



**ESKİŞEHİR TECHNICAL
UNIVERSITY**

**DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING**

EEM 334 - Digital Systems II

**LAB 2 - INTRODUCTION TO VHDL AND
FPGA HARDWARE IMPLEMENTATION**

1. PURPOSE

In this lab, you will learn to write VHDL codes and embed the designs written in VHDL to FPGA. In the first experiment, four bit adder design was prepared with using schematic and VHDL design techniques. When the designed object started to get bigger, VHDL design is much easier to build and making changes on design is not complicated and time consuming according to schematic design techniques. So, next experiments we will use VHDL to build circuits.

In this lab, we will implement four bit adder circuit on Nexys4 DDR board. We will connect inputs to the switches and outputs to the seven segments. For implementation of four bit adder design, you must complete the first experiment and you should learn basic properties of Nexys4 DDR board. You can find basic information about board contents on background information section. For more information, you can visit www.xilinx.com and download board's data sheet and example circuits.

2. BACKGROUND

VHDL is an acronym which stands for VHSIC Hardware Description Language. VHSIC is another acronym which stands for Very High Speed Integrated Circuits.

Hardware description languages can be used in several ways; they can be an alternative way of representing a circuit diagram for a digital circuit or a higher level algorithmic 'program' that solves a particular problem. Such structural or behavioral representations are two ways of describing a model of a digital system.

VHDL can be used for documentation, verification, and synthesis of large digital designs. This is actually one of the key features of VHDL, since the same VHDL code can achieve all three of these goals, thus saving a lot of effort and reducing the introduction of errors between translating a specification into an implementation.

VHDL files are composed of Entity-Architecture pairs. The Entity portion of the file is analogous to a symbol for the design. It describes all of the external connections to the design. The Architecture portion of the file is analogous to the circuit diagram of the design. It defines the implementation of the design. Every VHDL design will have the following general appearance:

```
=====
-- Library and package declarations
=====
LIBRARY ieee;
USE ieee.std_logic_1164.all;
=====
-- The following is the Entity portion
=====
ENTITY name IS      --Keywords are capitalized(optional)
    PORT( list of all external connections);
END name;
=====
-- The following is the Architecture portion
=====
ARCHITECTURE anyname OF name IS
-- This is the declarative part of the Architecture
-- Declare signals, enumeration types, constants here
```

BEGIN

- This is where the implementation is described. Concurrent signal assignments go here. Therefore this is called the concurrent part.
- Order of the statements does not matter since all statements are executed -- concurrently.

PROCESS

- The architecture may contain zero or more processes.
- This is the declarative part of the process.
- Variables used in the process are declared here.
 - BEGIN -- Beginning of the process implementation.
- The process is implemented using sequential statements.
- For example, FOR LOOP, IF-THEN-ELSE, CASE END

PROCESS;

END anyname;

In the above, note the use of -- to indicate a comment. "C-style" comments are not used in VHDL. In the above keywords were capitalized but this is optional. The architecture may contain zero or more processes. Outside of the process all statements are executed concurrently. This region is called the concurrent part of the architecture. Statements in the concurrent part can be in any order because all statements are executed at the same time. This mimics the operation of a real circuit where all gates evaluate their inputs at the same time. Statements in a process on the other hand are executed sequentially. Therefore, the order of statements matters. We will see later that processes are necessary to implement sequential circuits. A purely combinational circuit requires only concurrent statements. Therefore, no processes are needed.

2.1. VHDL design

This design methodology is little different than schematic one. In this design methodology you describe the digital circuit by using hardware description language. When designed object starting to get bigger HDL design methodology advantages arise. Simulation of the designed object can be the same if you use "test bench waveform" file to simulate designs. For this part of the experiment you don't need to write VHDL code, instead of that you will use available VHDL codes which you can download on website. You can also use the codes below.

EXAMPLE CODE

Half adder

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity half_adder is
```

```
Port ( a : in STD_LOGIC;
```

```
      b : in STD_LOGIC;
```

```
      s1 : out STD_LOGIC;
```

```
      c1 : out STD_LOGIC );
```

```
end half_adder;
```

```
architecture Behavioral of half_adder is
```

```
begin
```

```
    s1 <= (a XOR b);
```

```
    c1 <= (a AND b); -- the logical operator "AND" and "XOR" is defined in VHDL.
```

```
end Behavioral;
```

Full Adder

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity full_adder is
```

```
Port ( in1 : in STD_LOGIC;  
       in2 : in STD_LOGIC;  
       cin : in STD_LOGIC;  
       cout : out STD_LOGIC;  
       sout : out STD_LOGIC);
```

```
end full_adder;
```

```
architecture Behavioral of full_adder is
```

```
    signal sum_low : std_logic;
```

```
    signal c_low : std_logic;
```

```
    signal c_high : std_logic;
```

```
    component half_adder
```

```
    Port ( a : in STD_LOGIC;  
          b : in STD_LOGIC;  
          s1 : out STD_LOGIC;  
          c1 : out STD_LOGIC );
```

```
end component;
```

```
-- Components define up side of
```

```
-- architecture begin end block
```

```
begin
```

```
    ha_low : half_adder  
    port map ( a => in1,  
              b => in2,  
              s1 => sum_low,  
              c1 => c_low  
            );
```

```
    ha_high : half_adder  
    port map (a => cin,  
             b => sum_low,  
             s1 => sout,  
             c1 => c_high  
            );
```

```
    cout <= (c_low OR c_high);
```

```
end Behavioral;
```

4 Bit Adder

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity four_bit_adder is

Port (in1 : in STD_LOGIC_VECTOR (3 downto 0);

in2 : in STD_LOGIC_VECTOR(3 downto 0);

sout : out STD_LOGIC_VECTOR(3 downto 0);

cout : out STD_LOGIC);

end four_bit_adder;

architecture Behavioral of four_bit_adder is

component full_adder

Port (in1 : in STD_LOGIC;

in2 : in STD_LOGIC;

cin : in STD_LOGIC;

cout : out STD_LOGIC;

sout : out STD_LOGIC);

end component;

signal w1,w2,w3 : STD_LOGIC;

begin

U1 : full_adder port map(in1(0), in2(0), '0', w1, sout(0));

U2 : full_adder port map(in1(1), in2(1), w1, w2, sout(1));

U3 : full_adder port map(in1(2), in2(2), w2, w3, sout(2));

U4 : full_adder port map(in1(3), in2(3), w3, cout, sout(3));

end Behavioral;

2.2.Nexsys4 DDR FPGA Board

The Nexys 4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®. The Artix-7 FPGA is optimized for high performance logic and offers more capacity, higher performance, and more resources than earlier designs. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys 4 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, speaker amplifier, and several I/O devices allow the Nexys 4 to be used for a wide range of designs without needing any other components.

Features:

- Xilinx Artix-7 FPGA XC7A100T-1CSG324C
- 15,850 logic slices, each with four 6-input LUTs and 8 flip-flops
- 4,860 Kbits of fast block RAM
- Six clock management tiles, each with phase-locked loop (PLL)
- 240 DSP slices
- Internal clock speeds exceeding 450 MHz
- On-chip analog-to-digital converter (XADC)
- 16Mbyte CellularRAM®
- Serial flash
- Digilent USB-JTAG port for FPGA programming and communication
- microSD card connector
- Ships with rugged plastic case and USB cable
- USB-UART Bridge
- 10/100 Ethernet PHY
- PWM audio output
- 3-axis accelerometer
- 16 user switches
- 16 user LEDs
- Two tri-color LEDs
- PDM microphone
- Temperature sensor
- Two 4-digit 7-segment displays
- USB HID host for mice, keyboards, and memory sticks
- Pmod for XADC signals
- Four Pmod ports
- 12-bit VGA output

You must correctly choose FPGA model on ISE because it compile the project and produce “.bit” file according to FPGA which was chosen. You can see the parts of the board in the figure below.

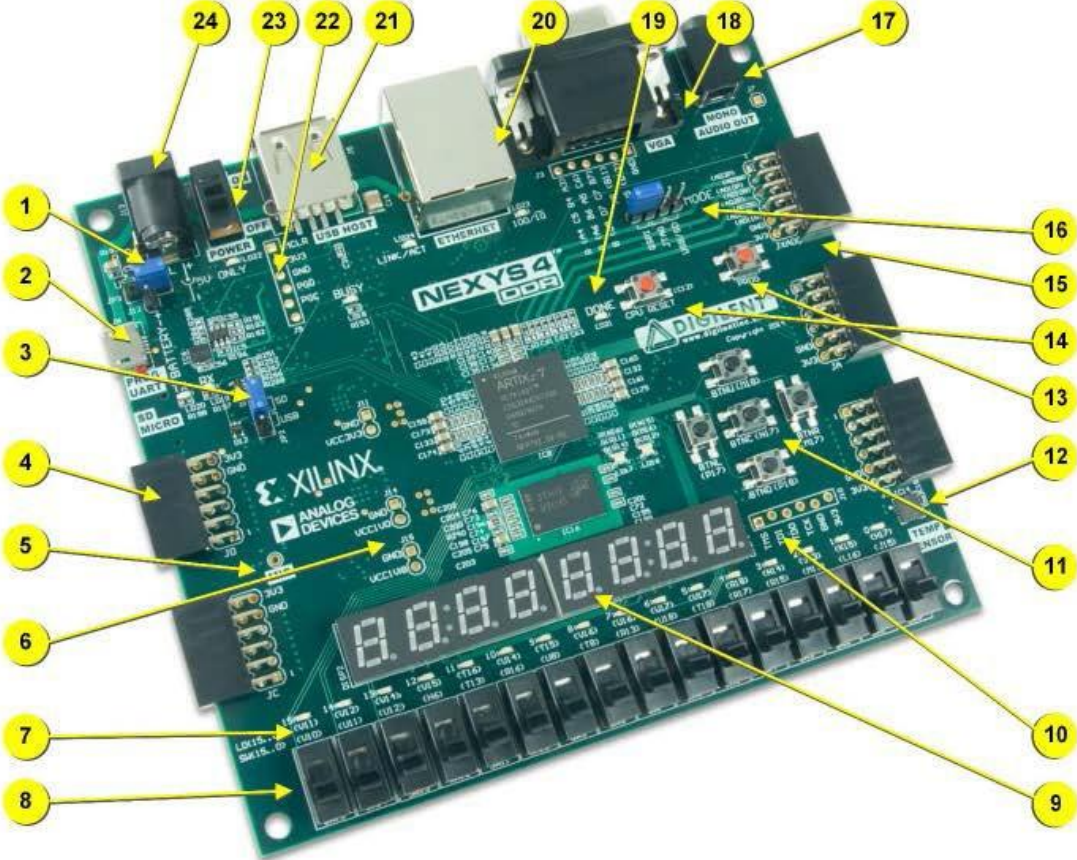
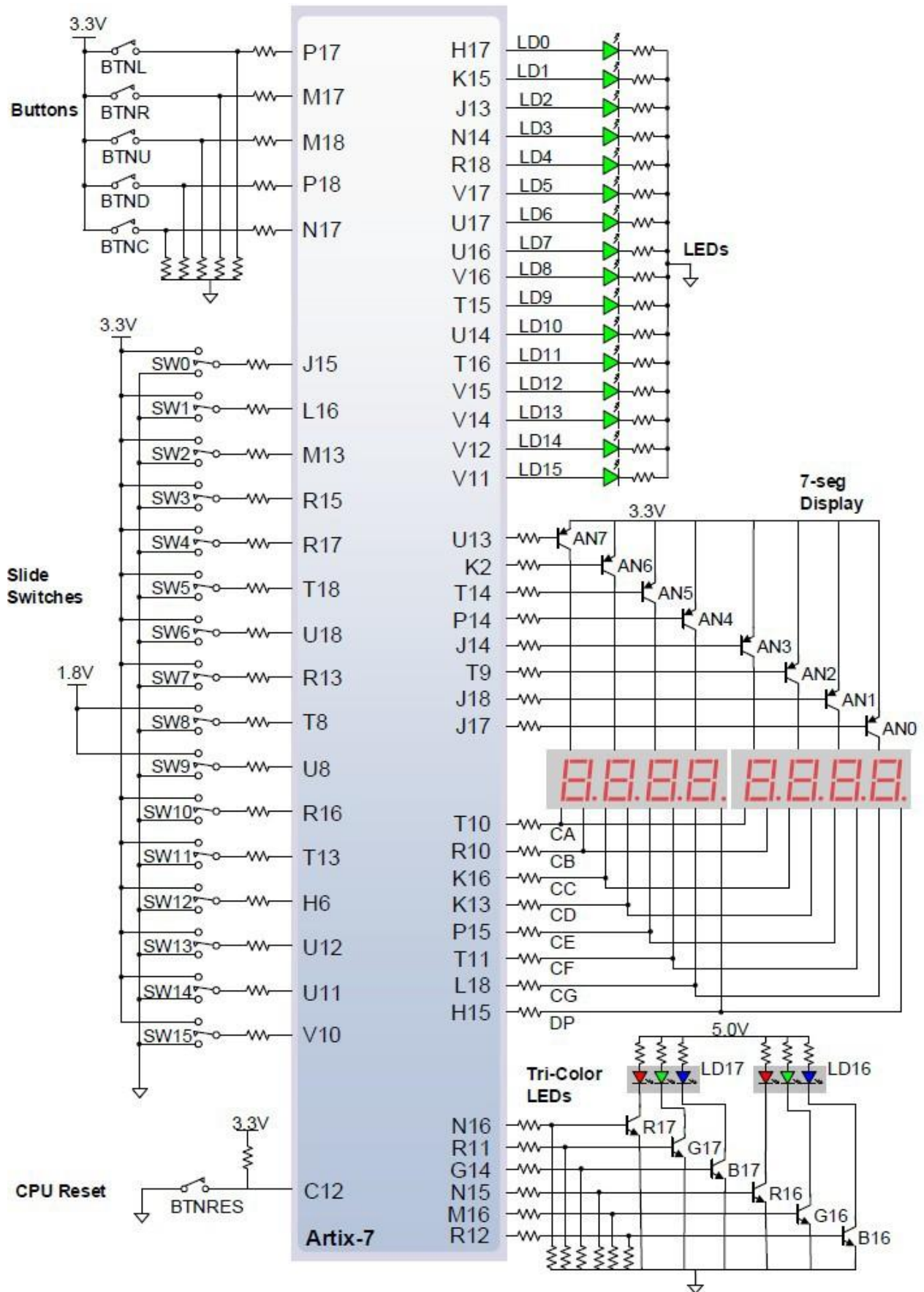


Figure 1. Nexys4 DDR board features.

| Callout | Component Description | Callout | Component Description |
|---------|--|---------|--------------------------------------|
| 1 | Power select jumper and battery header | 13 | FPGA configuration reset button |
| 2 | Shared UART/ JTAG USB port | 14 | CPU reset button (for soft cores) |
| 3 | External configuration jumper (SD / USB) | 15 | Analog signal Pmod port (XADC) |
| 4 | Pmod port(s) | 16 | Programming mode jumper |
| 5 | Microphone | 17 | Audio connector |
| 6 | Power supply test point(s) | 18 | VGA connector |
| 7 | LEDs (16) | 19 | FPGA programming done LED |
| 8 | Slide switches | 20 | Ethernet connector |
| 9 | Eight digit 7-seg display | 21 | USB host connector |
| 10 | JTAG port for (optional) external cable | 22 | PIC24 programming port (factory use) |
| 11 | Five pushbuttons | 23 | Power switch |
| 12 | Temperature sensor | 24 | Power jack |

Basic I/O pins that u may need to use is given in the figure below. Switches, LEDs and the seven segments are essential for you. Pins of the I/O elements are also given in the figure.

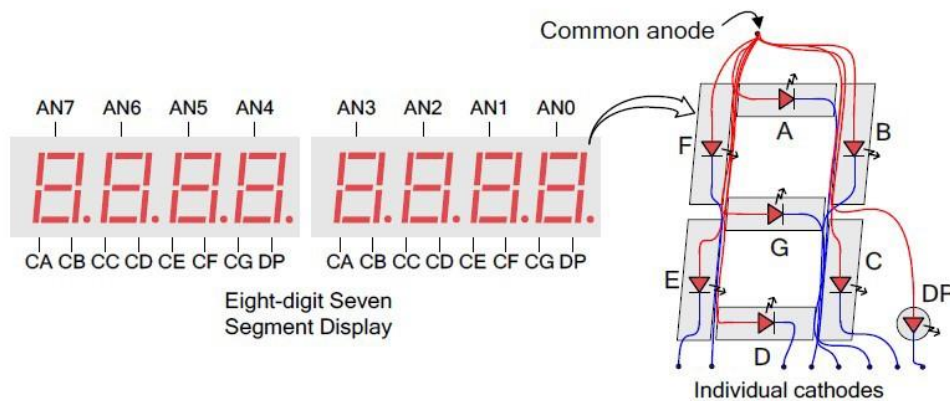


Seven Segments

The Nexys4 DDR board contains two four-digit common anode seven-segment LED displays, configured to behave like a single eight-digit display. Each of the eight digits is composed of seven segments arranged in a “figure 8” pattern, with an LED embedded in each segment.

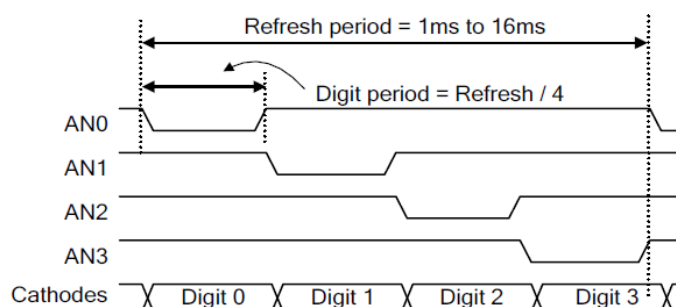
The LED control signals are time-multiplexed to display data on all four characters, as shown in below. Present the value to be displayed on the segment control inputs and select the specified character by driving the associated anode control signal Low. Through persistence of vision, the human brain perceives that all four characters appear simultaneously, similar to the way the brain perceives a TV display.

To illuminate a segment, the anode should be driven high while the cathode is driven low. However, since the Nexys4 DDR uses transistors to drive enough current into the common anode point, the anode enables are inverted. Therefore, both the AN0..7 and the CA..G/DP signals are driven low when active.



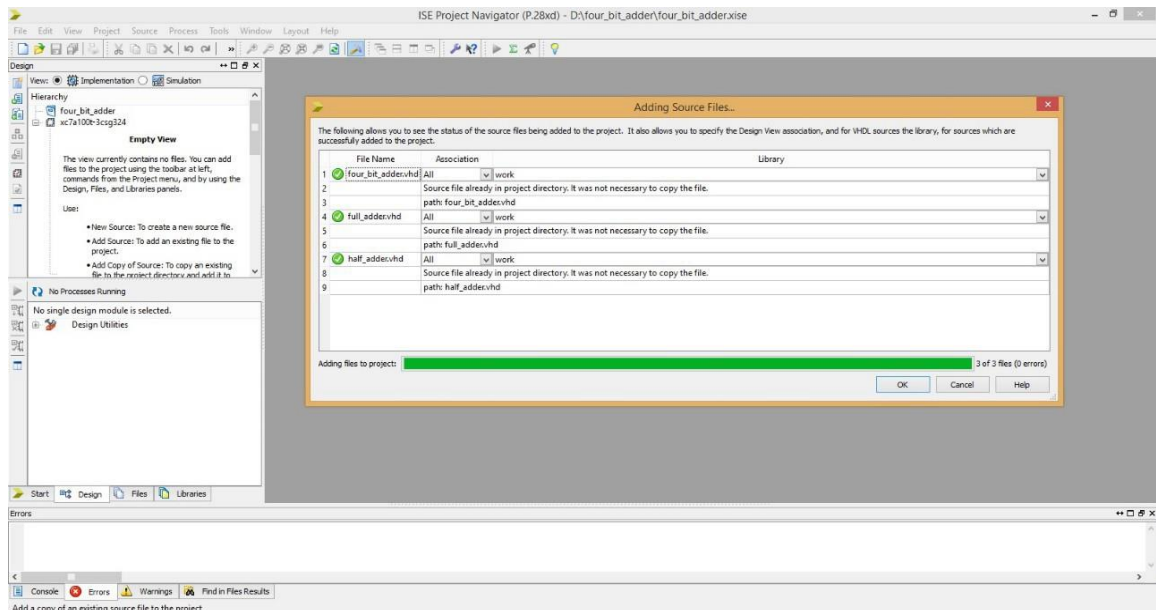
A scanning display controller circuit can be used to show an eight-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession at an update rate that is faster than the human eye can detect. Each digit is illuminated just one-eighth of the time, but because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears continuously illuminated. If the update, or “refresh”, rate is slowed to around 45Hz, a flicker can be noticed in the display.

For each of the four digits to appear bright and continuously illuminated, all eight digits should be driven once every 1 to 16ms, for a refresh frequency of about 1 KHz to 60Hz. For example, in a 62.5Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for 1/8 of the refresh cycle, or 2ms. The controller must drive low the cathodes with the correct pattern when the corresponding anode signal is driven high. To illustrate the process, if AN0 is asserted while CB and CC are asserted, then a “1” will be displayed in digit position 1. Then, if AN1 is asserted while CA, CB, and CC are asserted, a “7” will be displayed in digit position 2. If AN0, CB, and CC are driven for 4ms, and then AN1, CA, CB, and CC are driven for 4ms in an endless succession, the display will show “71” in the first two digits. An example timing diagram for a four-digit controller is shown in the figure below.

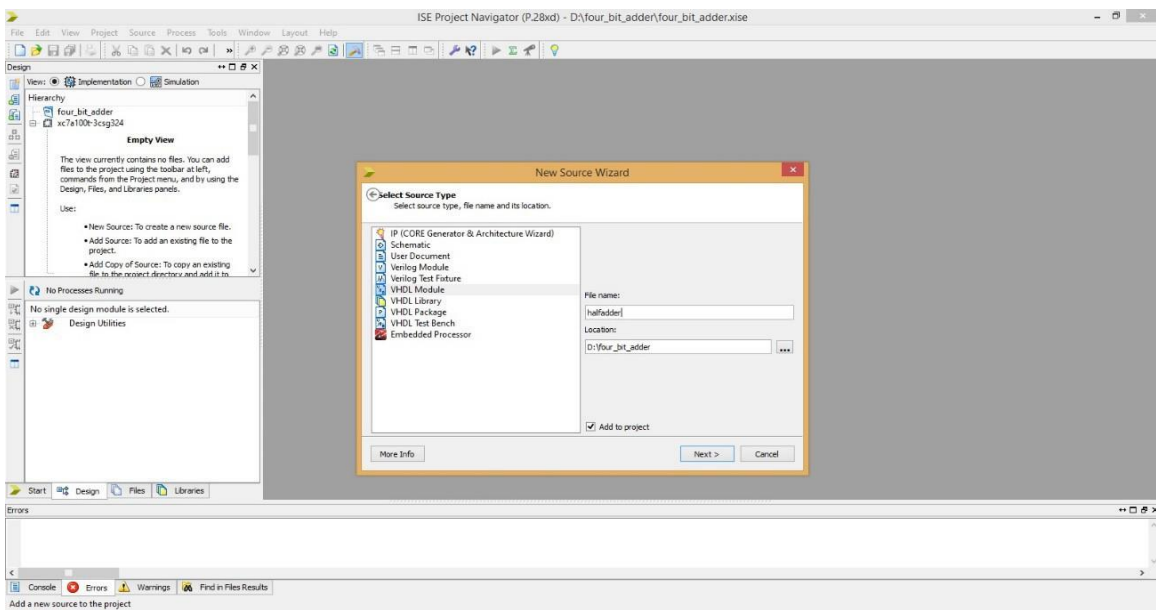


3. PROCEDURE

1. Open Xilinx ISE
2. If a project already opened you can close it by choosing “close project” on file menu
3. Create a new VHDL project like schematic one an name it “four_bit_adder_HDL”
4. After creation complete add two VHDL file to the project by using “Add copy of sources” on “project” menu.
5. Click ok to continue



6. Also you can create new VHDL module and copy the codes given above to build four bit adder circuit.



7. To test the design you should make simulation like schematic design. Add a VHDL test bench, Write the input combinations and then simulate. Simulation result must be the same as schematic design.

- We will add some other VHD files to the previous project to be able to see the results on seven segment displays. To assign package pins, you need to have a user constraint file in your design.

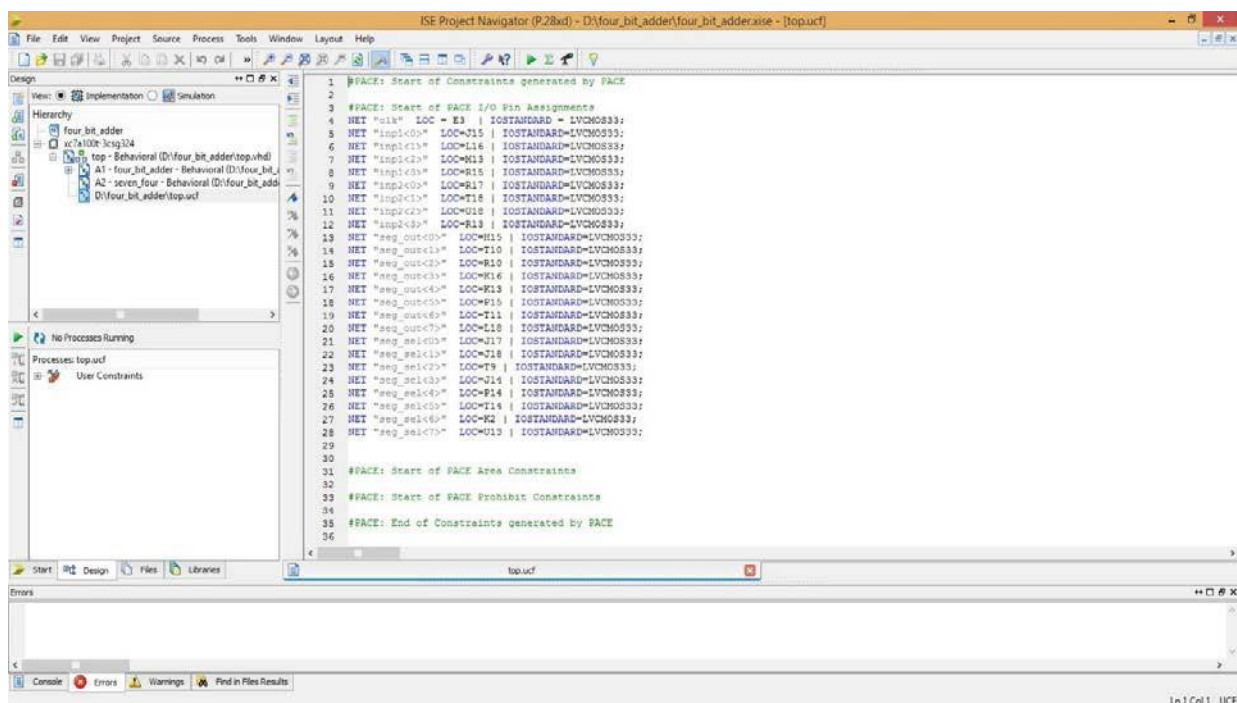
To do that, we must add given VHDL codes and UCF file given below.

- Top.vhd
- Seven_four.vhd
- Top.ucf

To assign the pins, you need a ucf file. To add this file, you can right click on your project, click on new source and select the Implementation constraints file. The figure below shows the ucf file for this project.

NET "clk" LOC = E3 | IOSTANDARD = LVCMOS33

Is an example pin assignment. User defined a input pin named clk and want to assign it to 100 MHz clock oscillator on the board. That pin is E3. The IOSTANDARD of that pin is LVCMOS33. This IOSTANDARD is not necessary can be turned off by some configurations.

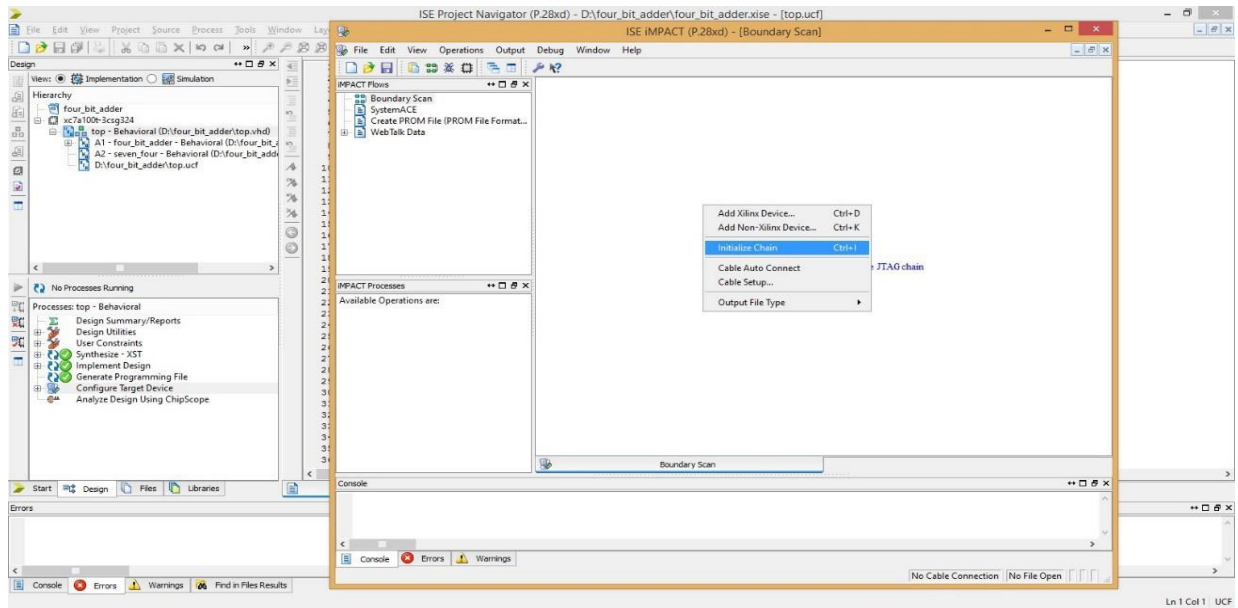


The screenshot shows the ISE Project Navigator interface. The main window displays the contents of a User Constraints File (UCF) named 'top.ucf'. The file contains the following text:

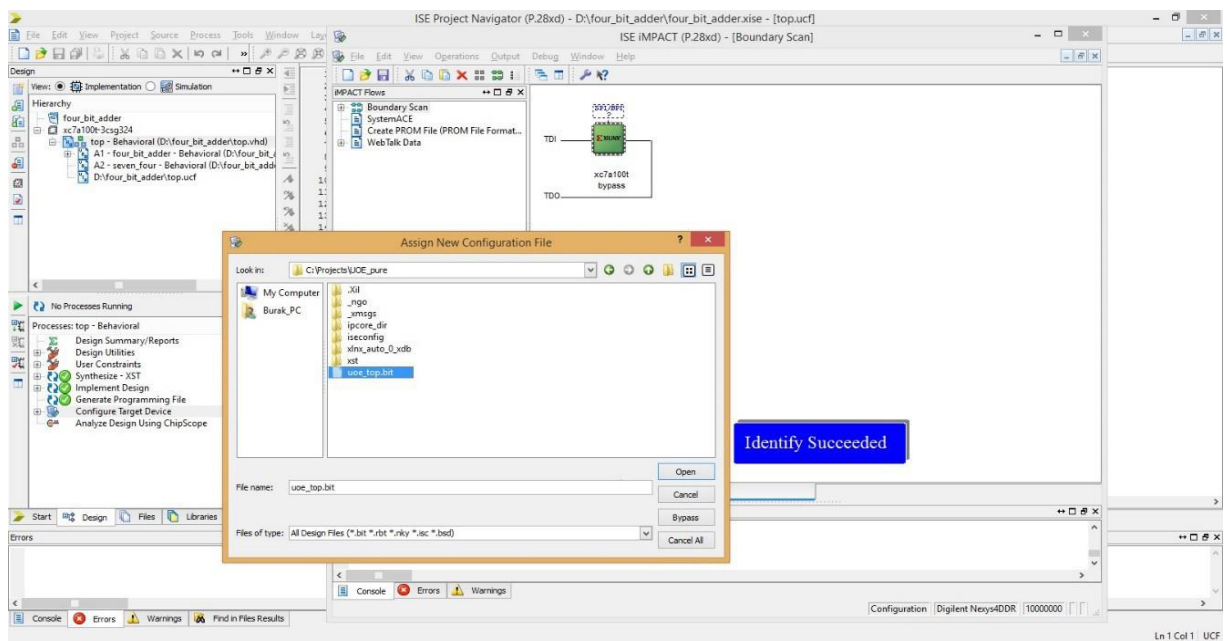
```
1 #FACE: Start of Constraints generated by FACE
2
3 #FACE: Start of FACE I/O Pin Assignment
4 NET "clk" LOC = E3 | IOSTANDARD = LVCMOS33;
5 NET "inp1<0>" LOC=R15 | IOSTANDARD=LVCMOS33;
6 NET "inp1<1>" LOC=R16 | IOSTANDARD=LVCMOS33;
7 NET "inp1<2>" LOC=R13 | IOSTANDARD=LVCMOS33;
8 NET "inp1<3>" LOC=R15 | IOSTANDARD=LVCMOS33;
9 NET "inp2<0>" LOC=R17 | IOSTANDARD=LVCMOS33;
10 NET "inp2<1>" LOC=T18 | IOSTANDARD=LVCMOS33;
11 NET "inp2<2>" LOC=R18 | IOSTANDARD=LVCMOS33;
12 NET "inp2<3>" LOC=R13 | IOSTANDARD=LVCMOS33;
13 NET "seg_out<0>" LOC=R15 | IOSTANDARD=LVCMOS33;
14 NET "seg_out<1>" LOC=T10 | IOSTANDARD=LVCMOS33;
15 NET "seg_out<2>" LOC=R10 | IOSTANDARD=LVCMOS33;
16 NET "seg_out<3>" LOC=R16 | IOSTANDARD=LVCMOS33;
17 NET "seg_out<4>" LOC=R13 | IOSTANDARD=LVCMOS33;
18 NET "seg_out<5>" LOC=R18 | IOSTANDARD=LVCMOS33;
19 NET "seg_out<6>" LOC=T11 | IOSTANDARD=LVCMOS33;
20 NET "seg_out<7>" LOC=L18 | IOSTANDARD=LVCMOS33;
21 NET "seg_sel<0>" LOC=U17 | IOSTANDARD=LVCMOS33;
22 NET "seg_sel<1>" LOC=T18 | IOSTANDARD=LVCMOS33;
23 NET "seg_sel<2>" LOC=T9 | IOSTANDARD=LVCMOS33;
24 NET "seg_sel<3>" LOC=U14 | IOSTANDARD=LVCMOS33;
25 NET "seg_sel<4>" LOC=P14 | IOSTANDARD=LVCMOS33;
26 NET "seg_sel<5>" LOC=T14 | IOSTANDARD=LVCMOS33;
27 NET "seg_sel<6>" LOC=R2 | IOSTANDARD=LVCMOS33;
28 NET "seg_sel<7>" LOC=U13 | IOSTANDARD=LVCMOS33;
29
30
31 #FACE: Start of FACE Area Constraints
32
33 #FACE: Start of FACE Prohibit Constraints
34
35 #FACE: End of Constraints generated by FACE
36
```

The interface also shows a Hierarchy window on the left with the project structure, and a Processes window at the bottom showing 'User Constraints'.

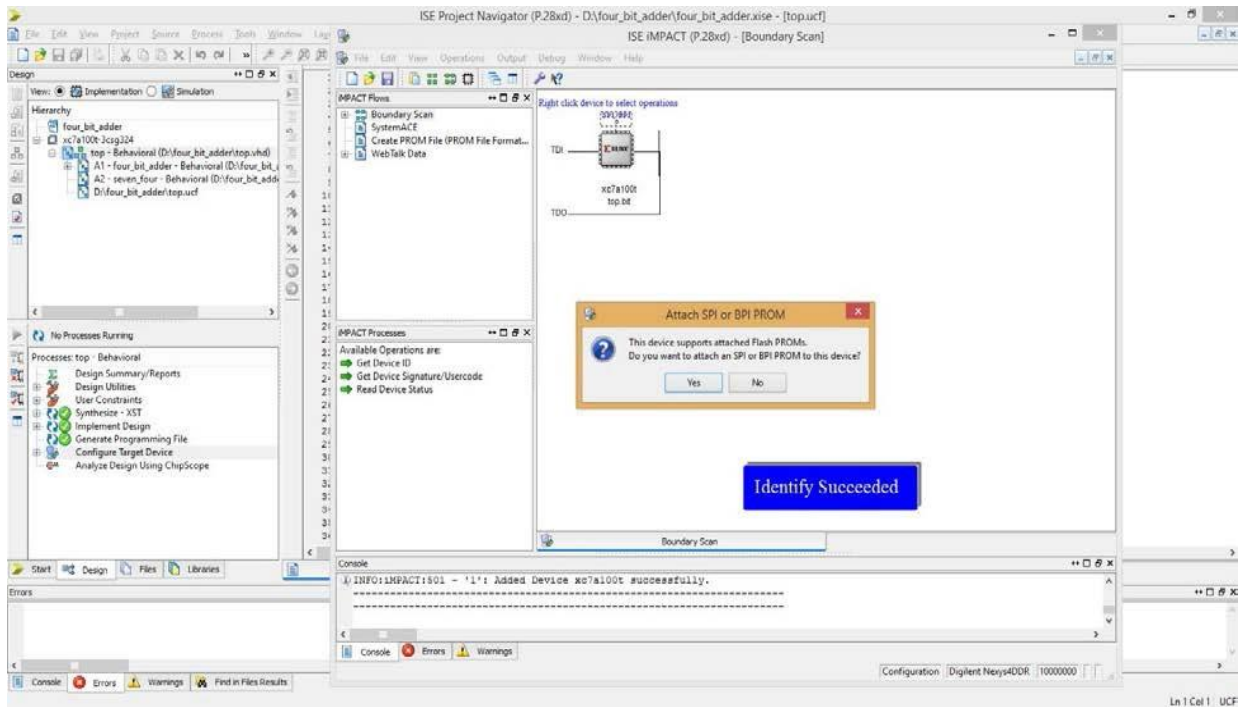
- After assigning pins we can load design to FPGA with using IMPACT utility. To open IMPACT you should double click "Configure Device (IMPACT)" on processes window under "Generate Programming file" menu. Before doing that you must synthesize and implement design. If you don't do that ISE do it automatically. Also you can open IMPACT independently on Xilinx Accessories menu to load ".bit" file. If there is an error in your code, program will not synthesize your code.



10. When you click on the configure target device, a new screen will open (a warning may appear, ignore it) and you need to double click on the boundary scan. A page with writing “Right click to add device or initialize jtag chain” on it will appear. You will right click and select the initialize chain after you connect your board to the computer and switch on the “ON/OFF” switch.



11. The program will find the device, then a box asking for adding a configuration file will appear. You can say yes to it and a screen like the image above will open. You will select the .bit file and click on the open button.



12. You will say no to the opened box. Right click on the device and click on the program and click ok on the opened box. Finally this will program the board and your code will be executed. Sum of the two inputs can be seen on the seven segments. You can change the switches to verify the functionality of your design.

