TFE4141 Design of Digital Systems 1

Assignment 4: Common VHDL pitfalls and misconceptions

# 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.

|  |  |
| --- | --- |
| **library IEEE;**  **use IEEE.STD\_LOGIC\_1164.ALL;**  **entity Latch is**  **port(**  **A: in std\_logic\_vector(1 downto 0);**  **Y: out std\_logic**  **);**  **end Latch;**  **architecture Behavioral of Latch is**  **begin**  **latch: process (A) is**  **begin**  **if (A = "00") then Y<= '1';**  **elsif (A = "01") then Y<= '0';**  **elsif (A = "10") then Y<= '1';**  **end if;**  **end process latch;**    **end Behavioral;** | **The previous code leads to the following component, which actually is a latch (register not managed by a clock).**  **As not all options are covered inside the “if” statement, the signal is saved as a latch in order to keep the value when the non-covered option appears.** |

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.

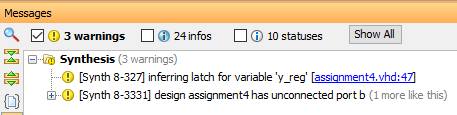


Figure 2. Warning produced by a synthesis tool.

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

**A very common way to create an unintentional latch is by defining an “if” statement (inside a process) and not covering all combinational possibilities.**

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

**Inside a combinational process, the best option to avoid latches would be to cover all combinational possibilities by using statements like “when others” or “else”.**

# 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.

**Both designs are slightly different each other, only in the “elsif” part.**

* **Design1 assigns the value of variable “t” (always defined inside a process) directly to signal “y” (always defined outside a process), and then variable “t” is updated with a xor operation.**
* **Design2 is the other way around: first of all the operation in “t” and then the assignation to “y”.**

**That means design1 uses variable “t” as a register because it must save its value until the next process cycle (next rising edge) in order to save such a new value into “y”.**

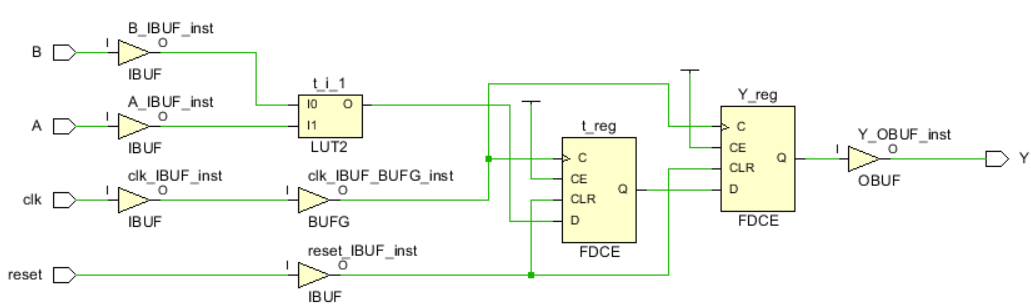
**On the other hand, design2 updates variable “t” (operation which is done immediately unlike with signals) before it is assigned to signal “y”. That means the value of “t” does not need to be saved for the next process cycle.**

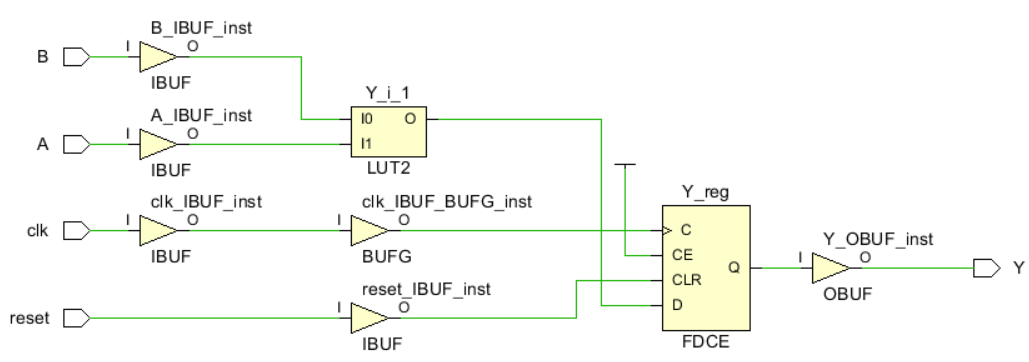
**As a final result, signal “y” is always updated one process cycle before in design2 than in design1.**

|  |  |
| --- | --- |
| 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; |

1. 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.

**Design1: Variable “t” is a register (flip-flop) so as to update the signal “y” in the next process cycle.**

**Design2: Variable “t” does not need to be a register (flip flop) because it is immediately assigned to the signal “y” within the same cycle.**



1. What determines whether a signal/variable will be implemented as a register or not?

**A signal working inside a synchronous (clocked) process will always be a register (flip flop), otherwise such a kind of process would be not possible. In the case of variables, they will be a register only if they are updated after (or not) being assigned to a signal/output, and that means it should be saved until the next process cycle.**

1. Can a signal or variable be written to in more than one process? Can they be read in more than one process?

**Signals are defined outside the processes and can be used in every one of them. In the case of variables, they must be defined inside a process and can be used only inside in such a process (so locally).**

# 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?

**The sensitivity list defines all signals that will execute such a process if they change their values. In this example, if “b” is not included, the process will work anyway, but a change in “b” will not execute such a process, but only a change in “a”.**

1. 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.

**Yes, it could be that the generated circuit behaves slightly different as the simulation.**

**In this example, if the test does not cover every single change of “b” (with and without changes in “a”), the final result would not be able to be validated.**

1. If yes, what should you do in this situation to make sure the circuit conforms to simulation results?

**To ensure not unexpected behaviour is discovered too late, two options are available: to put only the clock in the sensitivity list (so all processes would be synchronized by using flip flops), or to put every single input used inside the process in the sensitivity list (so every modification of each input would execute such a process).**

# 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.

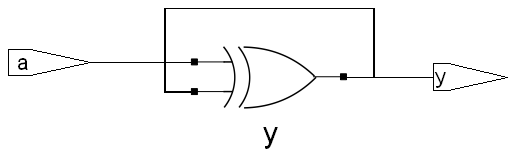


Figure 3. Combinational loop in standard cell technology.

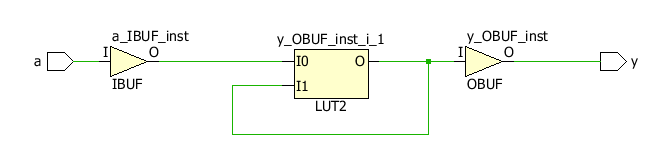


Figure 4. Combinational loop in FPGA technology with LUTs.

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:

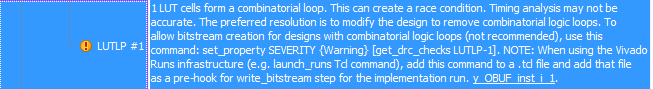


Figure 5. DRC violation in Vivado.

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).

**An example situation where the combinational loop would not be uncovered during simulation is if the feedback inputs are not included inside the sensitivity list, so an incomplete sensitivity list.**

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

**Related to the previous question, now more precisely, the “professional” way to manage a combinational loop would be to define it inside a clocked/synchronous process (so only the clock inside the sensitivity list). That means such a loop operation would be done only once per cycle (clock rising edge).**