Digital Circuits Lab

Search forums

Announcements

Summary notes for today's VHDL lecture .... disclaimer : the vhdl simulation algorithm is only coarsely described

◄ Lab lecture on Wednesday (March 6th)
The tasks for this week's lab sessions : Disclaimer : could have minor issues due to preparation on a short notice .... and minor corrections / improvements will be done later ►

Display replies flat, with oldest first | Move this discussion to ... | Move | Pin

Summary notes for today's VHDL lecture .... disclaimer : the vhdl simulation algorithm is only coarsely described
by Sachin B. Patkar - Wednesday, 6 March 2019, 7:33 PM

Some of the points brought up in today's lecture.
The intended "takeaway" of this lecture
has been just gaining familiarity
with "under-the-hood" activities during simulation, namely,
"driver", "transaction", "event", "suspend/resume process",
"advancing simulation time", "update signals" etc.

You will gain more insights as you go along.

0.
   An architecture-body consists of an (unordered) collection of
"concurrent statements"

   Each such "concurrent statement" models a digital logic block.
   These "logic blocks" are interconnected by "signals" within the
     "architecture"

   The body of "process ... end process;" consists of a sequenced
collection of (procedural) statements.

   A "process ... end process; " by itself is a "concurrent statement",
just like a "concurrent signal assignment" or a "component instantiation"

1.
   A concurrent signal assignment is really a short-hand disguise for a
"process statment"

```
   architecture ex1 of xyz is
     signal ....
     ....
   begin
     a <= not a after 5 ns ;  -- note "5 ns" and not "5ns"
     b <= not a after 1 ns ;
     c <= a or b after 1 ns ;
   end ex1 ;
```

IS EQUIVALENT TO

```
   architecture ex2 of xyz is
     ....
   begin
     aproc : process ( a ) begin a <= not a after 5 ns ; end process ;
     bproc : process begin b <= not a after 1 ns ; wait on a ; end process;
     cproc : process ( a , b ) begin
               c <= a or b after 1 ns ;
             end process ;
   end ex2 ;
```

2.
   A "process .... end process;" is itself a "concurrent statement" in the

sense that it models a piece of digital logic that "runs concurrently" with other logic blocks that are modeled by other "concurrent statements"

As seen in following contrived example, the USP of a "process" is the expressive power it facilitates to describe complex logic. Indeed, use of variables, "if--else" , "case ... end case;" , "for loop" etc.. helps in describing complicated logic behaviour.

```
architecture ex1 of pqr is
  ...
begin -- body of an architecture
  a <= not a after 5 ns ; -- a concurrent signal assignment

  -- the following "process ... end process;"
  -- -- is also a concurrent statement
  process ( a )
    variable v1, v2 : std_logic ;
  begin
    v1 := not a ;
    if ( true ) then v2 := not v1 ; else v2 := 'X' ; end if ;
    -- v2 is not ( not a )
    for I = 1 to 4 loop
      v2 := v2 xor 1 ;
    end loop ;
    -- v2 got toggled 4 times ... so results in just a
    b <= not v2 after 1 ns ; -- effectively b is just "not a"
  end process ;

  -- another concurrent statement
  -- -- in disguise really a simple concurrent signal assignment
  process begin
    c <= a or b after 1 ns ; wait on a, b ;
  end process ;
end ex1 ;
```

---

3. Some brief remark about simulation algorithm

A "process ... end process; " models a foreever-running logic block

"Abstract view" is that a "process block" runs until it suspends then runs again ... then suspends .... and so on

A process runs until it encounters explicit "wait" statement or an "implicit wait" at the end of a process with sensitivity list.

When a process "runs" (i.e. resumes after its suspension), the simulator the "drivers" in the statements within the process ... until the process gets "suspended" ... All these evaluation of drivers happens at the "same simulation time".

While a process "runs", transactions get created

```
   CREATING Transactions involve the following
   ... a transaction on a signal is a pair <future-time, future-value>
   --- "future-time" is "NOW + delay"
       .... "delay" is delta if not specified
   --- "future-value" is the value of the "driver" evaluated at time "NOW"

   All newly created transactions are "in future" ( either "delta" time
later or after a specified tangible delay, e.g. "5 ns" )

   When all processes are in suspended state,
   the simulator does "apply transactions"-phase


   "Apply transactions" involves the following
   --  advancing time , by delta  or by some
         tangible time  to time of next transaction
   --  updating target signals, if necessary
       .... when application of transaction "causes" an event ( i.e. change
in signal value )
   --  waking up processes that are suspended in "wait on"
       some event that happens on such updated signals

   "Awakened processes" are ready "run / executed" to create new
transactions ( as already outlined )


   .... and this "create transactions ... apply transactions" loop repeats


------------

4.
   Signals are always updated at a "future time" ( by default at NOW+delta)
   Variables are updated "instantaneously" ... so the effect of update on
     a variable is visible at the current time itself ... and hence
     while evaluating "drivers" of the later statements
     within a "process ... end process;"
```

Permalink | Edit | Delete | Reply

◄ Lab lecture on Wednesday (March 6th)
The tasks for this week's lab sessions : Disclaimer : could have minor issues due to preparation on a short notice .... and minor corrections / improvements will be done later ►