CpE 487 Digital Design Lab

Lab 1: Seven Segment Decoder

1. Getting Started

The purpose of this lab is to introduce you to the NEXYS2 FPGA development board and a software environment that will allow you to design a hardware system in VHDL, synthesize your design to create an FPGA configuration file, download the configuration file on to an FPGA and then run the configured FPGA to achieve the desired hardware functionality.

1.1 Hardware

We will be using the Digilent Nexys2 FPGA development board shown in Figure 1. It includes:

- 500k-gate Xilinx Spartan 3E FPGA
- 16 MB SDRAM
- 16 MB Flash ROM
- Platform Flash for holding startup configuration code
- 50MHz oscillator
- VGA, RS232 and PS/2 ports
- Simple I/O devices including 8 LEDs, 4-digit 7-seg. display, 4 buttons and 8 slide switches

The Nexys2 board connects to a host computer via a USB2 interface. The USB port provides board power (no external power supply is required) and a programming interface.

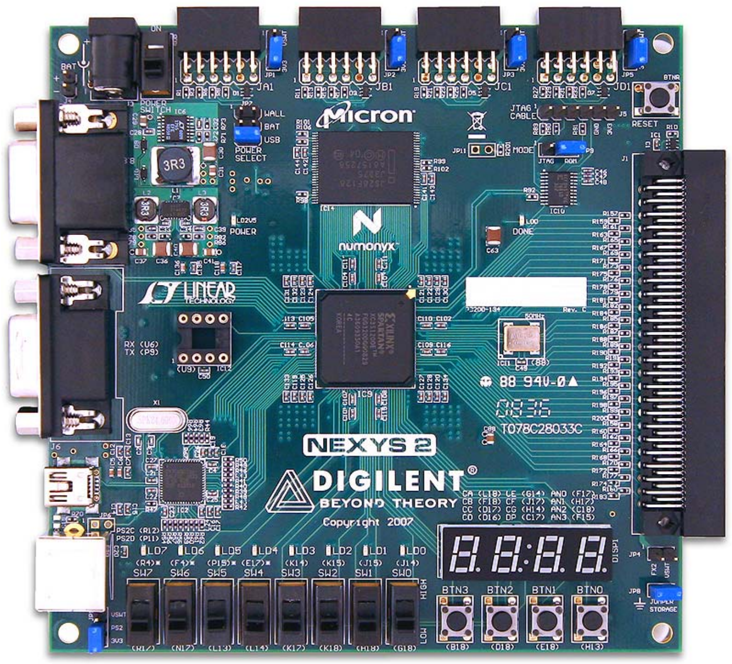


Figure 1 Nexys2 FPGA Development Board

You can find more detailed information on the Nexys2 board in the *Nexys2 Reference Manual* that is provided on the 487 Lab. web page.

1.2 Software

The Nexys2 is compatible with the *Xilinx ISE WebPack* tools that we have been using to enter and simulate VHDL designs in class. These tools will be used to (1) create a VHDL description of our desired hardware functionality, (2) synthesize the VHDL design to create an RTL description of our design, (3) map the RTL on to the configurable logic blocks and programmable routing resources of the FPGA and (4) generate a configuration file that can be used to program the FPGA according to this mapping.

In addition to the Xilinx software, we will also be using the *Digilent Adept* application. This provides a PC interface to the Nexys2 board via the USB port. *Adept* will allow you configure the FPGA, download files to the configuration FLASH and run tests to confirm correct operation. You can find more detailed information on the *Adept* application in the *Adept Users Manual* that is provided on the 487 Lab. web page.

1.3 Connecting and Testing the Nexys2 Board

Set the ON-OFF power switch on the Nexys2 board to OFF (Figure 2).

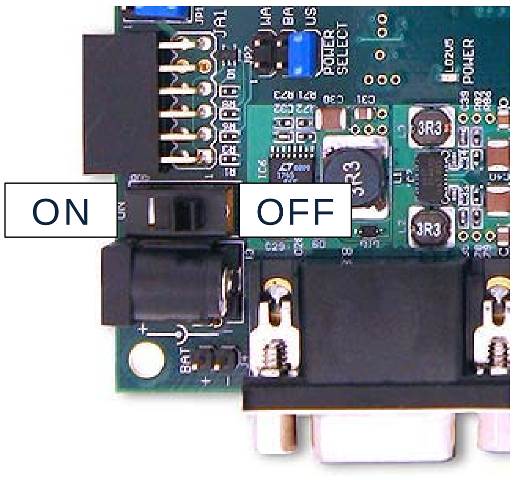


Figure 2 Power Switch

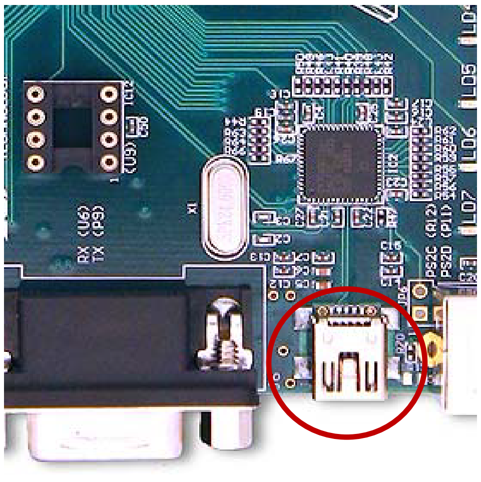
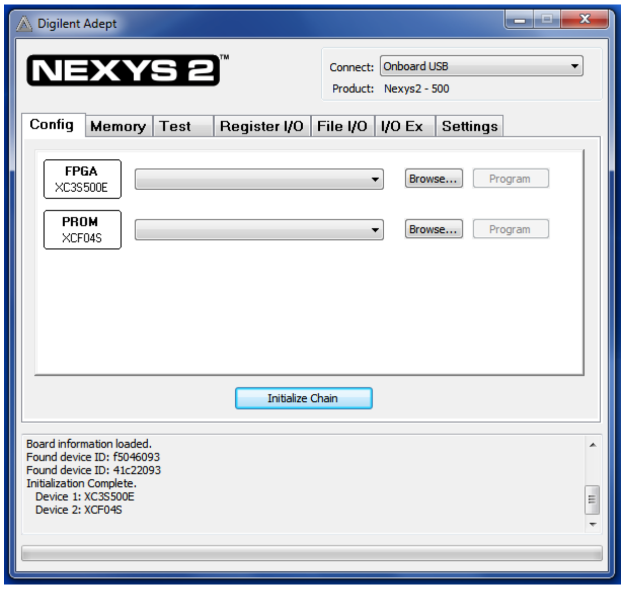


Figure 3 USB Port

Connect the board to the host computer using the supplied USB cable (Figure 3). Now turn the power switch to ON. The red POWER LED will light indicating that the board is being powered via the USB connection. Upon power up, the configuration code in the Platform Flash will be loaded into the FPGA. Once this has been completed (takes about 1 sec.), the yellow DONE LED will light up. What the FPGA does next depends on what is currently loaded in the Platform

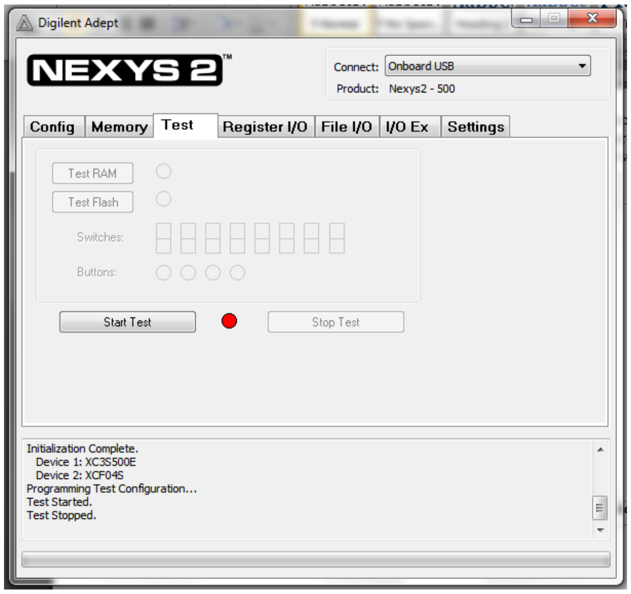
Flash ROM. It may contain self-test code (as originally loaded by the manufacturer) or it may contain code that has been loaded previously by another lab group.

Start the *Adept* software by double-clicking the Adept icon on the PC desktop. This brings up the Configuration window shown in Figure 4. The Adept software should recognize the board you have connected and show the type of FPGA and PROM that it has found on the board.



```
Board information loaded.
Found device ID: f5046093
Found device ID: 41c22093
Initialization Complete.
  Device 1: XC3S500E
  Device 2: XCF04S
```

Figure 4 Adept Configuration Window



```
Initialization Complete.
  Device 1: XC3S500E
  Device 2: XCF04S
Programming Test Configuration...
Test Started.
Test Stopped.
```

Figure 5   Adept Test Window

Select the Test tab. This will bring up the Test window shown in Figure 5. Click the **Start Test** button. This will initiate a self-test of the Nexys2 board. Once the test is complete, it will show “PASS” on the 7-segment displays and the red indicator on the Test page will change to green.

Once the test is complete, try turning on and off the eight black slide switches on the board (Figure 6). The corresponding LED on the board will turn on and off. Also the switch schematic on the Test window will change to indicate the selected switch position. Now try pushing each of the four test buttons and note its effect on the Test Window.

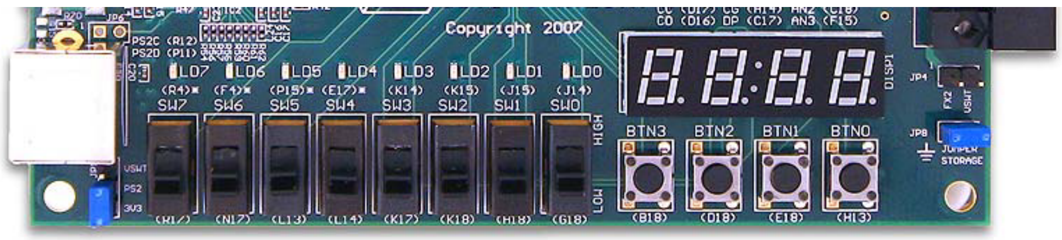


Figure 6   Switches, buttons & LEDs

2. LED Decoder

Our first FPGA design will be an LED Decoder that will take as input a 4-bit binary word from four of the slide switches and display its value as a HEX digit on one of the seven segment displays. Each of the seven segment displays consists of seven LED segments A through G which share a common anode as shown in Figure 7.

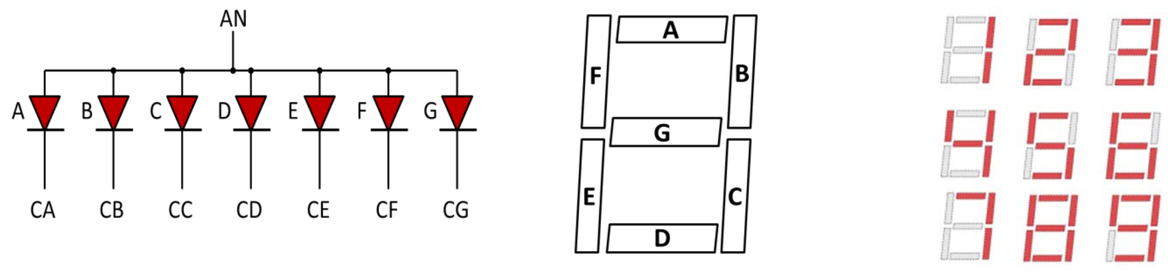


Figure 7 Common anode 7-segment display

Different patterns can be created by turning on selected segments as shown in the figure. For example, to make the digit “1’, we would turn on segments B and C. A positive voltage (logical ‘1’) is applied to the common anode AN. Each segment can then be turned on by setting the appropriate cathode to ground (logical ‘0’). The four 7-segment displays on the Nexys2 board share the segment cathode lines as shown in Figure 8.

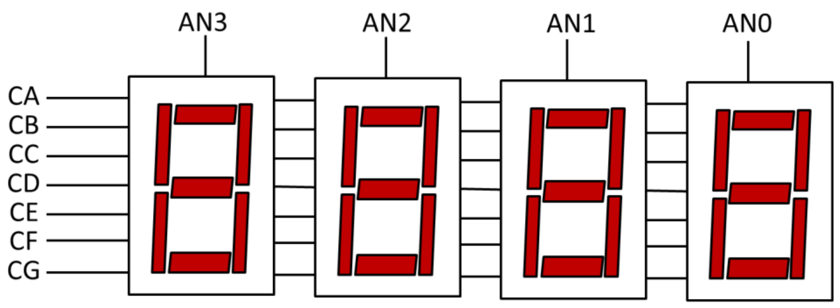


Figure 8 Shared cathode connection

Any one of the 7-segment displays can be enabled by setting the appropriate anode high and then driving the segment code on to the CA – CG lines. Multiple displays can be used by time multiplexing the AN0-3 signals. We will learn more about this in Lab. 2. For now, we will just be using one 7-segment display at a time.

The way in which the switches, button, LEDs and 7-segment displays are connected to the FPGA is shown in Figure 9. The codes inside the FPGA box show the actual FPGA pin that is connected to that input or output device. Note that PNP transistors that are used to drive the common anode signals for the 7-segment displays. In addition to providing current gain, these transistors invert the signal. So, for example, by applying a logical '0' to FPGA pin F17 and a '1' to H17, C18 and F15, we will "turn on" only the least significant 7-segment display.

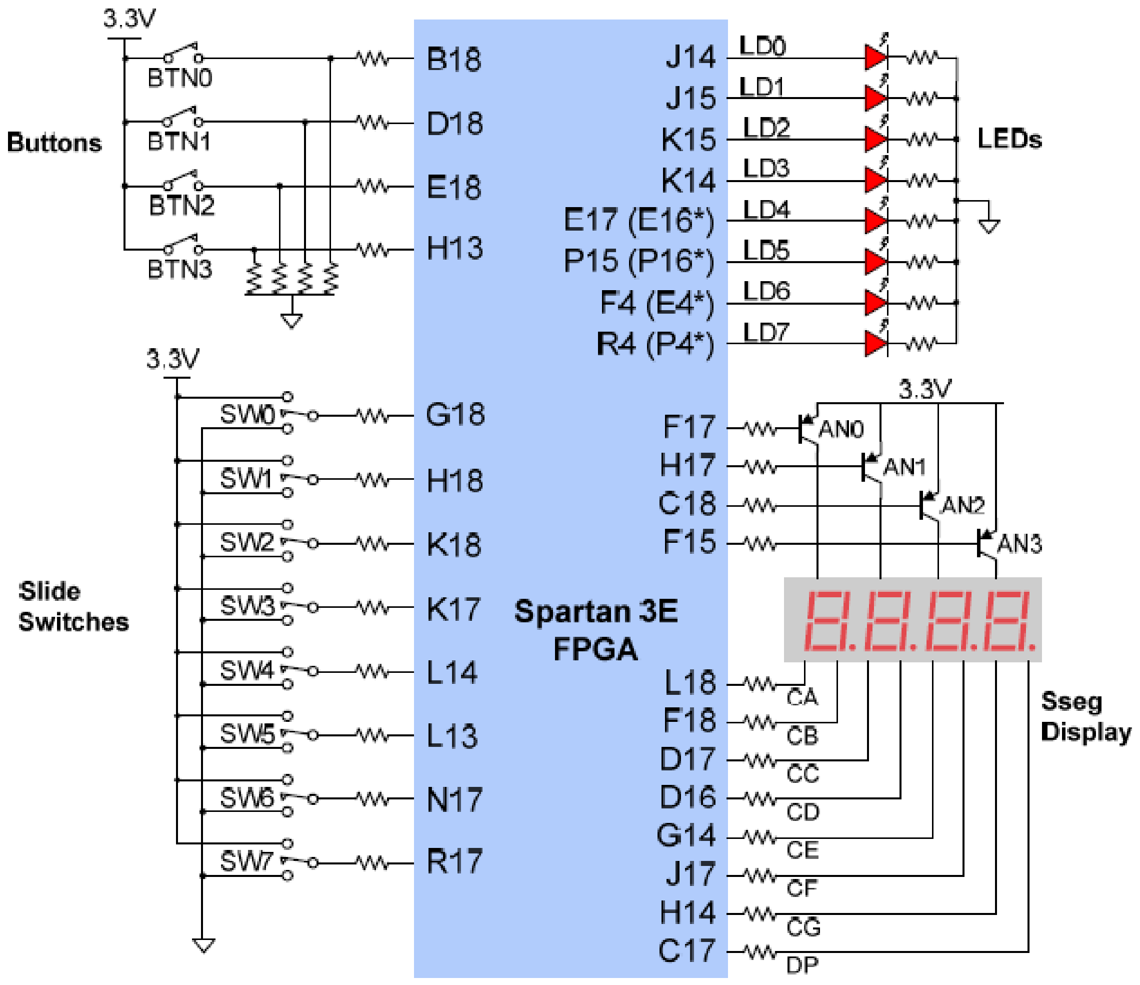


Figure 9 Nexys2 I/O Devices and Circuits

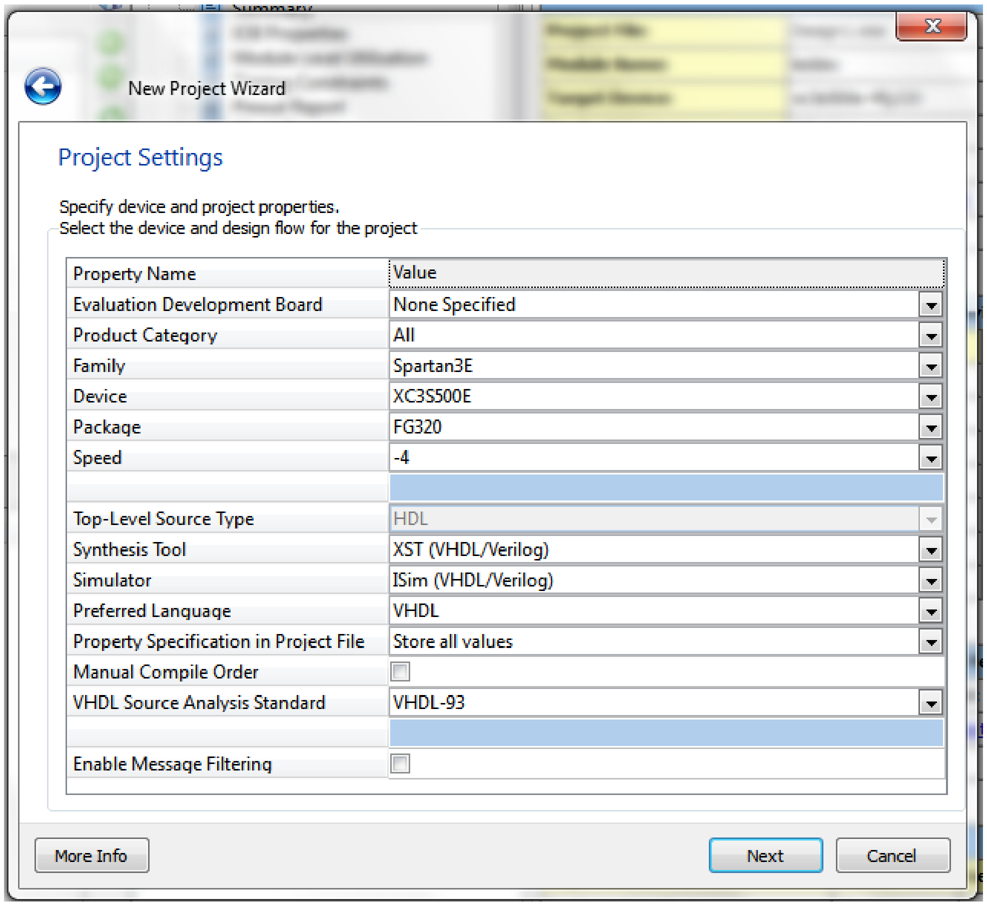
In this design, we will use slide switches SW6~SW7 to select which 7-segment display we wish to use. In addition, we will generate a 4-bit code using slide switches SW0~SW3. This code will be displayed on the selected 7-segment display using the following encodings (remember that a '0' on the cathode turns a segment "on"):

Four-bit Input	Hex Digit	LED Segment Code CA-CG
0000	0	0000001
0001	1	1001111
0010	2	0010010
0011	3	0000110
0100	4	1001100
0101	5	0100100
0110	6	0100000
0111	7	0001111
1000	8	00000000

1001	9	0000100
1010	A	0001000
1011	B	1100000
1100	C	0110001
1101	D	1000010
1110	E	0110000
1111	F	0111000

2.1 Entering VHDL Code

Start the WebPack software by double-clicking the ISE Project Navigator ICON on the desktop. Using the **File** pull-down menu select **New Project**. This brings up the *Create New Project* window. Name the new project *leddec* and click the **Next** button. This brings up the *Project Settings* window. This window is used to specify the characteristics of the FPGA that we will be using to implement our design. Enter the data shown in Figure 10 and click **Next**.



Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

Figure 10 Project Settings

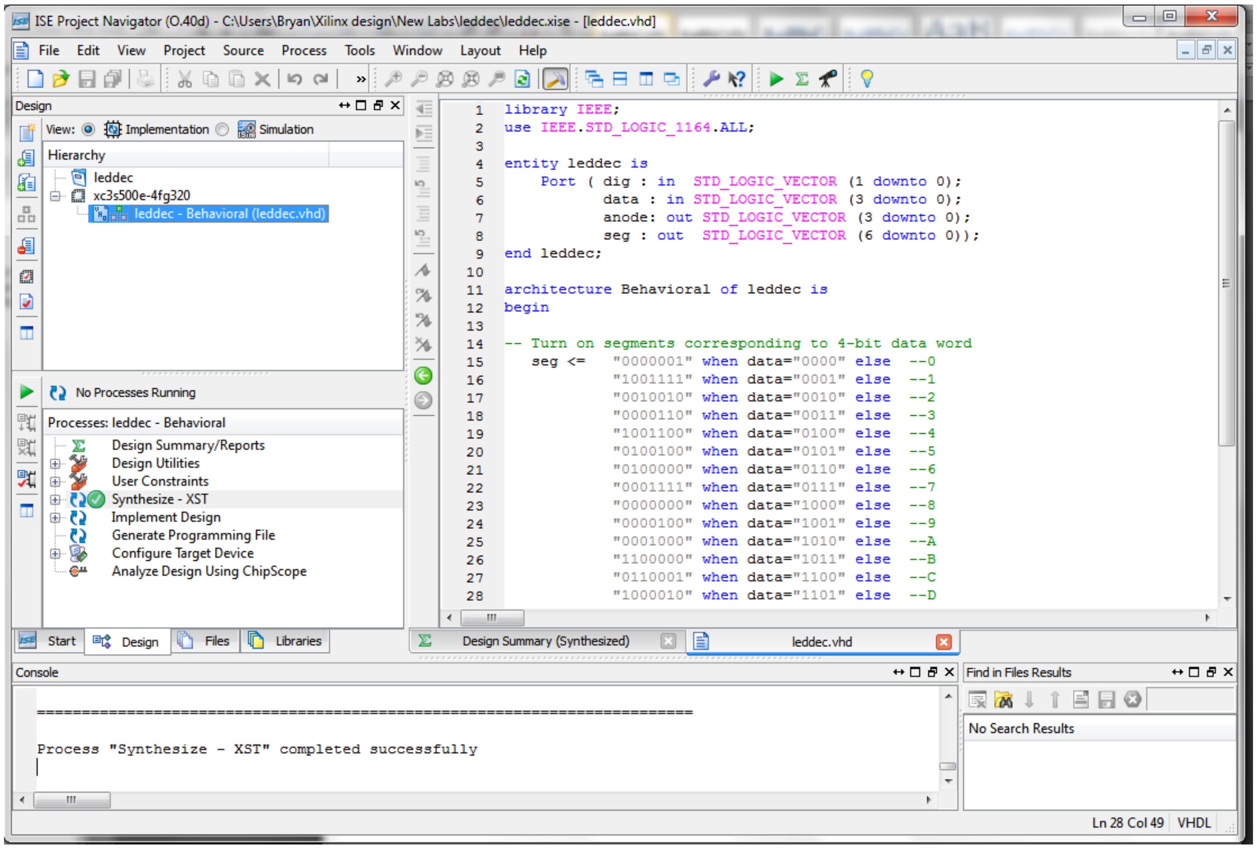
A summary window pops up. Click **Finish**. This brings up an “empty” design window for your new project. Using the **Project** pull-down menu select **New Source**. A *New Source Wizard* window pops up. Select **VHDL Module** and give it the name *leddec*. Click **Next**. Leave the *Define Module* page blank and click **Next** and then **Finish**. This generates an empty VHDL source code template for the module *leddec* in the editing pane. Delete all the code in this window and replace it with the following:

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

entity leddec is
   Port ( dig : in  STD_LOGIC_VECTOR (1 downto 0);
          data : in STD_LOGIC_VECTOR (3 downto 0);
          anode: out STD_LOGIC_VECTOR (3 downto 0);
          seg : out  STD_LOGIC_VECTOR (6 downto 0));
end leddec;

architecture Behavioral of leddec is
begin

-- Turn on segments corresponding to 4-bit data word
        seg <=  "0000001" when data="0000" else      --0
                           "1001111" when data="0001" else      --1
                           "0010010" when data="0010" else      --2
                           "0000110" when data="0011" else      --3
                           "1001100" when data="0100" else      --4
                           "0100100" when data="0101" else      --5
                           "0100000" when data="0110" else      --6
                           "0001111" when data="0111" else      --7
                           "0000000" when data="1000" else      --8
                           "0000100" when data="1001" else      --9
                           "0001000" when data="1010" else      --A
                           "1100000" when data="1011" else      --B
                           "0110001" when data="1100" else      --C
                           "1000010" when data="1101" else      --D
                           "0110000" when data="1110" else      --E
                           "0111000" when data="1111" else      --F
                           "1111111";

-- Turn on anode of 7-segment display addressed by 2-bit digit selector dig
        anode <=      "1110" when dig="00" else      --0
                           "1101" when dig="01" else      --1
                           "1011" when dig="10" else      --2
                           "0111" when dig="11" else      --3
                           "1111";
end Behavioral;
```

The module *leddec* has two inputs *dig* and *data*. The input *dig* selects which 7-segment digit should be turned on. A conditional signal assignment statement is used to select the appropriate common *anode*. The input *data* is the 4-bit code which defines the HEX number to be displayed. A second conditional signal assignment statement is used to select which cathodes should be enabled using the output *seg*.

2.2 Synthesizing the VHDL

Make sure the radio *View* button is set to *Implementation* in the *Design* window. Select the *leddec* object in the *Hierarchy* window and then double click the *Synthesis* process in the *Process* window as shown in Figure 11. Once the synthesis completes, a green tick mark will appear beside the process.


ISE Project Navigator (O.40d) - C:\Users\Bryan\Xilinx design\New Labs\leddec\leddec.xise - [leddec.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
leddec
xc3s500e-4fg320
leddec - Behavioral (leddec.vhd)
No Processes Running
Processes: leddec - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start
Design
Files
Libraries
Design Summary (Synthesized)
leddec.vhd
Console
Find in Files Results
No Search Results
Ln 28 Col 49
VHDL

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

entity leddec is
    Port ( dig : in  STD_LOGIC_VECTOR (1 downto 0);
           data : in STD_LOGIC_VECTOR (3 downto 0);
           anode: out STD_LOGIC_VECTOR (3 downto 0);
           seg : out  STD_LOGIC_VECTOR (6 downto 0));
end leddec;

architecture Behavioral of leddec is
begin

-- Turn on segments corresponding to 4-bit data word
   seg <=   "0000001" when data="0000" else  --0
            "1001111" when data="0001" else  --1
            "0010010" when data="0010" else  --2
            "0000110" when data="0011" else  --3
            "1001100" when data="0100" else  --4
            "0100100" when data="0101" else  --5
            "0100000" when data="0110" else  --6
            "0001111" when data="0111" else  --7
            "0000000" when data="1000" else  --8
            "0000100" when data="1001" else  --9
            "0001000" when data="1010" else  --A
            "1100000" when data="1011" else  --B
            "0110001" when data="1100" else  --C
            "1000010" when data="1101" else  --D
```

```
==========================================================================

Process "Synthesize - XST" completed successfully
```

Figure 11   Synthesis Process

2.3 Implementing the FPGA Configuration

The implementation process consists of three parts: (1) translating the synthesized RTL into primitives available on the FPGA, (2) mapping these primitives to actual physical locations in the FPGA and (3) Routing these physical primitives together. Before we map our design to the FPGA, we need to tell the software which FPGA I/O pins we wish to use for our inputs and outputs. This is done by creating an implementation constraints file.

Select **Project→New Source** and then select **Implementation Constraints File** with filename *leddec.ucf*. Click **Next** and **Finish**. A blank editing window appears. Insert the following code:

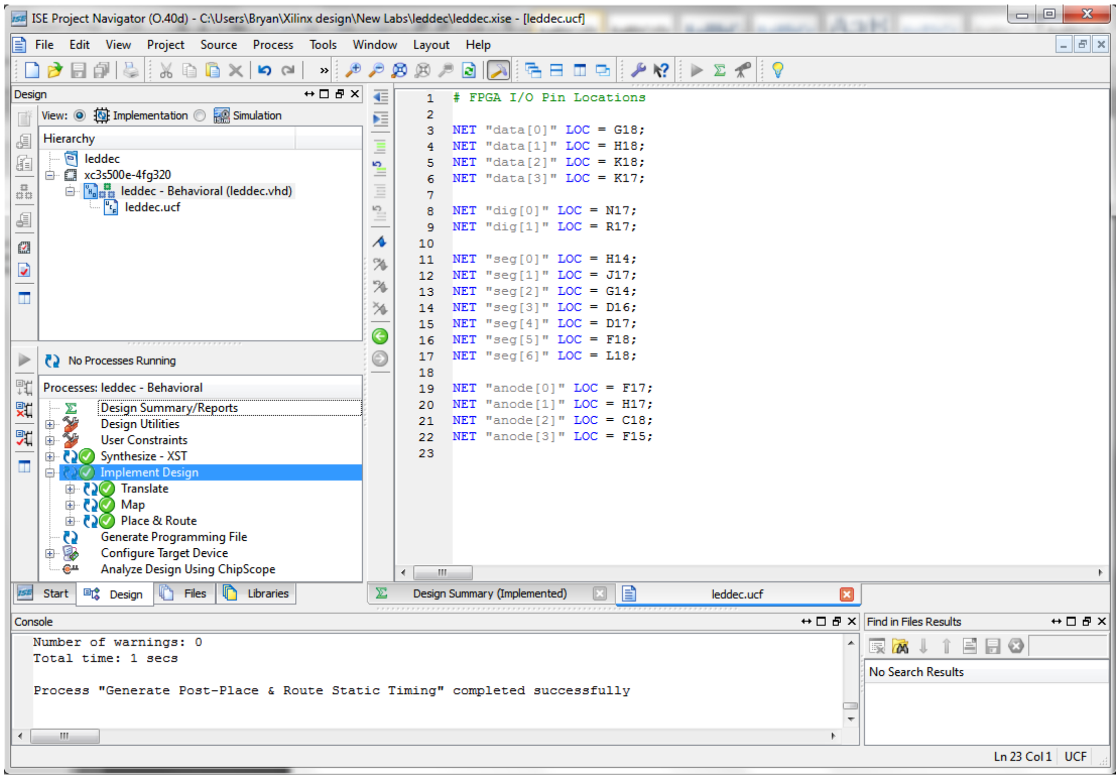
```
# FPGA I/O Pin Locations

NET "data[0]" LOC = G18;
NET "data[1]" LOC = H18;
NET "data[2]" LOC = K18;
NET "data[3]" LOC = K17;

NET "dig[0]" LOC = N17;
NET "dig[1]" LOC = R17;

NET "seg[0]" LOC = H14;
NET "seg[1]" LOC = J17;
NET "seg[2]" LOC = G14;
NET "seg[3]" LOC = D16;
NET "seg[4]" LOC = D17;
NET "seg[5]" LOC = F18;
NET "seg[6]" LOC = L18;

NET "anode[0]" LOC = F17;
NET "anode[1]" LOC = H17;
NET "anode[2]" LOC = C18;
NET "anode[3]" LOC = F15;
```

This assigns FPGA I/O buffers and pins to our inputs and outputs. These pin numbers can be found in the *Nexys2 Reference Manual*. Highlight the *leddec.vhd* module in the *Hierarchy* window. Expand the *Implementation* process in the *Process* window (so you can watch the progression of the implementation). Double click the **Implementation** process as shown in Figure 12. Green tick marks will indicate successful completion

2.4 Generating the FPGA Configuration File

The final step is to create the configuration file that will be actually loaded into the FPGA to realize our design. Before we do this we need to make one change to the default setup. Right click on the **Generate Programming File** process in the *Process* Window and select **Process Properties**. A *Process Properties* window pops up. Select **Startup Options**. Change *the FPGA startup clock* from **CCLK** to **JTAG Clock**. Click **OK**.

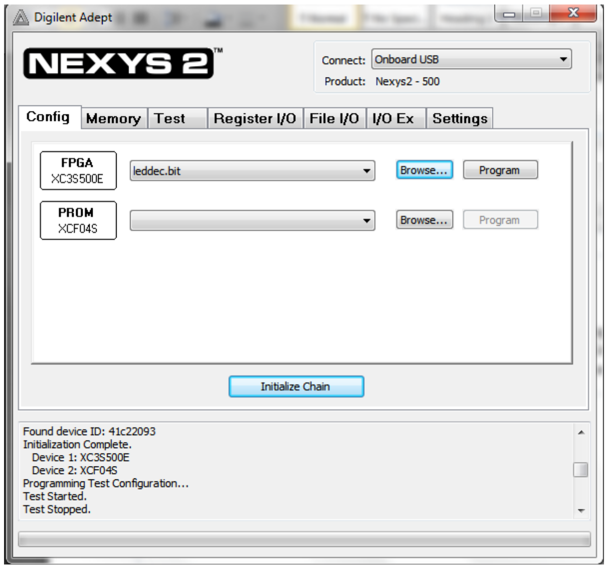
Now double-click the **Generate Programming File** process in the *Process* window. This will generate the *leddec.bit* file for downloading to the FPGA.


ISE Project Navigator (O.40d) - C:\Users\Bryan\Xilinx design\New Labs\leddec\leddec.xise - [leddec.ucf]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
leddec
xc3s500e-4fg320
leddec - Behavioral (leddec.vhd)
leddec.ucf
No Processes Running
Processes: leddec - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start Design Files Libraries
Design Summary (Implemented)
leddec.ucf
Console
Find in Files Results
No Search Results
Ln 23 Col 1 UCF

```
# FPGA I/O Pin Locations

NET "data[0]" LOC = G18;
NET "data[1]" LOC = H18;
NET "data[2]" LOC = K18;
NET "data[3]" LOC = K17;

NET "dig[0]" LOC = N17;
NET "dig[1]" LOC = R17;

NET "seg[0]" LOC = H14;
NET "seg[1]" LOC = J17;
NET "seg[2]" LOC = G14;
NET "seg[3]" LOC = D16;
NET "seg[4]" LOC = D17;
NET "seg[5]" LOC = F18;
NET "seg[6]" LOC = L18;

NET "anode[0]" LOC = F17;
NET "anode[1]" LOC = H17;
NET "anode[2]" LOC = C18;
NET "anode[3]" LOC = F15;
```

```
Number of warnings: 0
Total time: 1 secs

Process "Generate Post-Place & Route Static Timing" completed successfully
```

Figure 12   Implementation Process



Connect: Onboard USB
Product: Nexys2 - 500

FPGA XC3S500E: leddec.bit
PROM XCF04S:

```
Found device ID: 41c22093
Initialization Complete.
  Device 1: XC3S500E
  Device 2: XCF04S
Programming Test Configuration...
Test Started.
Test Stopped.
```

Figure 13   Configuration file ready for downloading

2.5 Downloading the FPGA Configuration File

We use the Adept application to download the configuration file. Bring up the Adept application. Select the **Config** tab. Use the **Browse** button on the FPGA line to locate the configuration file *leddec.bit* . The Adept window should now appear as Figure 13. Click the **Program** button. The yellow DONE LED will go out on the Nexys2 board while the configuration is loading. Once loading is complete, the DONE LED will light up and your FPGA program will be running.

There should now be a single 7-segment decoder illuminated with a HEX digit. Slider switches 0~3 will select the value of 4-bit HEX digit. Slider switches 6~7 will determine which display is illuminated.

3. <u>Hex Counter</u>

Our second design will be a HEX counter as shown in Figure 14. We will build a 4-bit counter that is driven by a clock derived from the on-board 50 MHz oscillator. We will display the 4-bit output as a HEX character using the LED decoder we have already built. We will set the *dig* input to the LED decoder to be "00" so that the count will display on the least significant 7-segment display.

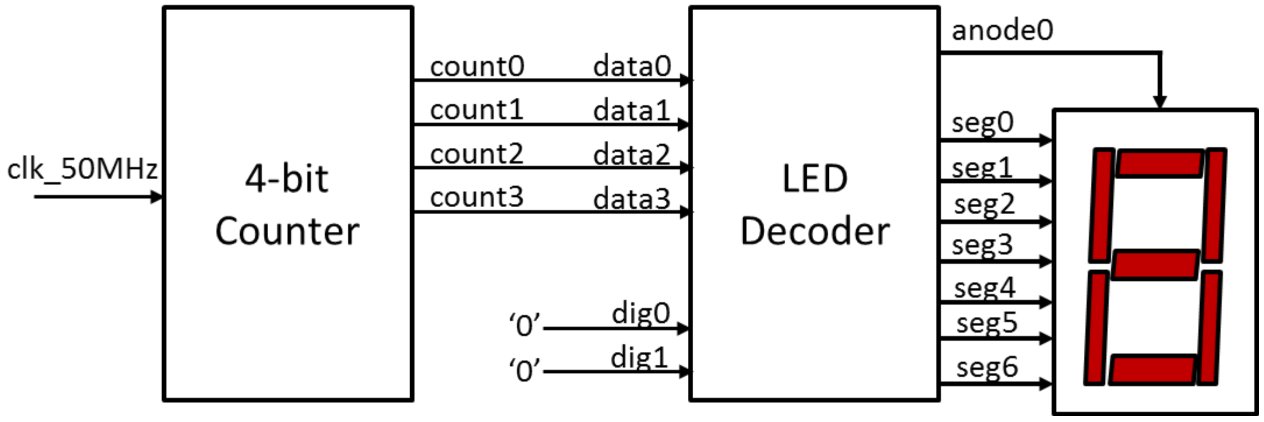


Figure 14 HEX Counter

2.1 Re-using LED Decoder VHDL Code

Create a new project named *hexcounter* and enter the project settings as described previously. Select **Project→Add Copy of Source**. This will bring up a browsing window. Browse to the *leddec* project and double-click on *leddec.vhd*. Click **OK** on the Adding Source Files pop-up. This will add a copy of leddec.vhd to your new project. It should now be visible in the *Hierarchy* window.

2.2 Building Counter Module

Now we have to build the counter module. Use **Project→New Source** to create a new VHDL module named *counter*. Replace the template code for *counter.vhd* with the following:

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

entity counter is
   Port ( clk : in  STD_LOGIC;
          count : out  STD_LOGIC_VECTOR (3 downto 0));
end counter;

architecture Behavioral of counter is
signal cnt: STD_LOGIC_VECTOR (28 downto 0);        -- 29 bit counter
begin
        process(clk)
        begin
                if clk'event and clk='1' then       -- on rising edge of clock
                        cnt <= cnt+1;               -- increment counter
                end if;
        end process;
        count <= cnt (28 downto 25);
end Behavioral;
```

This module builds a 29-bit binary counter *cnt* clocked by the input *clk*. We will use bits 25~28 of our counter to generate a 4-bit value count which will be used to drive the LED decoder. If the input clock is running at 50MHz, the value of *count* will change at a frequency of $50 \text{ MHz} \div 2^{25} \approx 1.5$ Hz which will be slow enough for us to see.

2.3 Putting it All Together

We still need to connect the counter module to the LED decoder. We do this using structural modeling. We create a top level cell called *hexcount* that includes an instance of each of the lower level cells. Create a new VHDL module named *hexcount* and enter the following VHDL code into its edit window:

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


entity hexcount is
   Port ( clk_50MHz : in  STD_LOGIC;
          anode : out  STD_LOGIC_VECTOR (3 downto 0);
          seg : out  STD_LOGIC_VECTOR (6 downto 0));
end hexcount;

architecture Behavioral of hexcount is

component counter is
```

```
            Port (    clk : in  STD_LOGIC;
                      count : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component leddec is
            Port (    dig : in  STD_LOGIC_VECTOR (1 downto 0);
                      data : in STD_LOGIC_VECTOR (3 downto 0);
                      anode: out STD_LOGIC_VECTOR (3 downto 0);
                      seg : out  STD_LOGIC_VECTOR (6 downto 0));
end component;

signal S: STD_LOGIC_VECTOR (3 downto 0);

begin
C1:       counter port map (clk=>clk_50MHz, count=>S);
L1:       leddec port map (dig=>"00", data=>S, anode=>anode, seg=>seg);
end Behavioral;
```

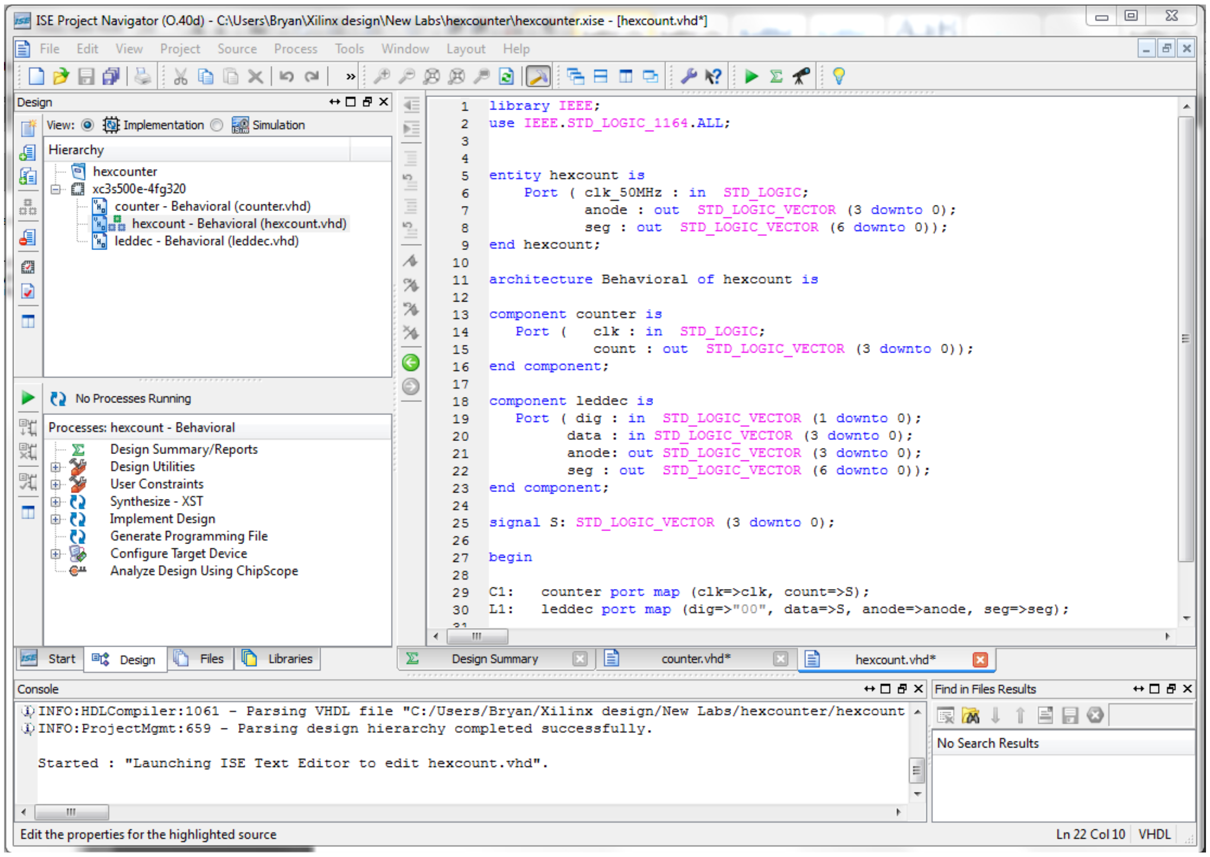
C1 and L1 are instances of the counter and LED decoder respectively. Signals are connected using the port map construct. The 4-bit internal signal *S* is used to connect the counter output to the decoder data input. Once you have entered the VHDL code, right click on the module *hexcount.vhd* in the *Hierarchy* window and select **Set as Top Module**.

2.4 Creating FPGA Implementation

Create an implementation constraint file *hexcount.ucf* and enter the following I/O constraints:

```
 # FPGA I/O Pin Locations

NET "clk_50MHz" LOC = B8;

NET "seg[0]" LOC = H14;
NET "seg[1]" LOC = J17;
NET "seg[2]" LOC = G14;
NET "seg[3]" LOC = D16;
NET "seg[4]" LOC = D17;
NET "seg[5]" LOC = F18;
NET "seg[6]" LOC = L18;

NET "anode[0]" LOC = F17;
NET "anode[1]" LOC = H17;
NET "anode[2]" LOC = C18;
NET "anode[3]" LOC = F15;
```

The ISE window should now as appear as in Figure 15.



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


entity hexcount is
    Port ( clk_50MHz : in  STD_LOGIC;
           anode : out  STD_LOGIC_VECTOR (3 downto 0);
           seg : out  STD_LOGIC_VECTOR (6 downto 0));
end hexcount;

architecture Behavioral of hexcount is

component counter is
   Port (   clk : in  STD_LOGIC;
            count : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component leddec is
   Port ( dig : in  STD_LOGIC_VECTOR (1 downto 0);
          data : in STD_LOGIC_VECTOR (3 downto 0);
          anode: out STD_LOGIC_VECTOR (3 downto 0);
          seg : out  STD_LOGIC_VECTOR (6 downto 0));
end component;

signal S: STD_LOGIC_VECTOR (3 downto 0);

begin

C1:   counter port map (clk=>clk, count=>S);
L1:   leddec port map (dig=>"00", data=>S, anode=>anode, seg=>seg);
```

```
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/Bryan/Xilinx design/New Labs/hexcounter/hexcount
INFO:ProjectMgmt:659 - Parsing design hierarchy completed successfully.

Started : "Launching ISE Text Editor to edit hexcount.vhd".
```

ISE Project Navigator (O.40d) - C:\Users\Bryan\Xilinx design\New Labs\hexcounter\hexcounter.xise - [hexcount.vhd*]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
hexcounter
xc3s500e-4fg320
counter - Behavioral (counter.vhd)
hexcount - Behavioral (hexcount.vhd)
leddec - Behavioral (leddec.vhd)
No Processes Running
Processes: hexcount - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start Design Files Libraries
Design Summary counter.vhd* hexcount.vhd*
Console
Find in Files Results
No Search Results
Edit the properties for the highlighted source
Ln 22 Col 10 VHDL

Figure 15 Ready to Synthesize *hexcount*

Highlight the module *hexcount.vhd* in the *Hierarchy* window and double click on the **Synthesis** process in the *Process* window. Now execute the **Implement Design** Process. Finally execute the Generate Programming File process (remembering to change the FPGA startup clock from **CCLK** to **JTAG Clock**).

2.5 Download & Run

Use the Adept application to download the configuration file *hexcount.bit* into the FPGA. The least significant 7-segment decoder should now count 0,1,2,3,4,5,6,7,8,9,0,A,B,C,D,E,F with a complete cycle taking approx. 10.7 secs