Management and analysis of physics datasets, Part. 1

Third Laboratory

Antonio Bergnoli bergnoli@pd.infn.it

16/11/2021

Laboratory Introduction

Goals

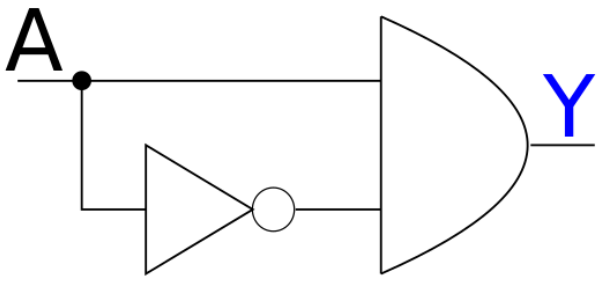
- make some practice with sequential circuits.
- Exploit the configuration flash memory chip of the Arty7 board.

VHDL naming convention

Signals/components	Name
Clock	clk
Reset	rst
Input Port	$port\_in$
Output Port	$port\_out$
VHDL file name	$entityname.vhd$
Test bench file name	$tb\_entityname.vhd$
Signal between 2 comps	$sign\_cmp1\_cmp2$
process name	p_name
...	...

Sequential Logic Circuits

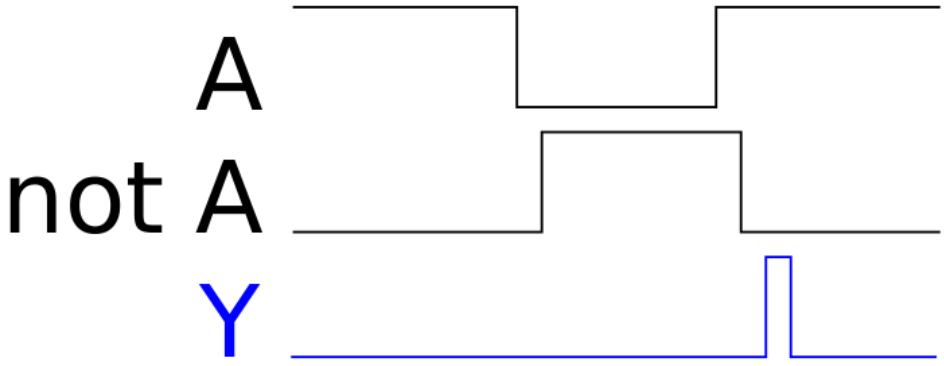
Glitch (1)



$$Y = A \text{ and } (\text{ not } A) = '0'$$

So far we have assumed that the logic gates do not introduce a delay. In practice this does not happen.

Glitch (2)



Actually there is a brief delay between the input changing and the respective changing.
The spurious '1' is a **glitch**.

How reduce glitches ?

- With extra combinatorial hardware. This design is not so simple.
- Design circuit glitches free. The logic: wait a certain amount of time, during with you do not consider the glitches. This is the idea behind the clocked circuits or **sequential circuits**.

Are they a problem? Depends on the circuit downstream the glitch. I.e., if there is a pulse counter, the final result of the counter will be wrong.

Processes

- Statements are executed one after the other sequentially, in order
- Processes are used to describe sequential behavior
- All processes in an architecture behave concurrently

Process Statements

The process includes the “sensitivity list”, which is a set of signals on which any change triggers the process itself

- Process runs when any of the signals in the sensitivity list changes
- A process with a sensitivity list must not contain any “wait” statement
- The sensitivity list can only include signals and input ports
- The execution of a process consists in the execution of its whole sequence of statements

```
library ieee;
use ieee.std_logic_1164.all;
entity process_ex is
  port (
    a : in  std_logic;
    b : in  std_logic;
    c : in  std_logic;
    y : out std_logic
    );
end entity process_ex;
architecture rtl of process_ex is
  signal x : std_logic;
begin  -- architecture rtl
  process (a, b, c, x)
  begin
    x <= (a and b) or c;
    y <= x;
  end process;
end architecture rtl;
```

Sequential Clocked Process

- Generates synchronous logic circuits
- All signals are evaluated at the rising edge of the clock, no need to add other signals on the sensitivity list

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    d   : in  std_logic;
    q   : out std_logic);
end entity dff;
architecture rtl of dff is
begin  -- architecture rtl
  flipflop : process (clk) is
  begin  -- process flipflop
    if rising_edge(clk) then   -- rising clock edge
      if rst = '0' then
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process flipflop;
end architecture rtl;
```

Classwork

Suggested excercise

- Build a testbench for the D Flip-Flop
- Design the architecture of a Toggle Flip-Flop
- Build a testbench to check its behavior

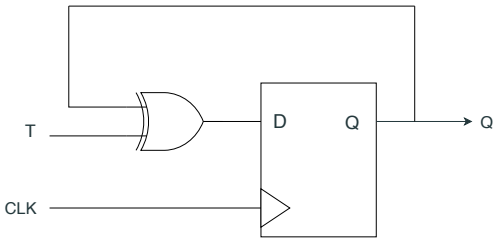


Figure 1: Toggle Flip-Flop Model

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2: Toggle Flip-Flop Truth Table

Sequential Elements - Code

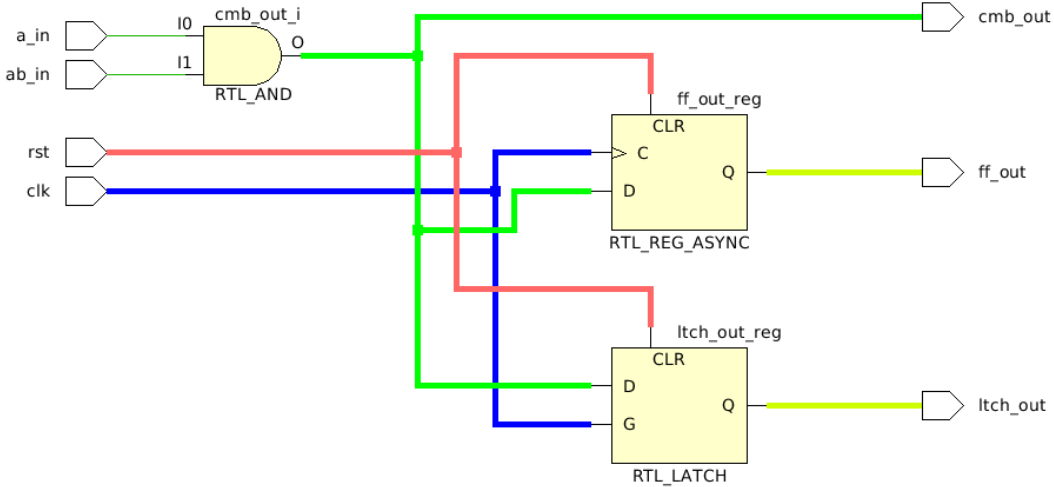
```
entity cmb_clk is
  Port (clk   :   in  std_logic;
        rst   :   in  std_logic;
        a_in  :   in  std_logic;
        ab_in :   in  std_logic;
        cmb_out  :  out  std_logic;
        ltch_out :  out  std_logic;
        ff_out   :  out  std_logic);
end cmb_clk;

architecture rtl of cmb_clk is

begin

p_ff: process(clk, rst, a_in, ab_in) is
   begin
   if rst = '1' then
      ff_out <= '0';
   elsif rising_edge(clk) then
      ff_out <= a_in and ab_in;
   end if;
end process;

p_ltch: process(clk, rst, a_in, ab_in) is
   begin
   if rst = '1' then
      ltch_out <= '0';
   elsif clk = '1' then
      ltch_out <= a_in and ab_in;
   end if;
end process;

cmb_out <= a_in and ab_in;
```

FLIP-FLOP

LATCH

COMBINATORIAL

Sequential Elements - Schematic

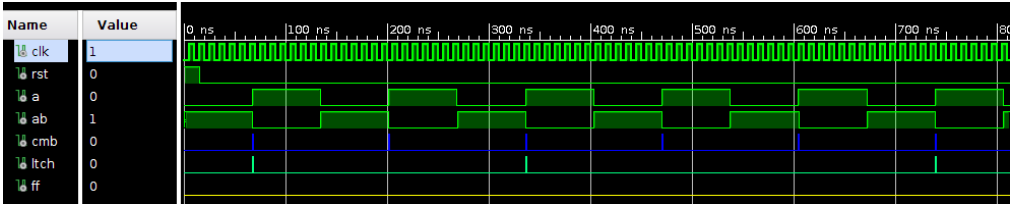


Sequential Elements - Testbench

```
architecture Behavioral of tb_cmb_clk is

component cmb_clk is
  Port (clk   :   in  std_logic;
        rst   :   in  std_logic;
        a_in  :   in  std_logic;
        ab_in :   in  std_logic;
        cmb_out  :  out  std_logic;
        ltch_out :  out  std_logic;
        ff_out   :  out  std_logic);
end component;

signal clk,rst : std_logic; signal a,ab : std_logic; signal cmb,ltch,ff : std_logic;

begin

uut : cmb_clk port map (clk=> clk, rst => rst, a_in => a, ab_in => ab, cmb_out => cmb, ltch_out => ltch, ff_out => ff);

p_clk : process -- 100 MHz
begin
   clk <= '0'; wait for 5 ns; clk <= '1'; wait for 5 ns;
end process;

p_rst : process
begin
   rst <= '1'; wait for 15 ns; rst <= '0'; wait;
end process;

p_cmb : process
begin
   a <= '0'; wait for 0.2 ns; ab <= '1';  wait for 67 ns;
   a <= '1'; wait for 0.2 ns; ab <= '0';  wait for 67 ns;

end process;

end Behavioral;
```

NOT delay

Sequential Elements - Simulation



- A latch is sensitive to the level. In this example to '1'.
- A Flip-Flop is sensitive to the edge. In this example to the rising edge $0 \rightarrow 1$.
- A Flip-Flop is immune to glitches.
- In the next laboratories you use the Flip-Flops !!!

Notes on the code

- Inside a process the code lines are executed sequentially. Inside a process the code, you consider the behavior similar to C, Python ...
- The processes run in parallel.
- In the sensitivity list of a process, you write all their inputs .
- In the testbench file is good practice to use a process for the clock signal, one for the reset signal and one for the others signals.

Notes on the Clock and Reset

- In this laboratory you use an asynchronous reset.

```
if rst = '1' then
    .  .  .  .  .  .  .
elsif rising_edge(clk) then
    .  .  .  .  .  .  .
end if;
```

- It is used to bring the circuit in a known state. That is the states (you will learn) and the output of the circuit are set to a default value.
- The clock in the board is provided by an oscillator at 100 MHz. It is connect to the pin E3 of the FPGA.

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=gclk[100]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];
```

Counter

Counter and Led Blink

```
use IEEE.NUMERIC_STD.ALL;

entity blink is
  Port (clk  :   in  std_logic;
        rst  :   in  std_logic;
        y_out:   out std_logic);
end blink;

architecture rtl of blink is
signal counter : unsigned (27 downto 0);
begin
p_cnt: process(clk, rst) is
   begin
   if rst = '1' then
      counter <= (others => '0');

   elsif rising_edge(clk) then
      counter <= counter + 1;
   end if;
end process;

y_out <= counter(26);

end rtl;
```

$$f_{\text{blink}} = \frac{f_{\text{clock}}}{2^{26}} = \frac{100\text{MHz}}{2^{26}}$$

Slower Counter

```
use IEEE.NUMERIC_STD.ALL;
entity cnt is
  Port (clk  :   in  std_logic;
        rst  :   in  std_logic;
        y_out:   out std_logic_vector(3 downto 0));
end cnt;
architecture rtl of cnt is
signal slow_clk, slow_clk_p : std_logic; signal counter : unsigned (27 downto 0);
signal slow_counter : unsigned (3 downto 0);
begin
p_cnt: process(clk, rst) is
   begin
   if rst = '1' then
      counter <= (others => '0');
   elsif rising_edge(clk) then
      counter <= counter + 1;
   end if;
end process;
slow_clk <= counter(26);
p_slw_cnt: process(clk, rst, slow_clk) is
   begin
   if rst = '1' then
      slow_counter <= (others => '0');
   elsif rising_edge(clk) then
      slow_clk_p <= slow_clk;
      if slow_clk = '1' and slow_clk_p = '0' then -- "RISING EDGE"
         slow_counter <= slow_counter + 1;
      end if;
   end if;
end process;
y_out <= std_logic_vector(slow_counter);
end rtl;
```

Slower Counter - few changes

1. Use $btn0$ as the reset button;
2. Use $sw0$ to change the counter incremental. If $sw0 =' 0'$ the counter increment itself, else it decrements itself.
3. Use $sw1$ to freeze the led status. If $sw1 =' 1'$ the counter is stuck.

In order to simulate with a testbench the entity cnt, $slow\_clk$ signal wuold be assigned at $counter(3)$.
Then you create the bitstream and you download it into the FPGA.

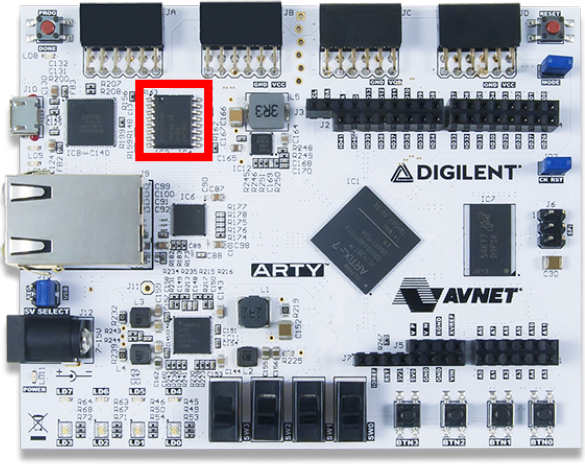
Use of the SPI Flash Memory

Board Configuration

- You noticed that when the board is programmed and you power off it, at the boot afterward the board has been reset.
- To avoid this, you have to load the configuration file (.mcs) in the SPI flash memory.
- So at the next boot, the FPGA loads the configuration file from the flash, without waiting you load the bitstream.

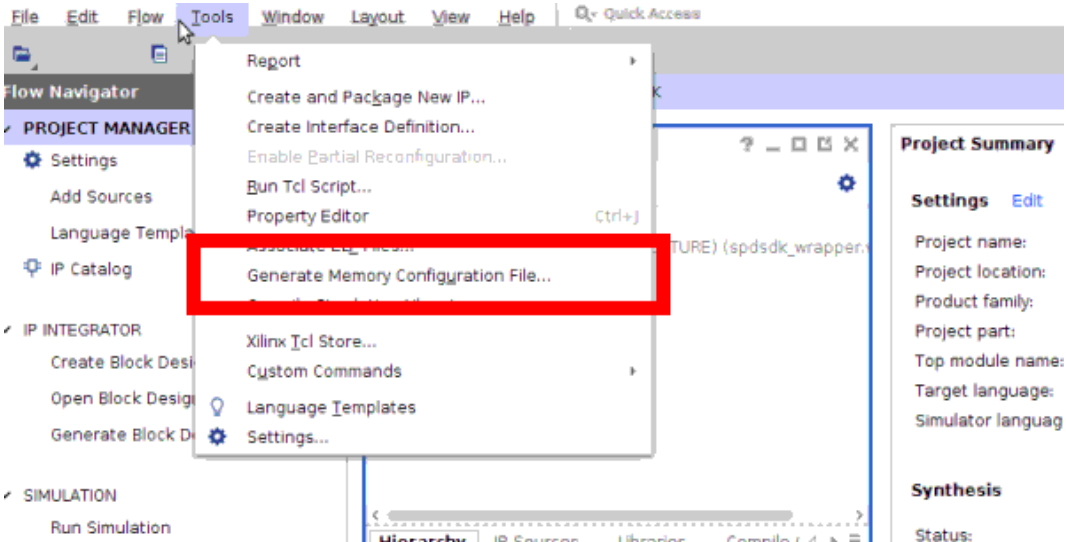
In the next slides is described how to generate the mcs file and how to load it in the flash memory.

Flash Chip



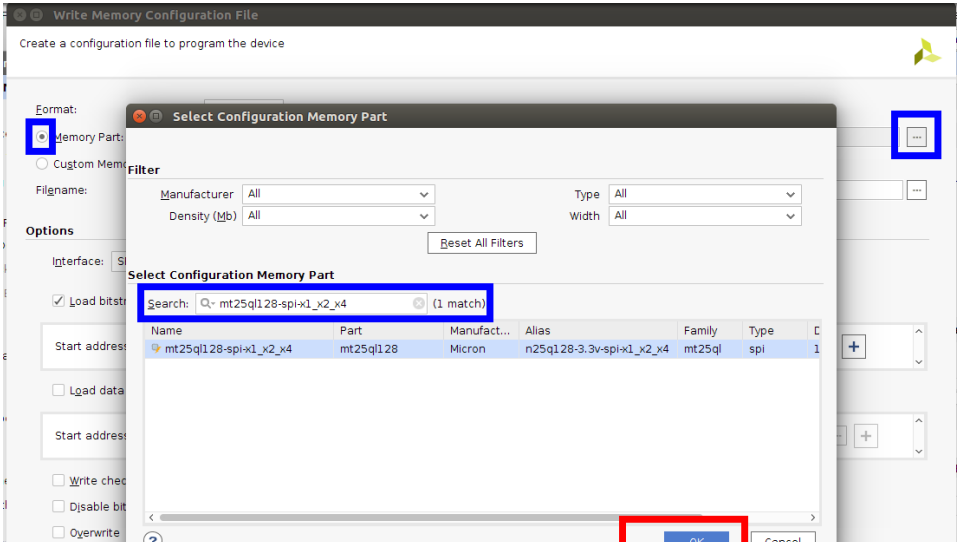
Flash configuration (1) Tools → Generate Memory

Configuration File →



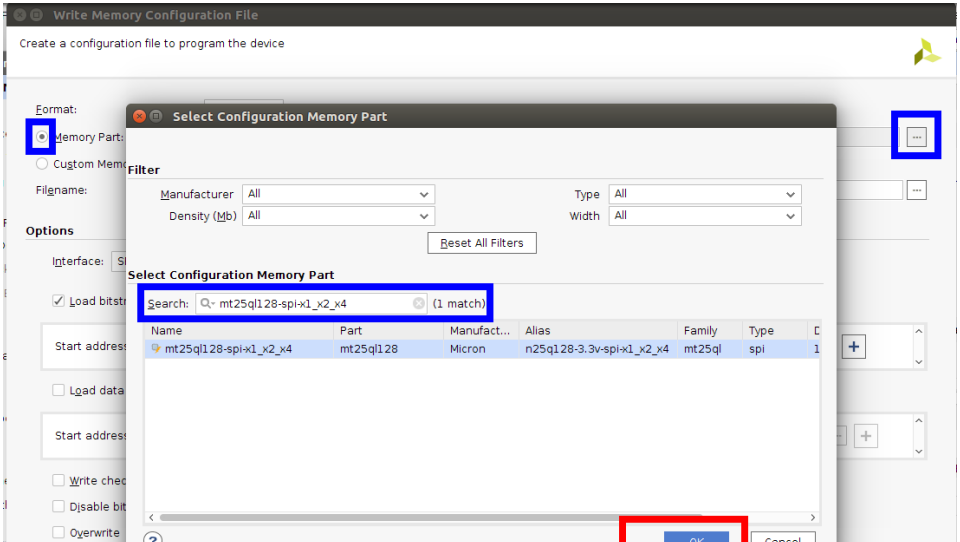
Flash configuration (2) Check ”Memory Part”, click ”...”, search

”**mt25ql128-spi-x1_x2_x4**” and ”OK”.



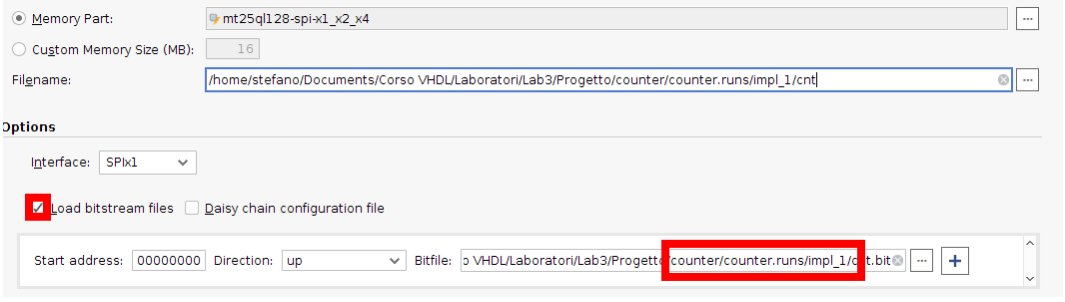
Flash configuration (2) Check ”Memory Part”, click ”...”, search

”**mt25ql128-spi-x1_x2_x4**” and ”OK”.



Flash configuration (3) In ”Filename” insert the name of file, in

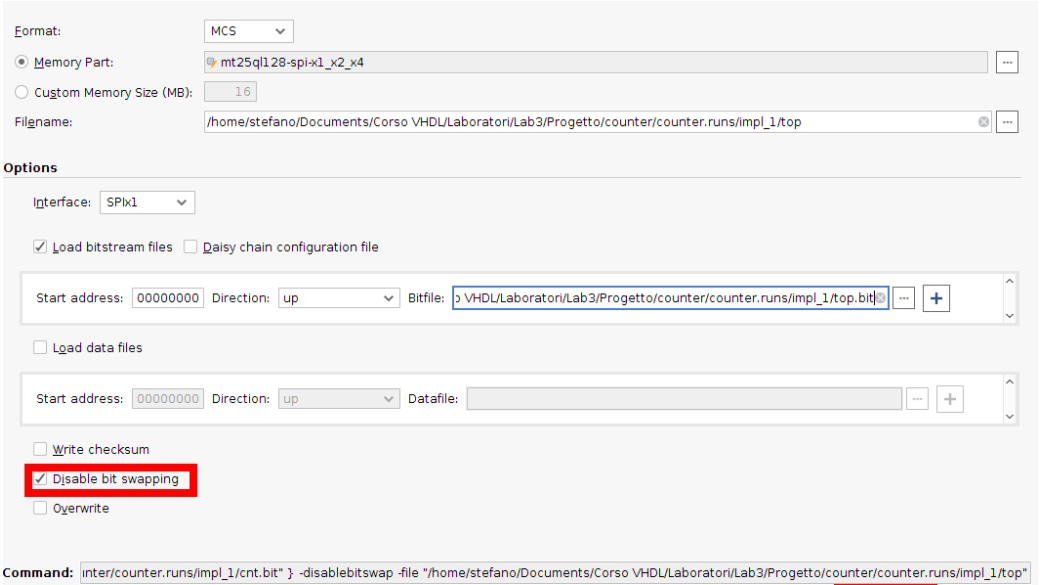
example case **top** and the path of the file. Suggestion: you create the file in the same folder of the bitstream file (.bit).



[ide]

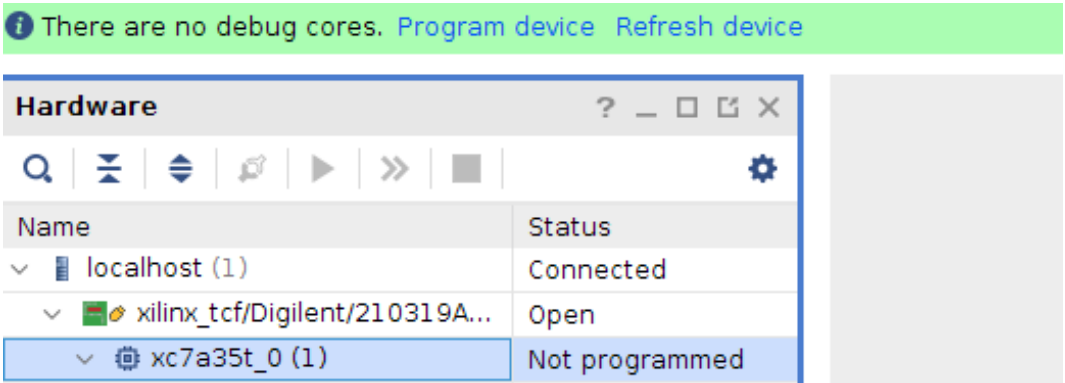
Then leave ”Interface field” at SPIx1, check ”Load bitstream files” and select the bitstream file to be *translated* in .mcs format.

Flash configuration (4) Check ”Disable bit swapping” and then ”OK”.



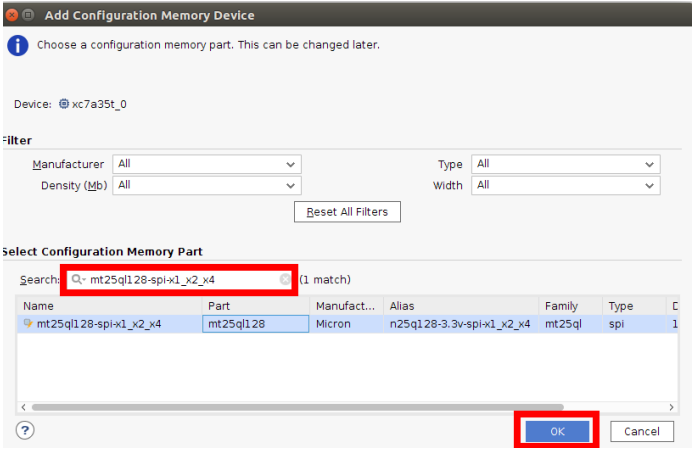
Flash configuration (5)

1. Connect the evaluation board to PC by the usb cable.
2. Open Hardware Manager →.
3. Open Target →.
4. Auto Connect.
5. Right Click on *"xc7a35t_0 (1)"*.



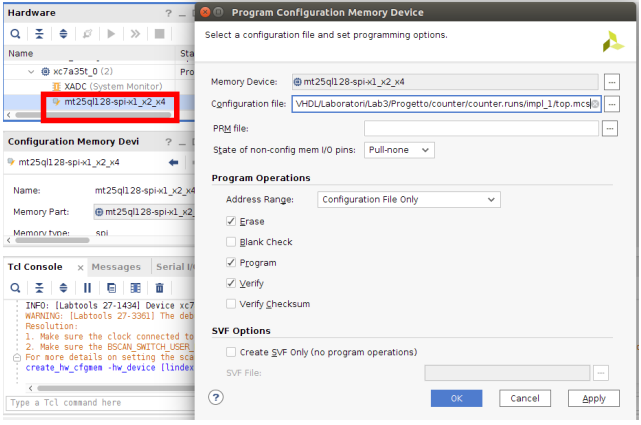
Flash configuration (6)

1. "Add Configuration Memory Device".
2. search "**mt25ql128-spi-x1_x2_x4**" →.
3. "OK" →.



Flash configuration (7)

1. Right click on the memory device.
2. Program Configuration Memory Device ... →
3. Select the mcs file just created.
4. OK.



Flash configuration (8)

Close the Hardware Manager.
Disconnect the board.
Reconnect it.
Wait ...

Homework

Suggested exercises

- Redo 1,2,3, ..., N times the exercises regarding the blink of the led and the slower counter.
- Modify the slower counter exercise in this way :
    1. if SW0 = '0' the counter increments itself else it decrements itself;
    2. if SW1 = '1' the counter doubles its blinking frequency;
    3. if SW2 = '1' the counter halves its blinking frequency;
    4. if SW3 = '1' the counter freezes its state.