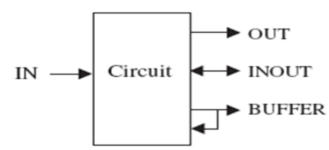
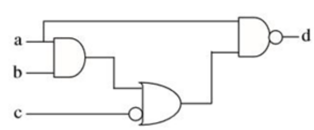
EE 213 Tutorial Lab
1

Contents

This is a VHDL tutorial that introduces basic concepts necessary for the mainlab #1 experiments. First, a basic VHDL structure is introduced. Then, the Vivado Design Suit is briefly explained, and it is described how to load a bitstream file into a Basys3 Xilinx board.

- What is VHDL (Very High –Speed Integrated Circuit Hardware Description Language)
  - It describes the behavior of an electronic, circuit or system, from which the physical circuit or system can then be implemented
  - VHDL is intended for circuit synthesis as well as circuit simulation. However, though VHDL is fully simulatable, not all constructs are synthesizable





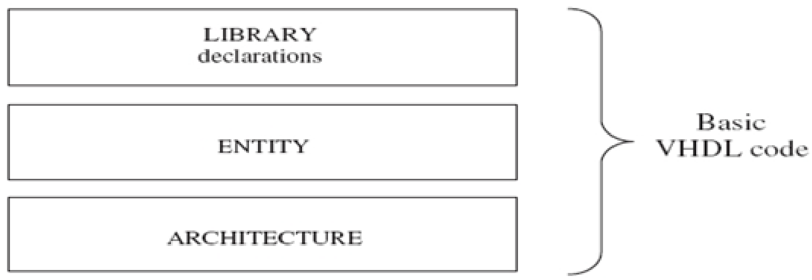

```
ENTITY example IS
   PORT(a,b,c: IN BIT
          d: OUT BIT);
END example ;
ARCHUTECTURE  behavior of example IS
BEGIN
   d <= NOT((NOT c OR (a AND b)) AND a);
END behavior;
```

VHDL Application Examples

1. Counters
2. Arithmetic Logic Units (ALU)
3. Registers
4. Flip-plops
5. Multiplexer

6. Decoder

Fundamental VHDL Units :



- LIBRARY declarations: Contains a list of all libraries to be used in the design. For example: ieee, std, work, etc.

- ENTITY: Specifies the I/O pins of the circuit.

- ARCHITECTURE: Contains the VHDL code proper, which describes how the circuit should behave (function).

1) **Entity:**

An ENTITY is a list with specifications of all input and output pins (PORTS) of the circuit. Its syntax is shown below:

ENTITY entity_name IS

PORT (

port_name : signal_mode signal_type;

port_name : signal_mode signal_type;

...);

END entity_name;

The mode of the signal can be: IN, OUT, INOUT, or BUFFER.

The type of signal can be: BIT, BOOLEAN, INTEGER, or REAL.

```
Entity entityName is
   Port ( I0, I1: in
          STD_LOGIC; O :
          out STD_LOGIC);
End entityName;
```

In entity, inputs and outputs of a circuit are declared. I0, I1, and O are port names, and entityName is the name of the circuit. See that the inputs I0 and I1 are designated as
‘in’ and the output O as ‘out’. There are two other types, “buffer” and “in/out” which will be explained in some other time.

STD_LOGIC is one of the data types in VHDL, which can take the values of ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'). For now, ‘0’ and ‘1’ are enough for the lab #1. Other data types include BIT, BOOLEAN, INTEGER, STD_LOGIC_VECTOR, and BIT_VECTOR.

General format for port declaration is:

```
signalName :mode data type;
E.g. dataBus :in STD_LOGIC_VECTOR(7 down to 0);
```

2) Architecture:

The ARCHITECTURE is a description of how the circuit should behave (function).
Its syntax is the following:

ARCHITECTURE architecture_name OF entity_name IS
[declarations]
BEGIN
(code)
END architecture_name;

As shown above, an architecture has two parts: a declarative part (optional), where signals and constants (among others) are declared, and the code part (from BEGIN down). Like in the case of an entity, the name of an architecture can be basically any name (except VHDL reserved words. Ex: İf,else,loop…), including the same name as the entity’s.

```
architecture  architectureName of
        entityName is type declarations
        signal declarations
        constant declarations
        function declaration
        procedure definitions
        component
        declarations
begin
        concurrent statements
end architectureName;
```

Architecture describes what a circuit does. Between “architecture” and “begin” there are declarations. In VHDL, there are predefined types that can be used. They are bit, bit_vector, Boolean, integer, real, time, string character, and time severity_level. For example, if a signal is to be declared, the following format is used.

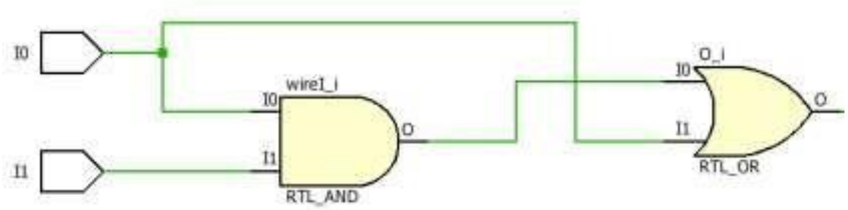
signal signalName : signal-type;

E.g. signal I0_ t : STD_LOGIC;

Notice the difference between a port declaration and a signal declaration. In the signal declaration, there is no need to define the mode of a signal. A signal can be considered as a wire in a circuit.

Example 1:

```
Entity entityName is
   Port ( I0 : in STD_LOGIC;
          I1 : in STD_LOGIC;
          O :out STD_LOGIC);
End entityName;
Architecture Behavioral of entityName is
signal wireI : std_logic;  (declared in
architecture part while ports are
declared in ENTITY part)
begin
wireI<= I0 and I1;
O<= wireI or I0;
end Behavioral;
```



In this example, "wireI" signal is used to connect the output of I0 AND I1 to another gate. The corresponding statement is "wireI<= I0 and I1;". Notice that VHDL is not case sensitive. You can use either STD_LOGIC or std_logic.

3) Component:

A COMPONENT is simply a piece of conventional code (that is, LIBRARY declarations + ENTITY + ARCHITECTURE). However, by declaring such a code as a COMPONENT, it can then be used within another circuit, thus allowing the construction of hierarchical designs.

A COMPONENT is also another way of partitioning a code and providing code sharing and code reuse.

In digital systems, smaller subsystems are generally used. In order to use these subsystems, 'component' is used. A declaration of component is inserted between "architecture" and "begin". Syntax for instantiation of component is,

```
Component componentName Port (…); end component;
label: component_name PORT MAP (port_list);
```

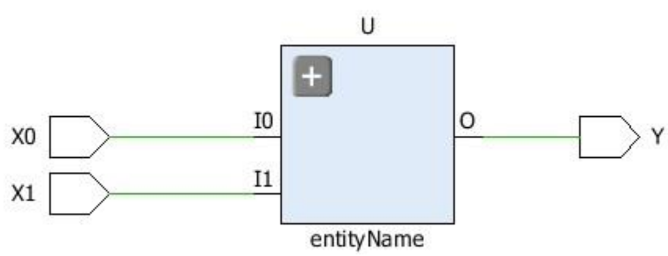
port_list is just a list relating the ports of the actual circuit to the ports of the pre-designed component which is being instantiated.

For example, to use the figure below as a component in another circuit, entityName2

```
Architecture Behavioral of entityName2 is

Component entityName Port ( I0,I1 : in STD_LOGIC; O : out STD_LOGIC); end component;

begin
```

Now this component's behavior can be used in another circuit. This is called structural description. Next step is to match the ports of entityName and entityName2. For this, port map is used.

```
U: entityName port map (inputs,outputs);
```



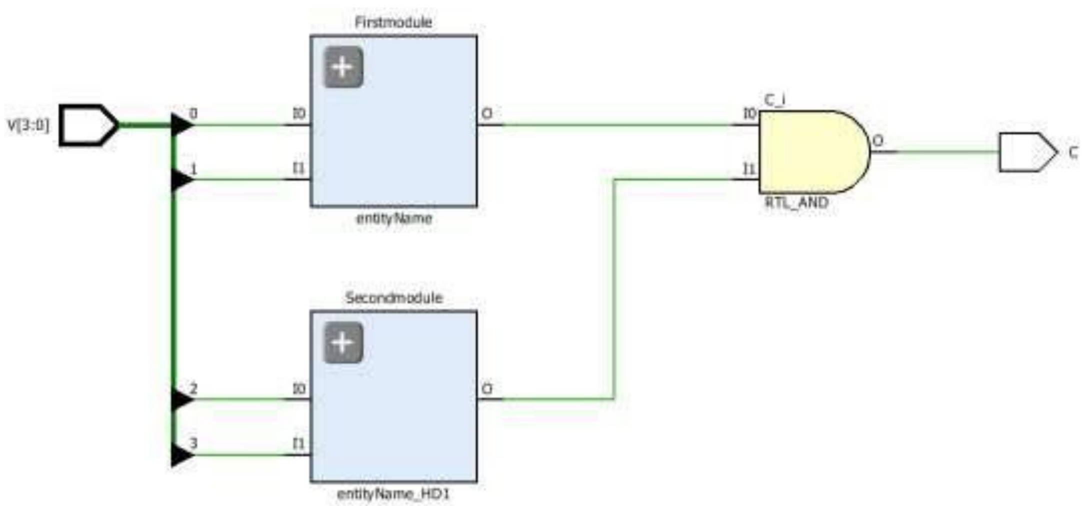

```
Entity entityName2 is

   Port ( X0 : in STD_LOGIC;

          X1 : in STD_LOGIC;

          Y :out STD_LOGIC);

End entityName2;

Architecture Behavioral of entityName2 is

Component entityName Port ( I0,I1 : in STD_LOGIC; O : out STD_LOGIC); end component;

begin

U: entityName port map(X0,X1,Y);  (Positional Mapping: Easier to write)

End Behavioral;
```

Now, a component entityName is included in another circuit entityName2, and their ports are matched. 'U' is used to label a component circuit, and port map is for one to one port mapping, such as X0=>I0, X1=>I1, Y=>O. Another way of port mapping is the following.

```
U: entityName port map(X0=>I0, X1=>I1, Y=>O)
```

In this way, you do not need to put ports in the order, and the code becomes less error-prone.

Now assume that the design is composed of two components, 4 inputs, and 1 output. Outputs of the entityNames are ANDed as the output of the larger circuit.




```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Entity entityName3 is
        Port ( V : in STD_LOGIC_VECTOR
        (3 downto 0); C : out
        STD_LOGIC);
End entityName3;
Architecture Behavioral of entityName3 is
        Component entityName Port ( I0,I1 : in STD_LOGIC; O : out STD_LOGIC); end component;
        Signal outputEntityName1,outputEntityName2 : STD_LOGIC;
begin
       Firstmodule: entityName port map(V(0),V(1),outputEntityName1);
       Secondmodule: entityName port map (V(2),V(3),outputEntityName2);
       C <= outputEntityName1  AND  outputEntityName2 ;
End Behavioral;
```

Here, just one component is instantiated because the same circuit is used twice. If the second component were different from the first one, different components would be instantiated. After the “begin” part, port map functions for the two components are used. Two signal outputEntityName1 and outputEntityName2 are used as the inputs for the last AND gate.

VIVADO

Vivado is software to write codes and put them on the board. With Vivado we will design circuits and run them on the FPGA board. Without gates or other circuit elements only by using software program we can create simple or complicated circuit and run on the hardware board. We will use VHDL as programming language.

In order to connect VHDL codes with a FPGA board (Basys3) we need a constraint file of a board. We need to add this file to the project.

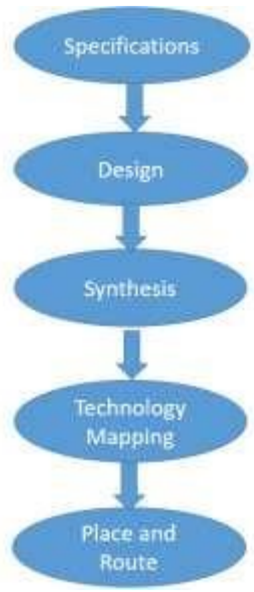
Test Bench:

Before generating a bitstream file, we need to be sure that our design is working correctly. That is why we are going to simulate for all the possible cases. Test benches are the simulation sources in Vivado.In the previous part, we created a design sources. You will see Vivado creates a simulation source for each design source you have generated. But, when you apply for simulation-
> Run behavioral simulation, you will see undefined ‘U’ inputs. That is why we need to write a test bench code.

Consider the first circuit, entityName with 2 inputs and 1 output. There are 4 possible input combinations because STD_LOGIC can be either 0 or 1. We need to test the output O for the cases of I0 = 0, 1 and I1 = 0, 1.

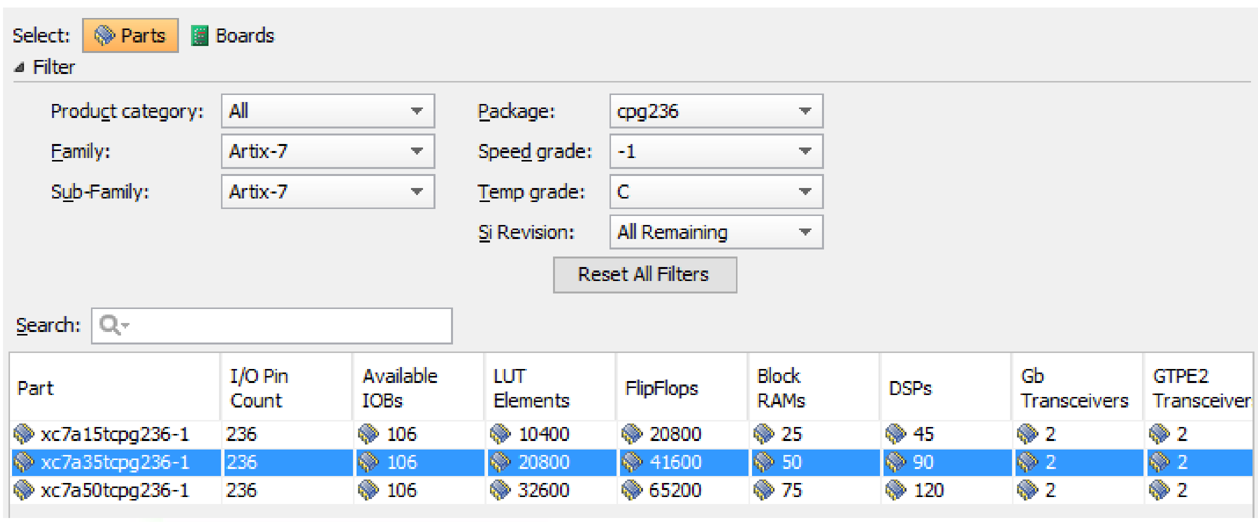
In the figure above, you can see the states of the output for the each combination of inputs. Remember the logic function is “O = (I0 and I1) or I0;” For instance, if I0 = 1 and I1=0, then output O is 1.

Vivado Design Suit

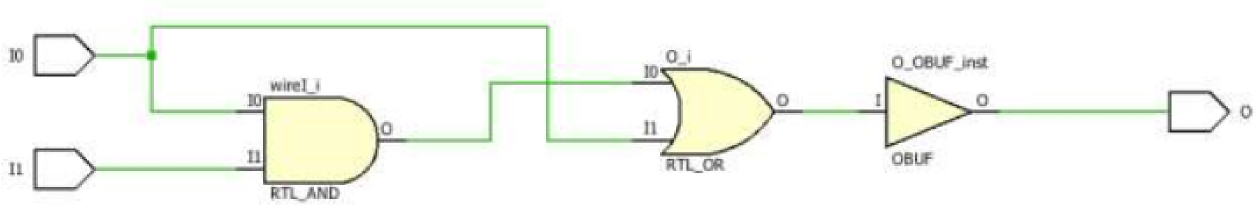

Specifications
Design
Synthesis
Technology Mapping
Place and Route

Shown above is the general flow of a digital system design. First, the source files are created. In the first two labs, we will use design sources, constrained sources and simulation sources. After writing a VHDL code, first thing to do is "Elaborated design." Then, you can see your design at the gate level. Second step is "Run simulation." You can see whether your design works properly or not. Even if you made all the connection correctly, functionality is another matter. After simulation, it is time to synthesize and implement the design for a selected hardware. Final step is to generate a bitstreamfile.

NOTE_1: While we are creating the project, it is better to select our board. In below you can see the description of our Basy3 board.

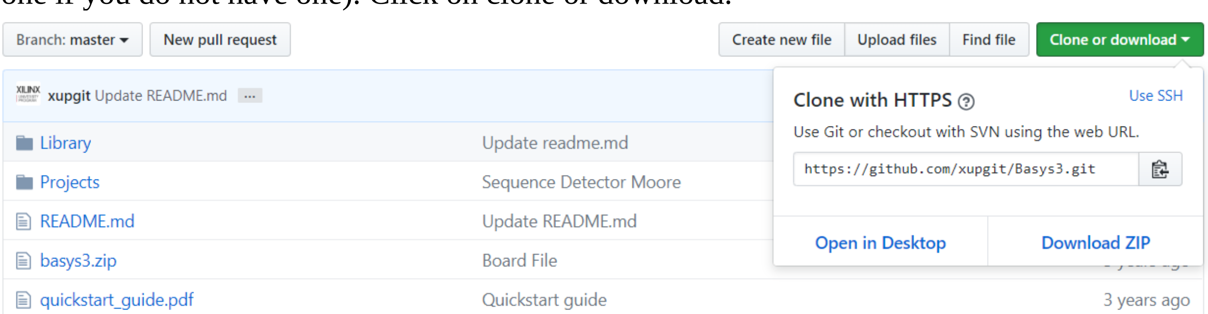


Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceiver
xc7a15tcpg236-1	236	106	10400	20800	25	45	2	2
xc7a35tcpg236-1	236	106	20800	41600	50	90	2	2
xc7a50tcpg236-1	236	106	32600	65200	75	120	2	2

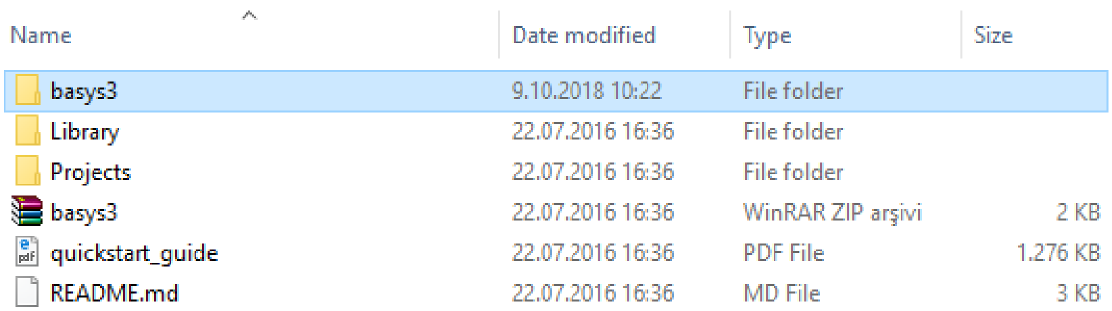


NOTE_2: To be able to integrate our code with Basy3 board we need to obtain XUP library. Details are shown in below.

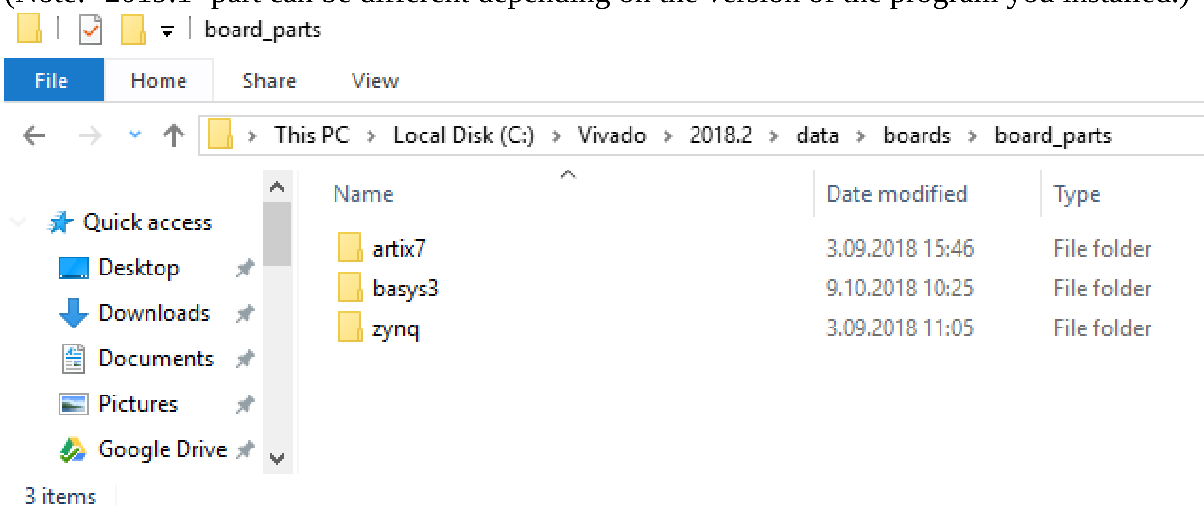
Visit https://github.com/xupgit/Basys3. Log in to GitHub with your account (you will have to create one if you do not have one). Click on clone or download.



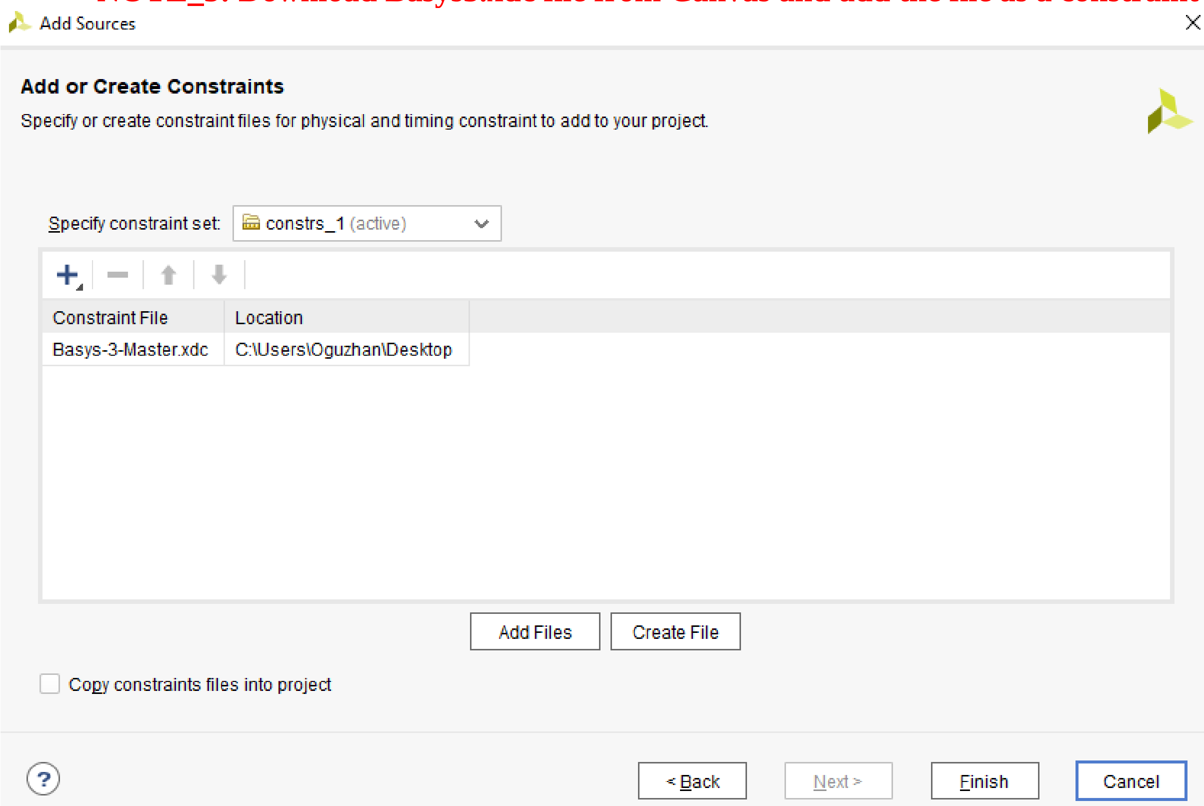
Extract the zip files.

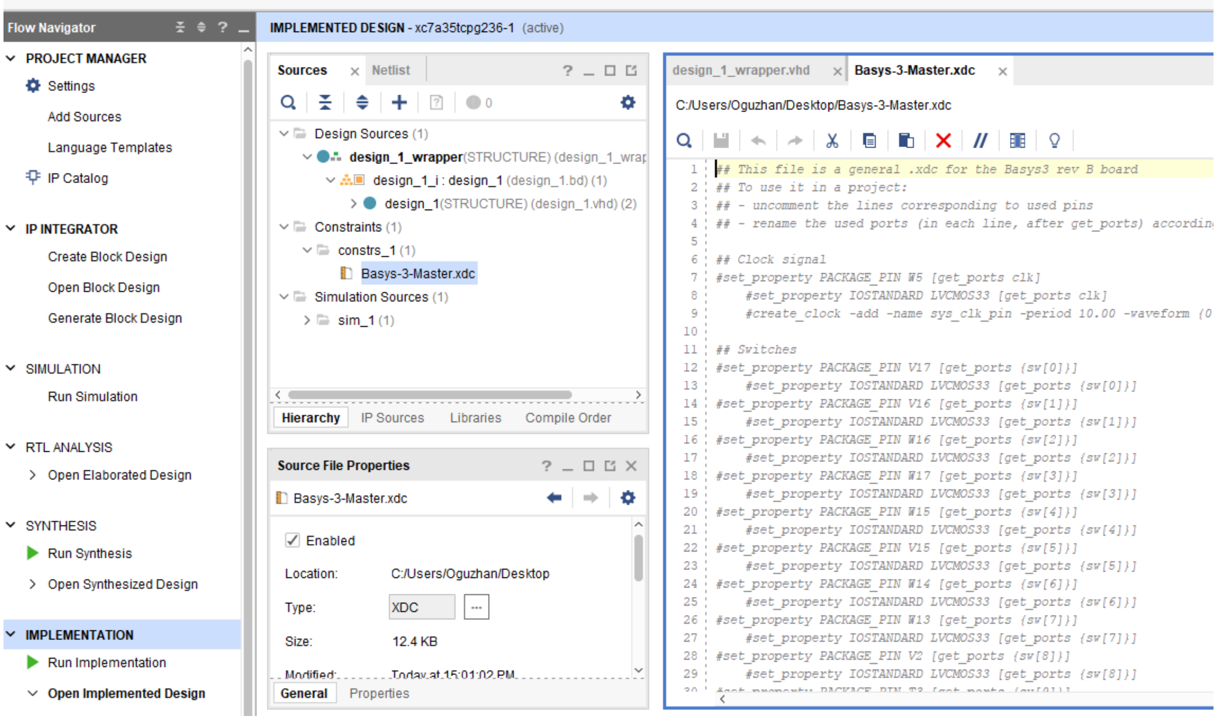


Now copy the basys3 folder to C:\Vivado\2019.1\data\boards\board_parts
(Note: ‘2019.1’ part can be different depending on the version of the program you installed.)

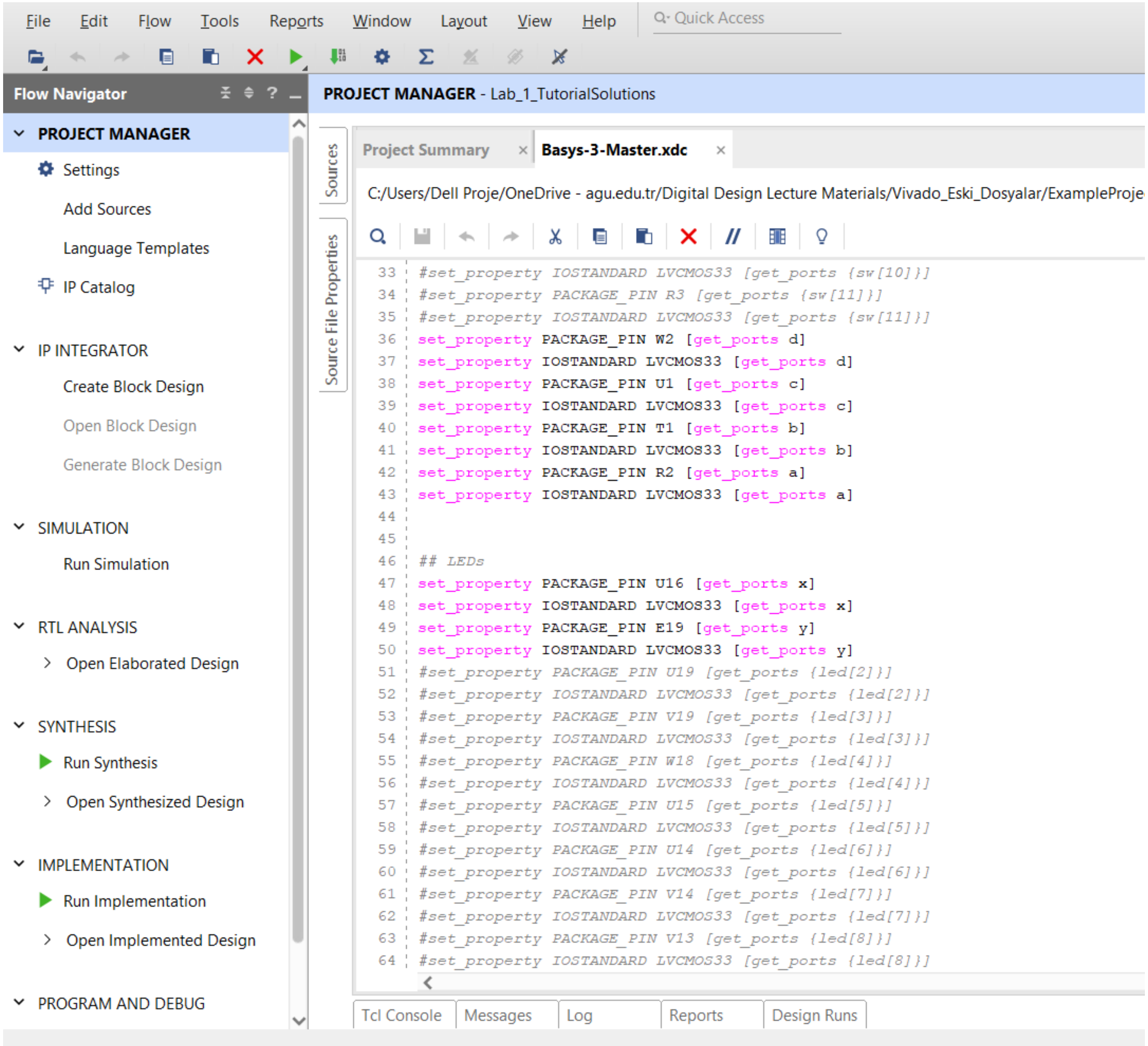


NOTE_3: Download Basys3.xdc file from Canvas and add the file as a constraint file.



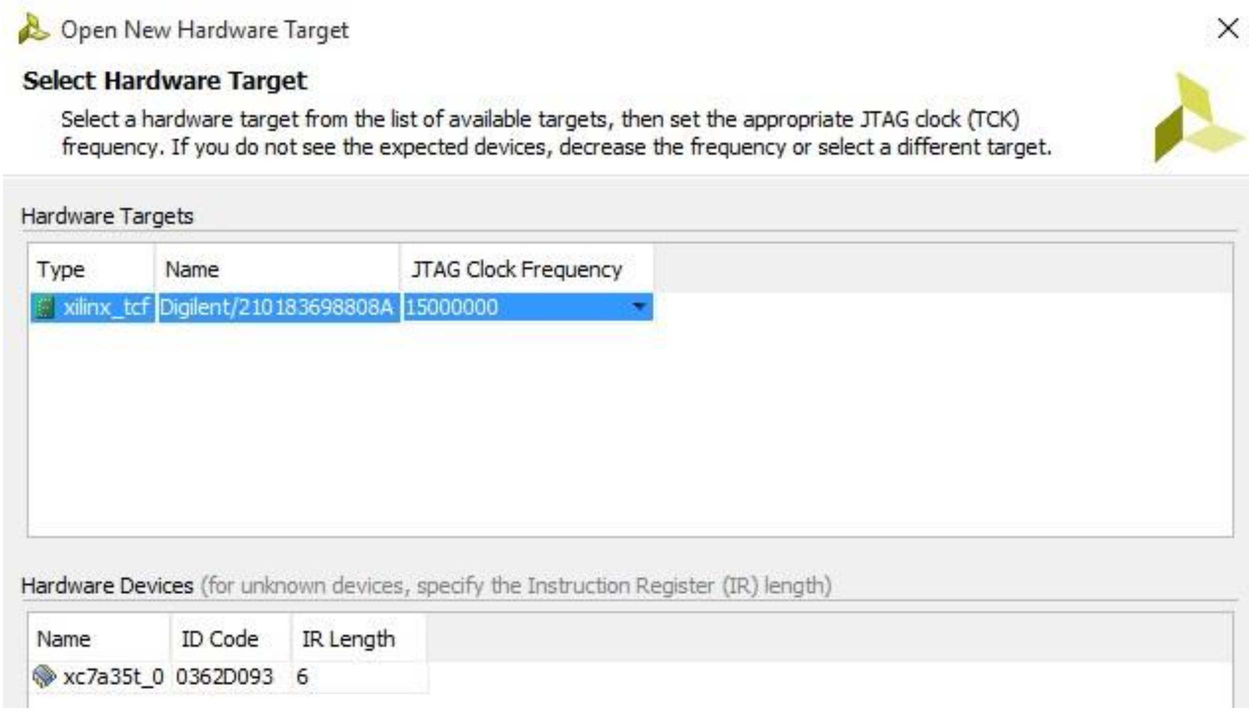


Now you need to define your inputs and outputs, and to match them with Basys3 peripherals. For instance, we have four binary inputs a, b, c, d and two binary output x and y in below example. We can use switches on the board for the inputs and leds for the outputs. In the Basys3 board, there are 16 switches (you can see from the constraint file that there are sixteen switches, from sw[0] to sw[15]). Modifications are shown in below;

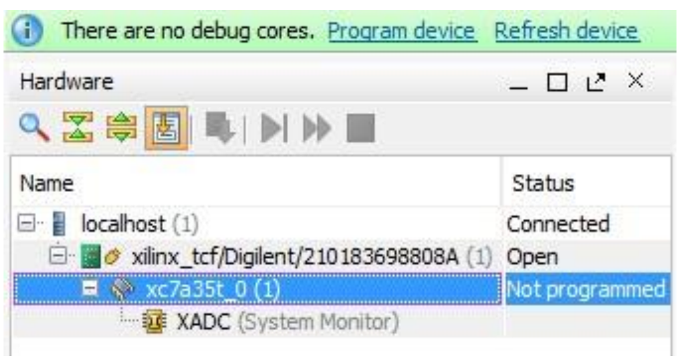


Basys3

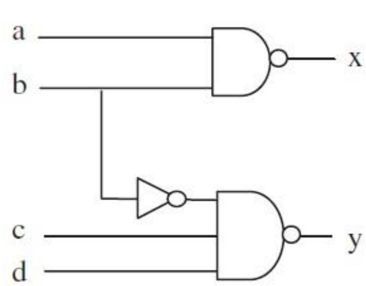
After generating the bitstream file, it is time to open the Hardware manager. If it is the first time, you need to define your device. Open Target -> Open New Target -> Next -> Connect to: Local server, then finalize the step.



After this step, you will see the board in the list. Now you can program your device.



EXAMPLE :




```
------ File inverter.vhd: -------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY inverter IS
PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END inverter;
------------------------------------
ARCHITECTURE inverter OF inverter IS
```

```
BEGIN
b <= NOT a;
END inverter;
---------------------------------------------
```

```
------ File nand_2.vhd: ---------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY nand_2 IS
PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
END nand_2;
------------------------------------
ARCHITECTURE nand_2 OF nand_2 IS
BEGIN
c <= NOT (a AND b);
END nand_2;
---------------------------------------------
```

```
----- File nand_3.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY nand_3 IS
PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
END nand_3;
------------------------------------
ARCHITECTURE nand_3 OF nand_3 IS
BEGIN
d <= NOT (a AND b AND c);
END nand_3;
---------------------------------------------
```

```
----- File project.vhd: ---------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY project IS
PORT (a, b, c, d: IN STD_LOGIC;
x, y: OUT STD_LOGIC);
END project;
------------------------------------
ARCHITECTURE structural OF project IS
-------------
COMPONENT inverter IS
PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
-------------
COMPONENT nand_2 IS
PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
END COMPONENT;
-------------
COMPONENT nand_3 IS
PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
END COMPONENT;
-------------
SIGNAL w: STD_LOGIC;
BEGIN
U1: inverter PORT MAP (b, w);
U2: nand_2 PORT MAP (a, b, x);
U3: nand_3 PORT MAP (w, c, d, y);
END structural;
---------------------------------------------
```

For more information about Vivado you can check :
[https://reference.digilentinc.com/vivado/getting_started/2018.2?s[]=source&s[]=files](https://reference.digilentinc.com/vivado/getting_started/2018.2?s[]=source&s[]=files) web site