It results a set of processes that communicate through signals
- The resulted structure of processes will be simulated under the control of an *event driven simulation kernel* named ***simulator***.

Simulation

- The simulator updates signal values, possibly resulting events on some signals
- Processes that had been suspended (with WAIT ON) and that are waiting for these events will resume execution.
- Processes that resumed will execute until they will be suspended again
- When all processes are suspended, the simulator will update again the signals' values.

Simulation

- Hence the simulation is a cyclic process with 2 phases:
  - signal update phase
  - process execution phase
- In this way it is modelled the functioning of hardware circuits, that:
  - First respond to the changes of the signal values at their inputs
  - Then they change their outputs, which may lead to the activation of other circuits

Simulation time

- Simulation is performed with the help of a global clock, which holds the current simulation time
- The simulation time is a purely conventional time, not related with
  - The physical time
  - The computer time (processor's clock, or other events related to the operating system)
- Simulation time is incremented during simulation, but
- The simulation time has only discrete values, namely:
  - The moments when values are scheduled for signals (specified by clauses AFTER time_value
  - The moments given by the time expressions from the statements WAIT FOR time_expression

Simulation: signals

- The simulator will update signals' values only at certain time moments.
- A signal assignment statement from a process
  - Has no direct effect on the signal's value
  - Only schedules one or more values that the signal will take in the future
- The simulation mechanism (the simulator) uses a data structure called the signal *driver*.
- The driver is the **source** of a signal.
- A process which assigns values to a signal will automatically create a driver for that signal.
- Normally a signal has a single driver:
  - A signal that has more than one driver is called a *resolved signal; a resolution function* is used in order to solve the conflict between drivers and to determine the signal's value
  - A resolution function is a special kind of function, that is written by the programmer, but is called by the simulator every time when a new value is scheduled to the resolved signal
  - Resolved signals are declared in a special way (will be discussed later)

Signals. Drivers

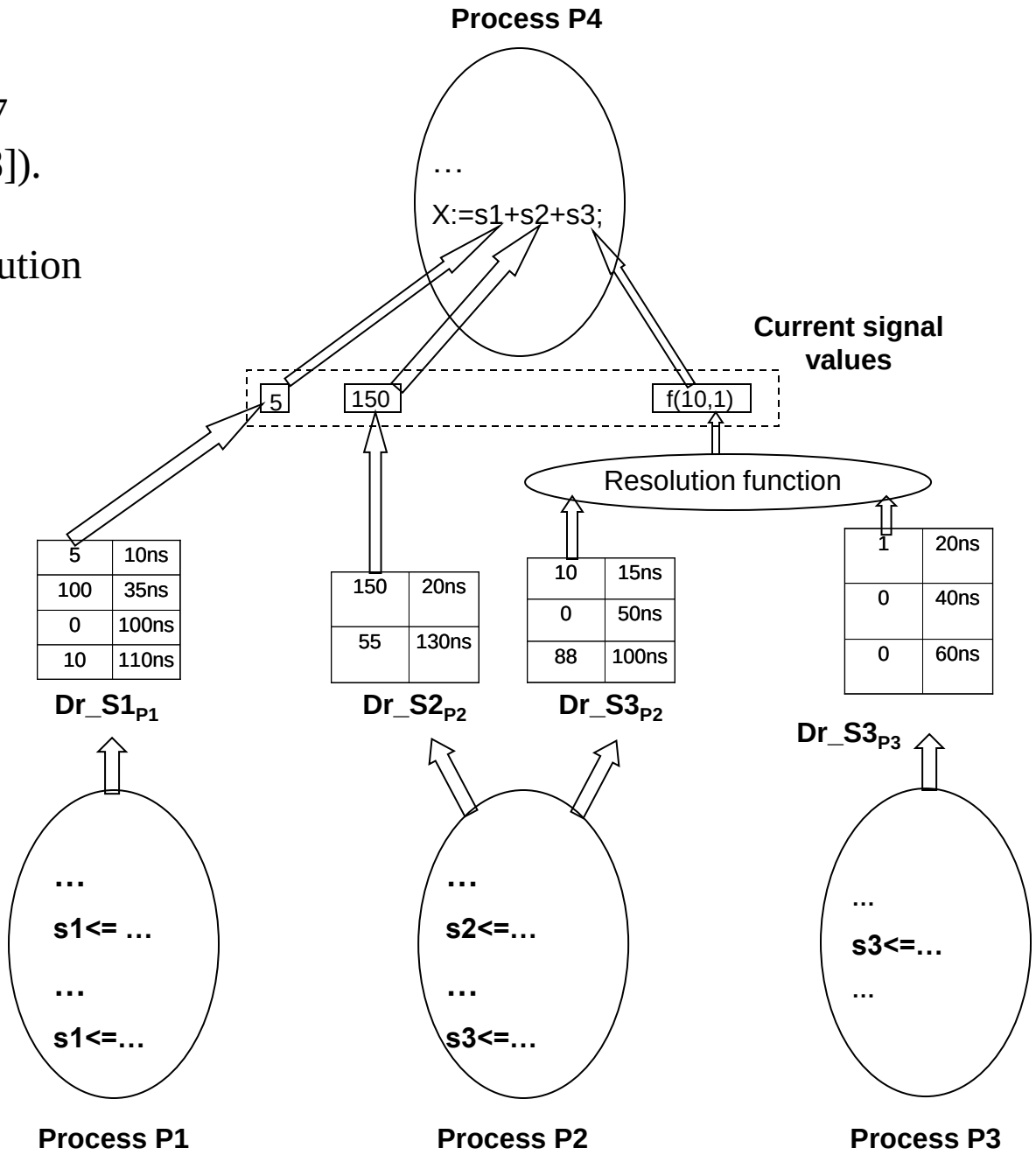
- The driver contains the *projected output waveform* of the signal
  - Which consists of a set of ***transactions***
  - A transaction is a pair `(value, time moment)`
  - The signal is scheduled to have these values at the corresponding time moments
- A signal assignment statement inside a process affects only the projected output waveform of the signal by:
  - Scheduling one or more transactions on the signal driver
  - Possibly by deleting some transactions from the signal driver
- In each driver the transactions are placed in increasing order of their time components

Signals. Drivers

- The current value of a driver is the value component of the first transaction from the driver.
- The current values of all signals are kept by the simulator in a data stucture created automatically at the beginning of the simulation.
- First transaction from a driver is the only transaction whose time component is not greater than the current simulation time.
- When the simulation time advances and becomes equal to the time component of the next transaction from the driver
  - First transaction is deleted and
  - Next transaction becomes the first transaction
  - In this way the driver takes new values and, no matter if the new value is different or not from the previous value, we say that the **signal is active in the current simulation cycle**
- Hence, during each simulation cycle the values of the active signals are updated by the simulator
- If, as a consequence of this update, the signal value is modified, than it is said that the signal has had an ***event***.
- Hence, **event = modification** of a signal value.

- Next figure presents a graphic illustration of these aspects.

Fig 2. (fig 2.7 from [EKP98]). Processes, drivers, resolution functions



The simulation cycle

- Simulation cycle contains in a more rigorous way what was discussed about simulation.
- The understanding of the simulation cycle is necessary for understanding the VHDL language and for understanding the functioning of any simulation model.
- In one form or another it will be required at the exam !

The simulation cycle

- Has three phases:
  - Phase 0: elaboration
  - Phase 1: initialization
  - Phase 2: the simulation cycle
- First 2 phases execute only once, the third phase repeats periodically.
- Notations:
  - Tc: current simulation time
  - Tn (time next): next simulation time.

The simulation cycle

- Phase 0: elaboration:
  - The components from structural descriptions are bound to entities
  - The hierarchical structure *flattens*
  - Resulting a set of processes which are connected (communicate) through signals
- Phase 1: initialization:
  1. The current simulation time is set to 0 ns (Tc ← 0ns)
  2. All signals are initialized with the value from the declaration, explicitly or implicitly, according to the rules for that type.
  3. All processes are executed until they suspend (all of them)
  4. Next simulation time (Tn) is computed in the same way like in phase 2.

The simulation cycle

- Phase 2: the simulation cycle:
  1. The current simulation time is set to Tn (Tc ← Tn) (simulation time advances)
  2. All active signals are updated (it was explained before how)
  3. Each process that was suspended waiting on signal events that occurred resumes. Will resume also the processes that were waiting for a certain time to elapse (due to WAIT FOR) and that time has completed.
  4. Processes that resumed will execute until they suspend
  5. The time Tn of the next simulation cycle is computed as the minimum between the following three values:
     - TIME'HIGH (maximum possible value of the type TIME)
     - The next time when a driver becomes active due to a clause AFTER time_expression
     - The next time when a process resumes (time specified by a WAIT with a FOR time_expression clause).

Delta delays. Delta cycles

In VHDL there is a mechanism that orders the events that take place at the same simulation time.

A delta delay is an infinitesimally small delay that separates events occurring in successive simulation cycles but at the same simulation time.

```
Example:
ENTITY delata_delay_example IS
        PORT(x,y: IN BIT; z: OUT BIT);
END ENTITY;
ARCHITECTURE ex OF delta_delay_ex IS
        SIGNAL s: BIT;
```

Delta delays. Delta cycles

```
BEGIN
    and_gate: PROCESS (x,y)
              BEGIN
                    s <= x AND y;
              END PROCESS and_gate;
   inv_gate:  PROCESS(s)
              BEGIN
                    z <= NOT s;
              END PROCESS inv_gate;
END ARCHITECTURE;
```

Delta delays. Delta cycles

- Suppose that y='1' and that at the moment T the signal x changes its value (has an event).
- In the first simulation cycle:
  - Signal x is updated
  - Process and_gate resumes and execute until it suspends
  - A new value is placed on the signal's s driver
  - Next simulation time is computed as:
    - Tn = T+0ns (practically Tn = T+Δ)

Example with delta cycles

- In the second simulation cycle:
  - Tc ←Tn (Tc = T + Δ)
  - Signal s is updated
  - The process inv_gate resumes and executes
  - An event is scheduled on signal's z driver at the moment Tn=Tc+0ns (=T+2 Δ)
- In the third simulation cycle:
  - Tc ←Tn (=T+2 Δ)
  - Signal's z driver is updated.

Delta delays. Delta cycles

- Between the moment when the signal x modifies its value and the moment when z modifies its value 3 simulation cycles take place, but without the modification of the simulation time.
- We say that the events from the 3 simulation cycles are separated by delta delays.
- Definition: a simulation cycle that is performed at the same simulation time as the previous cycle is called a *delta cycle*.
- Delta delay mechanisms very important when VHDL is used for synthesis.

Postponed processes

- Definition: a postponed process will be executed in the simulation cycle in which it resumes only if the next simulation cycle is not a delta cycle.
- Actually the postponed processes are executed only after all delta cycles that take place at a certain simulation time.
- Postponed processes are not allowed to generate new delta cycles: it is an error if a postponed process generates delta cycles (if it contains AFTER 0 ns or WAIT FOR 0ns)
- Due to postponed processes the simulation cycle is modified:
  - In phase 2, step 4: the processes that resumed will be executed if they are not postponed processes.
- New steps are added to the second phase:
  - step 6: if Tn /= TC (Tn is different from Tc), i.e., if next simulation cycle is not a delta cycle, then postponed processes are executed.
  - step 7: Tn is computed like in step 5.

3.3.Sequential signal assignment Transport delay and inertial delay mechanisms

Sequential signal assignment statement

The projected output waveform stored in the driver of a signal can be modified by a sequential signal assignment statement

Synthax:

Signal_assignment_statement::=

target <= [TRANSPORT | [REJECT time_expression] INERTIAL] waveform;

- target:
  - object of class signal (signal or port)
  - That has the same base type as the *value* components of the transactions produced by the waveform
- waveform ::= waveform_element {, waveform_element}
- waveform_elelement ::= value AFTER time_expression

Sequential signal assignment statement

•Evaluation of one waveform_element produces a single transaction.

•The time component of a transaction is equal with Tc + time_expression (from waveform element).

•The value component of a transaction is determined by the *value* expression from the waveform_element

• A sequential signal assignment will affect only the driver of the signal !

•The signal value will NOT be modified in the same simulation cycle in which the signal assignment statement is executed.

•Example. In a PROCESS there is the following code sequence:

```
…
X <= 1;
IF X=1 THEN
       sequence_1_of_statements
ELSE
       sequence_2_of_statements
END IF;
…
```

Sequential signal assignment statement

•If the signal's X value before the process execution was different from 1, then sequence_2_of_statements will be executed because the sequential signal assignment (X<= 1;) DOES NOT modify the current signal value.

•In order to be executed sequence_1_of_statements, a WAIT statement (either WAIT FOR 0 ns; or WAIT ON X; in this case) should have exist after the signal assignment statement.

• Then, when the WAIT statement is executed:

- •the process is suspended
- •Signal values are updated
- •The process resumes (in the next simulation cycle, which is a delta cycle in this case)
- •IF condition is evaluated, it is found true and it will be executed sequence_1_of_statements

•We can notice the difference between the execution of a sequential signal assignmen statement and the execution of a variable assignment statement (had X been a variable, the first sequence of statements from IF would had been executed, without being necessary to add a WAIT statement.

Inertial delay and transport delay mechanisms

- The evaluation of the waveform at a signal assignment statement
  - results in a sequence of transactions
  - each transactions corresponds to a waveform element
- These transactions are called *new transactions*
  - They must be in increasing order of their time components
- The sequence of new transactions will update the old transactions (i.e. the transactions that constituted the projected output waveform of the driver)
  - The way this update takes place depends on the *delay mechanism,* which can be *inertial delay* or *transport delay*.
  - The mechanism can be explicitly specified on the signal assignment statement using the keywords *inertial* or *transport*.
  - If not explicitly specified, implicitly it is INERTIAL delay.

Inertial delay and transport delay

- Transport delay
  - Models transmission lines (wires or conductive traces)
  - A signal (a pulse) present at one end will propagate to the other end no matter how short it is; at the other end the pulse will have the same shape as initially, being only delayed
  - Has to be specified using the keyword TRANSPORT
- Inertial delay
  - Models switching circuits
  - A signal from one input must have a certain duration (*called pulse rejection limit*) in order to be taken into consideration by the circuit (to propagate to the output) (For a real signal to be considered by a circuit, it must have a certain energy; in VHDL we consider the duration of the signal instead of its energy)
    - If the rejection limit is not explicitly specified, it will be equal to the time component of the first new transaction
  - It means that a signal that is too short does not propagate to the circuit's output (it is not taken into consideration by the circuit).
  - It is implicit in VHDL because:
    - We model more often circuits than transmission lines
    - Had the transport delay mechanism been implicit, then the simulator would have been unnecessarily loaded (the pulses that are too short would propagate, too).

Example

An entity having the input x and outputs z1, z2, z3, like in figure 3, contains the following statements in architecture:

Fig 3 (after figs 2.10 and 2.11 from [EKP98]) Buffer with transport delay and inertial delay

```
Z1 <= TRANSPORT x AFTER 15ns;

Z2 <= x AFTER 15ns;

Z3 <= REJECT 8ns x AFTER 15ns;

Where the input x is:

X<='1' AFTER 10ns, '0' AFTER 20ns, '1' AFTER
30ns, '0' AFTER 50ns, '1' AFTER 70ns, '0'
AFTER 75ns, '1' AFTER 85ns, '0' AFTER 90ns;
```

The resulting waveforms of the signals x, z1, z2, z3 are shown next:

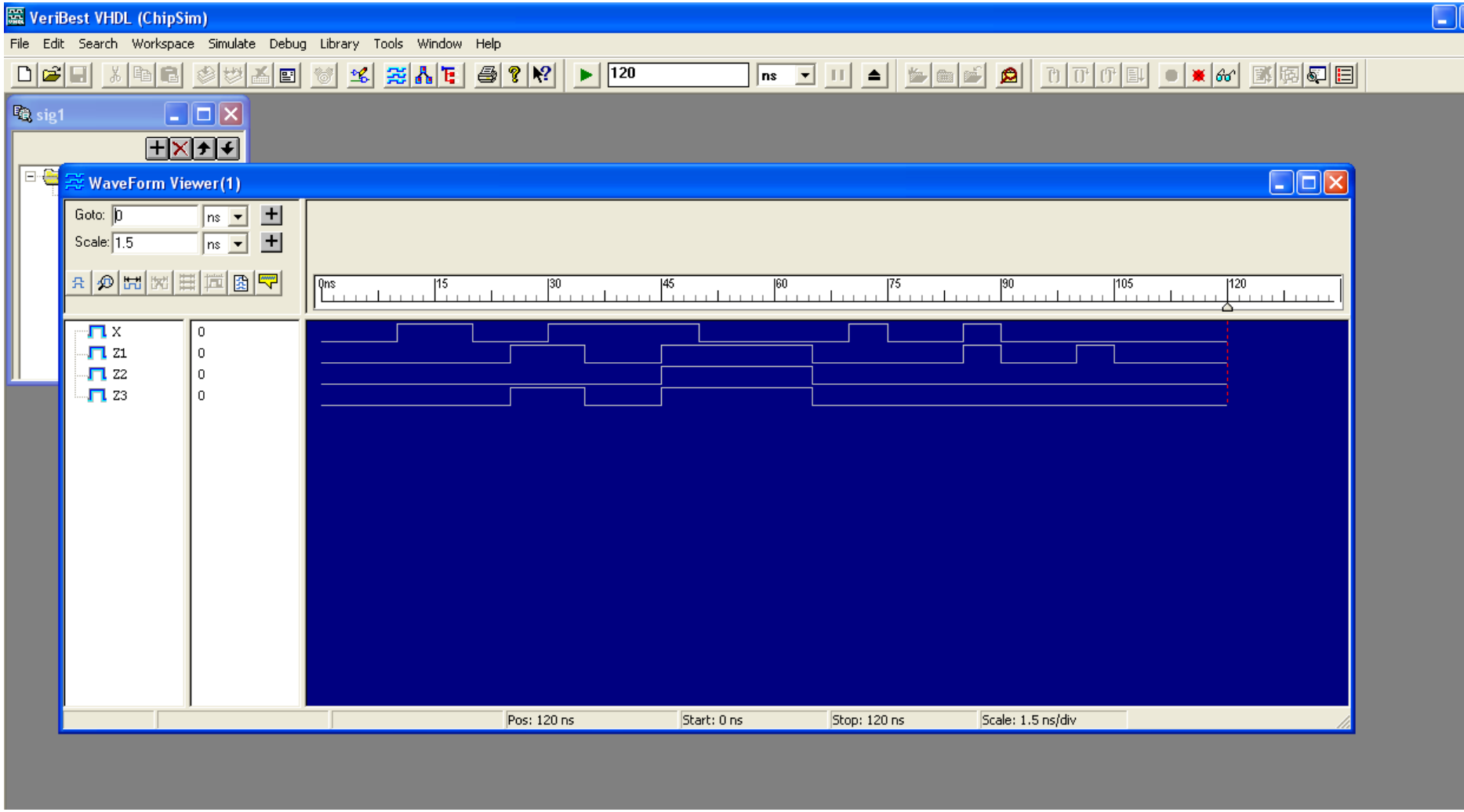


Fig 4. The waveforms for the signals from fig 3 (buffer with transport and inertial delay)

Rules for transport delay mechanism

- The projected output waveform of the signal is updated according to the following steps:
  1. All old transactions scheduled to occur at the same time or after the first new transaction are deleted
  2. The new transactions are appended at the end of the driver
- Consequence:
  - The old transactions scheduled before the first new transaction are not affected

Transport delay example

Suppose that inside a process there is the following code sequence:

```
S<=TRANSPORT 100 AFTER 20ns, 15 AFTER 35ns;

S<=TRANSPORT 10 AFTER 40ns;

S<=TRANSPORT 25 AFTER 38ns;
```

We assume that Tc=100 ns and that the projected output waveform consists of a single transaction with the value component S=0.

After the first two statements the driver of S will look like this:

0	100	15	10
100ns	120ns	135ns	140ns

Transport delay example (cont'd)

When the third statement is executed:

• last old transaction will be deleted, according to step 1.

•The transaction with the time component Tc + 38 ns is appended to the driver, according to 2.

The signal driver will look like this:

0	100	15	25
100ns	120ns	135ns	138ns

Rules for inertial delay mechanism

- The projected output waveform of the signal is updated according to the following steps:
  1. All old transactions scheduled to occur at the same time or after the first new transaction are deleted
  2. The new transactions are appended at the end of the driver
  3. all the old transactions that are scheduled to occur at times between the time of the first new transaction minus the pulse rejection limit and the time of the first new transaction ($[t_{new} – t_{rejection}, t_{new}]$) are deleted; excepted are those transactions which are immediately preceding the first new transaction and have the same value as it has.

Inertial delay example

Suppose that in a process there is the following code sequence:

```
S<=100 AFTER 20ns, 15 AFTER 35ns;--(1)

S<=8 AFTER 40ns, 2 AFTER 60ns, 5 AFTER 80ns, 1 AFTER 100ns;--(2)

S<=REJECT 35ns INERTIAL 5 AFTER 90ns;--(3)
```

Suppose that Tc=100 ns and that the projected output waveform consists of a single transaction with the value component S=0.

After the first transaction the driver of S will look like this:

0	100	15
100ns	120ns	135ns

Inertial delay example (2)

When statement (2) is executed:

- t_{new}=140ns, t_{rej}=40ns (why ?), t_{new}-t_{rej}=100ns
- Step 1: no transaction is deleted
- Step 2: the new transactions are appended to the driver
- Step 3: the old transactions in the interval [100ns, 140ns] are deleted.

Driver of S will look like:

0	8	2	5	1
100ns	140ns	160ns	180ns	200ns

When statement (3) is executed:

- t_{new}=190ns, t_{rej}=35ns , t_{new}-t_{rej}=155ns
- Step 1: the last transaction is deleted, because it has t=200ns>t_{new}=190ns
- Step 2: the new transactions are appended to the driver

Inertial delay example (3)

After step 2 the driver will look like:

0	8	2	5	5
100ns	140ns	160ns	180ns	190ns

•Step 3: the old transactions in the interval [155ns, 190ns] are deleted, except the transaction (5, 180ns) which is immediately preceding the first new transaction and has the same value like it.

The driver of S will look at the end this way:

0	8	5	5
100ns	140ns	180ns	190ns

Passive processes

- Have the following characteristics:
  1. **DO Not assign values to signals**
  2. Can appear in the statement part of an entity declaration:
     - More precisely, in the statement part of an entity declaration can appear ONLY passive processes or equivalent statements
  3. Most often they are used for verifying conditions.

Important observation

- ***Inside a process the simulation time does not change between WAIT statements.***
  - The simulation time is modified only when a WAIT statement is executed.
- A process which resumes after it was suspended by a WAIT statement will be executed in the next simulation cycle
  - Before the process resumes a signal update takes place (signal values are updated).
- Sensitivity lists are assimilated to WAIT statements