VHDL additional note 1 (khw)

The purpose of this document is to discuss collectively various cases when signals are used in different parts of a VHDL program. Since I cannot find a single short article in reference books on this matter so I compile this document to allow students to have a quick look into this issue, if you have any comments and points you want me to clarify, please send mail to me at khwong@cse.cuhk.edu.hk)

**Discussion of the use of signals**

**Introduction:** This document discusses the use of signals in concurrent statements, combinational processes and clocked processes. It describes how you should use the signals, and the effect of whether you include them in the sensitivity list of a process or not.

**An example:**

library IEEE;

use IEEE.std\_logic\_1164.all;

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

entity test11 is -- a typical vhdl program

port ( clk,in1,in2,reset: in STD\_LOGIC;

 out1,out2,out3,out\_con1,out\_con2: out STD\_LOGIC);

 end test11;

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

architecture test11\_arch of test11 is

begin --Concurrent statements here

 out\_con1<=in1 and in2; -- concurrent statement

 out\_con2<=in1 or in2; -- concurrent statement

---- out\_con2<= in2; -- multiple assignment to one signal (out\_con2) is not allowed in tools

-- without resolved logic support.

-------------------(1) clocked process with asynchronous inputs ---------------------------------------------------------------

clocked\_process1:process(clk, reset) – clocked process with asynchronous inputs

-- must have a clock triggering condition inside

---clock and asynchronous inputs must be in sensitivity list

-- it allows asynchronous input signals, such as reset or load etc.

begin if reset='1' then out1<='0'; -- asynchronous Input reset

 elsif clk='1' and clk'event -- the triggering point

then

 out1 <=in1 xor in2; -- sync. Signals and expressions

 end if;

end process;

-------------------(2) clocked pure synchronous process without sensitivity list -------------------------------------------------

clocked\_process2:process -- clocked pure synchronous process without sensitivity list -- inferring synchronous logic

--clocked process using wait until clock=’1’ when you use wait-until for a clock signal, the

--triggering expression wait-until must be the first expression, hence, all other signals

--of this process must be synchronous input signals. There is no room for asynchronous input signals

begin wait until clk = ‘1’; -- wait for positive rising edge

 out2<=’0’; -- multiple assignment is ok, but this statement has no effect since out2 has another assignment later

 out2<=in1 nand in2; --only the last assignment state for a signal (i.e. out2 here) counts

end process;

--------------------(3) combination process -------------------------------------------------------------------------------------------

combinational\_process3: process(in1, in2) -- combination process: must NOT have clock trigger condition inside

-- no (if clk='1' and clk'event) or ( wait until clock=’1’) etc.

-- input signals must be in sensitivity list, otherwise result is not predictable.

begin --

 out3<=in1 xor in2;

 out3<=’1’;--out3 will always be ‘1’ but not “in1 xor in2”, because it is the last signal assignment expression in a process

end process;

end test11\_arch;

**Signals in processes:**

###### Different cases of using signals

In concurrent statements outside processes

Signals Combinational processes – (no clock triggering inputs,

all inputs should be in sensitivity list)

each “<=” is a combinational logic

 Inside processes

 Clocked processes,

 each“<=” is a flip-flop clocked processes with synchronous inputs

( syn. inputs should not be in sensitivity list)

Clocked processes with asynchronous inputs (asyn. inputs must be in sensitivity list)

1. **Signals and variables in concurrent statement outside processes**. In the above VHDL example, the first two statements

out\_con1<=in1 and in2; -- concurrent statement,

out\_con2<=in1 or in2; -- concurrent statement,

just after “architecture begin” in the above VHDL example, we have the following discussions.

* 1. When signals are used in concurrent statements outside processes, statements are concurrent. Signals can be assigned multiple values if the tool you are using supports “resolved logic, e.g. std\_logic etc.” It is the job of the resolved logic function to resolve the conflict as in the following examples. However, this topic is out of the scope of this course.

E.g. 1. in the following example, the result is 1.

Sig1<=’L’; -- weak low, concurrent statement

Sig1<=’1’ – forcing 1, concurrent statement

E.g.2 in the following example, the result is ‘X’ unknown.

Sig1<=’1’; --forcing high, concurrent statement

Sig1<=’0’; -- forcing 0, concurrent statement

* 1. Variables can only live in processes. Variables are sequential, their values change intermediately.
1. **Signals and variables in processes.** (see p80 of [1])
	1. Variables are treated as the same as that in a sequential program. No further discussion is necessary here.
	2. Signals in Combinational processes: (see combinational\_process3)
		1. All input signals of a combinational process must be in the sensitivity list: In a combinational process, no edge triggered expressions are found, see combinational\_process3: of the above example. In there, all inputs (all to the right of <= or in if/case expression) of the process must be in the sensitivity list. If they are not, then your program is for the purpose of simulation or theoretical investigation only, and the VHDL simulation and the synthesized hardware will behave differently. So don’t forget to put signals used in combinational process in the sensitivity list.
		2. Multiple signal assignments in combinational process: When a signal is used in a combinational process, you can assign multiple values to one signal, however only the last assignment inside the process is effective. Quoted from chapter 6 {Sequential statements} of [2]:

*When a signal is assigned a value, the assignment does not necessarily take effect because the value of a signal is determined by the processes (or other concurrent statements) that drive it.*

* If several values are assigned to a given signal in one process, only the last assignment is effective. Even if a signal in a process is assigned, read, and reassigned, the value read (either inside or outside the process) is the last assignment value.
* *If several processes (or other concurrent statements) assign values to one signal, the drivers are wired together. The resulting circuit depends on the expressions and the target technology. The circuit might be invalid, wired* AND, wired OR, or a three-state bus. See “Driving Signals” in Chapter 7, “Concurrent Statements,” for more information.
	1. Signals in clocked process: (see clocked\_process1 and clocked\_process2 of the above example)

Note from Xilinx VHDL reference manual vhdlref.pdf [2] section 8-3, pp.169: *An edge expression must be the only condition of an if or an elsif statement. You can have only one edge expression in an if statement, and the if statement must not have an else clause. An edge expression cannot be part of another logical expression nor used as an argument.*

In a clocked process, the statement “if clk= ‘1’ and clk’event; or wait until clk= ‘1’¨ is for sensing the triggering clock “clk” input signal. The other signals in a clocked process can be classified into 2 types: (i) synchronous input signals, (ii) asynchronous input signals,.

* + 1. Synchronous input signals in a clocked process (for D inputs of flip-flops) : The signal inputs that are inside (located right after the if clk=’1’ and clk’event ) the clock triggered expression; or under wait “until clk=’1’¨ are called synchronous input signals. (In fact wait-until for clock edges must be the first expression in a process, so in a wait-until clocked process all inputs are synchronous. See page 6-34 of [2].) These inputs are the inputs of the flip-flops triggered by “clk”., should these signals be included in the sensitivity list?
			1. Each “<=” under a clock condition is treated as a flip-flop -- the right side is assigned to the left side upon the arrival of the clock edge.
			2. To avoid confusion, try not to put synchronous input signals in the sensitivity list, otherwise they will not be called synchronous inputs anyway. Because when these signals change values outside the clocked process, you don’t want to update the result immediately, you should wait until the next rising edge of the clock “clk” to update the outputs of the clocked process. That is why they are called synchronous input signals. (In fact there is no way you can put anything in the sensitivity list of a “wait until clock=’1’ ” process because there is no sensitivity list in such a process, the clock in “wait until clock=’1’ ” is the only triggering point by default, see pp.6-34 of [2].) That is why such a process is called inferred synchronous process.
			3. In fact there is no effect even if you put synchronous inputs in the sensitivity list (in the case of using “if clk=’1’ and clk’event” clock triggering.). Since these inputs are under the edge triggering condition, only at the clk rise edge should the inputs affect the outputs of a clocked process. Logically, if synchronous input signals found in the sensitivity list change values outside the clocked process, it will trigger the process to execute once. But it affects nothing since these inputs are ‘protected or embraced’ by edge triggered clock input control statement. It is simply -- no clock edges, no effect for synchronous clocked processes.
		2. Asynchronous input signals in a clocked process (they are not D inputs of flip-flops but rather RESET, LOAD inputs). They refer to those signal inputs that are not inside (not right after the “if clk=’1’ and clk’event ¨ the clock triggered expression, or under the wait until clk=’1’statement¨) clocked expressions. See reset of clocked\_process1.

Note that, such asynchronous inputs never exist in wait-until clocked processes. Because wait-until must be the first expression in a inferred synchronous process discussed earlier, so in such a process all inputs are synchronous, so there is no asynchronous input signals in a wait-until process. However if you use “if clk=’1’ and clk’event “expression for a clocked process, these asynchronous inputs, e.g. reset, load etc., can exist. So should they be included in the sensitivity list?

* + - 1. Surely the triggering clock signal, e.g. clk in our example, must be in the sensitivity list.
			2. Yes. Asynchronous signals (such as RESET, LOAD etc.) of clocked processes must be found in the sensitivity list if you are using the “if clk=’1’ and clk’event” clock triggering expression. Since they are not controlled by the clock clk, it affects the output of the clocked process asynchronously—anytime they change their values. So you should put them in the sensitivity list. Multiple assignments of these signals should be treated as those signals in a combinational process – for a particular signal, only the last assignment operator “<=” counts (see the argument in the above section on “signals in combinational processes”).
			3. WARNING: It is bad not to put asynchronous signals (e.g. reset etc.) of a clocked process in the sensitivity list. You should not do this. Because, VHDL is used for representing hardware, if these signals are neither synchronous inputs of flip-flops, (see asynchronous input signals defined earlier), nor asynchronous signals (because you miss them in the sensitivity list) there is no such hardware exists. So avoid doing it.
1. Conclusions:
	1. Combinational processes (no edge triggering element “if clk=’1’ and clk’event” or “wait until clk=’1’”): all input signals must be in the sensitivity list. For multiple signal assignments, only the last assignment operator “<=” for a particular signal counts.
	2. Clocked processes (have edge triggering element “if clk=’1’ and clk’event” or “wait until clk=’1’ ”):
		1. Do not put synchronous input signals (D inputs to flip-flops) in the sensitivity list.
		2. All asynchronous inputs should be in the sensitivity list. Otherwise the result may not agree with the hardware implementation.

**Reference:**

[1] VHDL for designers by Stefan Sjoholm, Prentice hall

[2] VHDL reference manual of Xilinx-foundation

Exercise:

1. What are these processes p1,p2 (clocked or combinational ) and why?
2. Which are synchronous inputs and which are asynchronous?
3. How to add asynchronous reset to p1?

------------example 1: lig1\_sr syn. reset based on lightA.vhd -------------------------------------------------------------

library IEEE; -- ok for foundation1.5

use IEEE.std\_logic\_1164.all;

entity traffic is

 port (out\_light :out std\_logic\_vector( 2 downto 0);

-- out\_light uses type out because no feedback requirement

 inB: in std\_logic ;----------\*\*\*\*\*\*\*\*\*

 clock: in std\_logic);

end traffic;------------------------------------------------

Architecture lightA of traffic is

type traffic\_state\_type is (s0, s1,s2,s3);

signal L\_stateA: traffic\_state\_type;

begin

----------------------- p1 -----------------------------------------------------------------------------------------

p1:process --exec.once when clock rises;the sensitivity list is empty,it implies only the clock will trigger the process

begin wait until clock='1'; --edged-clock triggering point

 if inB='1' -- reset

 then L\_stateA <=s0;

 else

 case L\_stateA is

 when s0 => L\_stateA<=s1;

 when s1 => L\_stateA<=s2;

 when s2 => L\_stateA<=s3;

 when s3 => L\_stateA<=s0;

 end case; end if; end process; --to be continued , see next page

----------------------- p2 --------- convert L\_statesA to out\_light -------------------------------------------

p2:process(L\_stateA) -- combin. process

 begin

 case (L\_stateA) is

 when s0 => out\_light <="100";

 when s1 => out\_light <="110";

 when s2 => out\_light <="001";

 when s3 => out\_light <="010";

 end case; end process; end lightA;