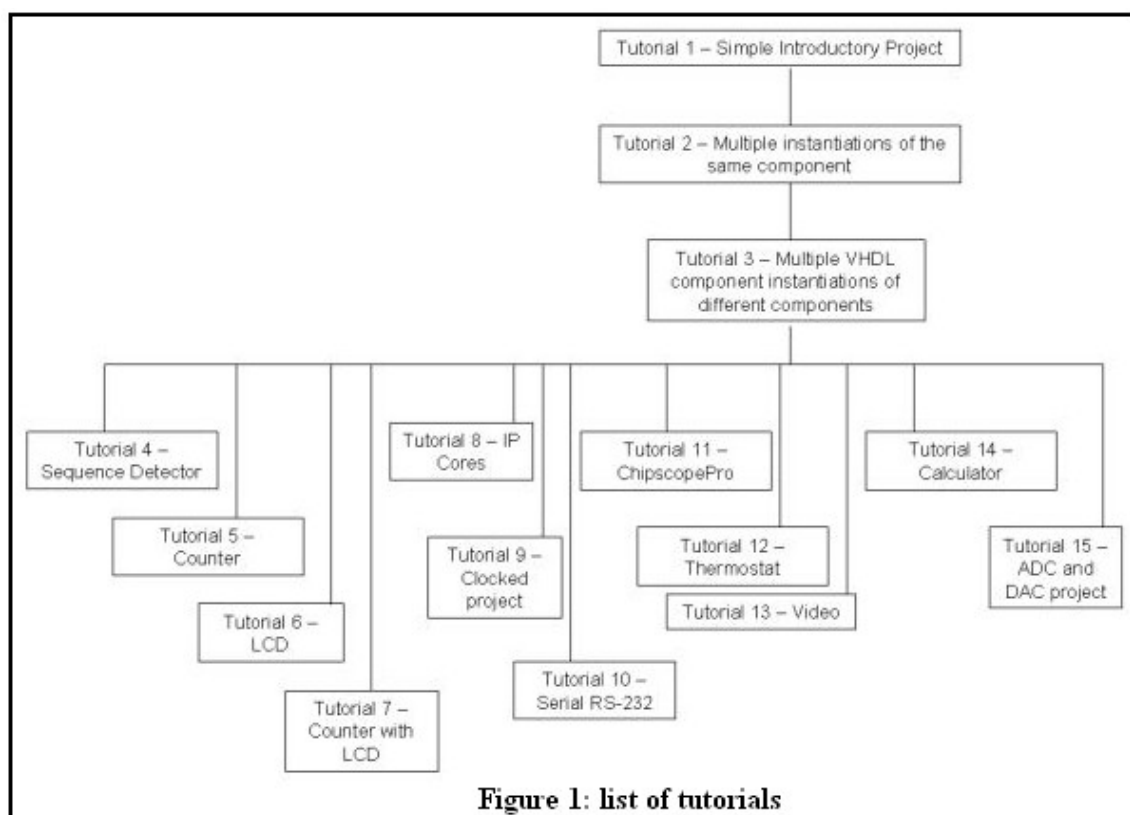


Tutorial 2

Multiple Instances of the Same Component with VHDL, ISE 10.1 on the Digilent Spartan-3E Starter Kit board

Introduction

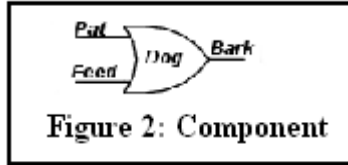
This tutorial will be the second in a series of tutorials as shown in figure 1.



One of the advantages of VHDL is the ability to develop and use packages of code repeatedly. Many experienced VHDL designers develop a component to perform a specific task. They debug and rework this component until it is perfect. Once this is done, the component can be used again and again for new projects.

This lab will be an introduction to reusing the same components in a single file using a hardware descriptive language (VHDL) design. Xilinx ISE 10.1 is the design tool provided by Xilinx for this purpose. The board will be a Digilent Spartan3E board with a XC3S500E chip.

There are two sections of design for this lab. The first is the VHDL program design of what the project has to do, where the circuit in question will be implemented.



The component to be used is the “Dog” component shown in figure 2. If you pat the dog or feed the dog, it will bark. This is a logic OR gate. Although this component is very simple, the same concepts and instantiation techniques can apply to much more complicated designs. The software package used for both design and simulation will be the Xilinx ISE 10.1. For additional details on the various tools and their uses, view the video on the FMAC website (www.fpgamac.com).

Objective

The objective is to understand how to create and simulate a single component. Once this is done, then the single component will be replicated three times. The resulting simulation will help in understanding the process.

Process

1. Analyze the VHDL of a single component.
2. Replicate the component.
3. Program FPGA and verify proper operation.

Implementation

1. Open the **Xilinx ISE Project Navigator** by selecting it from the start menu or from the desktop icon.
2. Within the project navigator, select from the **File Menu**, **New Project**. Name the project “tutorial2”. To put the project in a different location, either directly type the file path into **Project Location** or search by clicking the **Three-dotted Browse** button. The location of the project can always be seen on the top of the Project Navigator screen. Avoid spaces in the project name or path. Click next and it will show the device properties window.
3. Set the device properties exactly as shown in figure 3 and continue to select **Next** until the project is created.

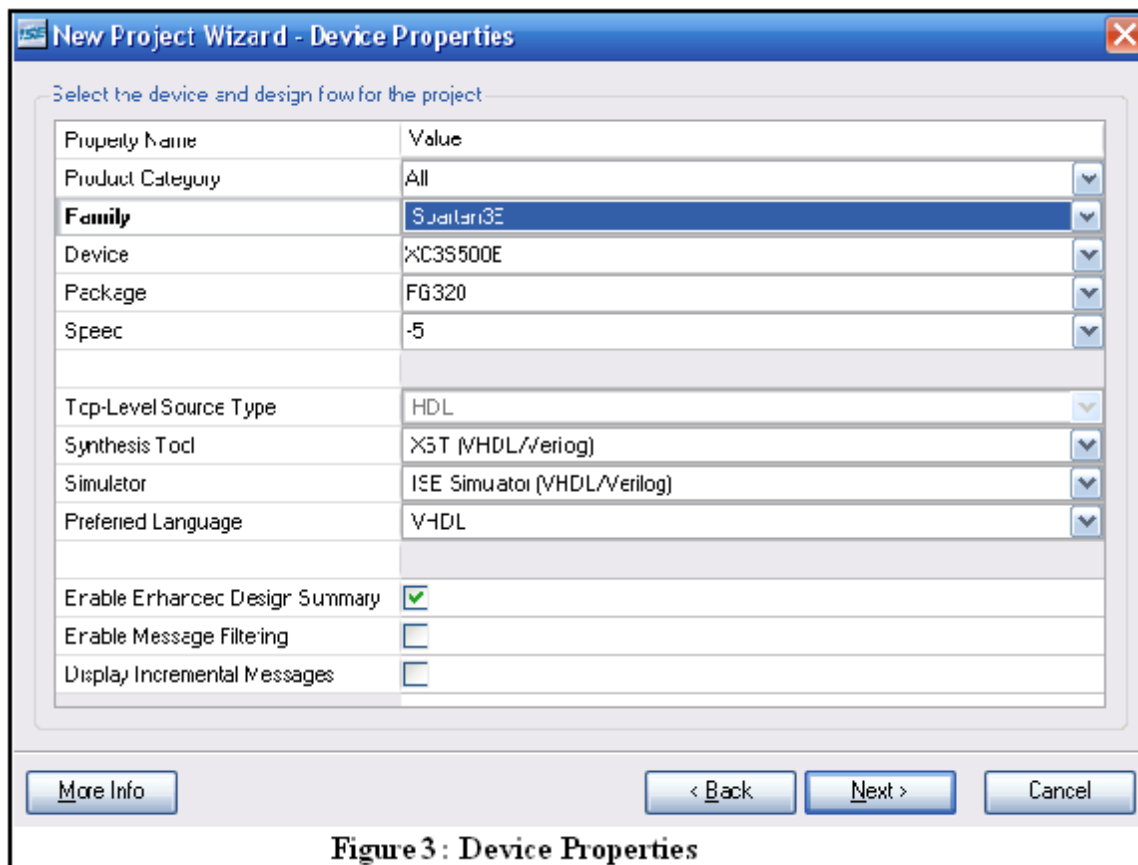


Figure 3: Device Properties

- The project will be a blank project. For this project, there are five source files (two source files, two simulation files and a ucf file). Copy them into the directory for Tutorial 2. The files are: Dog.vhd, Dog_tb.vhd, ManyDog.vhd, ManyDog_tb.vhd and ManyDog.ucf. When naming testbenches, it is common practice to add the “_tb” to the file name. To start, a project will be created with a single instance of Dog. This will allow for a more complete understanding of the basic building block that will be used in further instantiations. Right click into the Sources pane and select Add Source as shown in figure 4.

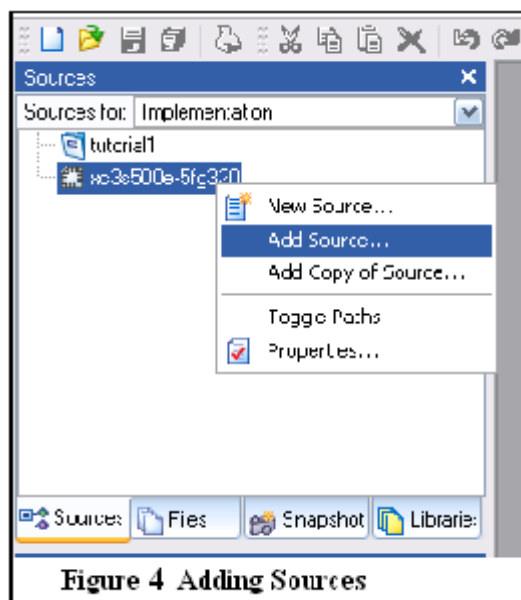
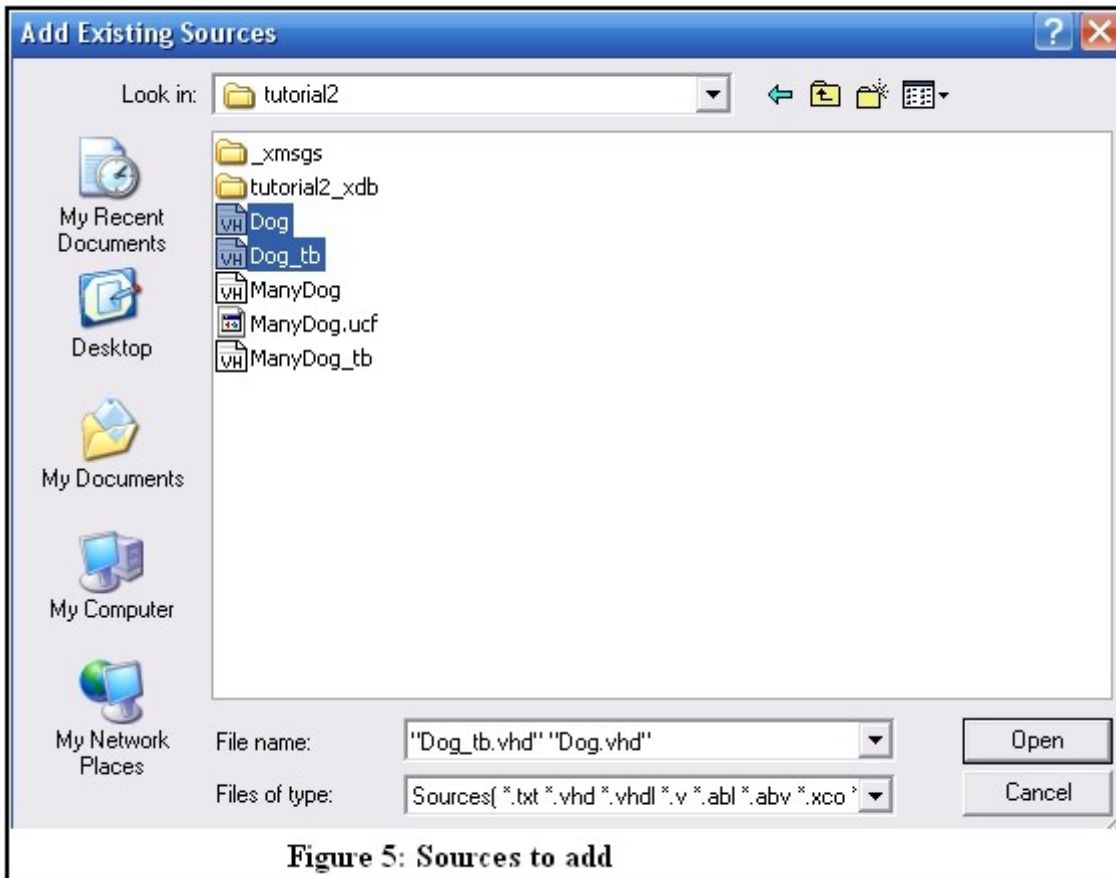
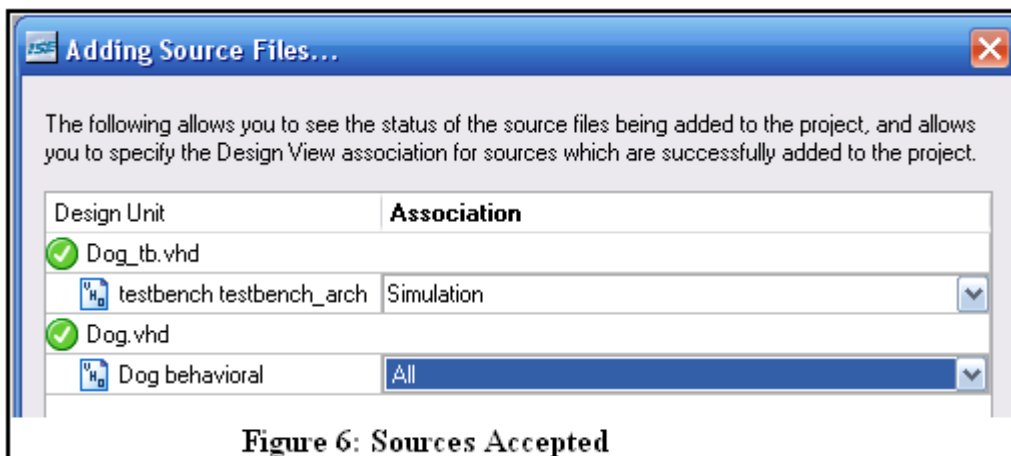


Figure 4 Adding Sources

5. Select the two files shown in figure 5.



6. The green check marks identify that the program knows what type of files they are and what to do with them. See figure 6.



- Using the drop down selector in “Sources for:”, choose the Behavioral Simulation option. See figure 7.

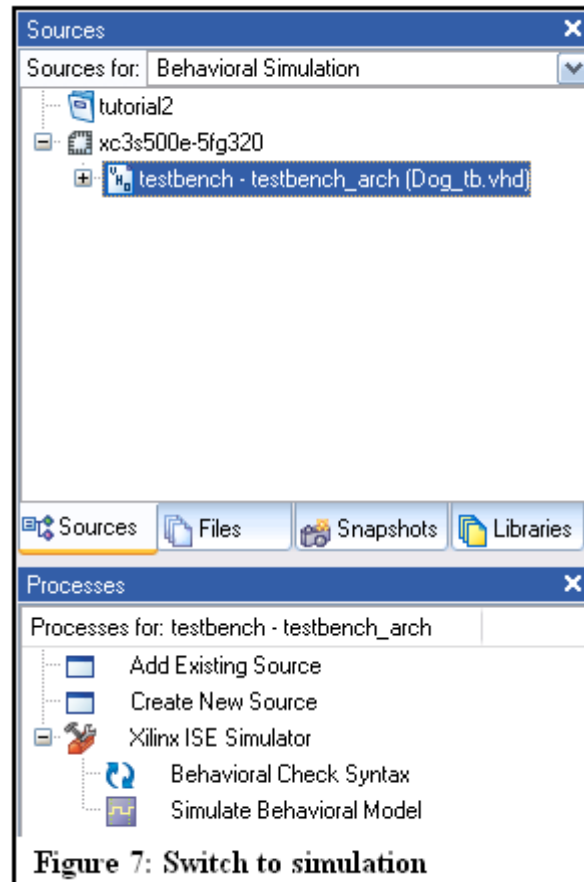


Figure 7: Switch to simulation

- Following the procedure from Tutorial 1, run the simulation using the applicable testbench. The resulting waveform confirms that anytime pat or feed are high, bark is also high. See figure 8.

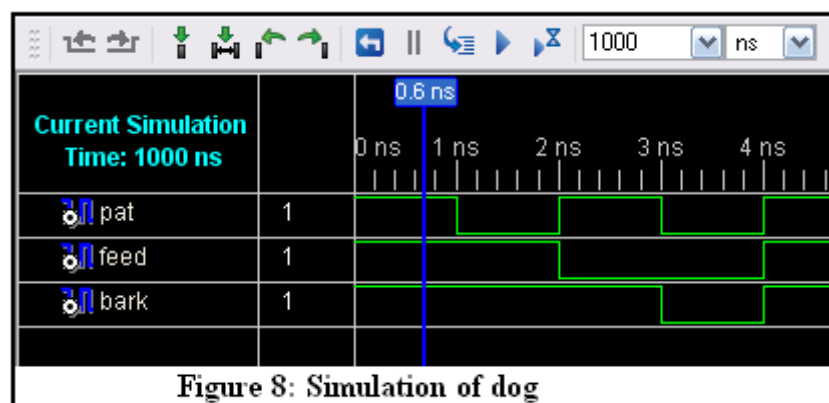


Figure 8: Simulation of dog

9. The next step will be to transition to a multiple instantiation of the dog. First, remove the Dog_tb.vhd file. Do this by highlighting the file, right clicking on it and choosing “remove.” See Figure 9. Next, add the new files. Right click in the sources pane and add the three files shown in figure 10.

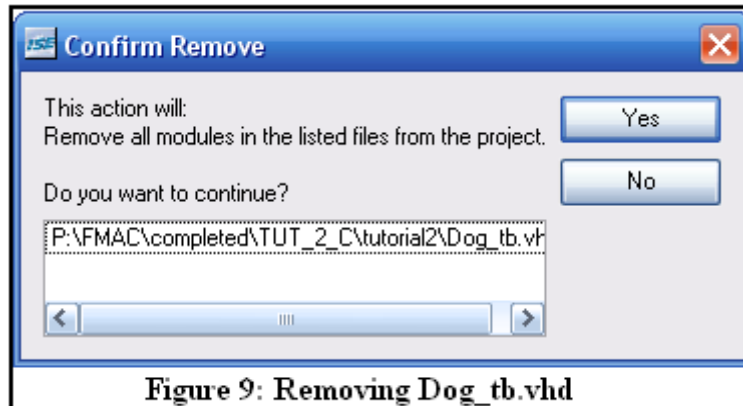


Figure 9: Removing Dog_tb.vhd

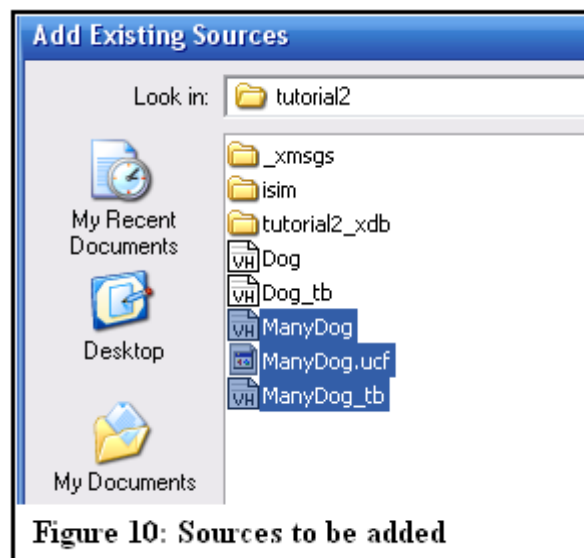


Figure 10: Sources to be added

10. The files added can be explored by double clicking on the file title in the sources window. They are the source file, the associated testbench and the constraints file for programming the board.
11. The expansion of the file hierarchy in the Behavioral Simulation pane shows how the ISE software believes the components are developed. It shows that ManyDog has three identical components called Dog.vhd. The three instantiations are: Poodle, Chow and Collie. See Figure 11.

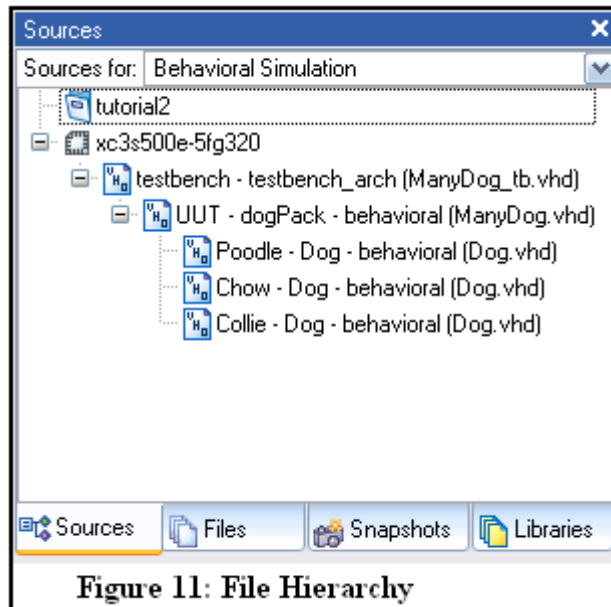


Figure 11: File Hierarchy

- Run the simulation. When compared to the Dog simulation, the ability to visualize that all three instantiations are identical is clear. Three identical instantiations of the dog have been created. The ManyDog.vhd code shows how this was accomplished. This is the key to component reuse. See Figure 12.

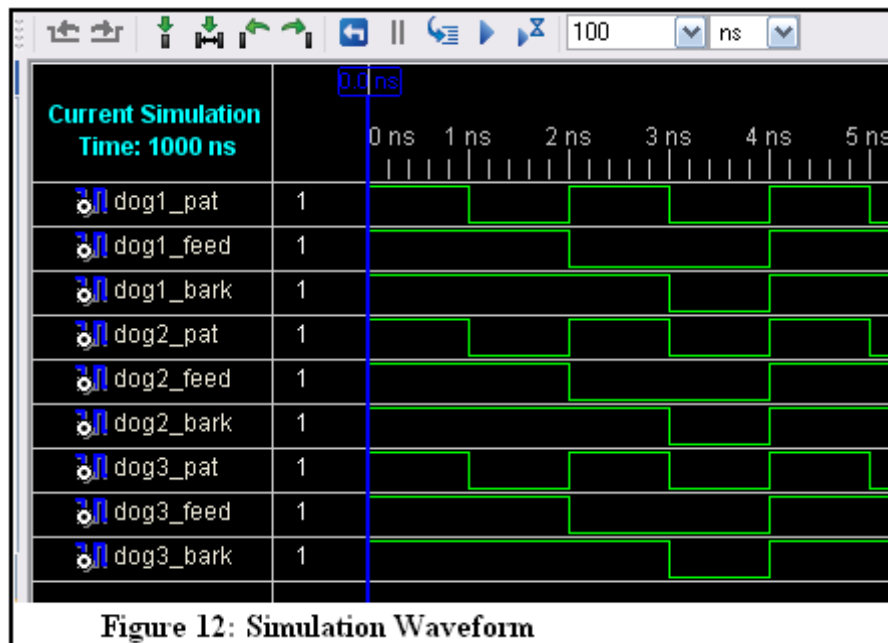
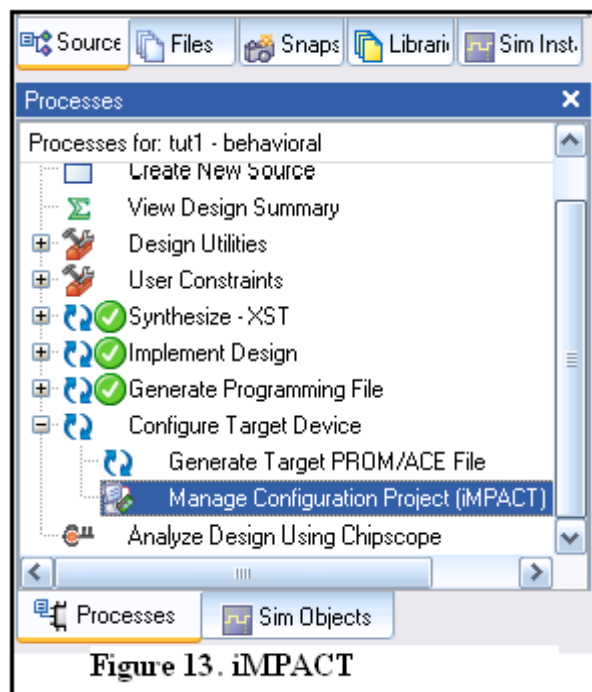


Figure 12: Simulation Waveform

13. The UCF shows the pin assignments used to display the developed code on the S3E board. See the attached Code.

14. Ensure the “Sources for:” selection is set for Implementation. Highlight the ManyDog.vhd code. The design portion is now finished. It would be possible to now click on Synthesize. When it is done, double click on Implement Design. Once both process have completed double click on generate programming file. This will create the “.bit” file that will be used to program the FPGA. See Figure 13. Finally open the “configure target device” and select iMPACT. iMPACT is the software that the Xilinx tools will use to communicate through the JTAG chain to the programming pins on the FPGA.expand the Generate Programming File selection and double click on the iMPACT choice.



15. Choose the JTAG option, and click finish. See Figure 14

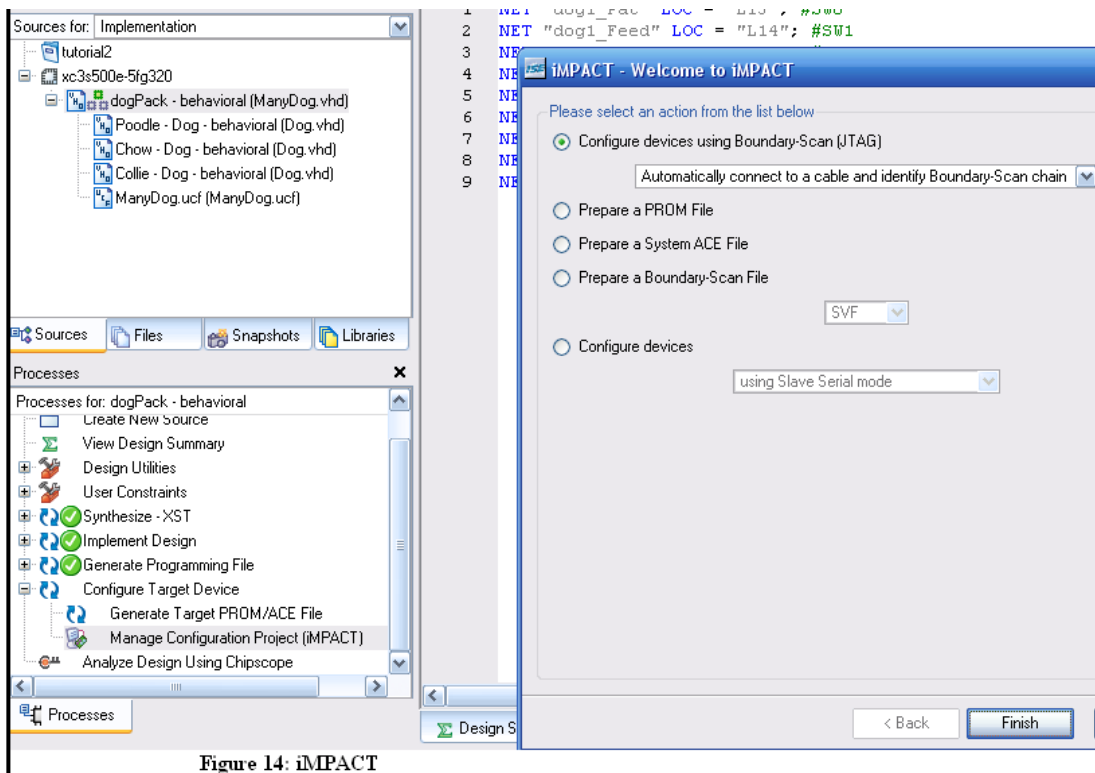


Figure 14: iMPACT

16. When the xc3s500e FPGA device is highlighted, choose dogpack.bit and Open. Bypass the other two devices. On the next screen press OK. See figure 15.

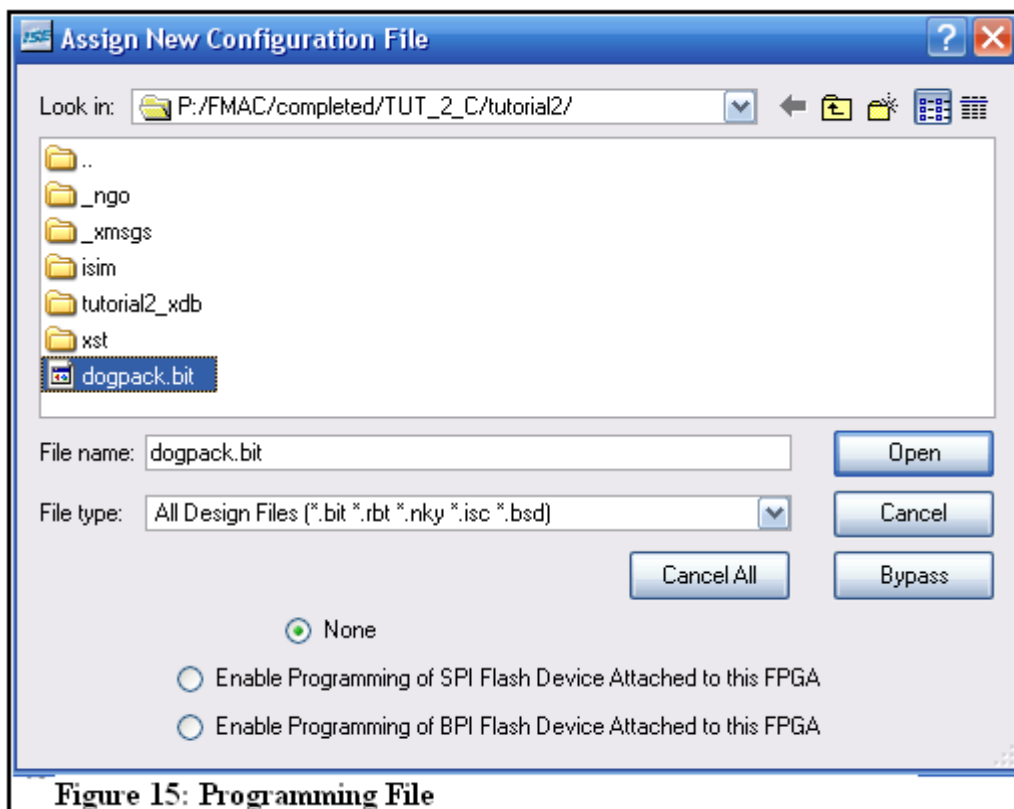


Figure 15: Programming File

17. When done, right click on the FPGA icon and program the device. According to the UCF settings, switches 0 and 1 are pat and feed for dog 1 and flipping either switch high will result in LED 0 being activated. The buttons will be used for Dog 3. However since they are subject to a bounce, LED2 may stay on. See Figure 16 and 17.

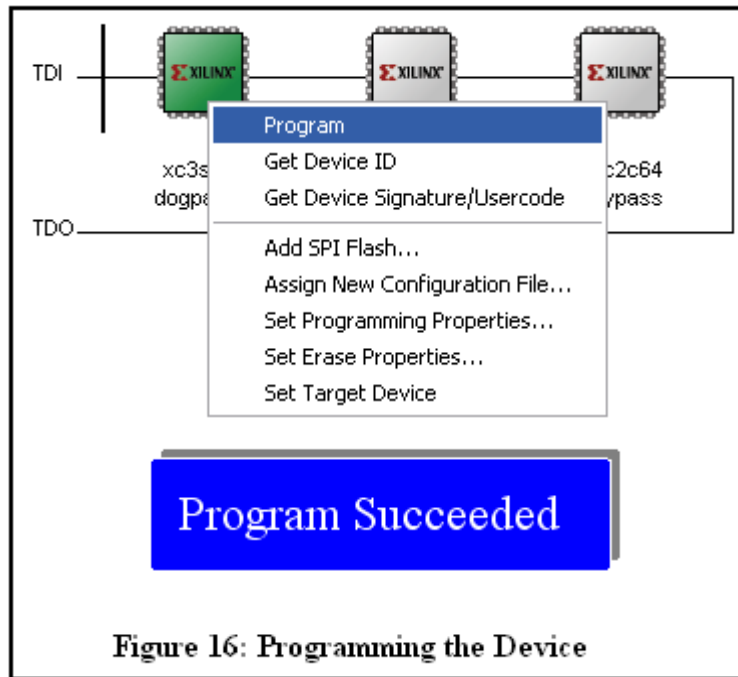


Figure 16: Programming the Device

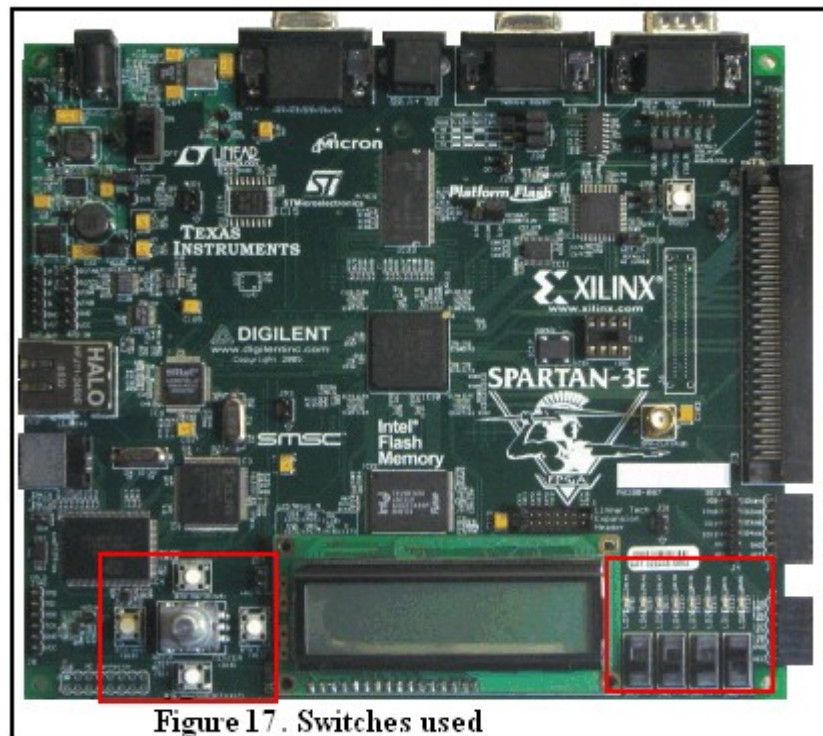


Figure 17. Switches used

About the author: Craig Kief is the Deputy Director of the FPGA Mission Assurance Center and can be reached at DeptDirector@fpgamac.com.

First Source File

```
--VHDL files contain three parts: library declarations, entity and
--behavior. The library describes what each function will do. Entity
--describes the inputs and outputs. Behavior is the working
--portion of the project.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Declaration of the module's inputs and outputs for part 1 of tutorial 2
entity Dog is port (

pat: in std_logic;
feed: in std_logic;
bark: out std_logic

);
end Dog;

--Defining the modules behavior
Architecture behavioral of Dog is
begin
process (pat, feed) begin

bark <= pat OR feed;

end process;
end behavioral;
```

First Testbench

--This is a VHDL testbench. It is used to stimulate the inputs to
--file we are testing. A good design practice is to create a testbench
--for each component you are developing. Test each component separately
--before combining them.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_unsigned.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE testbench_arch OF testbench IS

COMPONENT Dog
PORT (
pat                               : in std_logic;
feed                              : in std_logic;
bark                              : out std_logic
);
END COMPONENT;

--develop signals to stimulate
SIGNAL pat : std_logic := '0';
SIGNAL feed : std_logic := '0';
SIGNAL bark : std_logic := '0';

--UUT means unit under test
BEGIN
UUT : Dog

--map signals on right to entitys on the left
PORT MAP (
pat => pat,
feed => feed,
bark => bark
);

signal_Pat: process
begin
pat <= NOT pat;
wait for 1 ns;
end process;

signal_Feed: process
```

```
begin
feed <= NOT feed;
wait for 2 ns;
end process;

END testbench_arch;
```

Second Source File

```
--VHDL files contain three parts: library declarations, entity and
--behavior. The library describes what each function will do. Entity
--describes the inputs and outputs. Behavior is the working
--portion of the project.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Declaration of the module's inputs and outputs for part 1 of tutorial 2
entity dogPack is port (

dog1_Pat: in std_logic;
dog1_Feed: in std_logic;
dog1_Bark: out std_logic;
dog2_Pat: in std_logic;
dog2_Feed: in std_logic;
dog2_Bark: out std_logic;
dog3_Pat: in std_logic;
dog3_Feed: in std_logic;
dog3_Bark: out std_logic

);
end dogPack;

--Defining the modules behavior
Architecture behavioral of dogPack is

component Dog is port (

pat: in std_logic;
feed: in std_logic;
bark: out std_logic

);
end component Dog;

begin

Poodle : Dog
port map (
pat => dog1_Pat,
feed => dog1_Feed,
bark => dog1_Bark
);

Chow : Dog
```

```
port map (  
  pat => dog2_Pat,  
  feed => dog2_Feed,  
  bark => dog2_Bark  
);
```

```
Collie : Dog  
port map (  
  pat => dog3_Pat,  
  feed => dog3_Feed,  
  bark => dog3_Bark  
);
```

```
end Behavioral;
```

Second Testbench

--This is a VHDL testbench. It is used to stimulate the inputs to
--file we are testing. A good design practice is to create a testbench
--for each component you are developing. Test each component separately
--before combining them.

```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.all;  
USE IEEE.std_logic_arith.all;  
USE IEEE.std_logic_unsigned.all;  
USE IEEE.STD_LOGIC_TEXTIO.ALL;  
USE STD.TEXTIO.ALL;
```

```
ENTITY testbench IS  
END testbench;
```

```
ARCHITECTURE testbench_arch OF testbench IS
```

```
COMPONENT dogPack  
PORT (  
dog1_Pat: in std_logic;  
dog1_Feed: in std_logic;  
dog1_Bark: out std_logic;  
dog2_Pat: in std_logic;  
dog2_Feed: in std_logic;  
dog2_Bark: out std_logic;  
dog3_Pat: in std_logic;  
dog3_Feed: in std_logic;  
dog3_Bark: out std_logic  
);  
END COMPONENT;
```

```
--develop signals to stimulate  
SIGNAL dog1_Pat : std_logic := '0';  
SIGNAL dog1_Feed : std_logic := '0';  
SIGNAL dog1_Bark : std_logic := '0';  
SIGNAL dog2_Pat : std_logic := '0';  
SIGNAL dog2_Feed : std_logic := '0';  
SIGNAL dog2_Bark : std_logic := '0';  
SIGNAL dog3_Pat : std_logic := '0';  
SIGNAL dog3_Feed : std_logic := '0';  
SIGNAL dog3_Bark : std_logic := '0';  
--UUT means unit under test  
BEGIN  
UUT : dogPack
```

```
--map signals on right to entities on the left  
PORT MAP (  
dog1_Pat => dog1_Pat,
```

```
dog1_Feed => dog1_Feed,  
dog1_Bark => dog1_Bark,  
dog2_Pat => dog2_Pat,  
dog2_Feed => dog2_Feed,  
dog2_Bark => dog2_Bark,  
dog3_Pat => dog3_Pat,  
dog3_Feed => dog3_Feed,  
dog3_Bark => dog3_Bark  
);
```

```
signal_Pat: process  
begin  
dog1_Pat <= NOT dog1_Pat;  
dog2_Pat <= NOT dog2_Pat;  
dog3_Pat <= NOT dog3_Pat;  
wait for 1 ns;  
end process;
```

```
signal_Feed: process  
begin  
dog1_Feed <= NOT dog1_Feed;  
dog2_Feed <= NOT dog2_Feed;  
dog3_Feed <= NOT dog3_Feed;  
wait for 2 ns;  
end process;
```

```
END testbench_arch;
```

UCF File

```
NET "dog1_Pat" LOC = "L13"; #SW0
NET "dog1_Feed" LOC = "L14"; #SW1
NET "dog1_Bark" LOC = "F12"; #LED0
NET "dog2_Pat" LOC = "H18"; #SW2
NET "dog2_Feed" LOC = "N17"; #SW3
NET "dog2_Bark" LOC = "E12"; #LED1
NET "dog3_Pat" LOC = "D18"; #BTN West
NET "dog3_Feed" LOC = "H13"; #BTN East
NET "dog3_Bark" LOC = "E11"; #LED2
```