

EE 213 Tutorial Lab

1

Contents

Reminding previous class

Component

Introduction to VIVADO Software

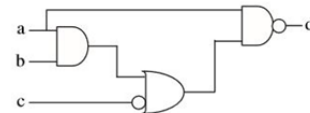
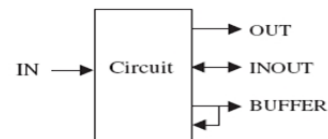
Simple Example

Simple Assignment

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



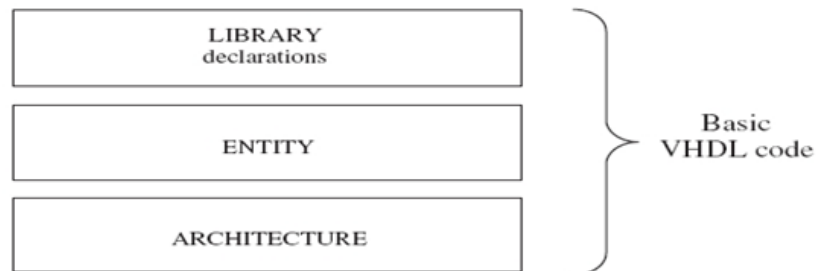
```
ENTITY example IS
  PORT(a,b,c: IN BIT
        d: OUT BIT);
END example ;
ARCHITECTURE behavior of example IS
  BEGIN
    d <= NOT((NOT c OR (a AND b)) AND a);
  END behavior;
```

VHDL Application Examples

1. Counters
2. Arithmetic Logic Units (ALU)
3. Registers
4. Flip-flops
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: If,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 IO_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` (IO : `in` STD_LOGIC;

I1 : `in` STD_LOGIC;

O : `out` STD_LOGIC);

`End` entityName;

`Architecture` Behavioral `of` entityName `is`

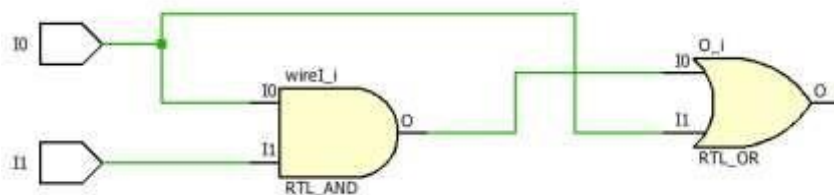
`signal` wire1 : `std_logic`; (`declared in`
`architecture` part while ports are
`declared in ENTITY` part)

`begin`

wire1<= IO and I1;

O<= wire1 or IO;

`end` Behavioral;



In this example, “wire1” signal is used to connect the output of I0 AND I1 to another gate. The corresponding statement is “wire1<= 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.

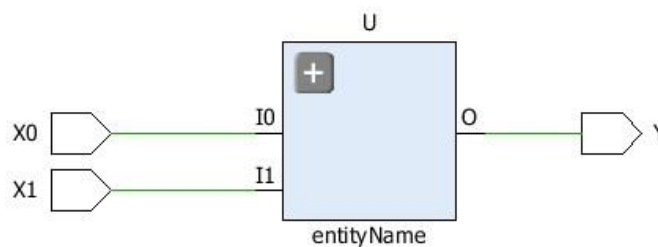
For example, to use the figure below as a component in another circuit, entityName2

```
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)
```

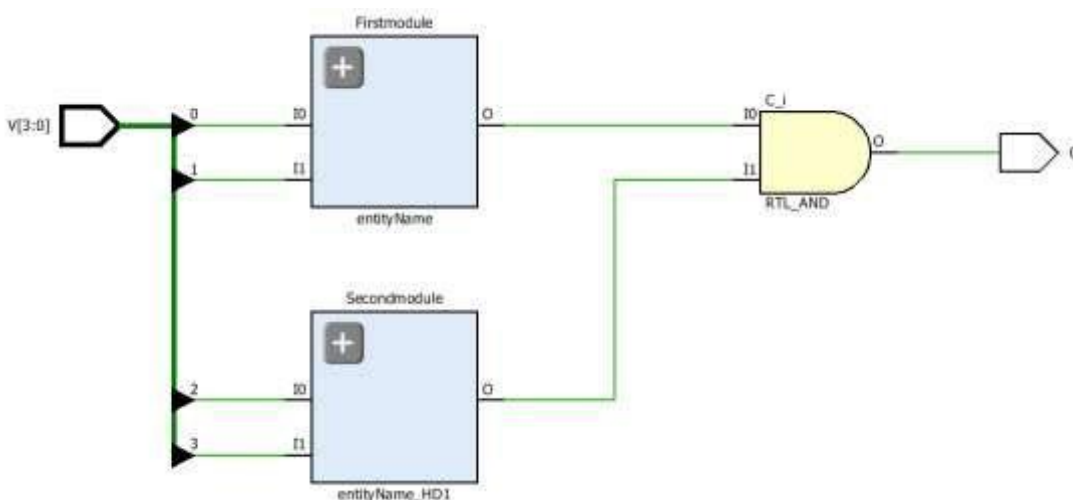
```
End Behavioral;
```

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.

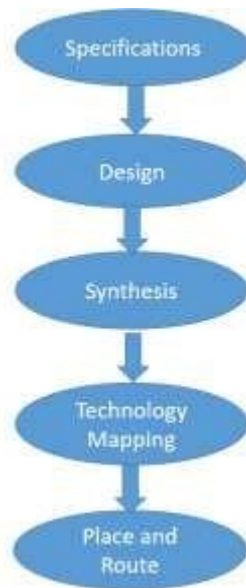
Name	Value	Data Type
I0t	1	Logic
I1t	1	Logic
Ot	1	Logic

Name	Value
I0t	1
I1t	1
Ot	1



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

Vivado Design Suit



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.

Select: **Parts** **Boards**

Filter

Product category: All Package: cpg236

Family: Artix-7 Speed grade: -1

Sub-Family: Artix-7 Temp grade: C

Si Revision: All Remaining

Reset All Filters

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceiver
xc7a15tcbg236-1	236	106	10400	20800	25	45	2	2
xc7a35tcbg236-1	236	106	20800	41600	50	90	2	2
xc7a50tcbg236-1	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.

Branch: master New pull request

Create new file Upload files Find file Clone or download

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/xupgit/Basys3.git>

Open in Desktop Download ZIP

Library Update readme.md







Projects Sequence Detector Moore

README.md Update README.md




basys3.zip Board File

quickstart_guide.pdf Quickstart guide 3 years ago

Extract the zip files.




Name	Date modified	Type	Size
 basys3	9.10.2018 10:22	File folder	
 Library	22.07.2016 16:36	File folder	
 Projects	22.07.2016 16:36	File folder	
 basys3	22.07.2016 16:36	WinRAR ZIP arşivi	2 KB
 quickstart_guide	22.07.2016 16:36	PDF File	1.276 KB
 README.md	22.07.2016 16:36	MD File	3 KB

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.)

 |   board_parts

File Home Share View

← → ⌵ ⬆ > This PC > Local Disk (C:) > Vivado > 2018.2 > data > boards > board_parts

Name	Date modified	Type
 artix7	3.09.2018 15:46	File folder
 basys3	9.10.2018 10:25	File folder
 zynq	3.09.2018 11:05	File folder

3 items

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

Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Constraint File	Location
Basys-3-Master.xdc	C:\Users\Oguzhan\Desktop

☐ Copy constraints files into project

Flow Navigator

- PROJECT MANAGER
 - Settings
 - Add Sources
 - Language Templates
 - IP Catalog
- IP INTEGRATOR
 - Create Block Design
 - Open Block Design
 - Generate Block Design
- SIMULATION
 - Run Simulation
- RTL ANALYSIS
 - Open Elaborated Design
- SYNTHESIS
 - Run Synthesis
 - Open Synthesized Design
- IMPLEMENTATION
 - Run Implementation
 - Open Implemented Design

IMPLEMENTED DESIGN - xc7a35tqpg236-1 (active)

Sources

- Design Sources (1)
 - design_1_wrapper (STRUCTURE) (design_1_wrapper) (1)
 - design_1 (design_1.bd) (1)
 - design_1 (STRUCTURE) (design_1.vhd) (2)
- Constraints (1)
 - constrs_1 (1)
 - Basys-3-Master.xdc
- Simulation Sources (1)
 - sim_1 (1)

Hierarchy IP Sources Libraries Compile Order

Source File Properties

Basys-3-Master.xdc

☒ Enabled

Location: C:\Users\Oguzhan\Desktop

Type: XDC

Size: 12.4 KB

Modified: Today at 15:01:02 PM

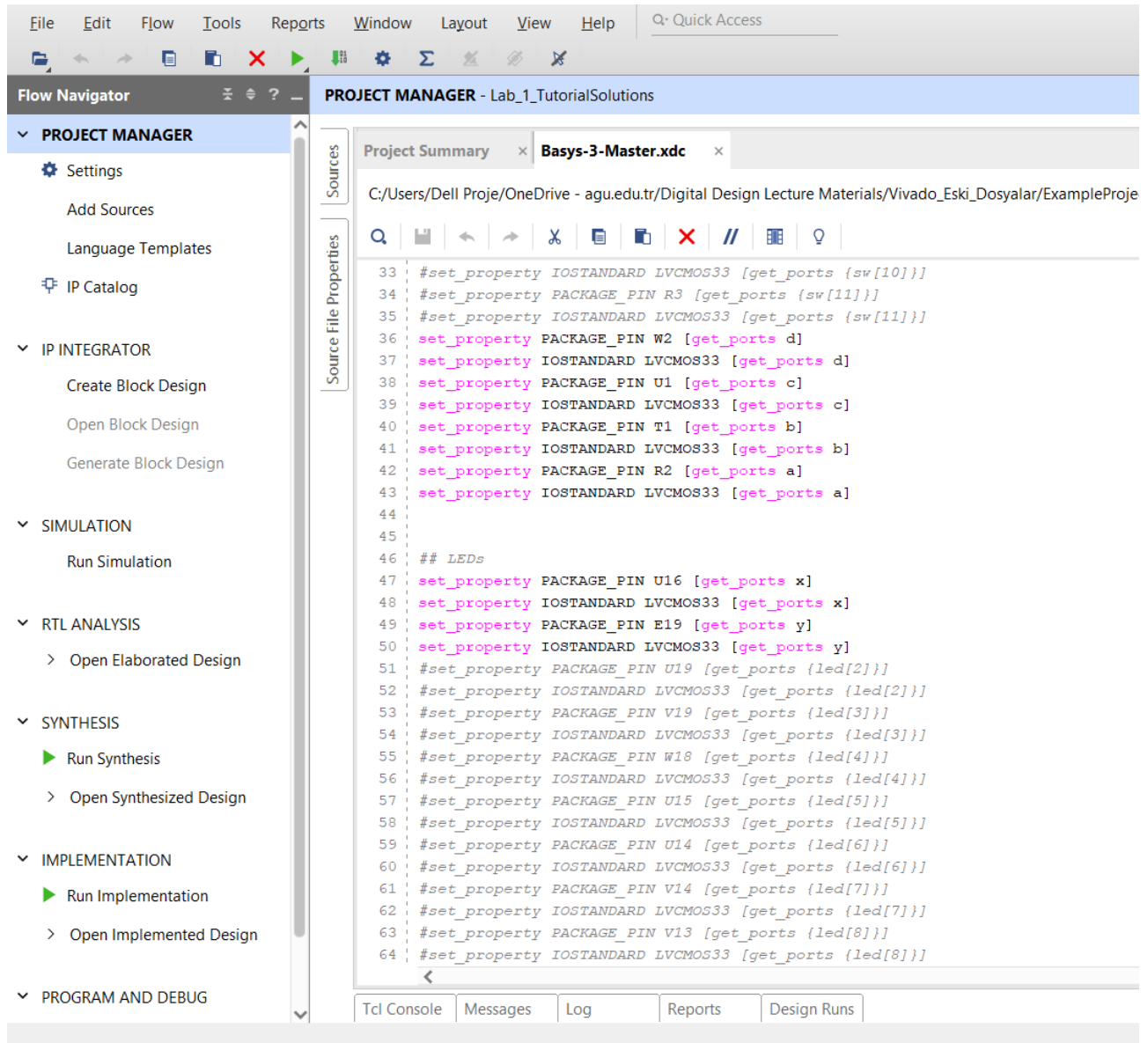
General Properties

design_1_wrapper.vhd Basys-3-Master.xdc

C:\Users\Oguzhan\Desktop\Basys-3-Master.xdc

```
1 ## This file is a general .xdc for the Basys3 rev B board
2 ## To use it in a project:
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according
5
6 ## Clock signal
7 #set_property PACKAGE_PIN W5 [get_ports clk]
8 #set_property IOSTANDARD LVCMOS33 [get_ports clk]
9 #create_clock -add -name sys_clk_pin -period 10.00 -waveform {0
10
11 ## Switches
12 #set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14 #set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
16 #set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
17 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
18 #set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
19 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
20 #set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
21 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
22 #set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
23 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
24 #set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
25 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
26 #set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
27 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
28 #set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
29 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
30 #set_property PACKAGE_PIN W2 [get_ports {sw[9]}]
```

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.

Open New Hardware Target

Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

Type	Name	JTAG Clock Frequency
xilinx_tcf	Digilent/210183698808A	15000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
xc7a35t_0	0362D093	6

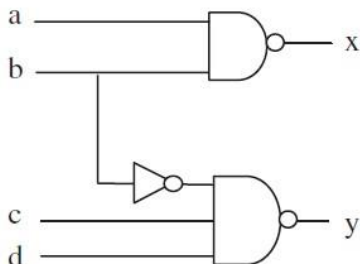
After this step, you will see the board in the list. Now you can program your device.

There are no debug cores. [Program device](#) [Refresh device](#)

Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183698808A (1)	Open
xc7a35t_0 (1)	Not programmed
XADC (System Monitor)	

EXAMPLE :



```

1 ----- File inverter.vhd: -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY inverter IS
6 PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
7 END inverter;
8 -----
9 ARCHITECTURE inverter OF inverter IS

```

```

10 BEGIN
11 b <= NOT a;
12 END inverter;
13 -----

1 ----- File nand_2.vhd: -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY nand_2 IS
6 PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
7 END nand_2;
8 -----
9 ARCHITECTURE nand_2 OF nand_2 IS
10 BEGIN
11 c <= NOT (a AND b);
12 END nand_2;
13 -----

1 ----- File nand_3.vhd: -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY nand_3 IS
6 PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
7 END nand_3;
8 -----
9 ARCHITECTURE nand_3 OF nand_3 IS
10 BEGIN
11 d <= NOT (a AND b AND c);
12 END nand_3;
13 -----

1 ----- File project.vhd: -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY project IS
6 PORT (a, b, c, d: IN STD_LOGIC;
7 x, y: OUT STD_LOGIC);
8 END project;
9 -----
10 ARCHITECTURE structural OF project IS
11 -----
12 COMPONENT inverter IS
13 PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
14 END COMPONENT;
15 -----
16 COMPONENT nand_2 IS
17 PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
18 END COMPONENT;
19 -----
20 COMPONENT nand_3 IS
21 PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
22 END COMPONENT;
23 -----
24 SIGNAL w: STD_LOGIC;
25 BEGIN
26 U1: inverter PORT MAP (b, w);
27 U2: nand_2 PORT MAP (a, b, x);
28 U3: nand_3 PORT MAP (w, c, d, y);
29 END structural;
30 -----

```

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

