Laboratory Exercises

Michael Psarakis

DEPARTMENT OF INFORMATICS - UNIVERSITY OF PIRAEUS

COURSE: "COMPUTER SYSTEMS DESIGN"

Introduction

The laboratory exercises of the course "Computer Systems Design" will help you learn the Xilinx Vivado tool for design entry, simulation, synthesis of digital circuits and their implementation on FPGA (Field Programmable Gate Arrays) training platforms. The objectives of the workshop are:

- become familiar with an integrated environment for designing, simulating and synthesizing circuits (note: you will also use this environment to implement the coursework).
- learn through a series of laboratory exercises of increasing complexity the design of computer systems using the VHDL hardware description language.
- get in touch with a hardware training and development platform and using programmable logic technology to implement the circuits on Field Programmable Gate Arrays (FPGAs).

The laboratory exercises are divided into the following sections:

- Introduction to the tool: You will become familiar with using the Xilinx Vivado tool by designing and simulating elementary circuits.
- Familiarization with the use of the FPGA training board: you will use a training platform (Basys 3 Artix-7 FPGA board) and get familiar with its use by implementing some elementary circuits.
- Implementation of simple circuits: you will deal with the design, simulation and implementation on the FPGA board of simple combinational (e.g. multiplexers, decoders, adders, etc.) and sequential circuits (e.g. counters, slip registers, etc.).
- Implementation of complex circuits: you will deal with the design, simulation and implementation on the FPGA board of more complex circuits (e.g. finite state machines, memories, etc.)

The Xilinx Vivado tool supports several hardware description languages. The lab exercises focus on the VHDL hardware description language.

Note: This brochure is for the 18.3 version of the Xilinx Vivado 18.3 tool.

Lab Exercise 1: Introduction to the tool

In this lab exercise you will become familiar with the Xilinx Vivado environment. In particular, you will learn:

- How to design circuits using a hardware description language (e.g. VHDL)
- How to simulate the circuit (functional simulation) using the Vivado simulator.

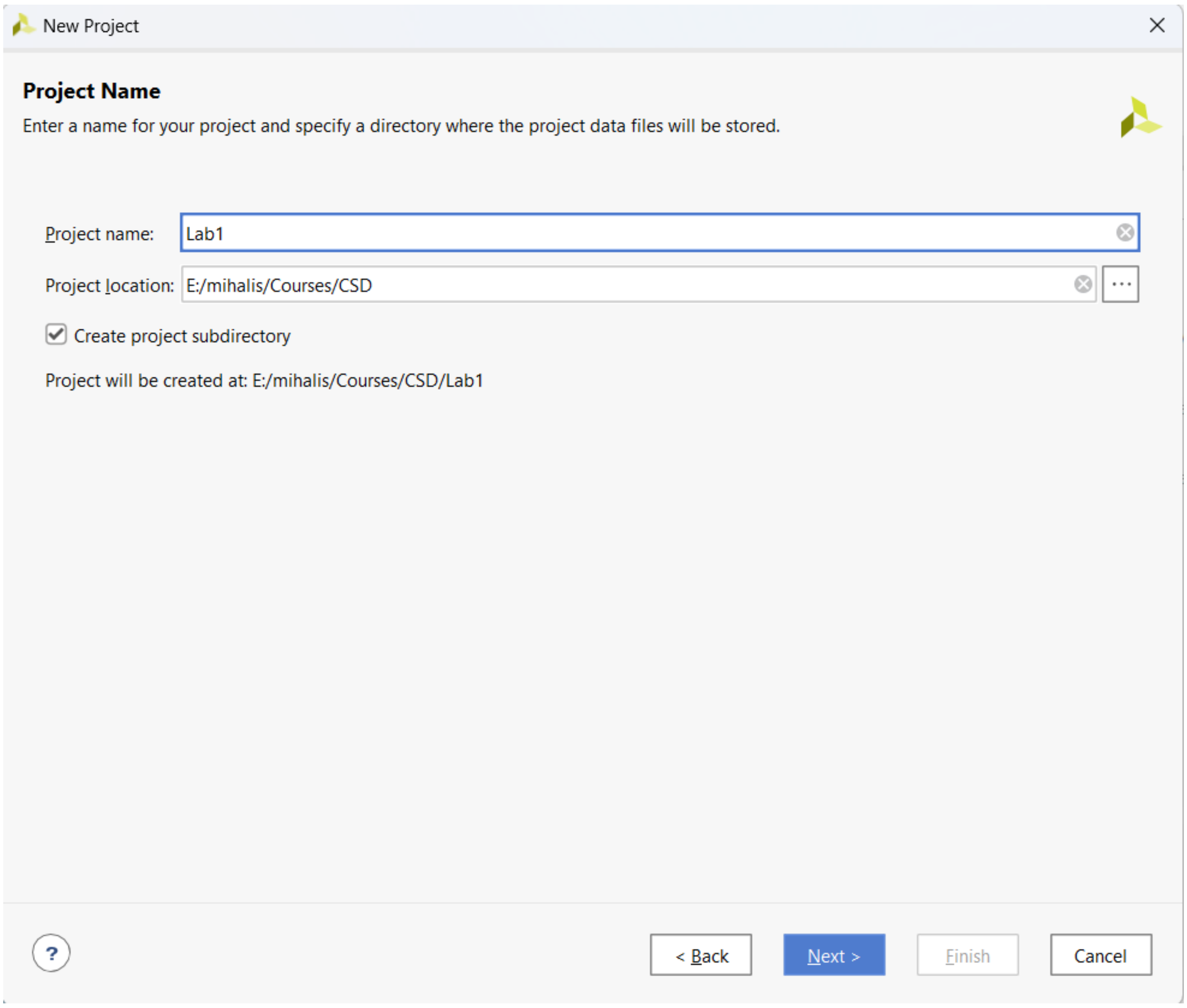
Exercise 1: 4-bit parity generator (with VHDL)

The circuit you will implement in this lab exercise is a simple circuit to calculate the even parity of a 4-bit message.

1.1 Create a new project (new project)

Select All Programs→Xilinx Design Tools→Vivado 2018.3→Vivado 2018.3. Select Create New Project. The first New Project dialog box will appear, as shown in Image from 1.

The dialog box prompts you to enter the name and directory of the project, as shown in the Image from 1. Once you have filled in the details, click Next.



New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: Lab1

Project location: E:/mihalis/Courses/CSD

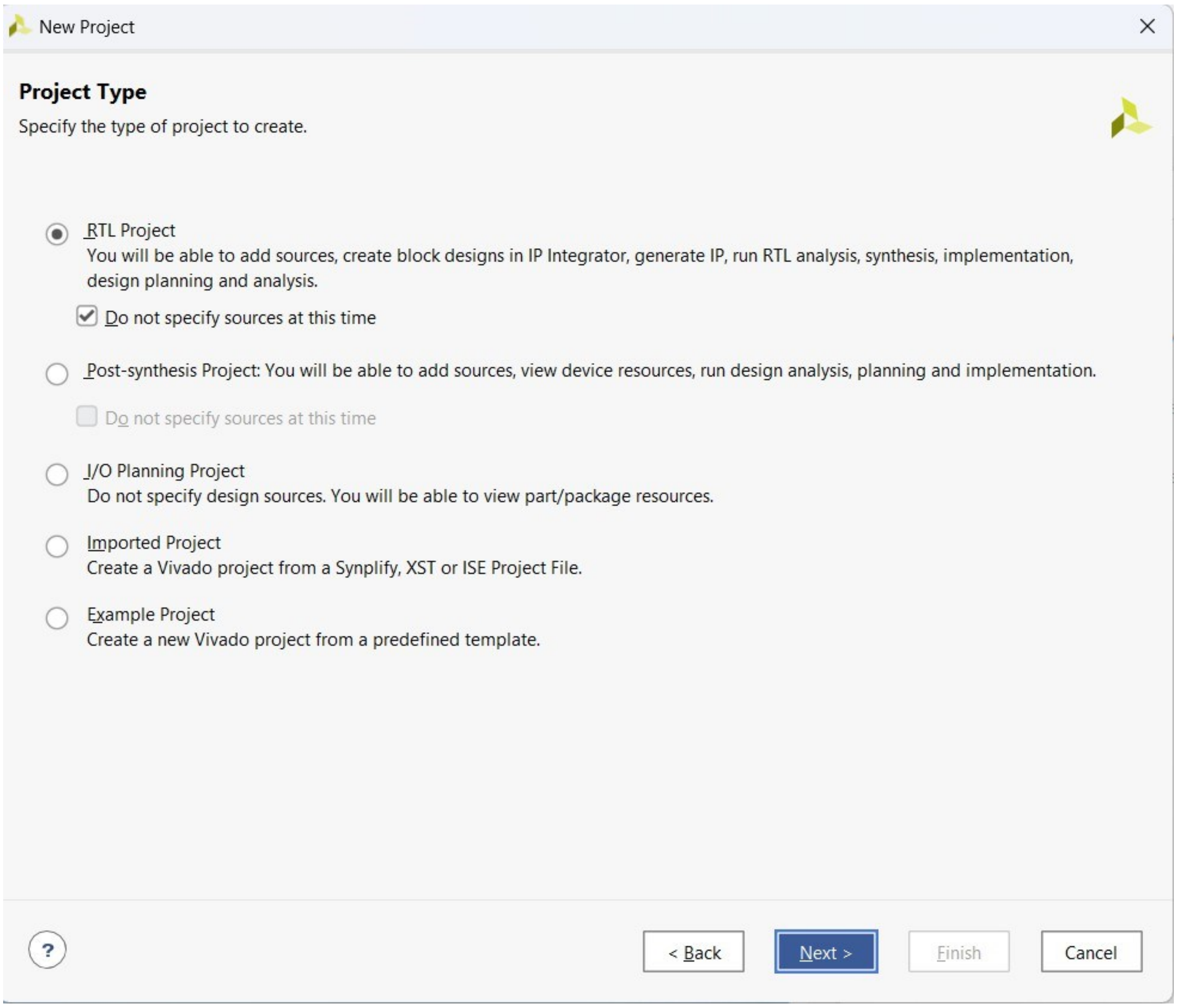
☑ Create project subdirectory

Project will be created at: E:/mihalis/Courses/CSD/Lab1

Image from 1: New Project - Project name (1 of 4)

Note: Do not use file or folder names that contain spaces and folder names in Greek.

The next dialog box allows you to select the type of project. Select RTL Project type and enable the Do not specify sources at this time option, as shown in Image from 3, and click Next.


New Project

Project Type

Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
[✓] Do not specify sources at this time

( ) Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.
[ ] Do not specify sources at this time

( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

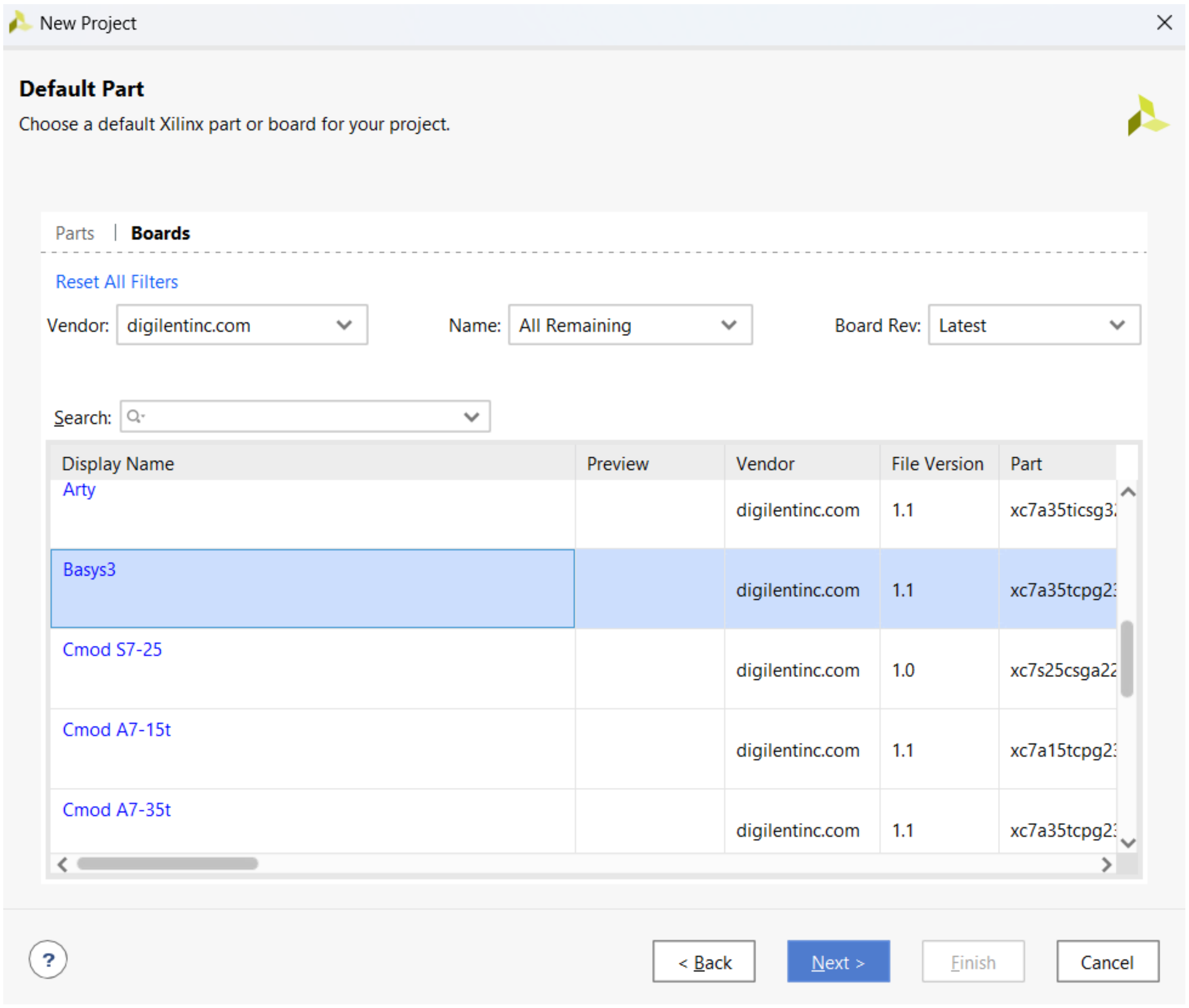
( ) Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Example Project
Create a new Vivado project from a predefined template.

Image from 2: New Project - Project type (2 of 4)

The next dialog box allows you to specify the type of FPGA device to use. Select Boards and in the Filter/Preview field select Vendor: digilentinc.com. Then, select Display Name: Basys3, as in Image from 3. After selecting the device type, click Next.

Note: If the digilentinc.com vendor boards are not available, select Parts and then in the Filter field select Family: Artix-7, in the Package field: cpg236, and in the Speed grade field: -1. Finally, select the device xc7a35tcpg236 -1.


New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | **Boards**

Reset All Filters

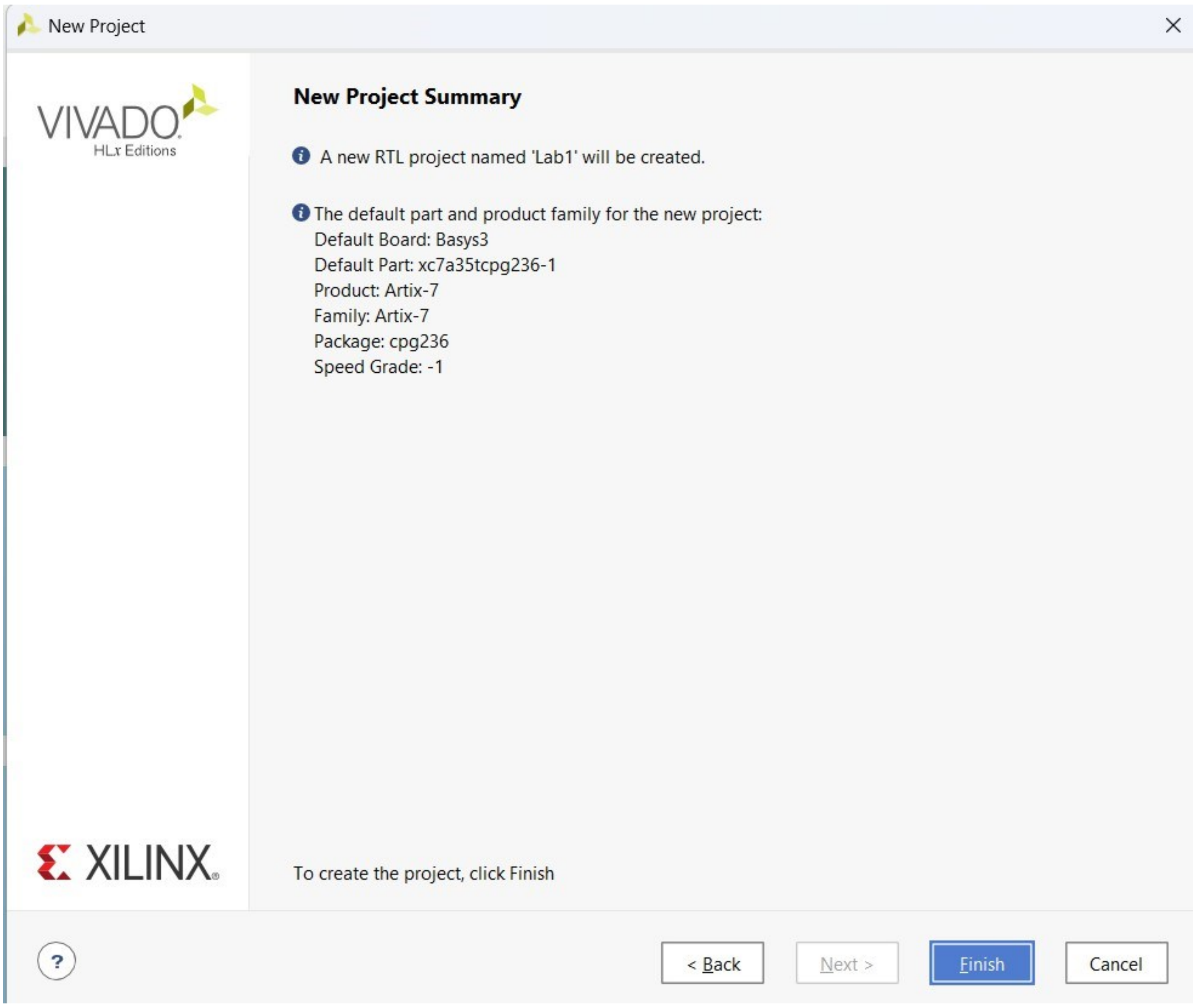
Vendor: digilentinc.com   Name: All Remaining   Board Rev: Latest

Search:

Display Name	Preview	Vendor	File Version	Part
Arty		digilentinc.com	1.1	xc7a35ticsg3:
Basys3		digilentinc.com	1.1	xc7a35tcpg2:
Cmod S7-25		digilentinc.com	1.0	xc7s25csga22
Cmod A7-15t		digilentinc.com	1.1	xc7a15tcpg2:
Cmod A7-35t		digilentinc.com	1.1	xc7a35tcpg2:

Image from 3: New Project - Default Part (3 of 4) : Zybo board selection

The final dialogue framework in the process of creating a new project, shown in Image from 4, provides a summary of the project that Vivado will create based on your settings. Check the summary to make sure it matches what is shown in the Image from 4. If not, click Back to correct any errors. Otherwise, click Finish to complete the process.



New Project Summary

A new RTL project named 'Lab1' will be created.

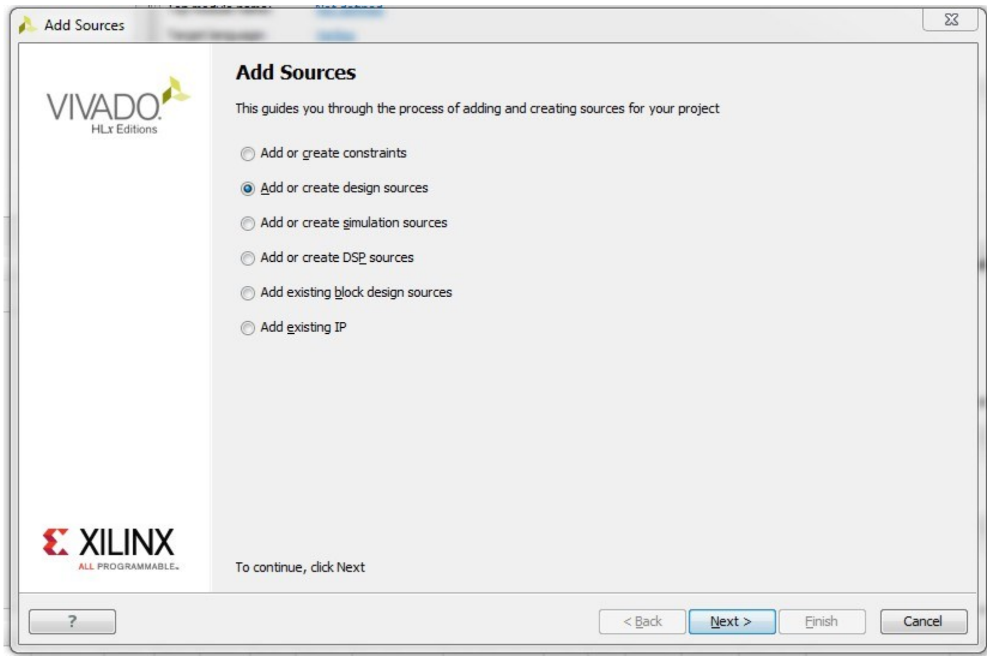
The default part and product family for the new project:
Default Board: Basys3
Default Part: xc7a35tcpg236-1
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1

To create the project, click Finish

Image from 4: New Project - Summary (4 of 4)

1.2 Design introduction

At this point, the created project does not contain any source files. Create a new source file for the design of the even parity calculation circuit. In the Project Manager window, select Add Sources. In the next steps you will need to specify the source file. The first of the new dialog boxes asks you to specify the type of file, as shown in Image from 5. Select Add or create design sources.


Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

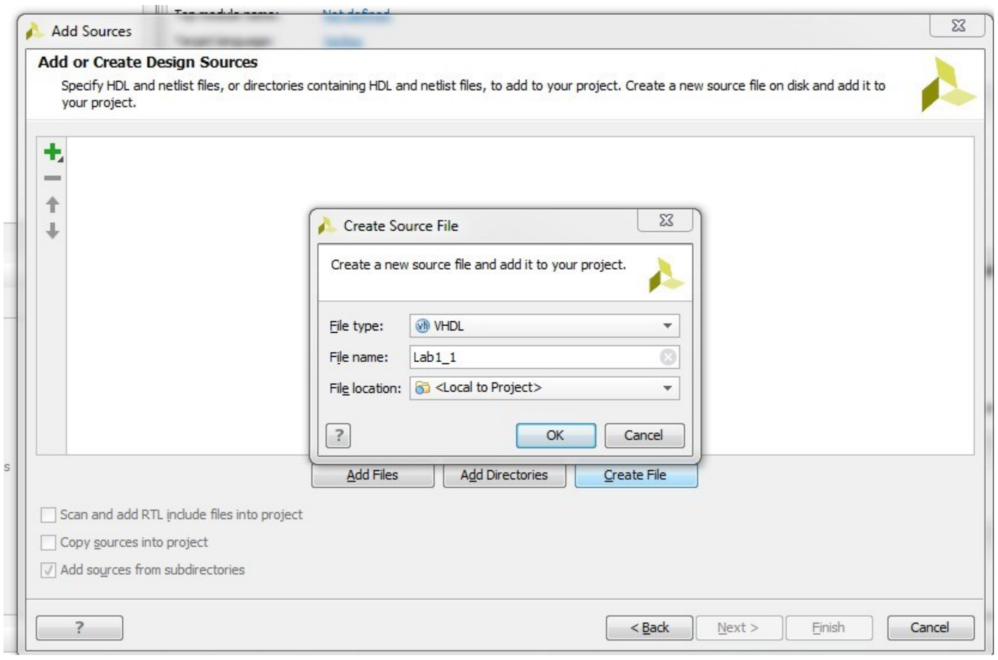
- ( ) Add or create constraints
- (●) Add or create design sources
- ( ) Add or create simulation sources
- ( ) Add or create DSP sources
- ( ) Add existing block design sources
- ( ) Add existing IP

To continue, click Next

Image from 5: Add Sources (1 of 3)

You can then choose to add a new file (Add Files), a new directory containing one or more source files (Add Directories), or create a new file (Create File). Select Create File.

The next dialog box asks you to specify the type and name of the new file. Select VHDL type and name it Lab1_1, as in Image from 6.


Add Sources

Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

Create Source File

Create a new source file and add it to your project.

File type: VHDL

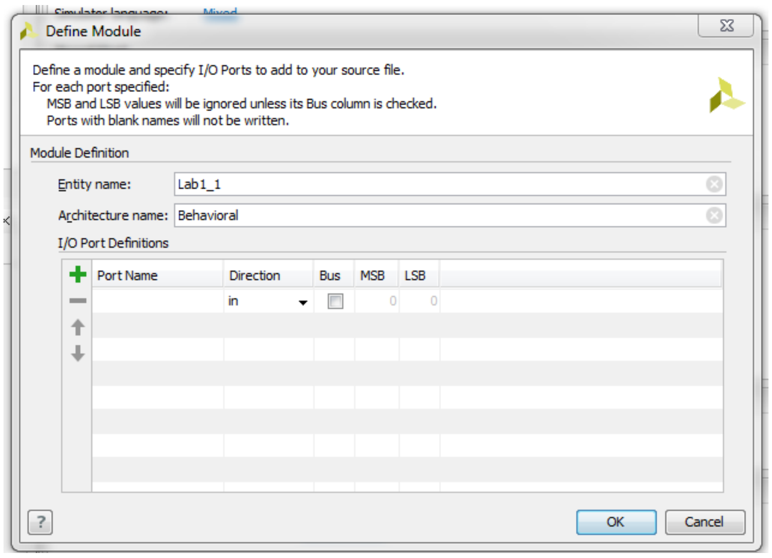
File name: Lab1_1

File location: <Local to Project>

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [✓] Add sources from subdirectories

Image from 6: Add Sources (2 of 3)

The next window allows you (optionally) to specify the ports of the drive. This can also be done in the text editor when editing the module, so ignore it at this stage. Just confirm that the settings match those shown in the Image from 7 and click Next.


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name: Lab1_1

Architecture name: Behavioral

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	in	☐	0	0

Image from 7: Add Sources (3 of 3)

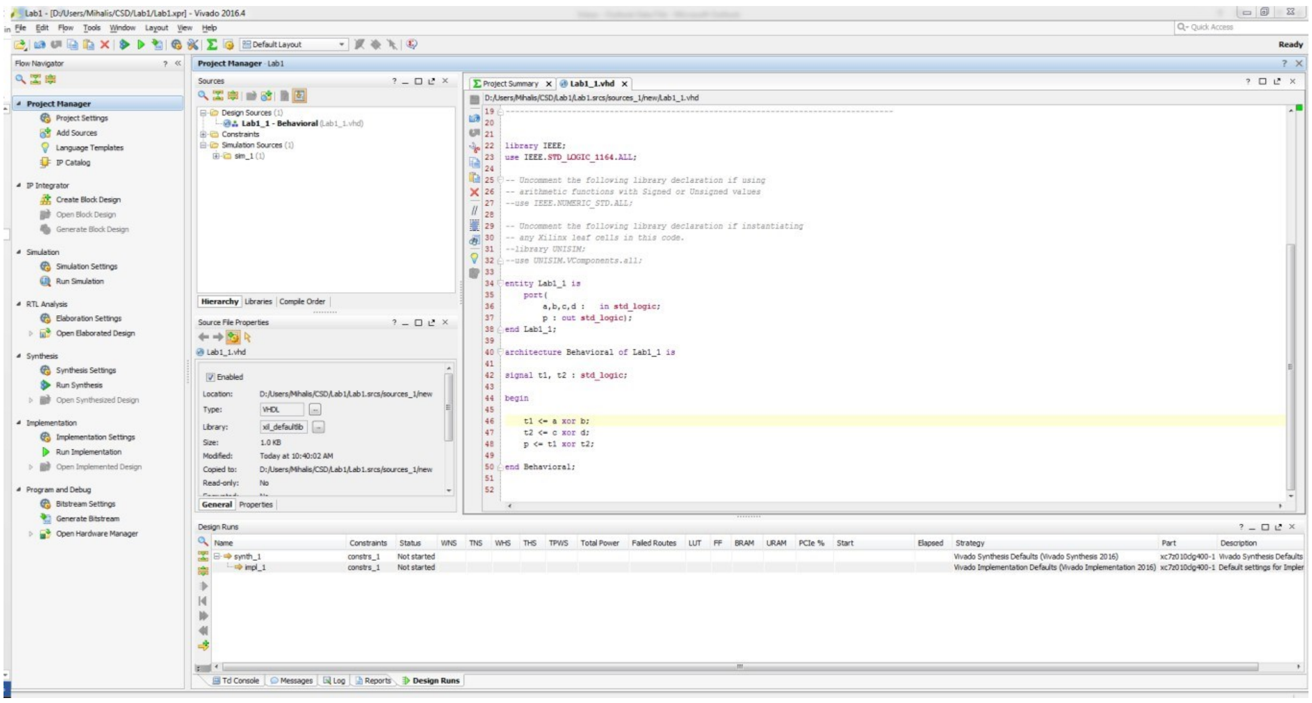
In the text editor, some of the basic VHDL file structures are already written. Language keywords are shown in blue, data types in red, comments in grey, and values in black. This color coding enhances the readability of the VHDL file and the detection of typos. Now, enter the description of the even parity calculation circuit.

Note: You can download the following model (Lab1_1.vhd) from the course website (contained in Lab1.zip).

```
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 primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Lab1_1 is
   port( ,b,c,d : in std_logic;
          p :    out std_logic);;
end Lab1_1;

architecture Behavioral of Lab1_1 is

signal t1, t2 : std_logic;

begin

   t1 <= a xor b;
   t2 <= c xor d;
   p <= t1 xor t2;
```

```
end Behavioral;
```

At this point, you should end up with a window that looks like the one shown in Image from 8. Once you are done, save the file and close the window. There are options in the main menu to save either individual files or the entire project.



```
---------------------------------------------------------------------------------


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 Lab1_1 is
    port(
        a,b,c,d :   in std_logic;
        p : out std_logic);
end Lab1_1;

architecture Behavioral of Lab1_1 is

signal t1, t2 : std_logic;

begin

    t1 <= a xor b;
    t2 <= c xor d;
    p <= t1 xor t2;

end Behavioral;
```

Lab1 - [D:/Users/Mihalis/CSD/Lab1/Lab1.xpr] - Vivado 2016.4
Flow Navigator
Project Manager
Sources
Project Summary
Lab1_1.vhd
D:/Users/Mihalis/CSD/Lab1/Lab1.srcs/sources_1/new/Lab1_1.vhd
Design Runs
synth_1
impl_1
Not started

Image from 8: Integrated design

1.3 Simulation (Simulation)

You can download the model (Lab1_bench.vhd) from the course website (contained in Lab1.zip).

1.4 Circuit Analysis (RTL Analysis)

In the RTL Analysis window, select Open Elaborated Design. Observe the schematic diagram of the parity circuit (consisting of 3 2-input XOR gates).

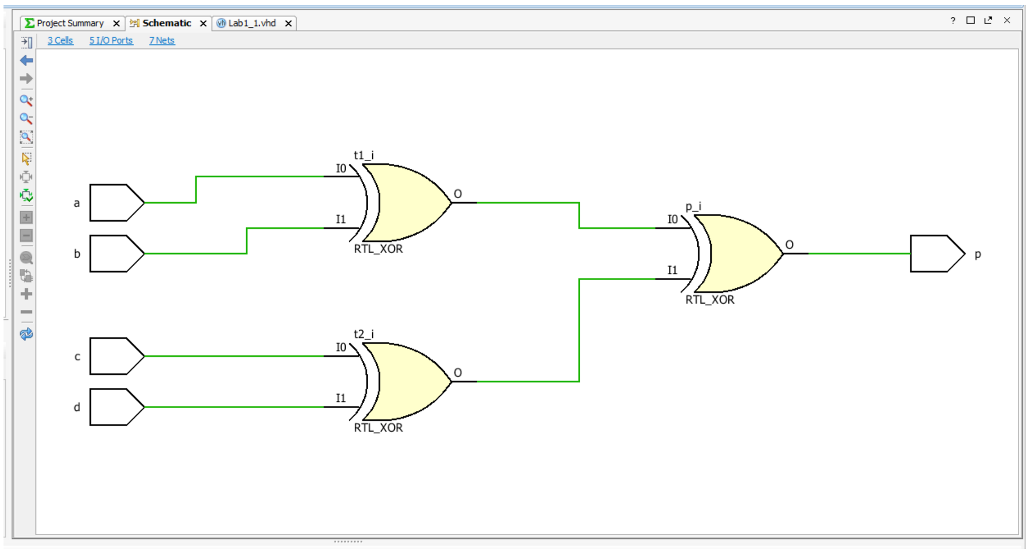


Image from 9: Schematic diagram

Lab Exercise 2: Programming the FPGA device

In this lab exercise you will get familiar with the use and programming of the FPGA training board. More specifically, you will learn:

- The features of the training and development board (Digilent Basys-3 FPGA board) that you will use in the lab
- How to create an FPGA programming file
- How to program an FPGA device

Exercise 1: Implementing the 4-bit parity generator circuit on the FPGA board

You will implement the circuit that calculates the even parity of a 4-bit message (designed in Lab Exercise 1) on the FPGA board. So open the Lab 1 project (Lab1 project).

In lab exercise 1 you worked on the steps: design introduction, simulation and circuit analysis. The next steps to program the FPGA device are: synthesis, implementation, and FPGA programming and debug.

1.1 Educational board Basys-3

But first, get familiar with the FPGA board you will be using. You can find a user manual for the board on the lab's website.

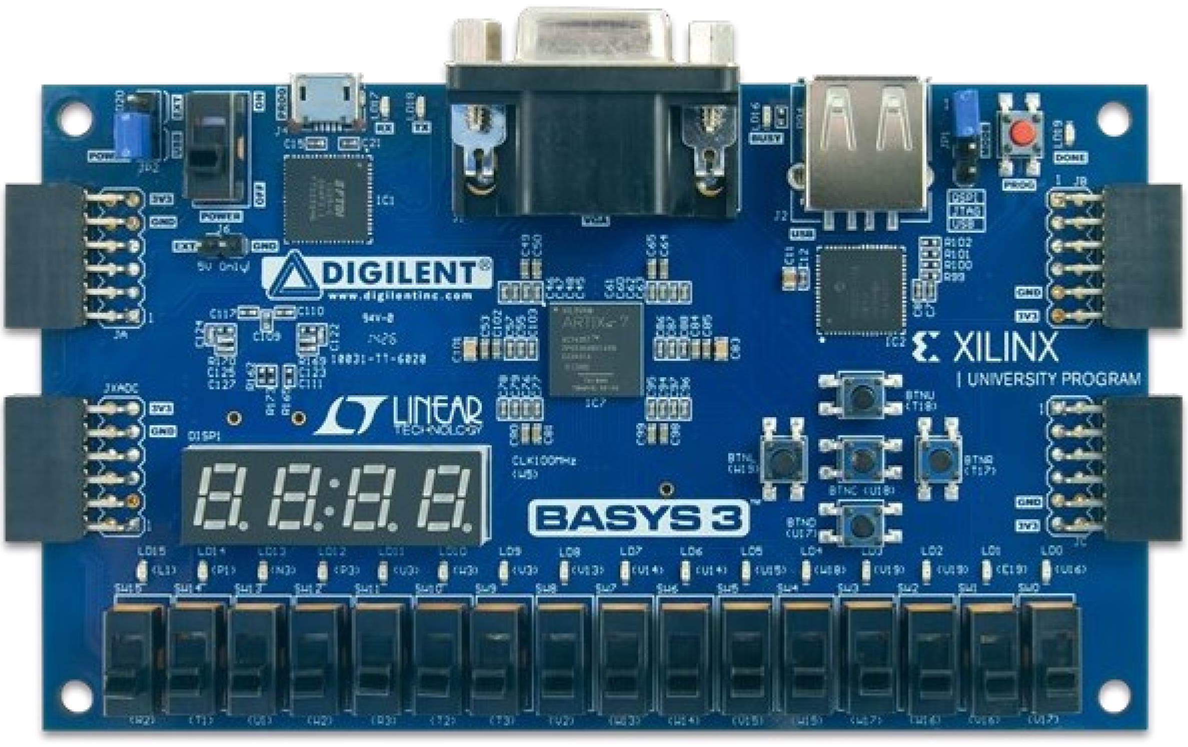


Image from 10 Basys-3 board

The Basys-3 board contains the following (in the context of the lab you will use what is in bold):

- **Artix-7 35T FPGA**

- **16 user switches**
- **16 user LEDs**
- **5 user pushbuttons**
- **4-digit 7-segment** display
- Three Pmod ports
- Pmod for XADC signals
- 12-bit VGA output
- USB-UART Bridge
- serial flash
- **Digilent USB-JTAG port for FPGA programming and communication**
- USB HID Host for mice, keyboards and memory sticks
- And other elements that will not be used in the workshop

1.2 Design synthesis (design synthesis)

After simulating and verifying that your circuit works correctly, the next step is to use a synthesis tool to transform your description into a netlist. A netlist is a schematic representation that can be read by automatic tools.

In the Synthesis window, select Run Synthesis.

When the synthesis process is complete select View Report. In the Reports window select Utilization Report and view the FPGA device resources used by your circuit. How many Slice LUTs, how many Slice Registers, how many Memory blocks (Block RAMs), how many DSPs, how many IOBs (Input-Output Blocks)?

Then in the Synthesis window select Schematic and see the schematic diagram of your circuit using the programmable resources of the FPGA. Select LUT4 and view the Truth Table.

Close Synthesized Design.

1.3 Design implementation (design implementation)

The design implementation is the sequence of events that translates the synthesized design netlist into a programming file for the FPGA device. The description of the circuit you have now synthesized has a number of ports at the top level. The implementation tools need to know how to assign the ports at the top level of your design to the physical pins of the FPGA, which are connected to various resources on the board. If you don't specify explicit assignments, the tools will randomly assign the pins for you. Obviously, this is a bad idea since the random assignments will be incorrect.

The highest level of the example design has 4 input ports (a, b, c, d), and one output port (p). So, we want to **have 4 switches, SW0, SW1, SW2 and SW3**, connected to the inputs. In addition, we want the output to be connected to an indicator light (LED) so that we can observe it - **LD0** is suitable for this purpose.

If you inspect the top side of your board, you will notice that almost every port is annotated with some text that identifies which pins of the FPGA it is connected to. This information is also available in the board's User Guide. Try to determine on your board which FPGA pins are used for SW0, SW1, SW2, SW3, and LD0, and then check your results against those shown below:

SW0 → FPGA Pin V17

SW1 → FPGA Pin V16

SW2 → FPGA Pin W16

SW3 → FPGA Pin W17

LD0 → FPGA Pin U16

You now have enough information to create what is called a user constraint **file, or UCF**. This file contains the design constraints that you did not specify in the VHDL description, such as pin location and design performance constraints. It is convenient to provide them in a UCF rather than in the VHDL description. For example, if you make a mistake in pin assignments, you don't have to go back and reassemble your circuit.

Download from gunet the XDC constraint file of your board (Basys-3-Master.xdc).

You can add a UCF to the project using the same process you used to add the drawing. In the Project Manager window select Add Sources → Add or create constraints. In the next window select Add Files and select the Basys-3-Master.xdc file.

Open the xdc file and uncomment the lines that refer to the 4 switches you want to use and the one LED. Give the corresponding signals the input and output names of your circuit, i.e. a instead of sw[0], b instead of sw[1], c instead of sw[2], d instead of sw[3], p instead of led[0],

Now that you have a constraints file in your project, you can implement the design. In the Implementation window, select Run Implementation. When the implementation process is complete, select View Report.

1.4 FPGA programming (Program and Debug)

At this point, you are ready to program the FPGA with your design.

In the Program and Debug window, select Generate Bitstream. When the process is complete, select View Reports.

Connect your board via the USB cable. Power off the board. Select Open Hardware Manager. Select Open Target. select Program Device.

Now, you can test your design on the material. Locate switches SW0-SW3 on the board, and examine your circuit by testing the 16 possible combinations of switch values and observing LD0. Does your circuit behave as you expect? If not, ask the teacher for help. If it works correctly, you have successfully completed the exercise.

Lab Exercise 3: Combinational circuits

In this lab exercise you will work on the design of combinational circuits. More specifically, you will design:

- A 3-input majority circuit
- A 4-bit adder
- A 3-in-8 decoder
- A 4-bit two-number comparator
- A binary to BCD number converter

Exercise 1: 3-input majority circuit

Design a 3-input majority circuit using switches SW2-SW0 as inputs and LED0 as output.

- Create a new project.
- Create a new file (vhdl module) named lab3_majority.vhd.
- The entity of the unit is as follows:

```
entity lab3_majority is
   port (
        a, b, c : in STD_LOGIC;
        f : out STD_LOGIC
   );
end lab3_majority;
```

- To calculate the logistic function of the output f, use a truth table and Karnaugh map.
- Create a testbench for the project and enter all possible test vectors of the circuit.
- Simulate the circuit.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and check the operation of the circuit.

Exercise 2: 4-bit adder

Design an adder that adds two 4-bit unsigned numbers A=A3 A2 A1 A0 and B= B3 B2 B2 B1 B0 and produces a 4-bit sum of S= S2 S2 S2 S1 S0 and a carry output C. Use switches SW7-SW0 for inputs A and B and LEDs LED4-LED0 for outputs S and C.

- Create a new project.
- Create a new file (vhdl module) named lab3_adder.vhd.
- The entity of the unit is as follows:

```
entity lab3_adder is
   port (
          A,B   :UNSIGNED(3DOWNTO 0);
          S     : UNSIGNED(3 DOWNTO 0);;
          C     STD_LOGIC
     );
   end lab3_adder;
```

- Add the following library to the file (contains functions for the UNSIGNED data type):

  ```
  USE ieee.numeric_std.ALL;
  ```

- Create a testbench for the project and enter various test scenarios of the circuit.
- Simulate the circuit.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and check the operation of the circuit.

Exercise 3: 3-in-8 decoder

Design a 3-in-8 decoder (3x8 decoder) using switches SW2-SW0 as inputs and LED7-LED0 as outputs.

Note: An n-in-2^n decoder for each input combination turns on (sets to active value, e.g. 1) one output and turns off (sets to 0) all the others.

- Create a new project.
- Create a new file (vhdl module) named lab3_decoder_3x8.vhd.
- The entity of the unit is as follows:

```
    entity lab3_decoder_3x8 is
       port (
            d : in STD_LOGIC_VECTOR (2 downto 0);
            q : out STD_LOGIC_VECTOR (7 downto 0)
       );
   end lab3_decoder_3x8;
```

- Add the following libraries to the file (they contain functions for the std_logic data type):

  ```
  use IEEE.STD_LOGIC_ARITH.ALL;
  use IEEE.STD_LOGIC_UNSIGNED.ALL;
  ```

- Write the decoder design using either the **with-select** assignment command or the **when-else** assignment command.
- Insert the testbench lab3_decoder_3x8_tb.vhd into the project. Note: You can download the testbench from the course website (contained in Lab3.zip).
- Simulate the circuit.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and check the operation of the circuit.

Exercise 4: Binary number comparator

Design a comparator of 2 4-bit unsigned binary numbers. The circuit will compare the binary numbers A=A3 A2 A1 A0 (for input A use switches SW7, SW5, SW5, SW4) and B=B3 B2 B2 B1 B0 (for input B use switches SW3, SW2, SW1, SW0) and produce 3 outputs.

- Draw the circuit using the **when-else** command.
- Compose and implement the circuit by importing the appropriate ucf file.
- Program the board and check the operation of the circuit.

Exercise 5: Binary to BCD converter

Design a circuit that converts a 4-bit binary number B = B3B2B1B0 to the equivalent decimal number (BCD code) with 2 digits D = D1D0. The table below illustrates the conversion:

B3B2B1B0	D1 D0
0000	0 0
0001	0 1
0010	0 2
...	...
1001	0 9
1010	1 0
1011	1 1
1100	1 2
1101	1 3
1110	1 4
1111	1 5

- Draw the circuit.
- Compose and implement the circuit by importing the appropriate ucf file. For the 4 inputs use switches SW3-SW0 and for the 2 decimal digits use the leds on the board, e.g. D1 (LED7-LED4) and D0 (LED3-LED0)
- Program the board and check the operation of the circuit

Note: To design the circuit, do not use complex with-select, when-else, case, if-then-else commands. The purpose of the exercise is to implement the circuit using simple logic functions. You can use only logical (Boolean) operators, a 4-bit adder and a 2x1 multiplier (to implement the multiplier use a when-else instruction).

To design the circuit, rely on the following observation. When input B is less than 10 then output D1 is 0 and output D0 is equal to B. When input B is greater than 9 then output D1 is 1 and output D0 is equal to B+6 (ignore the quotient).

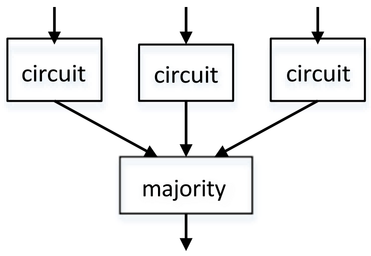
Lab Exercise 4: Structure Description

In this lab exercise you will work on designing combinational circuits using structural description. More specifically, you will design:

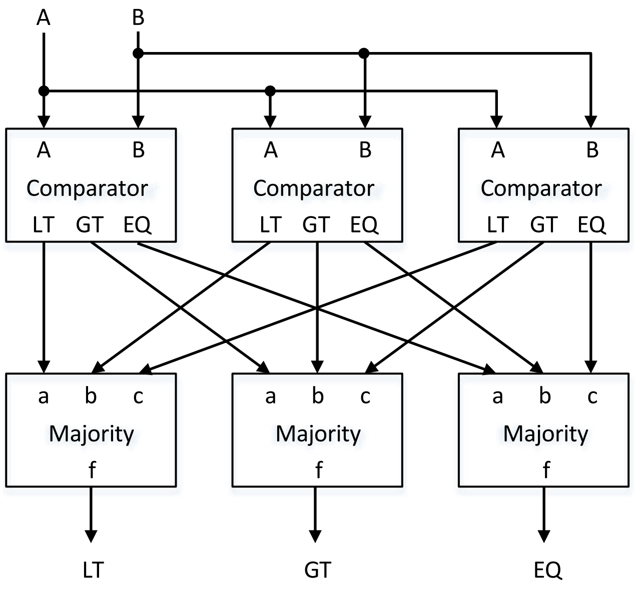
- The 2-bit 2-number comparator of Exercise 3.4 using the TMR (Triple Modular Redundancy) technique

Exercise 1: Binary number comparator using the TMR technique

The TMR (Triple Modular Redundancy) technique is used to design reliable systems and is based on tripling the circuit under design and using a 3-input majority voter circuit. The basic principle of TMR is illustrated in the following Figure. If one of the 3 copies of the circuit has an error then the majority circuit selects the correct output.



- Create a new project and insert the vhdl model (comparator) you implemented in Exercise 3.4.
- Insert a new vhdl unit (comparator_TMR) that has the same inputs & outputs as the comparator (of Exercise 3.4). Design the 2-bit two-number comparator using the TMR technique. Use a structural description where you insert 3 copies (instances) of the comparator and for each output a majority circuit (use the majority circuit of Exercise 3.1), as shown in the following Figure.



THE circuit will compare the binary numbers A=A1A0 (for input A use switches SW3, SW2) and B=B1B0 (for input B use switches SW1, SW0) and produce 3 outputs: LT (less than, LED2), GT (greater than, LED1) and EQ (equal to, LED0)

- Simulate the circuit (use the testbench of Exercise 3.4).
- Implement the circuit (use the ucf file from Exercise 3.4).
- Program the board and check the operation of the circuit.

To test the operation of the TMR mechanism, create a comparator_faulty circuit, which implements the same function as the comparator, but contains an error in one of its outputs (implement a logical error, where one output will have an incorrect value for some input combinations).

- Replace one of the three comparator_TMR modules with the comparator_faulty circuit. Implement and test the circuit.
- Replace two of the three comparator_TMR modules with the comparator_faulty circuit. Implement and test the circuit.

Lab Exercise 5: Processes and sequential instructions

In this lab exercise you will work on designing combinational circuits using processes and sequential instructions (if-then-else, for-loop). More specifically, you will design:

- A 2-bit two-number comparator
- An integer enumerator of a binary number

Exercise 1: Binary number comparator

Draw the 2-bit unsigned binary 2-bit comparator that you also drew in Exercise 3.4, but using the if-then-else instruction (instead of the when-else instruction you used in Lab 3).

The circuit will compare the binary numbers A=A1A0 (for input A use switches SW3, SW2) and B=B1B0 (for input B use switches SW1, SW0) and produce 3 outputs: LT (less than, LED2), GT (greater than, LED1) and EQ (equal to, LED0).

- Draw the circuit using the **if-then-else** command.
- Compose and implement the circuit by importing the appropriate ucf file.
- Program the board and check the operation of the circuit.

Exercise 2: Number of ones in a binary number

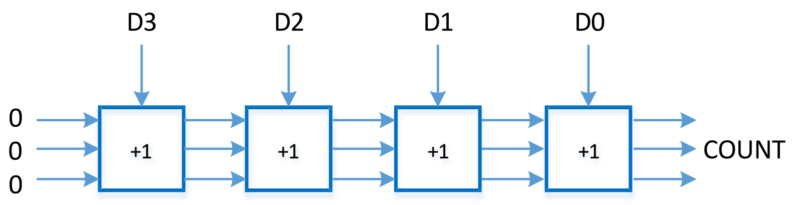
Design a circuit that will count the number of ones ('1') contained in a 4-bit binary number.

- The entity of the unit is as follows:

```
 entity ones_counter is
    port (
          D      :  in  STD_LOGIC_VECTOR(3  DOWNTO  0);  in  STD_LOGIC_VECTOR(3
DOWNTO 0);
          COUNT : out UNSIGNED(2 DOWNTO 0)
    );
end ones_counter;
```

- For the COUNT output you can also use the integer or std_logic_vector types.
- Draw the circuit using the **for loop** command.
- Compose and implement the circuit by importing the appropriate ucf file (for input D use switches SW3-SW0 and for output COUNT use LED2-LED0).
- Program the board and check the operation of the circuit.

Draw the same circuit using the structural description in the figure below. Comment on how the structural description relates to the original implementation using the for loop command.



Lab Exercise 6: Sequential circuits

In this lab exercise you will deal with the design of sequential circuits and in particular the implementation of

- Binary cash
- Frequency-division circuits
- Debouncing circuits (debouncing circuits)
- Accumulators (accumulators)

Notes: Adopt the entity and port names listed in the exercise pronunciation to avoid incompatibility problems with the vhdl files provided.

Exercise 1: 4-bit binary counter

Design a 4-bit binary counter that has the following functions: (a) resets to zero when the RESET input signal is asserted, (b) counts up or down depending on the value of an UP_DOWN input signal, and (c) freezes when the FREEZE input signal is asserted. For the inputs and outputs of the circuit use the following:

- RESET: BTN0
- FREEZE: BTN1
- UP_DOWN: SW0
- COUNT (counter outputs): LED3-LED0
- CLK: for the counter clock use the 50MHz clock of the board

- Create a new project.
- Create a new file (vhdl module) named counter_4b.vhd.
- The entity of the unit is as follows:

```
 entity counter_4b is
    port (
         clk, reset       : in STD_LOGIC;
         freeze, up_down in STD_LOGIC;
         count            : out unsigned(3 downto 0)
    );
end counter_4b;
```

- Design the meter architecture (Note: Consult the course slides).
- Create a vhdl testbench named counter_4b_tb.vhd and associate it with the counter_4b module. The clock should have a period of 20ns (frequency 50 MHz). Simulate the circuit.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and test all circuit functions. What do you observe?

Exercise 2: Clock divider

In this exercise you will use the freq_div.vhd file. The freq_div entity divides the frequency of the clock.

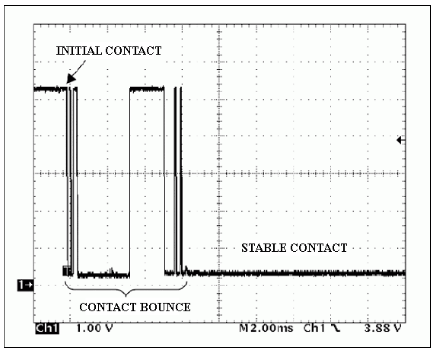
Note: You can download the freq_div.vhd and top_level_counter.vhd files from the course website (contained in Lab4.zip).

- Add the freq_div.vhd file to the Exercise 1 project
- Add the top_level_counter.vhd file to the project. Note: The file has the same entity (ports) as counter_4b and contains instances of counter_4b and freq_div. It essentially binds the divided clock to the counter instead of the 50MHz clock on the board.
- You can set the frequency divider according to the value of the CLK_DIVISOR constant. The input clock is set as 50MHz and the output clock is set as 2Hz.
- Compose and implement the circuit using the same ucf file.
- Program the board and check the operation of the circuit.
- You can change the clock divider to another output frequency and check the operation of the circuit.

Exercise 3: Shock Avoidance Circuit

In this exercise you will use a debouncer circuit.

The phenomenon of bouncing occurs in mechanical switches and results in the output of the switch changing many times between ON and OFF before it stabilizes at its final value (as in the adjacent picture). This can cause problems when the clock frequency of the circuit is higher than the frequency of the bouncing.



Design a 4-bit counter as in exercise 1, except this time the counter will change values manually (by pressing a button). The counter has the following functions: (a) it resets to zero when the RESET input signal is activated, (b) it counts up when the UP button is pressed (for each press of the button the counter should only increase once), and (c) it counts down when the DOWN button is pressed (for each press of the button the counter should only decrease once). For the inputs and outputs of the circuit use the following:

- RESET: BTN0
- UP: BTN1
- DOWN: BTN2
- COUNT (counter outputs): LED3-LED0
- CLK: on-board 50MHz clock (Attention: Do not use the divided clock)

- Create a new project.
- Create a new file (vhdl module) named counter_m_4b.vhd.
- The entity of the unit is as follows:

```
entity counter_m_4b is
   port (
        clk, reset : in STD_LOGIC;
        up, down   in STD_LOGIC;
        counter    : out unsigned(3 downto 0)
    );
end counter_m_4b;
```

For each press of the UP & DOWN buttons the meter should only sense one clock pulse (of 50MHz) so that it counts only once. Otherwise, for each button press it will count multiple times. To achieve this, insert the following circuit in your architecture (add the corresponding signals to the architecture).

```
up_button: process (clk)
begin
     if clk'event and clk = '1' then
          up_q1 <= UP;
          up_q2 <= up_q1;
     end if;
end process;
up_pulse <= up_q1 and (not up_q2);;

down_button: process (clk)
begin
     if clk'event and clk = '1' then
          down_q1 <= DOWN;
          down_q2 <= down_q1;
     end if;
end process;
down_pulse <= down_q1 and (not down_q2);;
```

The above circuit for each press of the UP or DOWN button produces a pulse of one CLK clock period.

- Use the up_pulse & down_pulse signals as enable signals for the meter instead of using the signals from the UP & DOWN buttons directly.
- Create a vhdl testbench named counter_m_4b_tb.vhd and associate it with the counter_m_4b module. The clock should have a period of 20ns (frequency 50 MHz). Simulate the circuit. For the UP & DOWN buttons simulate "pulses" of 10 pulses (200 ns) duration.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and test all circuit functions.

If you notice that the meter has a strange behaviour, i.e. it takes "undefined" values every time you press a button, this may be due to the phenomenon of oscillation. To avoid the oscillation phenomenon you can use the following circuit (modify the up_button process).

```
constant deb_length: integer :=16;
```

```
signal up_q: std_logic_vector(deb_length-1 downto 0);;
...
up_button_p1: process (clk)
begin
  if clk'event and clk = '1' then
       up_q(0) <= UP;
       up_q(deb_lenght-1 downto 1) <= up_q(deb_lenght-2 downto 0);;
  end if;
end process;

up_button_p2: process(up_q)
variable temp: std_logic;
begin
  temp := up_q(0);
  for i in 1 to deb_length-2 loop
       temp := temp and up_q(i);
  end loop;
  temp := temp and (not up_q(length-1));;
  up_pulse <= temp;
end process;
```

- Similarly modify the down_button process.
- Compose and reimplement the circuit.
- Program the board and check the operation of the circuit.
- You can adjust the depth of the circuit (deb_length) depending on the frequency of the clock and the duration of the oscillation pulses.

Exercise 4: 4-bit accumulator

Design a 4-bit accumulator that performs the operation ACC = ACC + DIN. The input DIN has a size of 4 bits and the output ACC has a size of 8 bits. The circuit should have the following functions: (a) it is reset when the RESET input signal is asserted, (b) it has an enable input (ENABLE) and reads the DIN input only when the ENABLE input is 1. For the inputs and outputs of the circuit, use the following:

- RESET: BTN0
- ENABLE: BTN1
- DIN (accumulator inputs): SW3-SW0

- COUNT (counter outputs): LED7-LED0
- CLK: on-board 50MHz clock (Attention: Do not use the divided clock)

Note: For each new input you want to add to the accumulator you must activate the ENABLE input (push button) after first setting the DIN input to switches SW3-SW0. For each press of ENABLE the accumulator should only sense one clock pulse (of 50MHz) so it will only read and add one value. For this purpose (and also to avoid oscillation) use the circuit of the previous exercise.

- Create a new project.
- Create a new file (vhdl module) named acc_4b.vhd.
- The entity of the unit is as follows:

```
 entity acc_4b is
    port (
         clk, reset       : in STD_LOGIC;
         enable           : in STD_LOGIC;
         din              in unsigned(3 downto 0);;
         acc              : out unsigned(7 downto 0)
    );
end acc_4b;
```

- Design the architecture of the accumulator (Note: Consult the course slides).
- Create a vhdl testbench named acc_4b_tb.vhd and associate it with the acc_4b module. The clock should have a period of 20ns (frequency 50 MHz). Simulate the circuit.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and test the operation of the circuit.

Draw the same circuit using a schematic diagram. Use an 8-bit adder (add8) and an 8-bit register (fd8re).

Lab Exercise 7: Finite State Machines

In this lab exercise, you will work on the design of finite state machines (FSMs) and in particular the implementation:

- A serial input parity generator (parity generator)
- A sequence generator (sequence generator)

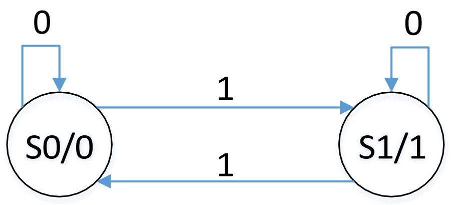
Exercise 1: Parity generator

Design a sequential circuit that accepts a DIN serial input and produces the parity of the input at the PARITY serial output, i.e., it produces the value 1 when it receives an odd number of aces at the serial input. The parity generator should have a RESET reset input and operate on a ½ Hz clock (2 sec period). For the inputs and outputs of the circuit use the following:

- RESET: BTN0
- DIN: SW0
- PARITY: LED0
- CLK: on-board 50MHz clock.

Instruction:Use the circuit from exercise 2 of lab exercise 4 (clock divider) to divide the 50 MHz clock on the board into a ½ Hz clock. You can also drive the divided clock signal to an LED (e.g. LED7) to monitor each time the clock changes.

The operation of the generator can be modeled as a Moore-type finite-state machine. (Note: the generator could also be implemented as a Mealy-type machine).



- Create a new project.
- Create a new file (vhdl module) named par_gen.vhd.
- The entity of the unit is as follows:

```
entity par_gen is
   port (
        clk, reset : in STD_LOGIC;
        din          in STD_LOGIC;.
        parity     : out STD_LOGIC
   );
end par_gen;
```

- Design an architecture that implements the above finite state machine (Note: Consult the course slides).
- Create a vhdl testbench named par_gen_tb.vhd and associate it with the par_gen module. Simulate the circuit.

Instruction: For the purposes of the simulation, modify the frequency divider to divide the input clock by a small divisor (e.g., by 4). Before proceeding to the next step, reset the clock divider to its original value.

- Implement the circuit by importing the appropriate ucf file.
- Program the board and test all circuit functions.

Experiment with the properties of the synthesis tool for the FSMs encoding type. (Synthesize-XST -> Process Properties -> Process Properties -> HDL Options -> FSM Encoding Algorithm). Test the One-Hot & Compact types and compare the synthesis results by selecting View RTL Schematic.

Exercise 2: Sequence detector

Design a sequential circuit that accepts a 2-bit DIN[1:0] input and produces a DOUT output of 1 when the total number of aces received at the DIN input is exactly divided by 3. The circuit should have the following functions: (a) it is reset when the RESET input signal is asserted, (b) it has an enable input (ENABLE) and reads the DIN input only when the ENABLE input is 1, and (c) it operates on a ½ Hz clock (2 sec period). For the inputs and outputs of the circuit use the following:

- RESET: BTN0
- ENABLE: BTN1
- DIN[1:0]: SW1-SW0
- DOUT: LED0
- CLK: on-board 50MHz clock.

Instruction:Use the circuit from exercise 2 of lab exercise 4 (clock divider) to divide the 50 MHz clock on the board into a ½ Hz clock. You can also drive the divided clock signal to an LED (e.g. LED7) to monitor each time the clock changes.

Model the operation of the circuit with a Moore-type finite-state machine. How many states does the machine have?

- Create a new project.
- Create a new file (vhdl module) named seq_det.vhd.
- Design the entity and architecture of the sequence detector.
- Create a vhdl testbench named seq_det_tb.vhd and associate it with the seq_det module. Simulate the circuit.

  Instruction: For the purposes of the simulation, modify the frequency divider to divide the input clock by a small divisor (e.g., by 4). Before proceeding to the next step, reset the clock divider to its original value.
- Implement the circuit by importing the appropriate ucf file.
- Program the board and test all circuit functions.

Lab Exercise 6: Memories

In this laboratory exercise you will implement: