

TFA4141-Assignment 4

Group 12

Erfan Abdolniafard

Shima Shakouri

Zahra Jenab Mahabadi

TASK 1

1)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity latch is
Port ( D : in STD_LOGIC;
      EN : in STD_LOGIC;
      Q : out STD_LOGIC);
end latch;


architecture Behavioral of latch is

signal DATA : STD_LOGIC;
begin

    DATA <= D when (EN = '1') else DATA;
    Q <= DATA;

end Behavioral;
```

▼  Synthesis (1 warning)

 [Synth 8-327] inferring latch for variable 'DATA_reg' [dlatch.vhd:45]

2)

```

library ieee;
use ieee.std_logic_1164.all;

entity bad_latch_2 is
  port (
    i_select : in  std_logic_vector(1 downto 0);
    o_latch  : out std_logic_vector(3 downto 0)
  );
end bad_latch_2;

architecture rtl of bad_latch_2 is
begin

  -- Incomplete Assignment:
  o_latch <= "0101" when i_select = "00" else
            "0111" when i_select = "01" else
            "1111" when i_select = "10";

end architecture rtl;

```

This will happen when our assignment is incomplete. The assignment of the output is not complete under all input possibilities. This is bad and should be avoided! Let's look at some VHDL example code to see how a latch is created.

3)

We forgot to specify all possible input combinations of i_select. What happens to o_latch when i_select = "11"? This creates an incomplete assignment and therefore creates a latch. A latch is almost certainly not what the Digital Designer intended to create with this code. To fix the code above, the Digital Designer can either specify what happens when i_select = "11" or can use the *else* clause with no *when* attached to it. I recommend the latter because it is more flexible. If the width of i_select changes from two bits to three bits your code will still not create a latch.

```

-- Fixed!
o_latch <= "0101" when i_select = "00" else
          "0111" when i_select = "01" else
          "1111" when i_select = "10" else
          (others => '0');

```

Source: <https://www.nandland.com/articles/how-to-avoid-transparent-latches-in-vhdl-and-verlog.html>

TASK2

1)

In both designs we set the value of "t" to zero when the if statement is true. but in design1 if the statement is not true we are telling the synthesis tool to remember the value of "t" and the only way to do that is by using a register because in design1, when we enter the elseif condition, we give the y, the value of t and

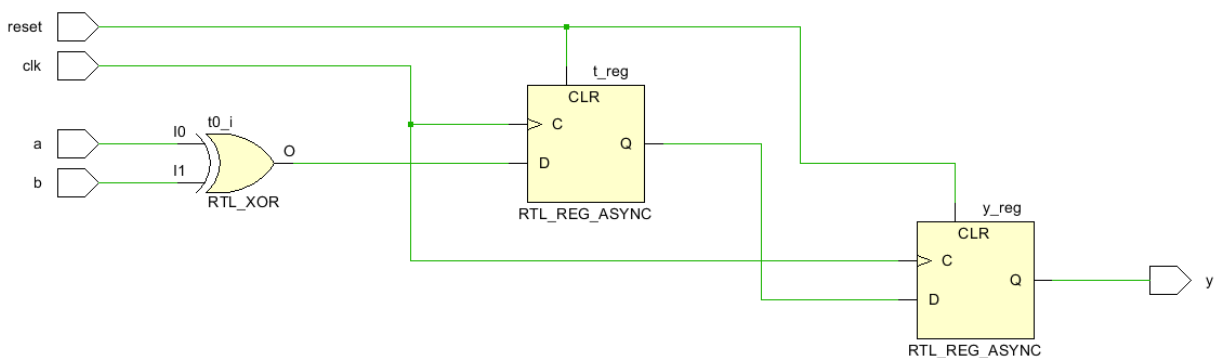
this way, we preserve the previous value of t in y and then in the next line, we store the result of the xor function in t. In this condition, t must have been implemented as a register because it is important to know it's initial value so that we can put into signal y. In design 2, it does not matter if we have a initial value for t or not. Since, at the first line of elseif, the result of a xor function b is stored in t and then the second line runs. Therefore, we do not need a register in this design.

In summary, in design 1, since we implement t as a register, we are able to have different values for t and y. This is not the case in design 2 and the values of both t and y will be equal at the end

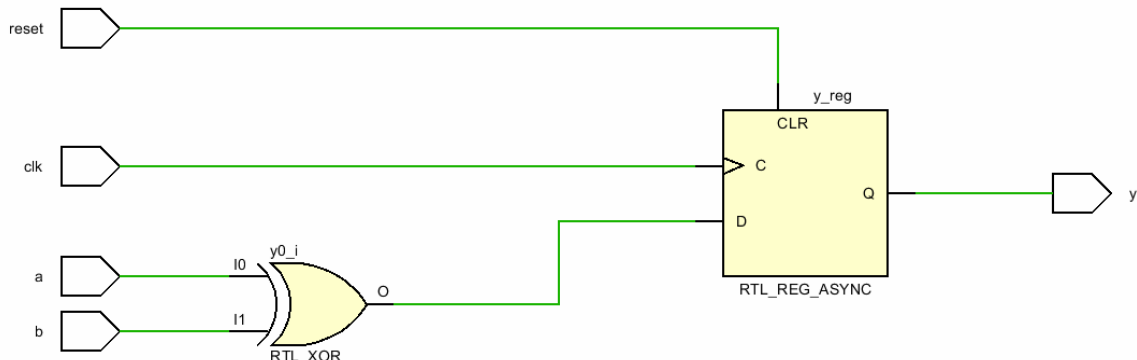
2)

It can be seen that the synthesis result completely matches the conclusion we made in the first section of this task and we can see that they both are valid and produce the same output.

Design 1:



Design 2:



3)

Whenever we are telling the synthesis tool to remember the value.

4)

Variables can only be used inside processes and are not visible outside the process in which they are declared; signals can be used inside or outside processes and are visible in all local processes if they are declared in an entity or architecture. Any variable that is created in one process cannot be used in another process, signals can be used in multiple processes though they can only be assigned in a single process.

Source:<https://www.nandland.com/vhdl/tips/variable-vs-signal.html#:~:text=Variables%20can%20only%20be%20used,assigned%20in%20a%20single%20process.>

TASK3

1)

Most, if not all, synthesis tools require that the processes used to describe combinational logic should have a 'complete' sensitivity list, while for synchronous logic only the reset, if any, and clock signals should be in the sensitivity list. Beside these two applications of processes with sensitivity lists, there is a vague support for other incomplete sensitivity lists, sometimes resulting in latches in the circuit and sometimes resulting in logic that does not have the proper behavior.

2)

When we use some signals in our process and they are in the sensitivity list, the output signal will compute and store a new value based on our change on the input signals but if we don't put some of the used signals inside our sensitivity list, the simulation might not show us the result that we expect because output will not produce a new value based on the change that we made on the signals which are not in our sensitivity list.

3)

We need to make sure that all the signals which are used within the process, are mentioned in the sensitivity list of the process.

TASK4

1)

In a combinational loop (having feedback from the output), if we have a complete sensitivity list, we will be able to see the oscillations in the output. (In this example no matter the value of b, we will have oscillations for a=1 and unknown for a=0) because the value of b must be updated in order to change the y (to produce the oscillation). However, if we have an incomplete sensitivity list, the value of b may not be updated and hence we won't have oscillations.

2)

Whenever our design is synchronized by using registers.