

L6.1 VHDL Statements Tues. (Mon.) February 19, 2002

Concurrent and sequential

Signal assignment

Concurrent

Instantiation

when-else

with-select-when

process (as a wrapper for sequential statements)

Sequential – ONLY within a process

if-then-elsif-else

case-when

L6.2 Signal Assignment Tues. (Mon.) February 19, 2002

outc <= ina AND (inb OR inc); - - One needs parentheses to define order.

Basic Operators:

Unary Arithmetic

-

Arithmetic

+, - * also, but it is not synthesizable!

Concatenation – defined for strings and signal values

&

Need a use clause for the next two – use ieee.std_logic_1164.all;

Logical

AND, NAND, OR, NOR, XOR, XNOR, NOT

Relational

=, /=, <, <=, >, >= Note that , <= and => have other meanings also.

L6.3 Concurrent Statements Tues. (Mon.) February 19, 2002

- - Instantiation by named association

h1: halfadd PORT MAP(a => ina, b => inb, sum => s1, c => s3);

- - Instantiation by positional association (not recommended)

h1: halfadd PORT MAP(ina, inb, s1, s3);

- - when-else

outc <= ina WHEN inc = '0' ELSE inb;

- - with-select-when

WITH inc SELECT

outc <= ina WHEN '0',
inb WHEN '1',
inb WHEN OTHERS;

L6.4 With – Select – When Tues. (Mon.) February 19, 2002

Assignment is based on a selection signal.

Use "signal_name <=" only once.

WHEN clauses must be mutually exclusive.

Always use a WHEN OTHERS at the end.

It is easy to not specify all signal values.

WITH sel_signal SELECT

signal_name <= v_1 WHEN v_1 of sel_signal,
v_2 WHEN v_2 of sel_signal,
v_3 WHEN v_3 of sel_signal,

....

v_n WHEN v_n of sel_signal,
v_x WHEN OTHERS;

L6.5 Process Tues. (Mon.) February 19, 2002

Multiple processes are possible within an architecture.
A process is concurrent with other concurrent statements.

A process is a wrapper for sequential statements.
Sequential statements model combinational or synchronous logic (or both).
Signal assignments are BOTH concurrent and sequential.

Only variables (more later) may be declared within a process.
Signals must be declared outside the process.

Statements within a process are executed sequentially.
Variables are updated when a statement is executed.
Signals are SCHEDULED to be updated when a statement is executed.
Signals are updated at the END of a process.

Synthesis of variables is problematic (or not possible). What is really meant?

L6.6 Process Syntax Tues. (Mon.) February 19, 2002

Keywords are in UPPER CASE.

label: PROCESS (sensitivity list)
- - VARIABLE declarations
BEGIN
- - sequential statements
END PROCESS label;

The process label and variable declarations are optional.

PROCESS (sensitivity list)
BEGIN
- - sequential statements
END PROCESS ;

The process executes when a signal in the sensitivity list has an event.
The sensitivity list is NOT used for synthesis.
Variables are updated when a sequential statement is executed.
Signals are SCHEDULED to be updated when a statement is executed.
Signals are updated at the END of the process.

L6.7 Sequential Statement Tues. (Mon.) February 19, 2002

If - then - elsif - else is used to select a set of statements to execute.

Selection based on a Boolean evaluation of condition(s).

Absence of ELSE may result in implicit memory
(if all possible conditions are not specified).

Conditions do not have to be mutually exclusive.

```
IF condition(s)
THEN do_something;
ELSIF other_conditdion(s)
- - any number of ELSIFs (including zero)
ELSE do_default;
- a final ELSE is optional but recommended
END IF;
```

```
- - for example
IF inc = '0' THEN outc <= ina;
ELSIF ina = '1' THEN outc <= inb;
ELSE outc <= inc;
END IF;
```

L6.8 Sequential Statement Tues. (Mon.) February 19, 2002

CASE - WHEN is similar to with - select - when BUT the
Selection signal determines which statement(s) to execute.

WHEN clauses must be mutually exclusive.

Always use a WHEN OTHERS at the end.
It is easy to not specify all conditions.

```
CASE sel_signal IS
WHEN v_1 of sel_signal => statement(s)1
WHEN v_2 of sel_signal => statement(s)2
WHEN v_3 of sel_signal => statement(s)3
....
WHEN v_n of sel_signal => statement(s)n
WHEN OTHERS => default_statement(s)
END CASE;
```

- - for example

```
CASE inc IS
WHEN '0' => outc <= ina;
WHEN '1' => outc <= inb;
WHEN OTHERS => outc <= ind;
end CASE;
```

L6.9 Use if you wish – we won't cover them. Tues. (Mon.) February 19, 2002

- - Unconditional loop
 loop_label: LOOP
 statement(s);
 END LOOP loop_label;
- - FOR loop
 loop_label: FOR loop_variable in RANGE LOOP
 statement(s);
 END LOOP loop_label;
- - WHILE loop
 loop_label: WHILE condition LOOP
 statement(s);
 END LOOP loop_label;
- - NEXT : Causes the loop to begin when condition is true
 NEXT loop_label WHEN condition;
- - EXIT : Terminates the loop, unconditionally or conditionally
 EXIT loop_label; - - or
 EXIT loop_label WHEN condition;

L6.10 Sensitivity List Tues. (Mon.) February 19, 2002

Signal changes (in the sensitivity list) cause the process to "run" or be evaluated.

All sequential statements in the process are "executed".

Variables are updated as statements are sequentially "executed".

Signal updates are scheduled as statements are sequentially "executed".

Signal updates occur when the process is suspended (finished).

A common error is to think that signal updates take place when a sequential statement is executed.

L6.11 Simulation and Synthesis Tues. (Mon.) February 19, 2002

The aim of synthesis is to produce hardware which will do what the concurrent statements specify.

This includes processes as well as other concurrent statements.

The aim of simulation is to produce outputs (signals, integers, etc.) from specified input signals.

Concurrent statements are evaluated whenever any input changes.

If the evaluation of any concurrent statement results in an input signal change for any concurrent statement then that concurrent statement is evaluated.

Processes are a little different in that you must list the conditions that initiate evaluation of the process.

This can be done by WAIT statements or by a SENSITIVITY LIST.

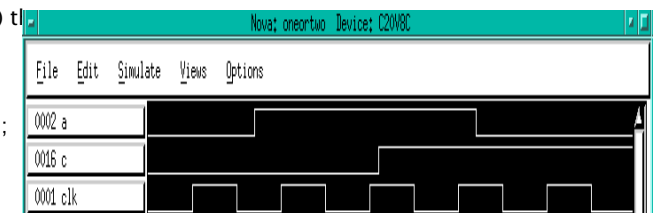
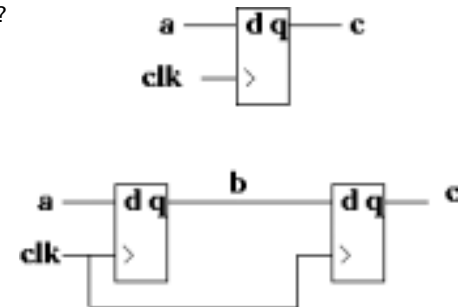
We will ONLY use a SENSITIVITY LIST.

Synthesis usually ignores the sensitivity list.

L6.12 One or Two Flip-Flops Tues. (Mon.) February 19, 2002

- - Does this have one or two flip-flops?

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
port (a, clk : in std_logic;
      c : out std_logic);
end reg;
architecture top of reg is
signal b : std_logic;
begin
reg2: PROCESS (clk)
BEGIN
if rising_edge(clk) then
b <= a;
c <= b;
end if;
END PROCESS reg2;
end top;
```



L6.13 Implicit Latch Tues. (Mon.) February 19, 2002

Memory is either implicit or explicit.

- - this is an example of an implicit latch
 library ieee;
 use ieee.std_logic_1164.all;
 entity reg is
 port (d, g : in std_logic;
 q : out std_logic);
 end reg;
 architecture top of reg is
 begin
 process (d, g)
 begin
 if g = '1' then q <= d;
 - - notice that there is no ELSE
 end if;
 end process;
 end top;
 - - this produces $q = /g * q + d * g$

L6.14 Explicit Latch Tues. (Mon.) February 19, 2002

- - this is an example of an explicit latch
 library ieee;
 use ieee.std_logic_1164.all;
 entity reg is
 port (d, g : in std_logic;
 q : out std_logic);
 end reg;
 architecture top of reg is
 signal s1 : std_logic;
 - - to get around not using q as an input
 begin
 q <= s1; -- Is this ok to have this first?
 s1 <= d when g = '1' else s1;
 end top;
 - - this produces $q = /g * q + d * g$
 - - the when-else statement could have been replaced with
 - - process (s1, d, g)
 - - begin
 - - if g = '1' then s1 <= d;
 - - else s1 <= s1;
 - - end if;
 - - end process;

L6.15 Implicit Clocked T Flip-Flop Tues. (Mon.) February 19, 2002

- - an example of a clocked T type register
 - - rising_edge() signifies a clocked register
 library ieee;
 use ieee.std_logic_1164.all;
 entity clked_reg is
 port (t, clk : in std_logic;
 q : out std_logic);
 end clked_reg;
 architecture top of clked_reg is
 signal s1 : std_logic;
 begin
 q <= s1;
 process (clk, s1)
 begin
 if rising_edge(clk) then
 if t = '1'
 then s1 <= not s1;
 end if;
 end if;
 end process;
 end top;
 - - this produces $q.D = t * /q.Q + /t * q.Q$

L6.16 Explicit Clocked T Flip-Flop Tues. (Mon.) February 19, 2002

Library ieee;
 use ieee.std_logic_1164.all;
 entity clked_reg is
 port (t, clk : in std_logic;
 q : out std_logic);
 end clked_reg;
 architecture top of clked_reg is
 signal s1 : std_logic;
 begin
 q <= s1;
 process (clk, s1)
 begin
 if rising_edge(clk) then
 if t = '1'
 then s1 <= not s1;
 else s1 <= s1;
 end if;
 end if;
 end process;
 end top;
 - - this produces $q.D = t * /q.Q + /t * q.Q$
 - - and $q.C = clk$

L6.17 Like an LS164 Tues. (Mon.) February 19, 2002

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ctr is
- - generic is used so that one need only change a single number
- - to change the width of the counter.
- - we will see later how to specify a
- - value for this generic
- - when the counter is instantiated.
generic (width : integer := 4);
port(
clk: in std_logic;
n_clr, n_ld, enp, ent: in std_logic;
data: in std_logic_vector
(width - 1 downto 0);
cnt: out std_logic_vector
(width -1 downto 0);
rco: out std_logic);
end ctr;
```

L6.18 Like an LS164 (cont'd) Tues. (Mon.) February 19, 2002

```
architecture behavioral of ctr is
signal intcnt, allones : std_logic_vector(width - 1 downto 0);
begin
clocked: process (clk)
begin
if rising_edge(clk) then
if n_clr = '0' then
intcnt <= (others => '0');
elsif n_ld = '0' then
intcnt <= data;
elsif (enp = '1') and (ent = '1') then
intcnt <= intcnt + 1;
end if;
end if;
end process clocked;
allones <= (others => '1');
rco <= '1' when ((ent = '1') and
(intcnt = allones))
else '0';
cnt <= intcnt;
end behavioral;
```

L6.19 Simulation Results Tues. (Mon.) February 19, 2002

Look at the left to see the precedence of n_clr over n_ld over enp and ent.

