NTNU
Department of Electronic Systems

# TFE4141 Design of Digital Systems 1

# Assignment 4: Common VHDL pitfalls and misconceptions

**Deadline: September 22 at 23:59**

## Task 1: Inferred latches

When inexperienced designers model hardware using combinatorial processes in VHDL, unintended latches are often created. The symbol for a latch will typically look like the ones in Figure 1.

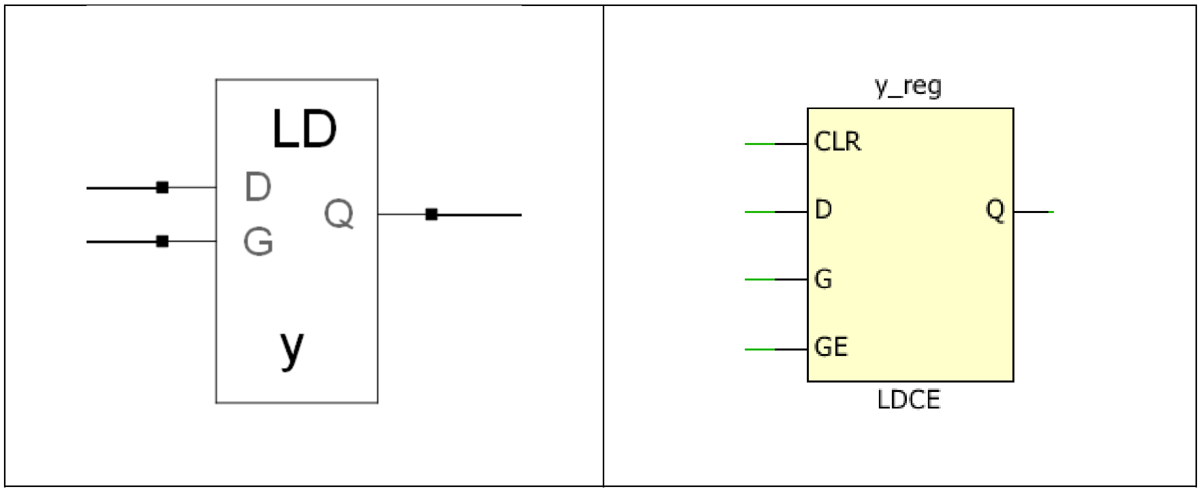
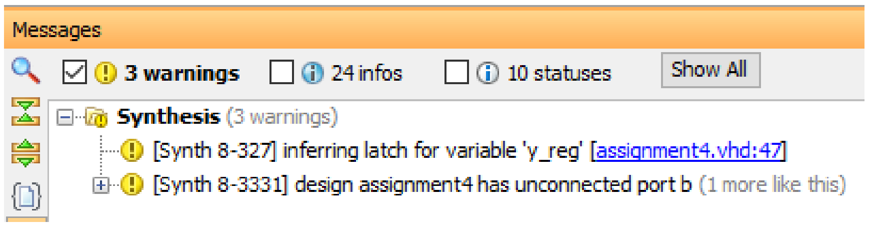


*Figure 1. Typical latch symbols in synthesis tools.*

1. Write a VHDL entity/architecture that implements at least one latch. Synthesize the design and look at the generated circuit (technology view.) Any latches in the generated circuit should be represented by a symbol similar to those shown in Figure 1.

Most, if not all synthesis tools will produce a warning such as "*Inferred latch for 'y' at design.vhd(10)*" or "*Latch generated from process for signal 'y'*" when you do this. This is because latches in designs are generally regarded as bad practice and should be avoided, for various reasons. The warnings produced by Xilinx Vivado is shown in Figure 2.



```
Synthesis (3 warnings)
    [Synth 8-327] inferring latch for variable 'y_reg' [assignment4.vhd:47]
    [Synth 8-3331] design assignment4 has unconnected port b (1 more like this)
```

*Figure 2. Warning produced by a synthesis tool.*

2. Inferred latches are often unintentional and generated from VHDL processes that were supposed to be purely combinatorial. Can you give an example?

3. How do you make sure that latches will not be generated from a combinatorial VHDL process?

## Task 2: Signals/variables and registers (flip-flops)

Signals or variables written to in synchronous (clocked) VHDL processes are *not* necessarily implemented as registers/flip-flops.

1. The two simple designs below are both perfectly valid, but only one will implement the variable ‘t’ as a register. Which one? Explain why this is so and how the two designs differ.

Note that both designs will implement the output signal ‘y’ as a register.

```
architecture design1 of assignment5 is
begin
        process (clk, reset) is
                variable t : std_ulogic;
        begin
                if (reset = ‘1’) then
                        t := '0';
                        y <= ‘0’;
                elsif (rising_edge(clk)) then
                        y <= t;
                        t := a xor b;
                end if;
        end process;
end architecture;
```

```
architecture design2 of assignment5 is
begin
        process (clk, reset) is
                variable t : std_ulogic;
        begin
                if (reset = ‘1’) then
                        t := '0';
                        y <= ‘0’;
                elsif (rising_edge(clk)) then
                        t := a xor b;
                        y <= t;
                end if;
        end process;
end architecture;
```

2. Synthesize the two designs. Look at the RTL or technology view of the synthesized circuits and try to understand why they implement the respective VHDL designs correctly.
3. What determines whether a signal/variable will be implemented as a register or not?
4. Can a signal or variable be written to in more than one process? Can they be read in more than one process?

## Task 3: Incomplete sensitivity lists

Consider the following VHDL process:

```
process (a) is
begin
  if (a = '1') then
    y <= b;
  else
    y <= '0';
  end if;
end process;
```

When this is synthesized, the following warnings will appear:

*[Synth 8-614] signal ‘b’ is read in the process but is not in the sensitivity list*

This means that the synthesis tool assumes that the signal ‘b’ is also supposed to be in the sensitivity list, because otherwise the process would not be synthesizable.

1. Why is the process not synthesizable without ‘b’ in the sensitivity list?
2. Say that you have written a design and tested it successfully by simulation. Then, you synthesize the design and get one or more of the warnings above. Is there any reason to suspect that the generated circuit will not work like during simulation? Explain.
3. If yes, what should you do in this situation to make sure the circuit conforms to simulation results?

## Task 4: Combinational loops

A combinational loop is any form of feedback in the combinatorial path of a synthesized circuit. This usually indicates a problem and therefore the synthesis tool will issue the following warning whenever this occurs:

*"CL180: Found combinational loop at y"*

The statement "b <= a xor b; y <= b;" is an example of a statement that will trigger this warning. Depending on the synthesis tool the generated netlist will look something along the lines illustrated by Figure 3 and Figure 4.

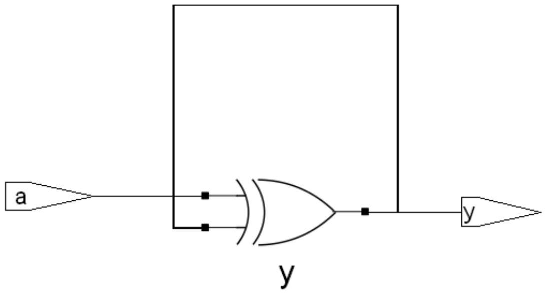


*Figure 3. Combinational loop in standardcell technology.*

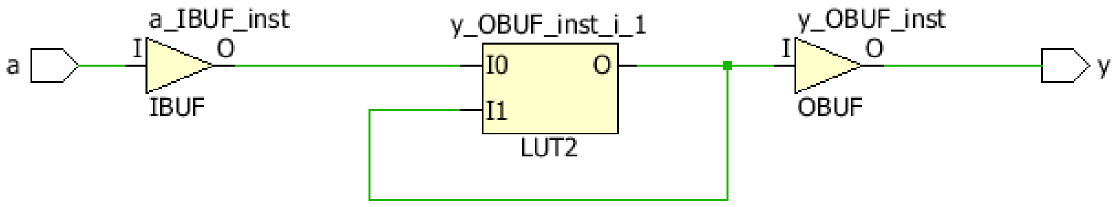
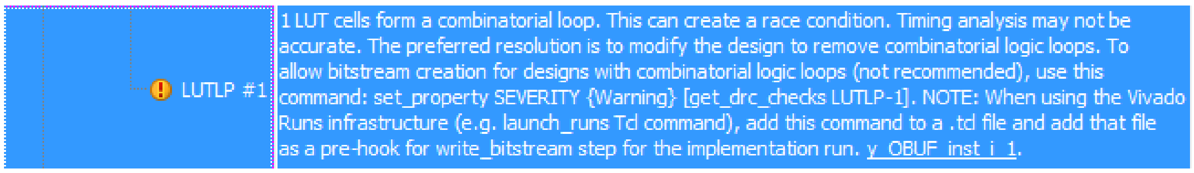


*Figure 4. Combinational loop in FPGA technology with LUTs.*

This particular circuit is unstable; *y* will oscillate when *a* is 1 and have an unknown value when *a* is 0.

In order to extract this warning from Xilinx Vivado Design Suite, it will be necessary to click on "Synthesis > Synthesized Design > Report DRC" (DRC=Design Rule Check). This will then generate the following warning:



LUTLP #1

1 LUT cells form a combinatorial loop. This can create a race condition. Timing analysis may not be accurate. The preferred resolution is to modify the design to remove combinatorial logic loops. To allow bitstream creation for designs with combinatorial logic loops (not recommended), use this command: set_property SEVERITY {Warning} [get_drc_checks LUTLP-1]. NOTE: When using the Vivado Runs infrastructure (e.g. launch_runs Tcl command), add this command to a .tcl file and add that file as a pre-hook for write_bitstream step for the implementation run. y_OBUF_inst_i_1.

*Figure 5. DRC violation in Vivado.*

1. Problems with oscillations from combinational loops will typically be uncovered during simulation. Can you think of any situation where this is not the case? (Hint: combine a combinational loop with an incomplete sensitivity list.)

2. In what situations is it permissible to have the same signal/variable on both sides of an assignment? (E.g., b <= a xor b;)