Management and analysis of physics datasets, Part. 1

First Laboratory

Stefano Pavinato
22/11/2018

Outline

Outline

Goals

- Become familiar with the Xilinx Vivado IDE.
- Implement the VHDL "Hello World". Then synthesize, simulate and download the project in the Artix-7 FPGA Development Board.

Run Vivado

Windows

Double click on the icon created during the installation project.

Linux

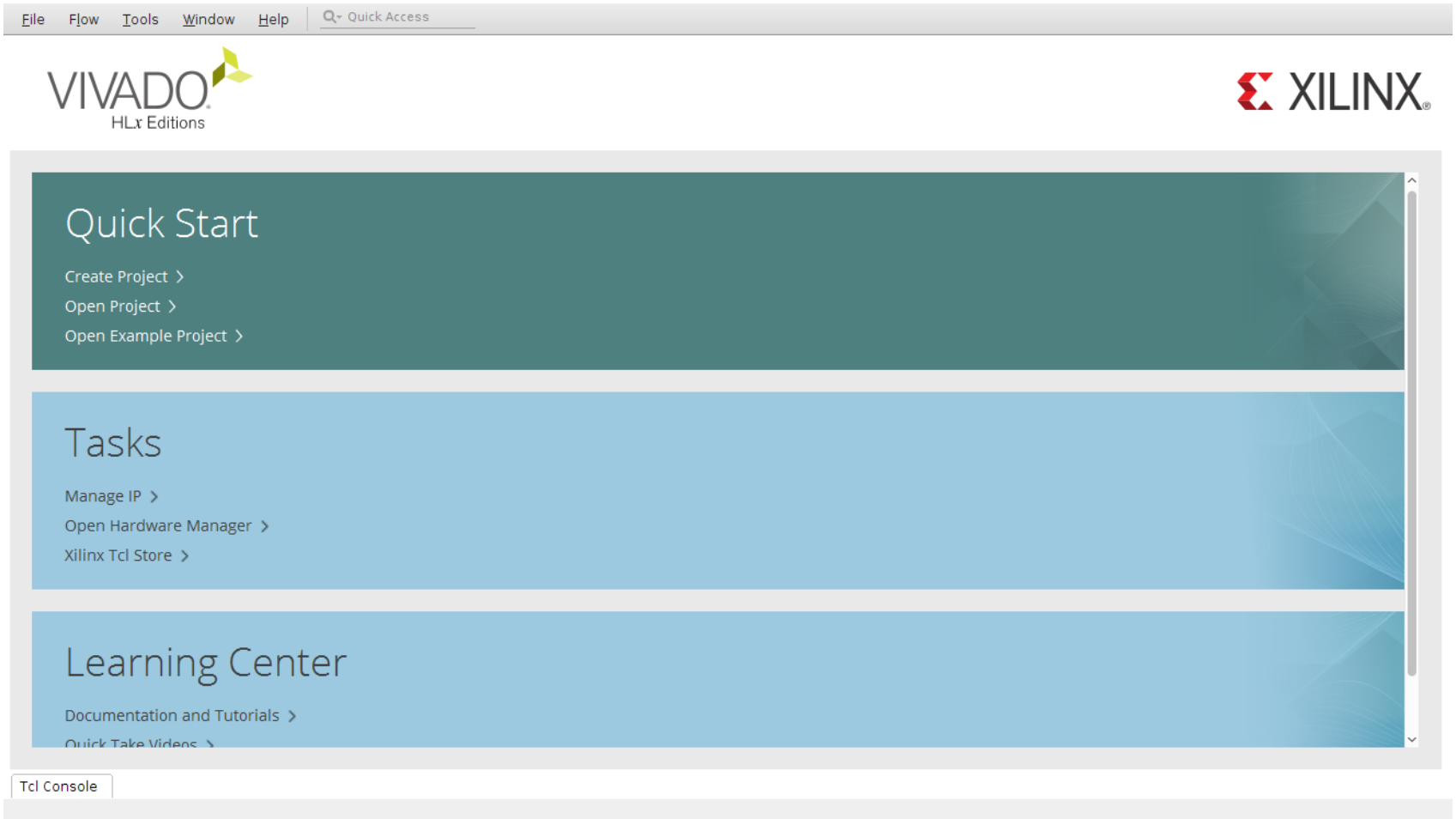
source <installpath>/Vivado/2018.2/settings64.sh && vivado

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*
...	...

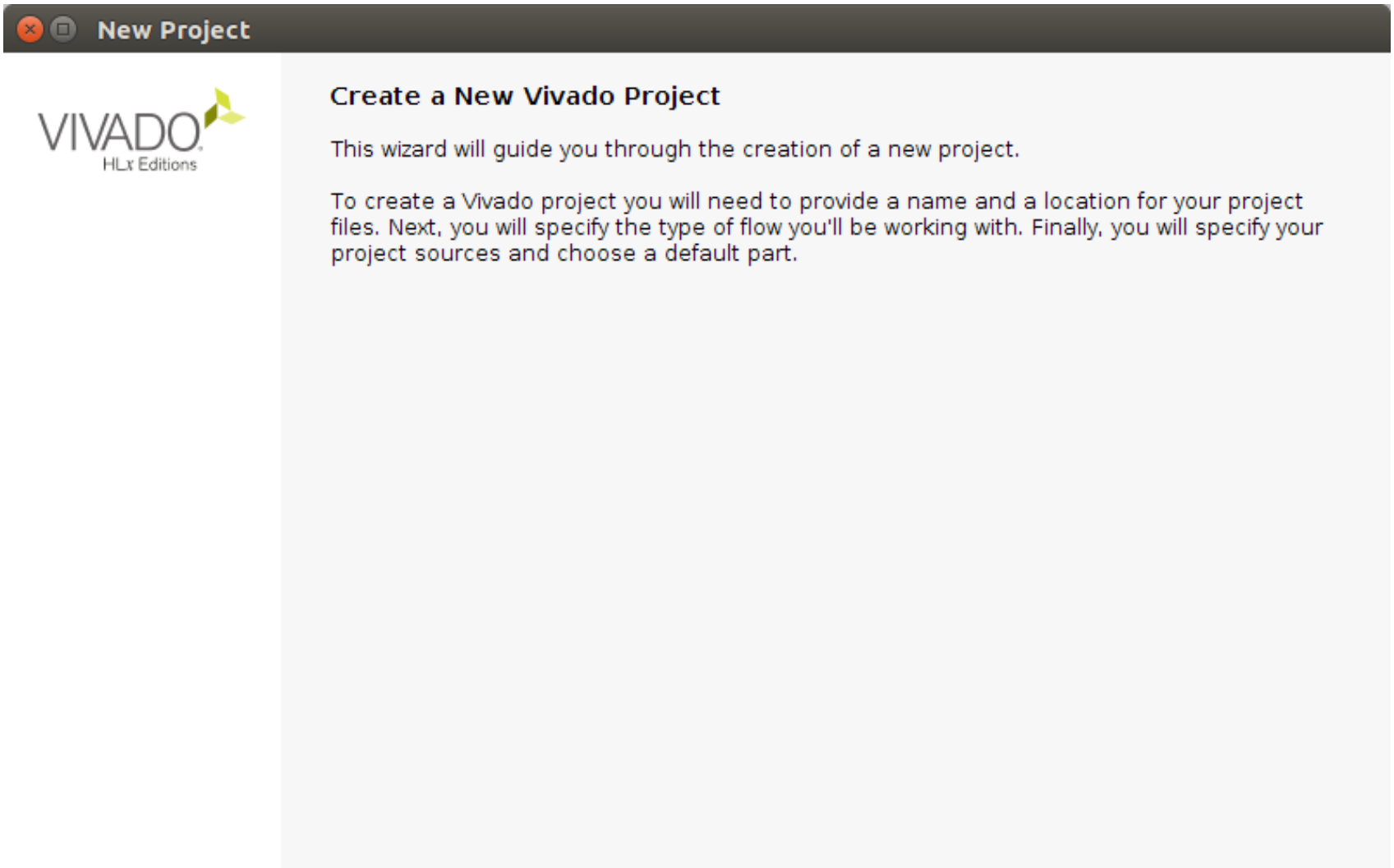
Outline

Presentation screen



Make a new project (1)

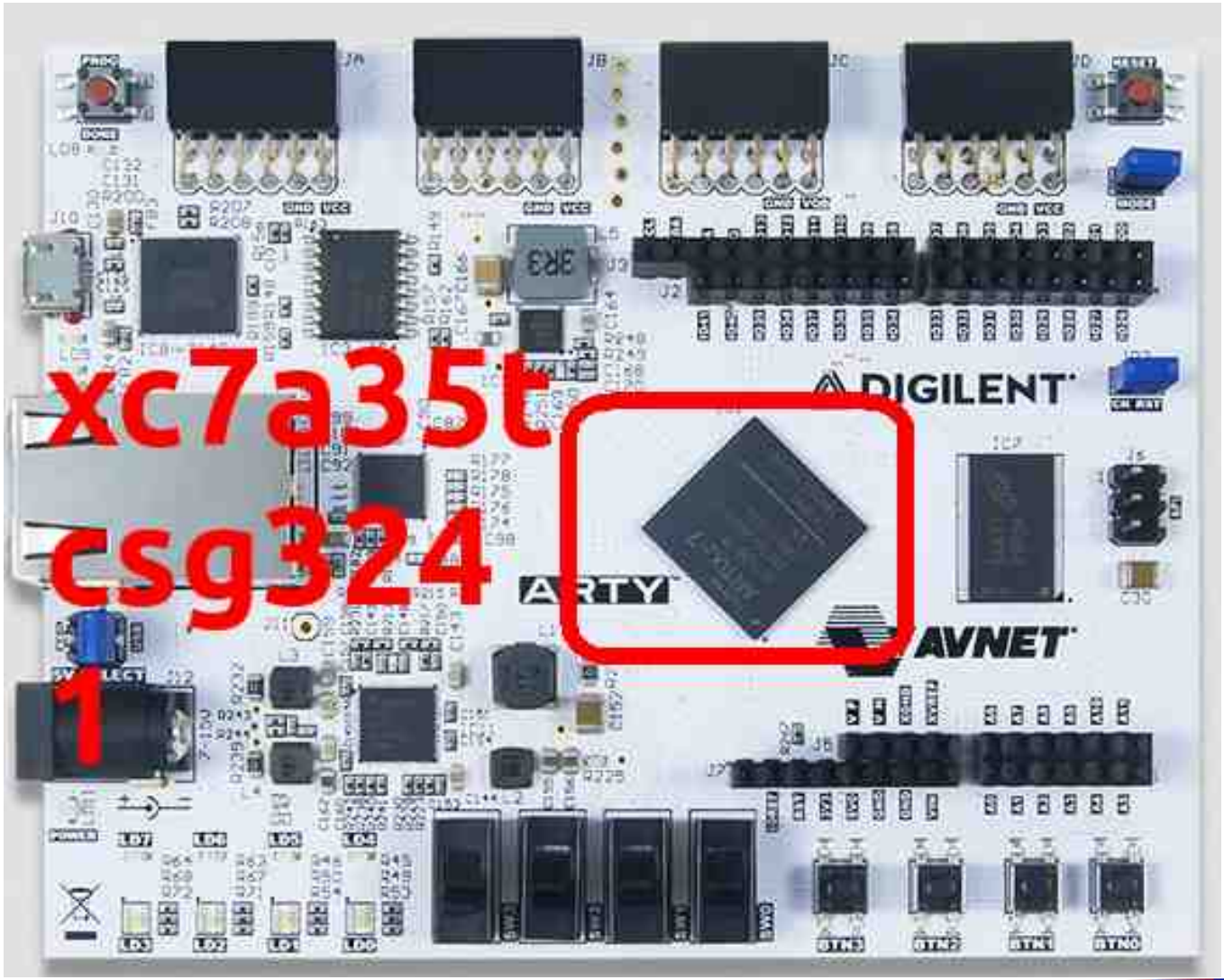
File → Project → New ...



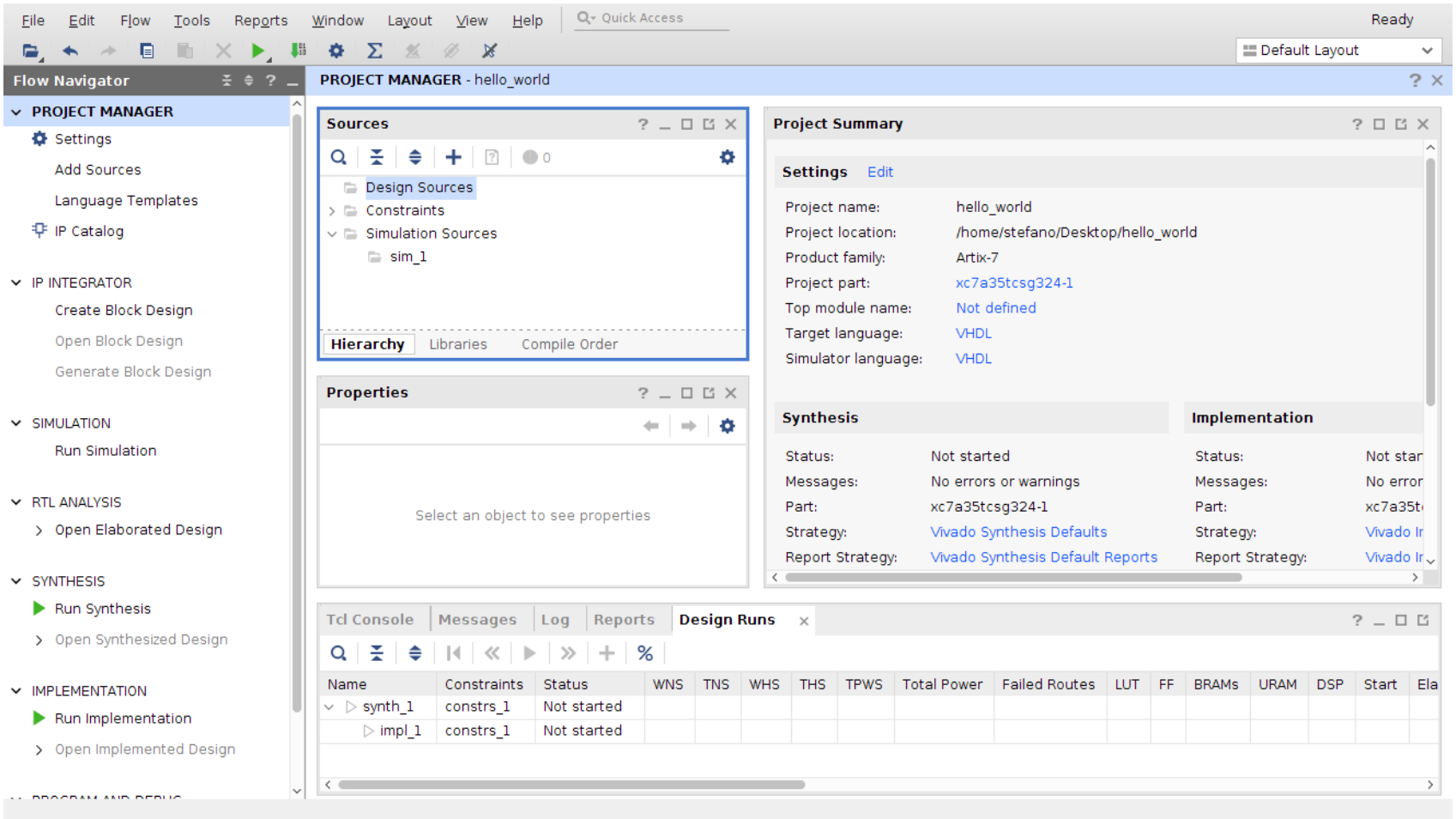
Make a new project (2)

1. Project name : **hello_world**;
2. Check "Create project subdirectory";
3. Next →;
4. Check "RTL Project";
5. Next →;
6. Target language : **VHDL**;
7. Simulator language : **VHDL**;
8. Next →;
9. Next →;
10. Search : **xc7a35tcsg324-1**;
11. Next →;
12. Finish →.

Evaluation Board - FPGA

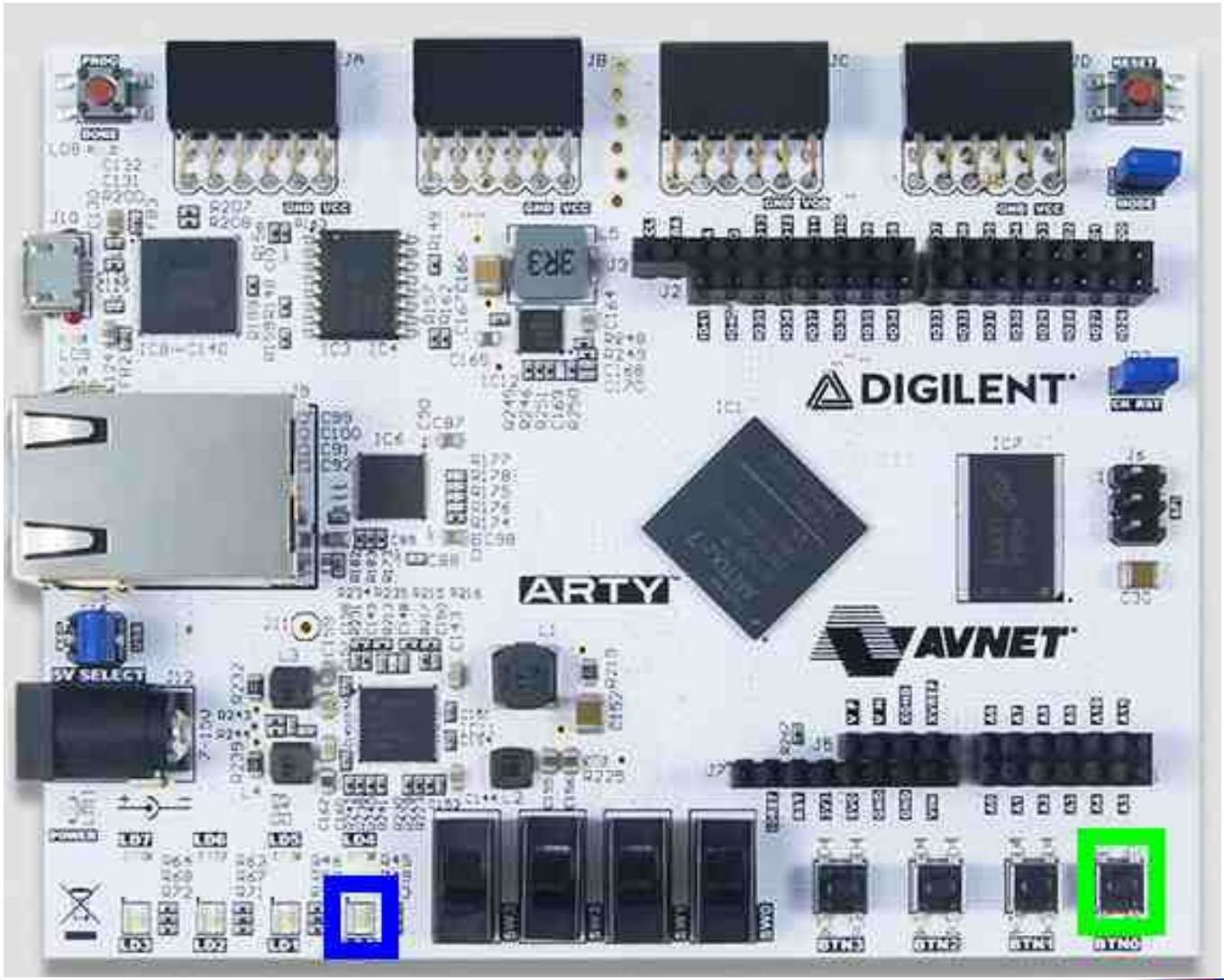


Make a new project (3)



Outline

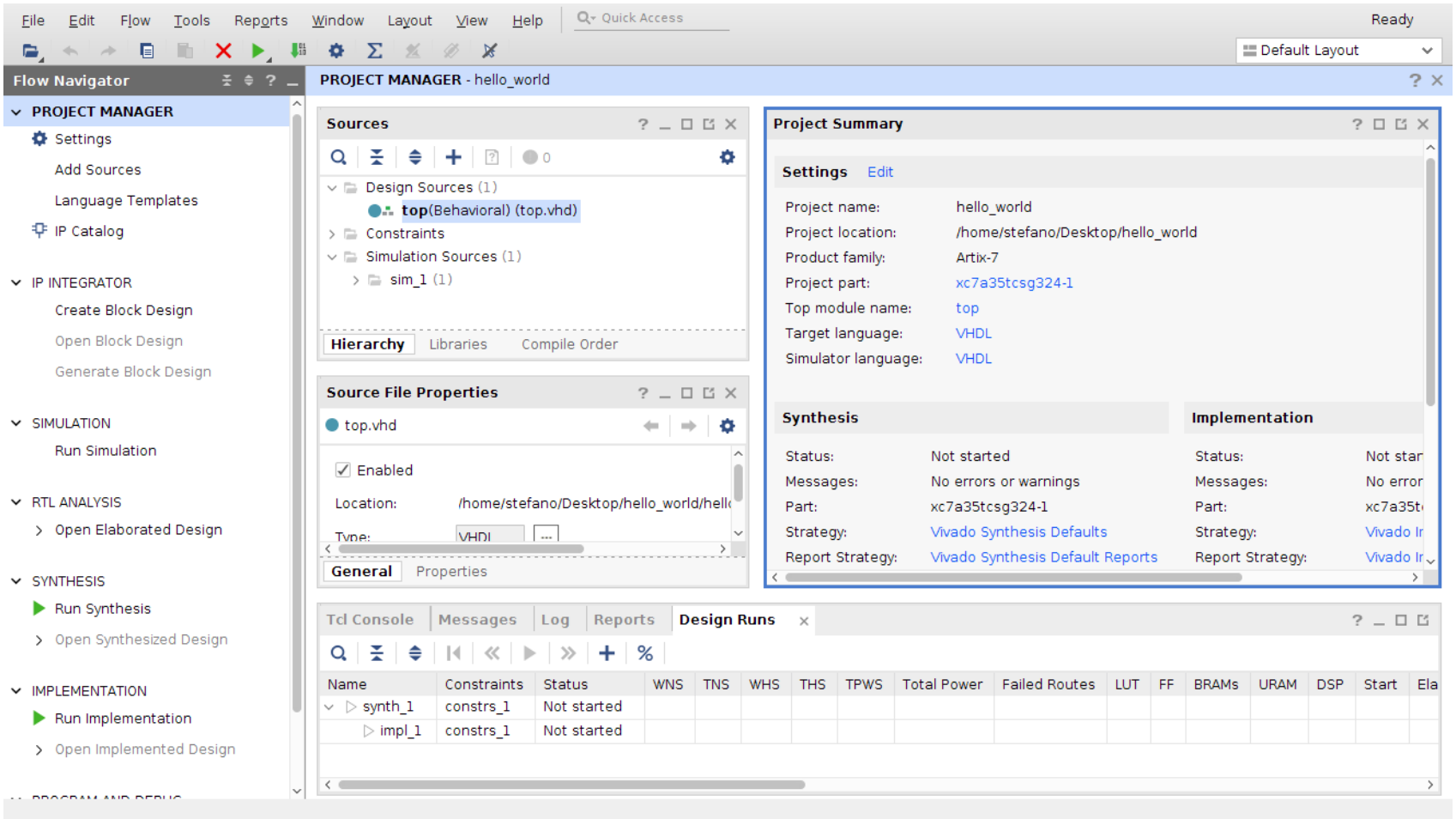
Hello World



Make the source file

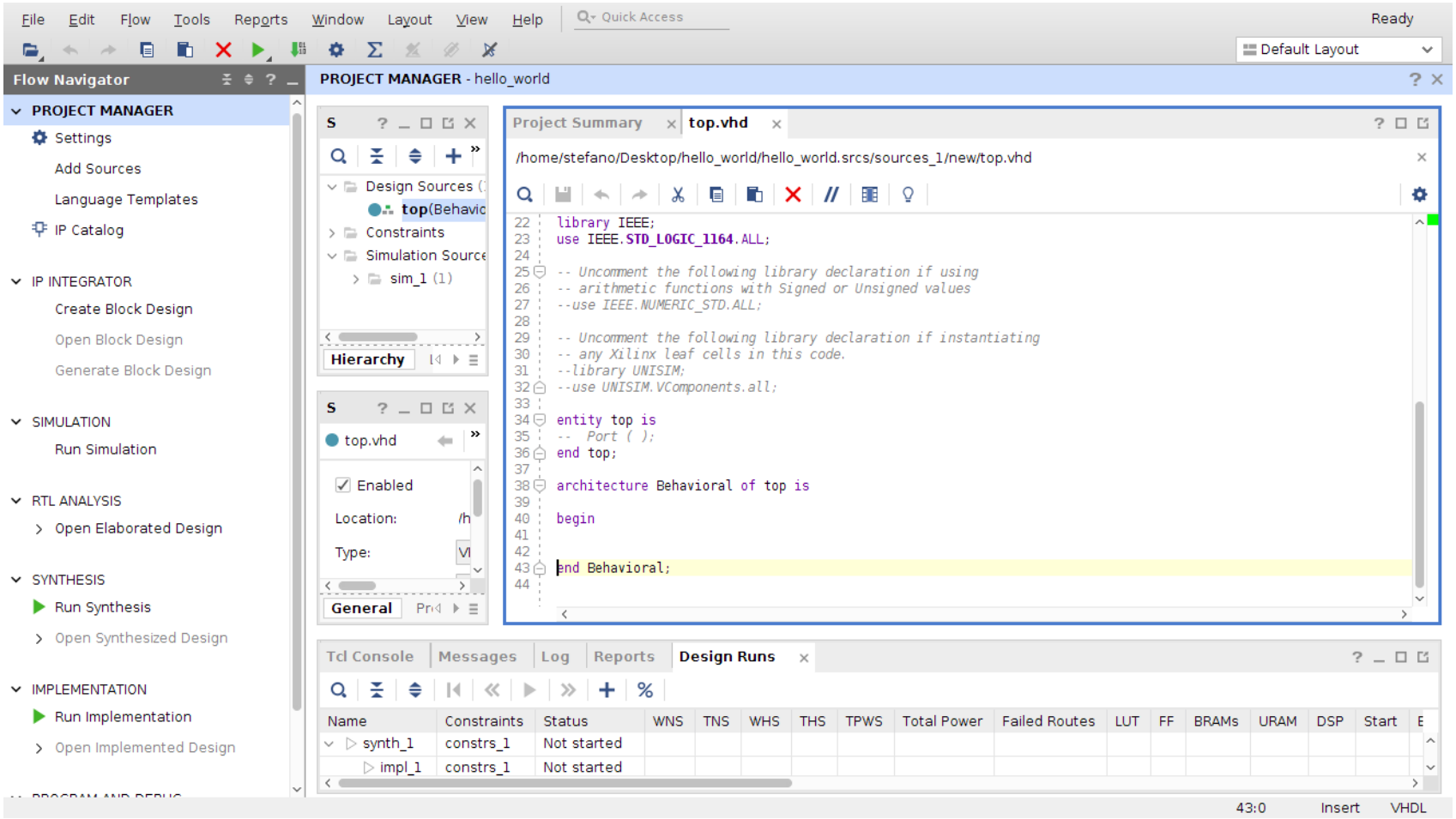
1. Add sources;
2. Check "Add or create design sources";
3. Next $\rightarrow$;
4. Check "Create File";
5. File name: "top";
6. OK $\rightarrow$.
7. Finish $\rightarrow$;
8. OK $\rightarrow$.
9. Yes $\rightarrow$.

Make the source file (1)



Make the source file (2)

Duoble click on top.vhd.



Make the source file (3)

```
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 top is
  Port (btn_in: in std_logic;
        led_out : out std_logic );
end top;

architecture Behavioral of top is

begin

led_out <= btn_in;

end Behavioral;
```

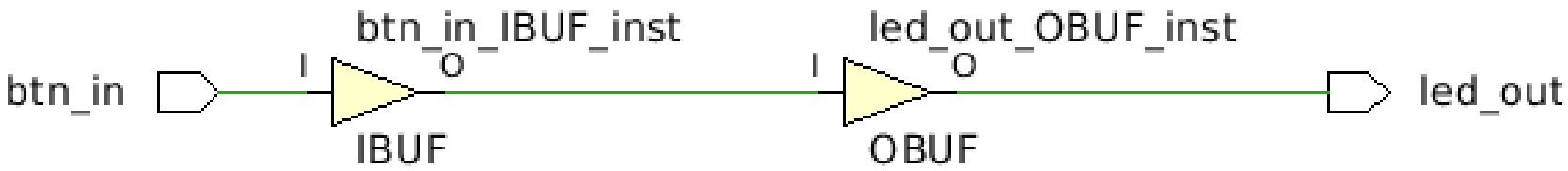
Synthesis (1)

Why we need it?

- To check possible errors in the code.
- To translate the VHDL code in a netlist. A netlist is a list of the logic gates, flip-flops, other components and a list of the connections between them.

Synthesis (2)

1. Run Synthesis →;
2. Cancel →;
3. Open Synthesized Design →;
4. Schematic.



Implementation (1)

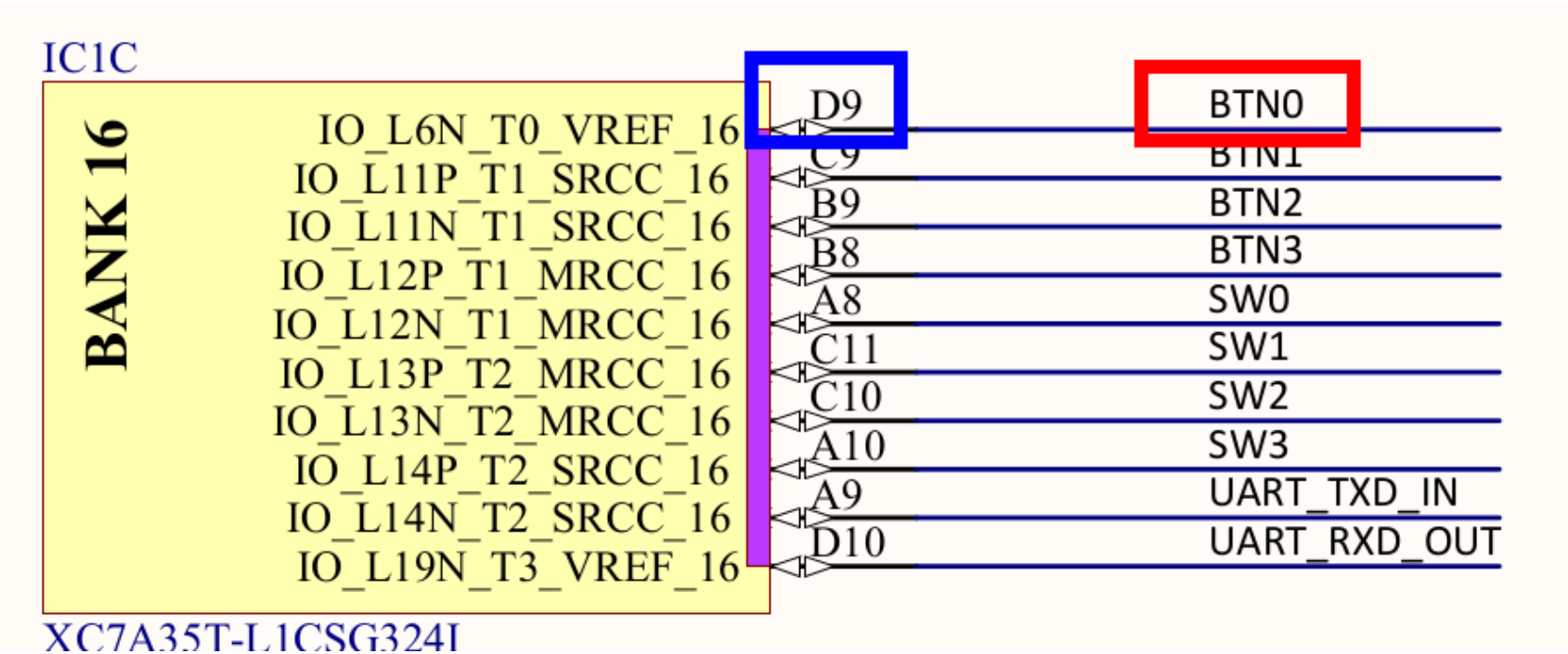
- In order to link the VHDL code to the FPGA pin, it is essential write a constraint file (.xdc);
- therefore we need two files: the schematic of the evaluation board and the xdc file of the FPGA mounted on the board;
- **SCHEMATIC**.
- **XDC**;

In this first example :

- The input (btn_in) has to be connected to the btn0 button.
- The output (led_out) has to be connected to the led0. (In particular to the blue led. $\rightarrow$ LED0_B)

Implementation (2)

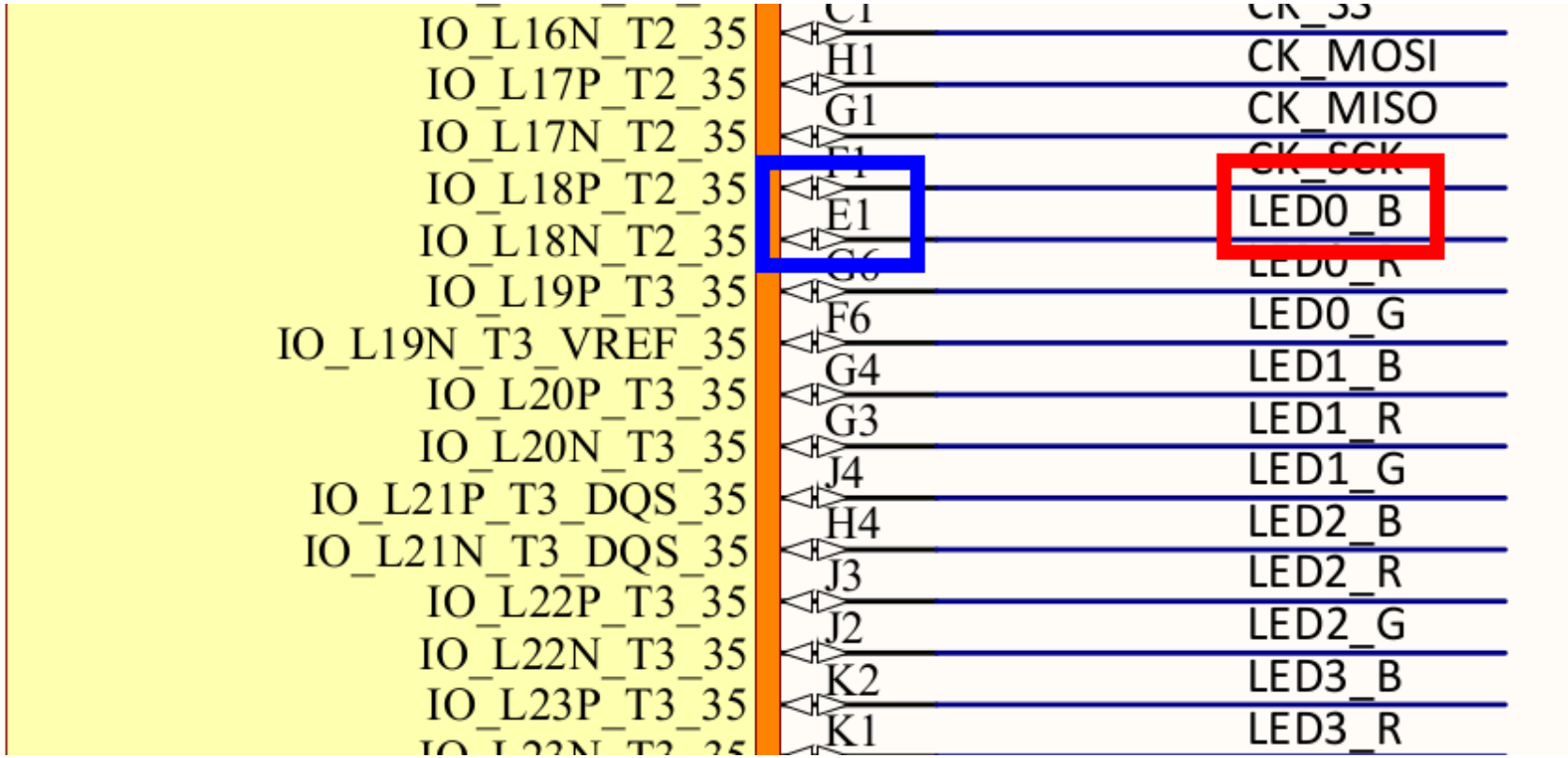
Find in the **schematic** btn0.



It is connected to the FPGA pin **D9**.

Implementation (3)

Find in the **schematic** led0_b.



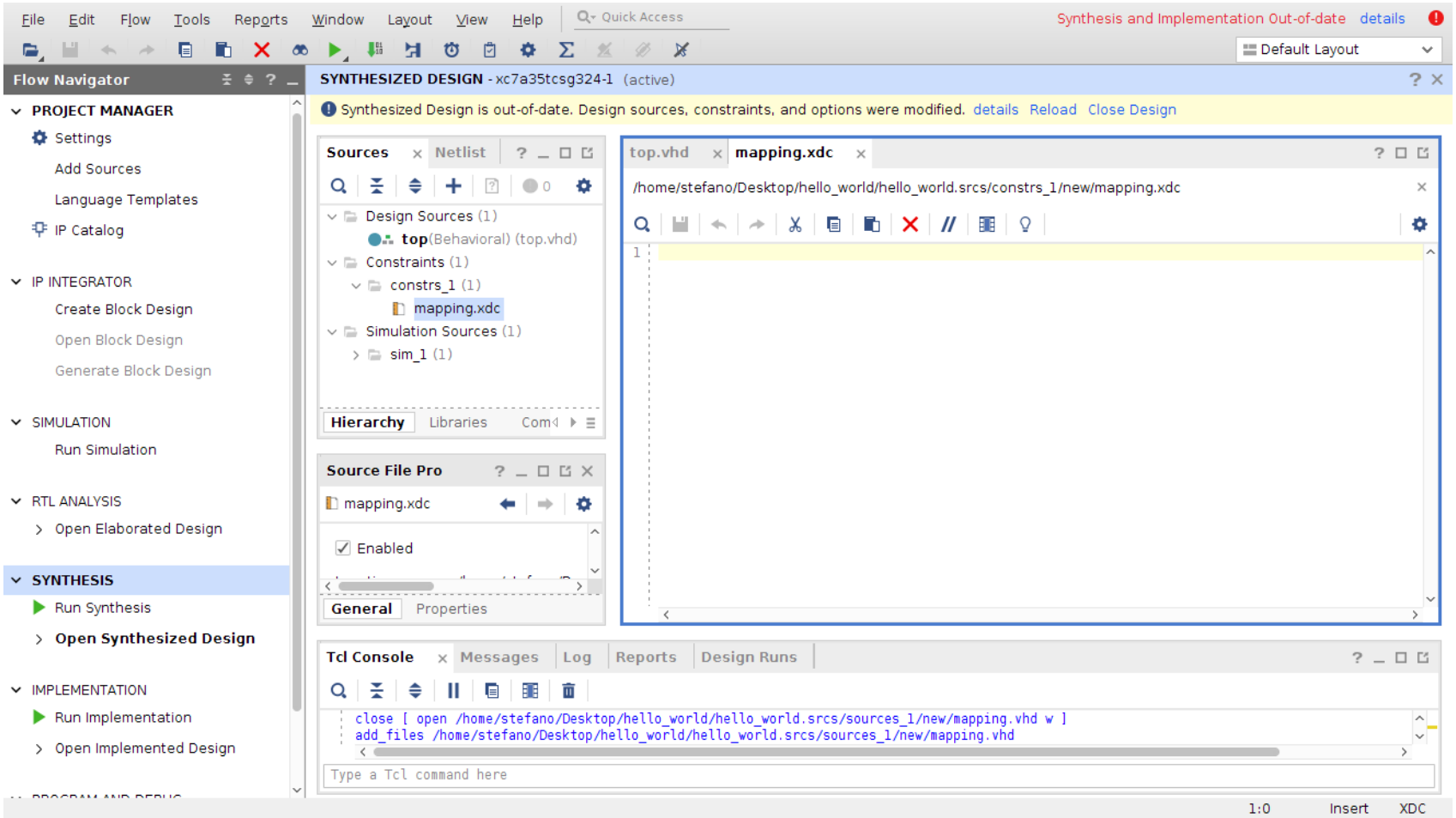
It is connected to the FPGA pin **E1**.

Implementation (4)

1. Add sources;
2. Check "Add or create constraints";
3. Next $\rightarrow$;
4. Check "Create File";
5. File name: "mapping";
6. Finish $\rightarrow$;

Implementation (5)

Open the file mapping.xdc.



Implementation (6)

1. Open the file **XDC** previously downloaded and find **D9** and **E1**;
2. Copy the equivalent lines in the file mapping.xdc;
3. Substitute led_out for led0_b;
4. Substitute btn_in for btn[0];
5. Uncomment the lines (Delete #).

```
set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 } [get_ports { led_out }];

set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { btn_in] }];
```

Implementation (7)

Why we need it?

- To "merge" the netlist and the constraint file, creating a unique design project file.
- To map the components listed in the netlist, in the resources provided by the FPGA.
- To place the resources in the chip and to route them together according to the constraints.

1. Run Implementation →;

Programming (1)

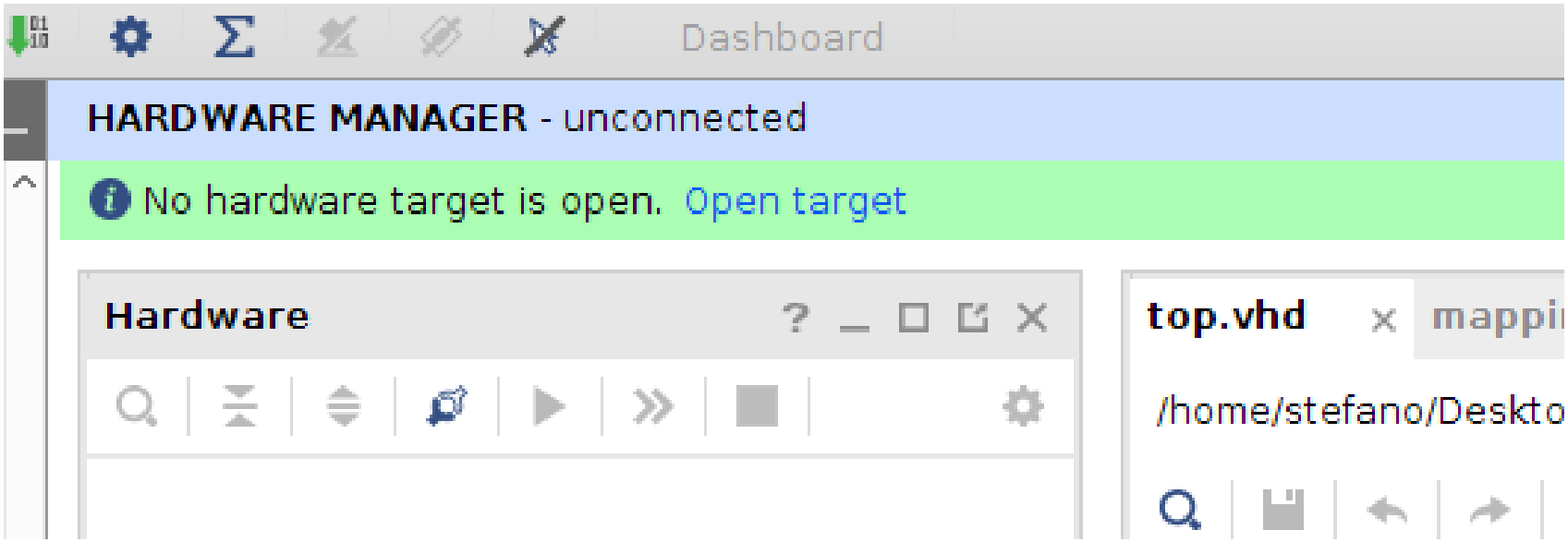
Generate Bitstream →;

Why we need it?

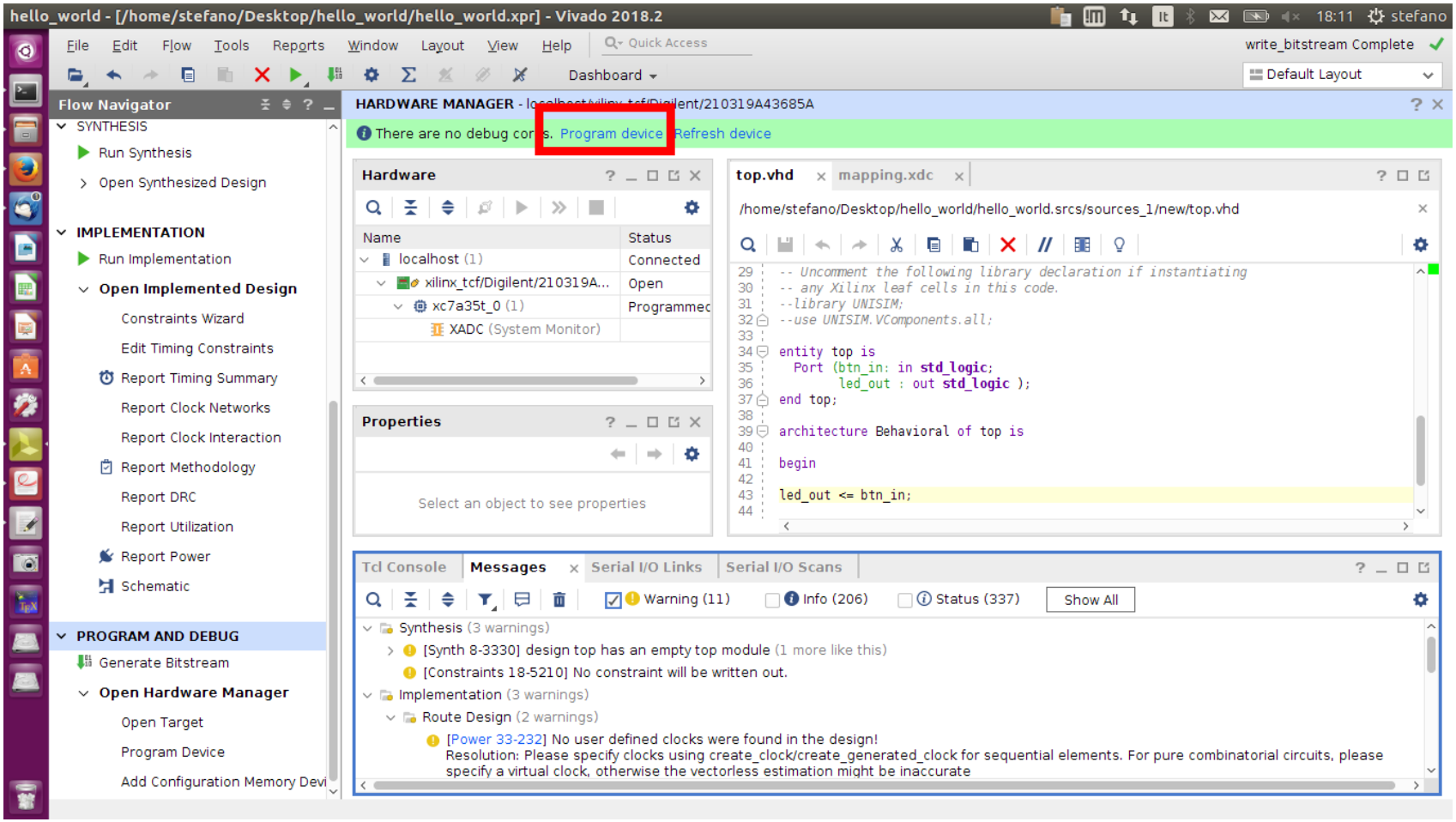
- To generate the bitstream file (.bit);
- this file represent the final configuration to set the FPGA;
- the file is then downloaded into the FPGA.

Programming (2)

1. Connect the evaluation board to PC by the usb cable.
2. Open Hardware Manager $\rightarrow$.
3. Open Target $\rightarrow$.
4. Auto Connect.

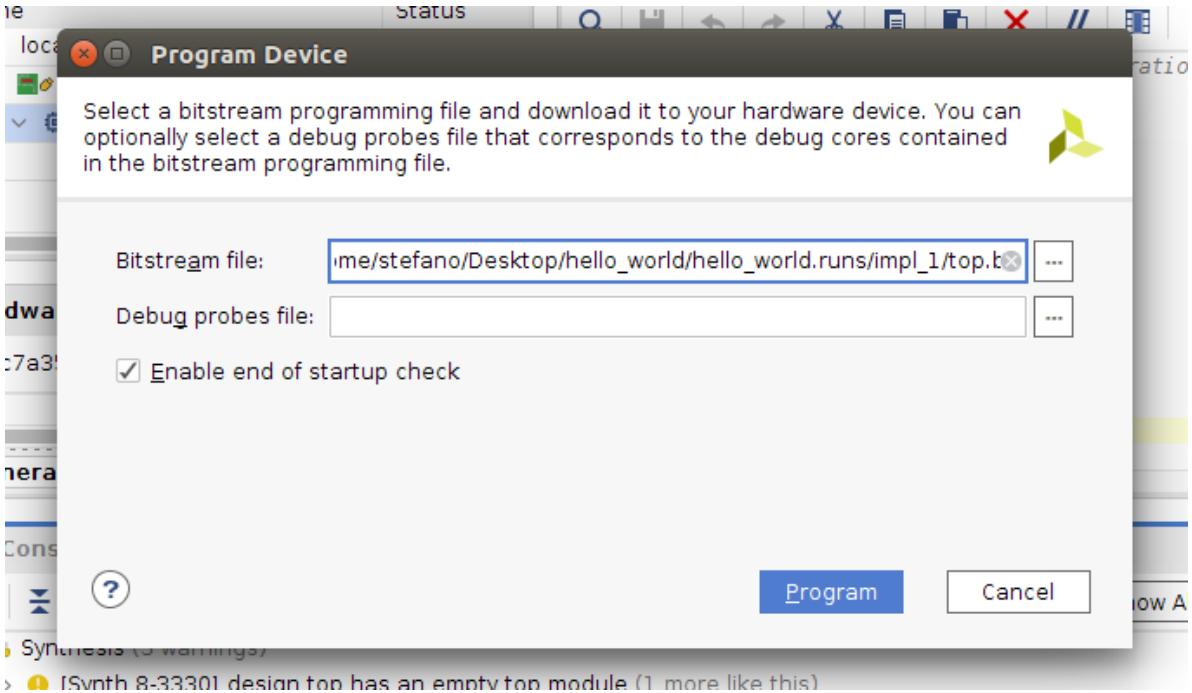


Programming (3)



Programming (4)

Program device →.



(If not checked, check "Enable end of startup check".)
Program device →.

Try to press btn0.

Outline

Suggested exercises

- Redo 1,2,3, ..., N times the exercise "Hello World".
- Get the code more complicated (a little bit). For example instantiate more inputs. Hence repeat each step of "Hello World". An example is reported in the next slide.
- Other complications. More inputs and more outputs.

Hello World with two inputs

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

entity top is
  Port (btn_in: in std_logic_vector(1 downto 0);
        led_out : out std_logic );
end top;

architecture Behavioral of top is

begin

led_out <= btn_in(0) xor btn_in(1);

end Behavioral;
```

```
set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 } [get_ports { led_out }]; #IO_L18N_T2_35 Sch=led0_b

set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { btn_in[0]}]; #IO_L6N_T0_VREF_16 Sch=btn[0]
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { btn_in[1]}]; #IO_L12N_T1_MRCC_16 Sch=sw[0]
```