Module: Electronics & Telecomunication, 5rd year

Programmable Logical Devices

Variables & signals Simulation

Agenda

Signals in VHDL
- processes, declarations, delays, hazards

Simulation
- Delta-Time cycle, sensitivity list, resolution function, attributes

Signals in VHDL
Signal declarations in VHDL

- Signals may be declared in:
  - packets – global signals
  - declaration section in entity - signals global for entity
    - ports: **in**, **out**, **inout**, **buffer**
    - other declarations
  - declaration section in architecture - signals local for architecture
- Signals are initialized with := operator
- Values are given to signals with <= operator

Signals in VHDL
Signal declarations in VHDL

Global signals:

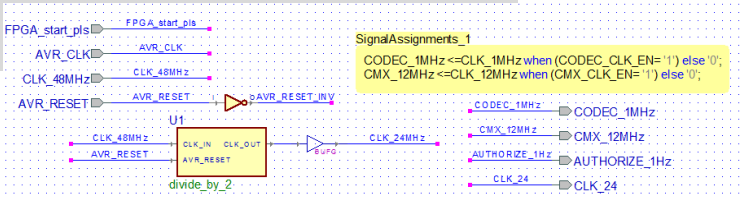
```
package SIGDEC is
  signal RESET: bit := '1';
  signal INIT: bit := '0';
end SIGDEC;
```

Signals global for entity:

```
entity BOARD_DESIGN is
  port (DATA_IN: in bit;
        DATA_OUT: out bit);
  signal SYS_CLK: bit := '1';
end BOARD_DESIGN;
```

Signals local for architecture:

```
architecture DATA_FLOW of BOARD_DESIGN is
  signal INT_BUS: bit;
begin
..........
```



Signals in VHDL

Signal declarations in VHDL

A declaration of signal in the `port` statement of `entity` should specify:
The name of the signal, its direction, type and optionally the initial value
Syntax:

`port` (*name* [,*more_names*]; *direction type* [:=*expression*][;*more ports*]);

Direction	Usage
`in`	The right side of the assignment to the variable or signal
`out`	The left side of the assignment to the variable or signal
`inout`	Both above
`buffer`	As above, but only one driver (*not used in practice*)

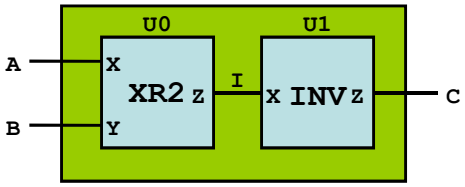
Examples:

```
port (DATA_IN: in bit; DATA_OUT: out bit);
port (B, A: in MyLib.MyPkg.MyType);
```


Signals in VHDL

Structural description

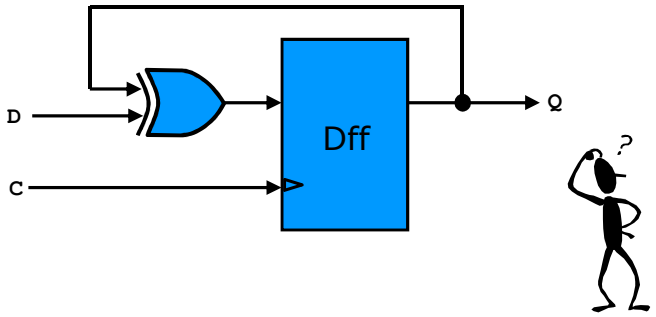
```
entity COMPARE is
  port (A, B: in bit;
        C: out bit);
end COMPARE;
```




```
architecture STRUCTURAL of COMPARE is
signal I: bit; -- internal signal - no direction!
  component XR2 port (X, Y: in bit; Z: out bit);
  end component;
  component INV port (X: in bit; Z: out bit);
  end component;
begin
  U0: XR2 port map (A, B, I);
  U1: INV port map (I, C);
end STRUCTURAL;
```

Signals in VHDL
Local Feedback

Question: What is the most likely direction of the signal Q?



Answer: `buffer`

Signals in VHDL
Buffer usage

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity with_buffer is
port( A : in unsigned(3 downto 0);
      B : in unsigned(3 downto 0);
      Clk : in std_logic;
      C : buffer unsigned(3 downto 0));
end with_buffer;


architecture BEHAVIORAL of with_buffer is

begin

process(Clk)
begin
    if (rising_edge(Clk)) then
        C <= A + B + C;
    end if;
end process;

end BEHAVIORAL;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity without_buffer is
port( A : in unsigned(3 downto 0);
      B : in unsigned(3 downto 0);
      Clk : in std_logic;
      C : out unsigned(3 downto 0));
end without_buffer;

architecture BEHAVIORAL of without_buffer is

signal C_dummy : unsigned(3 downto 0);

begin
C <= C_dummy;  --Assign the intermediate signal
process(Clk)
begin
    if (rising_edge(Clk)) then
        C_dummy <= A + B + C_dummy;
--Use the intermediate signal in actual
calculation.
    end if;
end process;

end BEHAVIORAL;
```

Signals in VHDL
Local Feedback

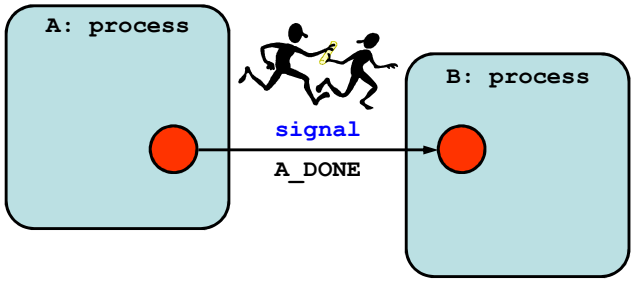
The rules of connections between the modules require corresponding modes of ports, eg: **buffer** ⇔ **buffer**. But this is not convenient...

```
entity ...
port(D: in ...
     C: in ...
     Q: buffer ... );
end entity;

architecture awkward
begin
 process(C)
 begin
  if C ='1' and C'event then
   Q <= Q xor D;
  end if;
 end process;
end awkward;
```

```
entity ...
port(D: in ...
     C: in ...
     Q: out ... ); -- OUT direction
end entity;

architecture good
 signal: T ...      -- T aux signal
begin
 process(C)
 begin
  if C ='1' and C'event then
   T <= T xor D;    -- T usage
  end if;
 end process;
 Q <= T;       -- concurrent assignment
end good;
```


Signals in VHDL
Communication between processes

```
architecture FIRST of ST_UNIT is
signal A_DONE: bit := '0';
begin
.....
```




```
A: process
begin
.....
if S1 then A_DONE <= '1';
```

```
B: process
begin
  wait until A_DONE = '1';
.....
```

Signals in VHDL

Communication between processes

- Processes can communicate with each other by giving the value to the signal(s).
- Process may suspend its operation and wait for the change on its input signal.
- Variables declared in the process may not pass their values to other processes.
- VHDL'93 defines the global variables that may be used for communication between processes.

Signals in VHDL

Transport delay

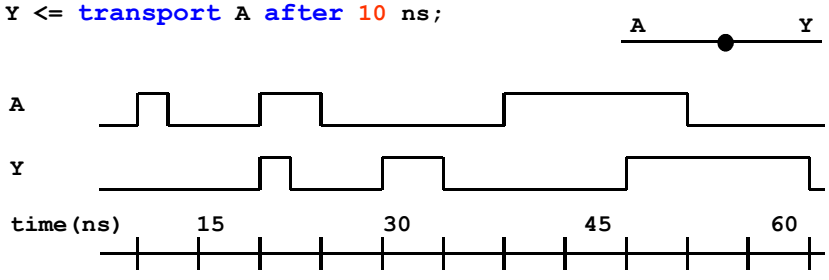
Modeling of connections is made using the transport delay.
All the changes of the signal value are propagated, regardless of their duration.

Syntax:

signal <= `transport` *expression* `after` *transport-delay*;

Example:

```
Y <= transport A after 10 ns;
```



Signals in VHDL
Inertial delay

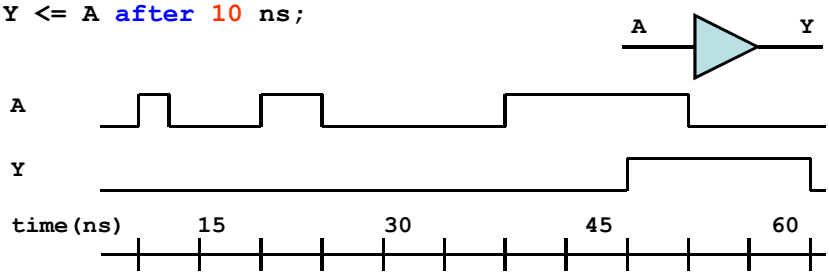
Modeling of elements is done using inertial delays (default).
Only these changes of the signal value, which last longer than the delay, are propagated.

Syntax:

signal <= [`inertial`] *expression* `after` *inertial-delay* ;

Example:

```
Y <= A after 10 ns;
```



Signals in VHDL
Inertial delay

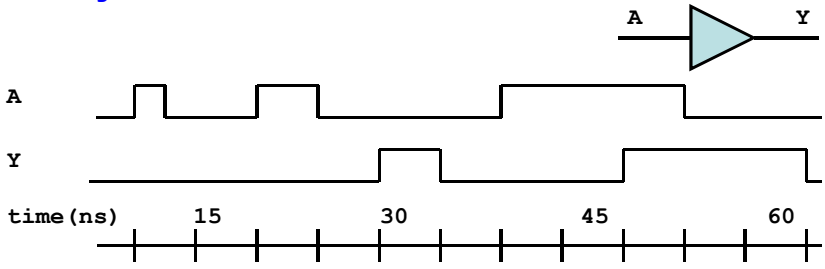
VHDL-93 allows the modeling of elements that respond to pulses shorter than the delay of these elements.

Syntax:

signal <= `reject` *reject-delay* `inertial` *expression* `after` *inertial-delay* ;

Example:

```
Y <= reject 4 ns inertial A after 10 ns;
```



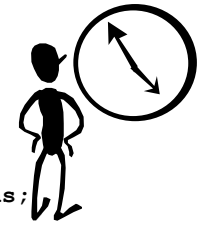
Signals in VHDL

Modeling the delays in process

Assigning values to the signals is **sequential** inside the processes, but **concurrent** outside of them.

Inside the processes, the signal assignments are stopped until the simulation cycle is run. This is triggered by the execution of the `wait` statement.

Example:



```
process
begin
   sys_clk <= not (sys_clk) after 50 ns;
   int_bus <= data_in after 10 ns;
   data_out <= my_function (int_bus) after 10 ns;
   wait .....
end process;
```


Signals in VHDL

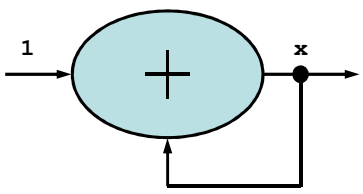
Modeling the delays with signals

```
signal X, Y: integer;
process
begin
  wait on Y;
  X <= Y + 1 after 10 ns;
```

In the above example, X takes the new value exactly after 10 ns, not after 9.999 or 10.001 ns.

Specification of delays is ignored by synthesis tools.

```
signal X: integer;
process .....
begin
.....
X <= X + 1 after 10 ns;
.....
```



Signals in VHDL

Modeling the delays with signals

Assignment statements may contain values of the signals for several various moments in time. This feature is useful for describing the clock signals and other repetitive waveforms.

Example:

```
S <= '1' after 4 ns, '0' after 7 ns;
T <= 1 after 1 ns, 3 after 2 ns, 6 after 8 ns;
```

Inside the process, each signal should have only one source at a time. Otherwise, only the last assignment is taken into account.

Example:

```
process
begin
  xyz <= 1 after 5 ns;   -- despite of the time...
  xyz <= 2 after 4 ns;   -- ...only this occures!...
  pqr <= 10 after 5 ns;  -- ...and of course this one.
wait .....
```


Simulation

Simulation and `wait` statement

Rather than **`wait`** statement, the user can specify the list of signals that activate a process (called a 'sensitivity list'). These signals are listed in parentheses after the **`process`** keyword.

This is equivalent to the **`wait`** statement, occurring **at the end** of the process. The process may contain **either** the list **or** such a wait statement.

Przykład:

```
process (CLK)
begin
.....
.....
<statements>
.....
.....
end process;
```

```
process
begin
..........
..........
<statements>
..........
wait on CLK;
end process;
```

Signals in VHDL
Zero delay

Example:

```
entity VAR is
port (A: in bit_vector (0 to 7);
      INDEX: in integer range 0 to 7;
      OUTPUT: out bit);
end VAR;

architecture VHDL_1 of VAR is
begin
  process
  begin
    OUTPUT <= A(INDEX); -- 0 ns delay
    wait .....;         -- wait initializes the assignment
.....
end VHDL_1;
```


Signals in VHDL
Variables versus signals

Main differences between the assignments of value to variables and signals

Signal assignments	Variable assignments
• according to time regimes	• no time regimes
• delays taken into account	• no delays
• after fulfilling the condition in **wait**	• immediate

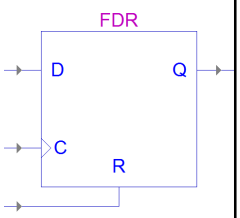
Signals in VHDL

Examples:

```
X <= 1;          -- 0 ns delay
wait .....;      -- assignment occures after execution of wait

X <= Y;          -- 0 ns delaty - signals swap
Y <= X;
wait .....;      -- both assignment occure after wait execution

V := 1;          -- variable assignment occures immediately
S <= V;
A := S;          -- A receives the previous value of S
wait .....;      -- S receives V value (=1) after wait executes

X <= 1;
X <= 2;
wait for 0 ns; -- after wait execution X recevies value =2
```


Signals in VHDL
Why do they behave that way?

```
-- D Flip Flop with asynchronous Reset
--      CLK: in STD_LOGIC;
--      RESET: in STD_LOGIC;
--      DIN: in STD_LOGIC;
--      DOUT: out STD_LOGIC;

process (CLK, RESET)
begin
  if RESET='1' then    --asynchronous RESET active High
     DOUT <= '0';
  elsif (CLK'event and CLK='1') then  --CLK rising edge
     DOUT <= DIN;
  end if;
end process;
```



Signals in VHDL

Variables versus signals

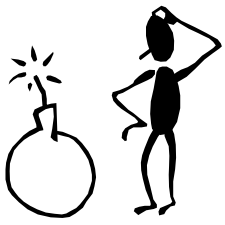
```
...
signal x,y,z : bit;
...
process (y)
begin
  x <= y;
  z <= not x;
-- wait on y;
end process ;
```

```
process (y)
variable x,z : bit;
begin
  x := y;
  z := not x;
end process;
```


Signals in VHDL

Hazards

```
entity MUX is
 port (Ain, Bin: in bit;
       SEL: in boolean;
       Y: out bit);
end MUX;

architecture WRONG of MUX is
 signal MUXVAL: integer range 0 to 1;
begin
 process
 begin
  MUXVAL <= 0;
  if (SEL) then
   MUXVAL <= MUXVAL + 1;
  end if;
  case MUXVAL is
   when 0 => Y <= Ain after 10 ns;
   when 1 => Y <= Bin after 10 ns;
  end case;
  wait on Ain, Bin, SEL;
 end process;
end WRONG;
```



Signals in VHDL

Hazards

```
entity MUX is
 port (Ain, Bin: in bit;
       SEL: in boolean;
       Y: out bit);
end MUX;

architecture BETTER of MUX is
begin
 process
  variable MUXVAL: integer range 0 to 1;   -- variable!
 begin
  MUXVAL := 0;                             -- variable!
  if (SEL) then
   MUXVAL := MUXVAL + 1;                   -- variable!
  end if;
  case MUXVAL is                           -- variable!
   when 0 => Y <= Ain after 10 ns;
   when 1 => Y <= Bin after 10 ns;
  end case;
  wait on Ain, Bin, SEL;
 end process;
end BETTER;
```


Simulation

Signals versus variables

```
architecture sig of counter is
signal SIG: ...
begin
process(CLK)
begin
  if CLK and CLK'event then
    SIG <= SIG + 1;
    if SIG = 9 then
      SIG <= 0;
    end if;
  end if;
end process;
```

```
architecture var of counter is
begin
process(CLK)
  variable VAR: ...
  begin
  if CLK and CLK'event then
    VAR := VAR + 1;
    if VAR = 9 then
      VAR := 0;
    end if;
  end if;
end process;
```

Question:
How will above counters count ?

Answer:
sig – from 0 to 9
var – from 0 to 8

Rule: Must the newly assigned value be used in the same run of simulation loop? If so, then use the variable. In other cases – use the signal (slower simulation of signals ☹).

Simulation cycle

- Former simulators: *One-List Algorithm* (evaluation and assignment).
- VHDL simulators: *Two-List Algorithm* (evaluation / assignment).

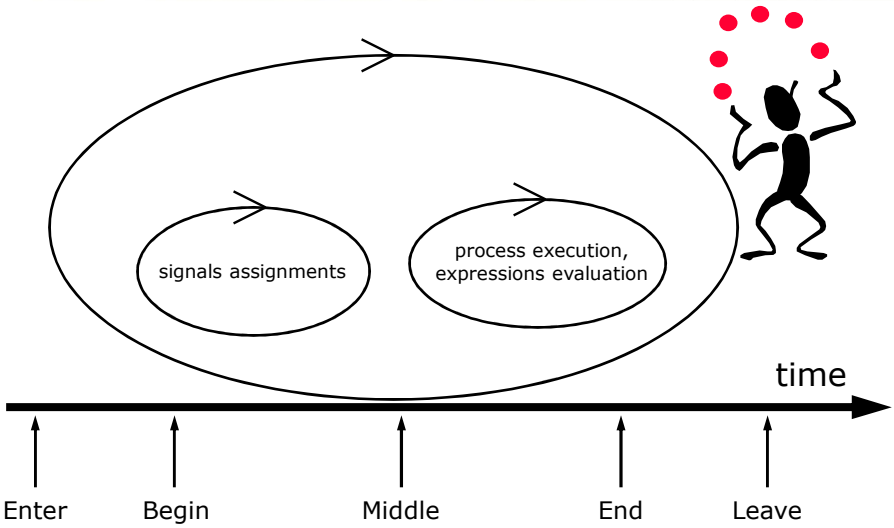
Example (in process):

```
A <= B;
B <= A;
```

Simulation of events with zero delay time is performed during the fictional time unit called *delta-time*. It is a complete cycle of a simulation, but without advancing the time counter:

- simulator models the events with zero delay time, using the *delta-time* cycle,
- events executed at the same time are simulated during the *delta-time* in a given order,
- logic connected with them is then resimulated to propagate changes for the next cycle,
- *delta-time* cycles are repeated until no changes are detected.

Simulation cycle

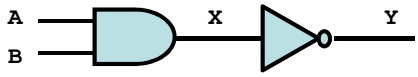


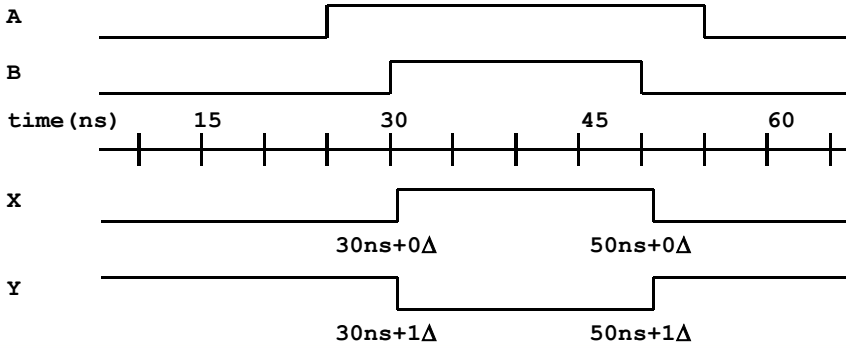
Simulation
Simulation cycle

Simulation of concurrent assignments with Δ-cycles.

Example:

```
X <= A and B;
Y <= not X;
```





Simulation
Simulation cycle

Question: What are the values of X and A after one cycle of the delta-time?

```
process
begin
  X <= 1;
  X <= 2;
  A <= X;
  X <= 3;
wait for 0 ns;
```

Answer: X has value of 3
A has a previous value of X

`wait` statement

`wait` statement variants:

- `wait on A, B;`
  Suspends the execution until the event on A or B occures.
- `wait until A > 10;`
  Suspends the execution until the event on A occures and the condition A > 10 is met.
- `wait for 10 ns;`
  Suspends the execution for 10ns time.
- `wait;`
  Suspends the execution forever. Used as 'kill'.

Sensitivity list

Question:

What is the difference in the behavior of the two following processes?

```
process (A, B)
begin
S <= A;
T <= B;
V <= S or T;
end process;
```

```
process (A, B, S, T)
begin
S <= A;
T <= B;
V <= S or T;
end process;
```

Answer:

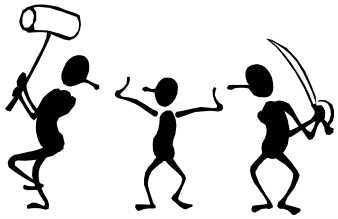
Left: 1 delta-cycle
Using previous values of S and T to calculate the value of V.

Right: 2 delta-cycles
Updating the value of S and T to calculate the value of V.

Simulation
Resolution function

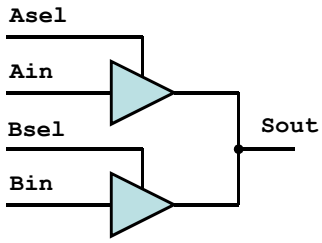
VHDL allows to drive signals from many sources, but:

- all drivers must be placed in a separate processes or in different concurrent assignments,
- for such signals there must be declared a resolution function,
- this function is pre-declared for the std_logic type. This is the preferred type of objects (it's easier to use one type in the whole project), but does not allow for detection (during compilation) of accidental connections of two or more drivers. This is only possible during the simulation.



Simulation
Resolution function

```
architecture SEQUENTIAL of TRISTATE is
 signal Ain,Bin,Asel,Bsel,Sout: STD_LOGIC;
begin

A:process (Ain,Asel)
  begin
   Sout <= 'Z';
   if (Asel='1') then
    Sout <= Ain;
   end if;
  end process;

B:process (Bin,Bsel)
  begin
   Sout <= 'Z';
   if (Bsel='1') then
    Sout <= Bin;
   end if;
  end process;
end SEQUENTIAL;
```




```
architecture CONCURRENT of TRISTATE is
signal Ain,Bin,Asel,Bsel,Sout: STD_LOGIC;
begin
 Sout <= Ain when Asel = '1' else 'Z';
 Sout <= Bin when Bsel = '1' else 'Z';
end CONCURRENT;
```

Simulation

Resolution function – STD_LOGIC_1164

```
TYPE std_ulogic IS ('U','X','0','1','Z','W','L','H','-');
TYPE std_ulogic_vector IS ARRAY (NATURAL RANGE <>) OF std_ulogic;
FUNCTION resolved (s : std_ulogic_vector) RETURN std_ulogic;
SUBTYPE std_logic IS resolved std_ulogic;
TYPE std_logic_vector IS ARRAY (NATURAL RANGE <>) OF std_logic;

TYPE stdlogic_table IS ARRAY (std_ulogic, std_ulogic) OF std_ulogic;
CONSTANT resolution_table : stdlogic_table := (
-------------------------------------------------
--| U   X   0   1   Z   W   L   H   -        |   |
-------------------------------------------------
  ('U','U','U','U','U','U','U','U','U'),  -- | U |
  ('U','X','X','X','X','X','X','X','X'),  -- | X |
  ('U','X','0','X','0','0','0','0','X'),  -- | 0 |
  ('U','X','X','1','1','1','1','1','X'),  -- | 1 |
  ('U','X','0','1','Z','W','L','H','X'),  -- | Z |
  ('U','X','0','1','W','W','W','W','X'),  -- | W |
  ('U','X','0','1','L','W','L','W','X'),  -- | L |
  ('U','X','0','1','H','W','W','H','X'),  -- | H |
  ('U','X','X','X','X','X','X','X','X')); -- | - |
```


Simulation

Resolution function – STD_LOGIC_1164

```
FUNCTION resolved (s: std_ulogic_vector ) RETURN std_ulogic IS
VARIABLE result : std_ulogic := 'Z';  -- weakest state default
BEGIN
-- The test for a single driver is essential otherwise the
-- loop would return 'X' for a single driver of '-' and that
-- would conflict with the value of a single driver unresolved
-- signal.
  IF (s'LENGTH = 1) THEN RETURN s(s'LOW);
  ELSE
    FOR i IN s'RANGE LOOP
      result := resolution_table(result, s(i));
    END LOOP;
  END IF;
  RETURN result;
END resolved;
```

Simulation

Resolution function

Function:

- returns one value,
- has all the arguments in input mode,
- passes the arguments by their values.

Resolution function:

- is required when the signal (node) is controlled by more than one driver,
- performs the arbitration of signals,
- is invoked in case of change in any of the signal drivers,
- receives an array of signals for arbitration,
- is a user-defined function,
- is associated with a subtype.

Signals in VHDL

Timing attributes of signals

- *signal_name*`event - returns TRUE if an event has occurred on signal in the current simulation cycle
- *signal_name*`last_event - returns the amount of time since last event occurred on signal
- *signal_name*`last_value - returns the previous value of signal before last event occurred on it

Examples:

```
if CLK'event and CLK='1' then ...

if SD_DAT'event and (SD_DAT='H' or SD_DAT='Z') and
   SD_DAT'last_value='1' then ...
```

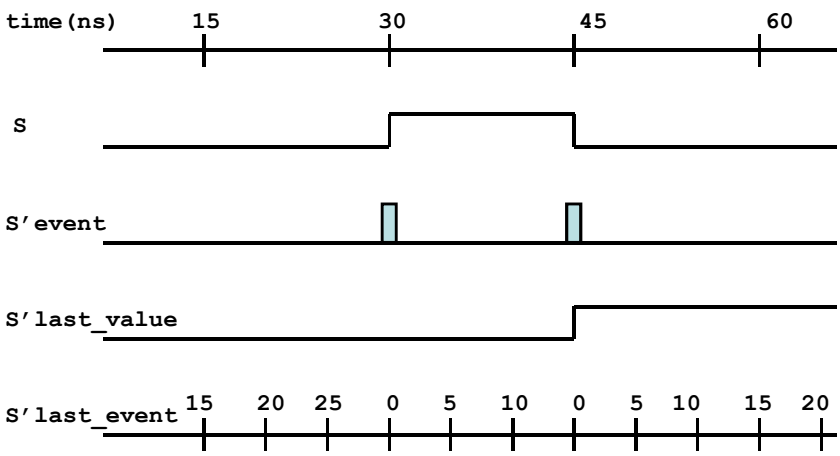
Defined in libraries functions `rising_edge` and `falling_edge`:

```
if rising_edge(CLK) then ...
```

are equivalent to statements like:

```
if CLK'event and CLK='1' and CLK'last_value='0' then ...
```

Signals in VHDL
Timing attributes of signals



Simulation
Predefined attributes

Predefined attributes allow to get information about objects, types, subprograms, etc.

Signal attributes:

Attribute	Result
S'Delayed(t)	implicit signal, equivalent to signal S, but delayed t units of time
S'Stable(t)	implicit signal that has the value True when no event has occurred on S for t time units, False otherwise
S'Quiet(t)	implicit signal that has the value True when no transaction has occurred on S for t time units, False otherwise
S'Transaction	implicit signal of type Bit whose value is changed in each simulation cycle in which a transaction occurs on S (signal S becomes active)
S'Event	True if an event has occurred on S in the current simulation cycle, False otherwise
S'Active	True if a transaction has occurred on S in the current simulation cycle, False otherwise
S'Last_event	the amount of time since last event occurred on S, if no event has yet occurred it returns Time'High
S'Last_active	the amount of time since last transaction occurred on S, if no event has yet occurred it returns Time'High
S'Last_value	the previous value of S before last event occurred on it
S'Driving	True if the process is driving S or every element of a composite S, or False if the current value of the driver for S or any element of S in the process is determined by the null transaction
S'Driving_value	the current value of the driver for S in the process containing the assignment statement to S

Simulation
Predefined attributes

Attributes of scalar types:

Attribute	Result type	Result
T'Left	same as T	leftmost value of T
T'Right	same as T	rightmost value of T
T'Low	same as T	least value in T
T'High	same as T	greatest value in T
T'Ascending	boolean	true if T is an ascending range, false otherwise
T'Image(x)	string	a textual representation of the value x of type T
T'Value(s)	base type of T	value in T represented by the string s

Attributes of discrete and physical types and subtypes:

Attribute	Result type	Result
T'Pos(s)	universal integer	position number of s in T
T'Val(x)	base type of T	value at position x in T (x is integer)
T'Succ(s)	base type of T	value at position one greater than s in T
T'Pred(s)	base type of T	value at position one less than s in T
T'Leftof(s)	base type of T	value at position one to the left of s in T
T'Rightof(s)	base type of T	value at position one to the right of s in T

Attributes of array types and array-type objects:

Attribute	Result
A'Left(n)	leftmost value in index range of dimension n
A'Right(n)	rightmost value in index range of dimension n
A'Low(n)	lower bound of index range of dimension n
A'High(n)	upper bound of index range of dimension n
A'Range(n)	index range of dimension n
A'Reverse_range(n)	reversed index range of dimension n
A'Length (n)	number of values in the n-th index range
A'Ascending(n)	True if index range of dimension n is ascending, False otherwise