

Tutorial 3

Multiple Instances of the Different Components with VHDL, ISE 9.2 and ModelsimXE on the Digilent Spartan-3E Starter Kit board

-Introduction

This tutorial will be the third in a series of tutorials as shown below. After someone finishes this tutorial, they should be able to go directly to any of the other tutorials (4-15).

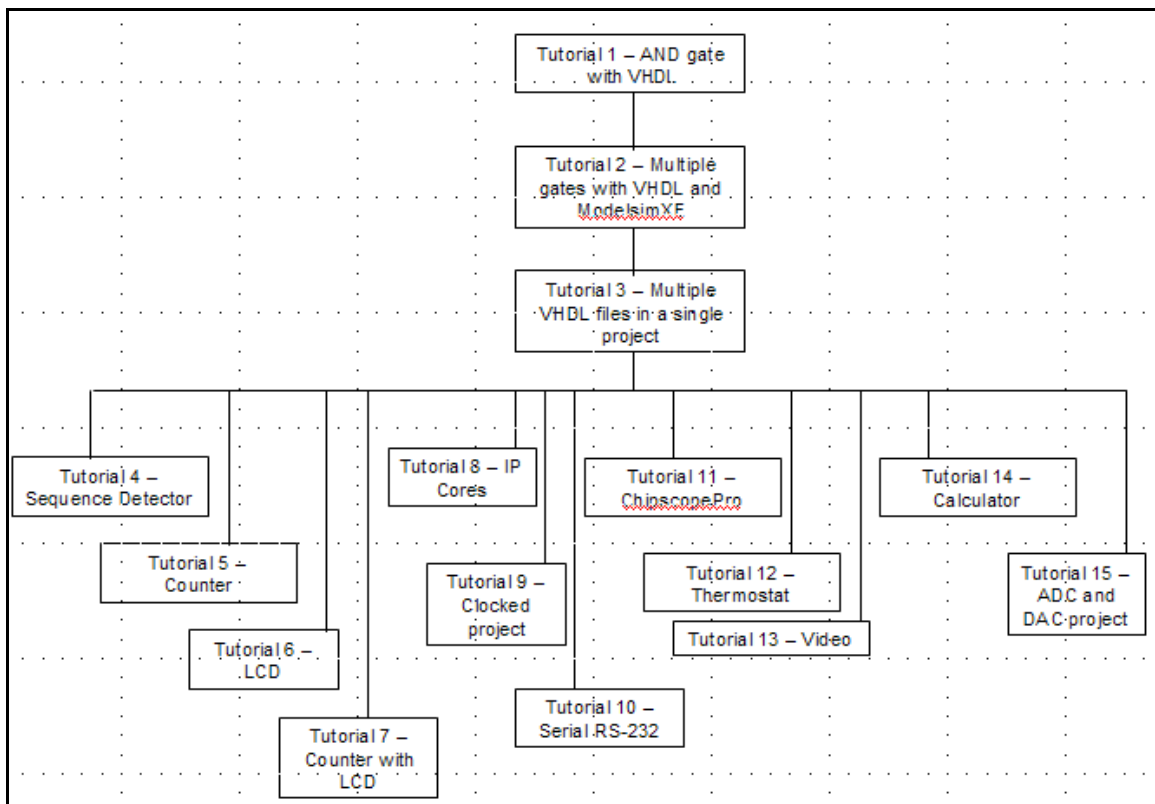


Figure 1. Entire list of tutorials

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 different components in a single file using a hardware descriptive language (VHDL) design. Xilinx ISE 9.2 is the design tool provided by Xilinx for this purpose. The board will be a Digilent Spartan3E board with a XC3S500E chip. Whereas Tutorial 2 developed multiple instances of the same component, Tutorial 3 will develop instances of different components.

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.

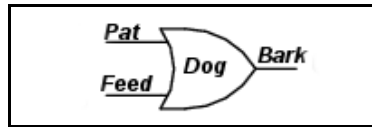


Figure 2. Dog Component

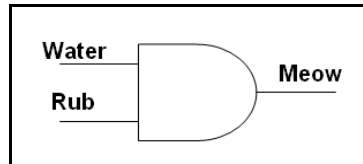


Figure 3. Cat Component

The components to be used are the “Dog” component and “Cat” component shown above. If you pat the dog or feed the dog, it will bark. This is a logic OR gate. If you water and rub the cat, it will meow. This is a logic AND gate. Although these components are very simple, the same concepts and instantiation techniques can apply to much more complicated designs. The software packages used are the Xilinx ISE and Mentor Graphics ModelsimXE (XE stands for Xilinx Edition). Although ISE has its own simulation engine, ModelsimXE is more powerful and provides greater functionality. ModelsimXE is used to verify the behavioral functionality of the design prior to programming the FPGA. 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 single components. Once this is done, then the single component will be replicated. 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 “tutorial3”. 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 the following picture and continue to select Next until the project is created.

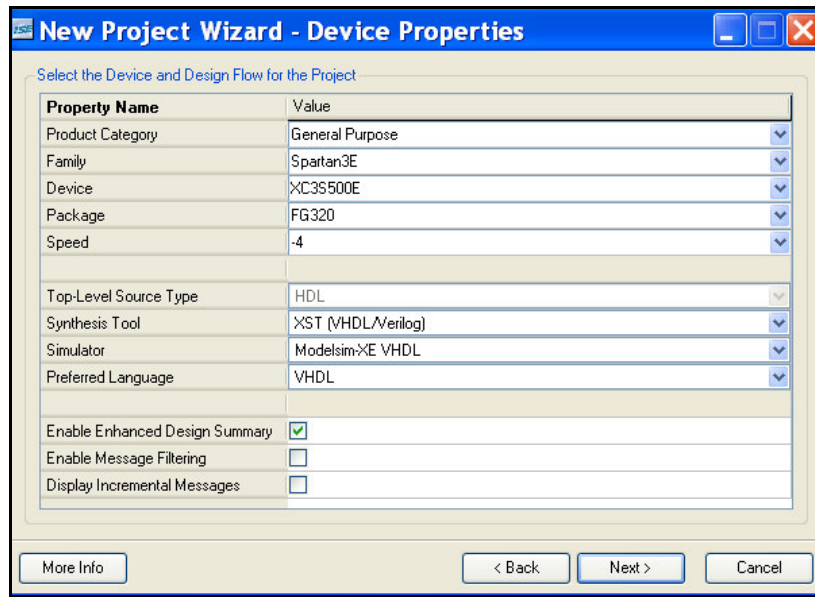


Figure 4. Device Properties

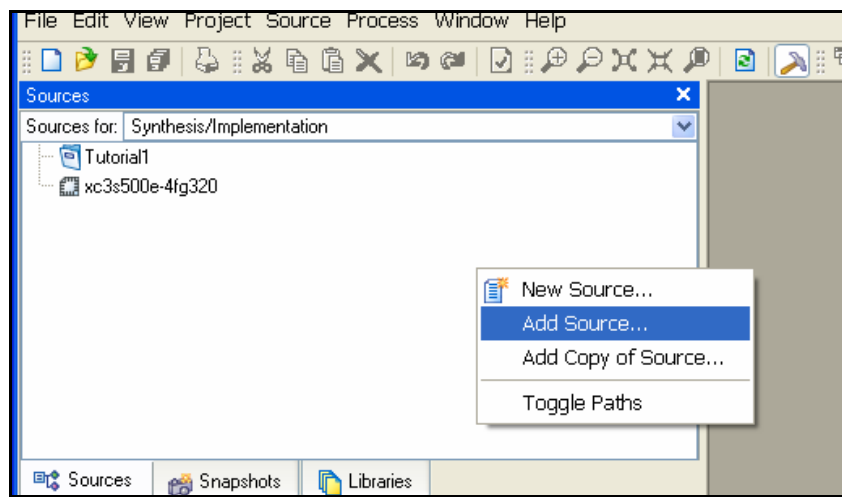


Figure 5. Adding Sources

4. The project will be a blank project. For this project, there are seven source files (three source files, three simulation files and a ucf file). Copy them into the directory for Tutorial 3. The files are: Dog.vhd, Dog_tb.vhd, Cat.vhd, Cat_tb.vhd, ManyPets.vhd, ManyPets_tb.vhd and ManyPets.ucf. When naming testbenches, it is common practice to add the “_tb” to the file name. Tutorial 2 covers all the specifics about the Dog component. It will not be repeated here. To start, a project will be created with a single instance of Cat. 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...

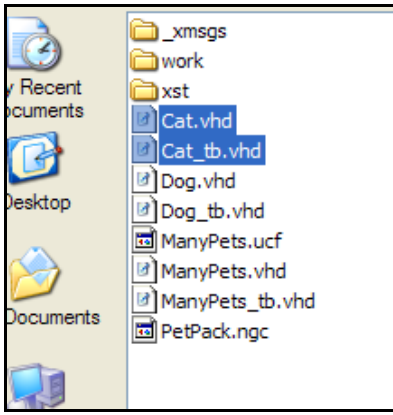


Figure 7. Selected Sources

5. Select the two files shown above.

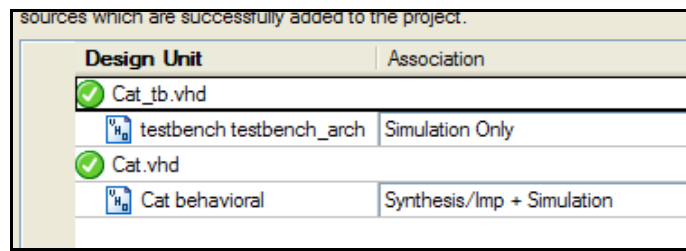


Figure 8. Accepted Source Files

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

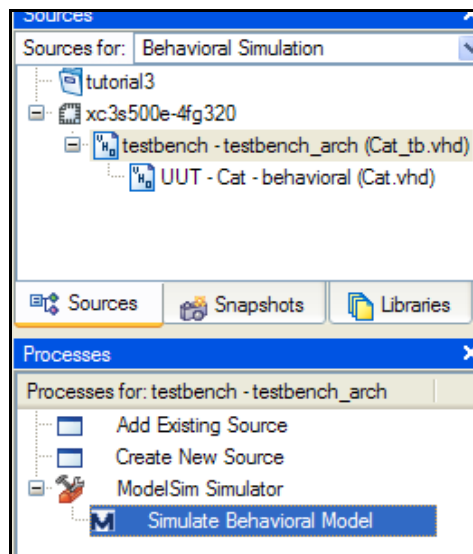


Figure 9. Switch to Simulation

7. Using the drop down selector in “Sources for:” choose the Behavioral Simulation option. Ensure that the Cat_tb.vhd file is highlighted.

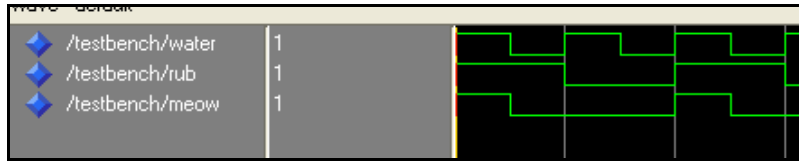


Figure 10. Simulation of Cat

- Following the procedure from Tutorial 1, run the simulation using the applicable testbench. The resulting waveform confirms that anytime water and rub are high, meow is also high.

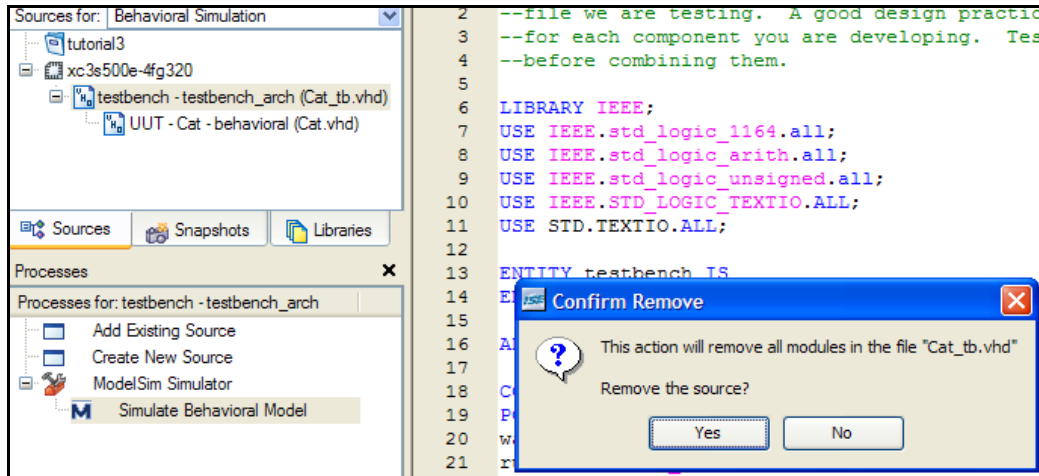


Figure 11. Remove Old Testbench

- The next step will be to transition to an instantiation of the dog and the cat. First, remove the Cat_tb.vhd file. Do this by highlighting the file, right clicking on it and choosing "remove." Next, add the new files. Right click in the sources pane and add the four files shown below.

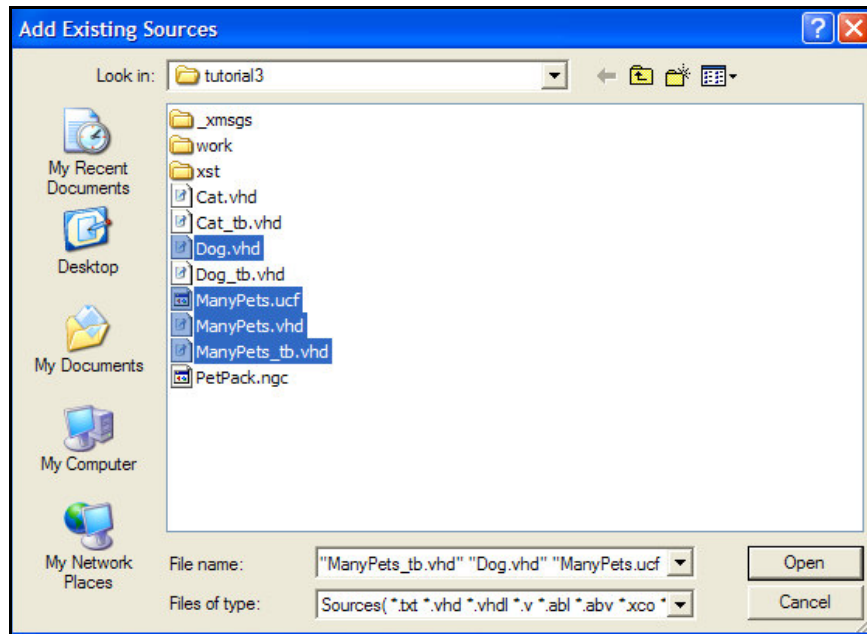


Figure 12. Adding New Files

10. The files added can be explored. They are the dog component and the ManyPets file that calls for an instance of dog and cat. This is the critical part. Just as dog and cat are created, verified and used individually, any creation of a quality (and verified) component can be reused.

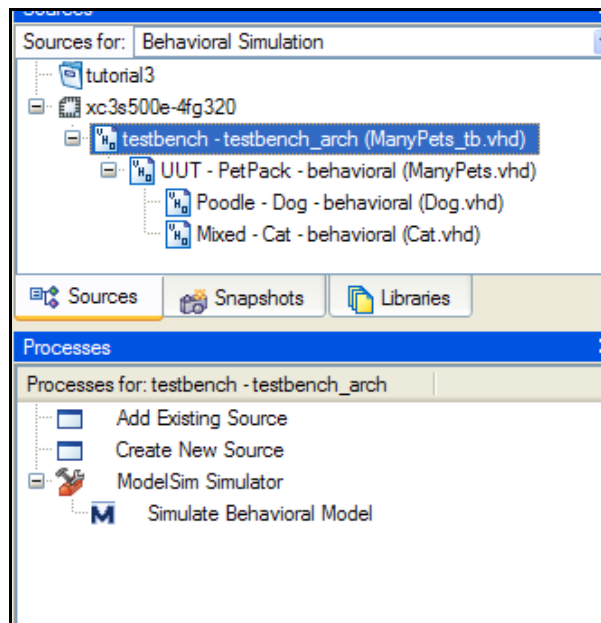


Figure 13. File Hierarchy

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 ManyPets has two identical components called Dog.vhd and Cat.vhd. The two instantiations are: Poodle and Mixed.

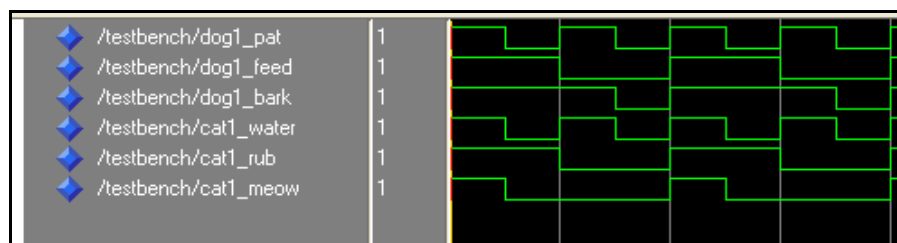


Figure 14. Simulation Waveform

12. Run the simulation. When compared to the Cat simulation, it is now possible to visualize both instantiations. The ManyPets.vhd code shows how this was accomplished. This is component reuse.

```

1 NET "dog1_Pat" LOC = "L13"; #SW0
2 NET "dog1_Feed" LOC = "L14"; #SW1
3 NET "dog1_Bark" LOC = "F12"; #LEDO
4 NET "cat1_Water" LOC = "H18"; #SW2
5 NET "cat1_Rub" LOC = "N17"; #SW3
6 NET "cat1_Meow" LOC = "E12"; #LED1
7

```

Figure 15. User Constraint File (UCF)

- The UCF shows the pin assignments used to display the developed code on the S3E board.

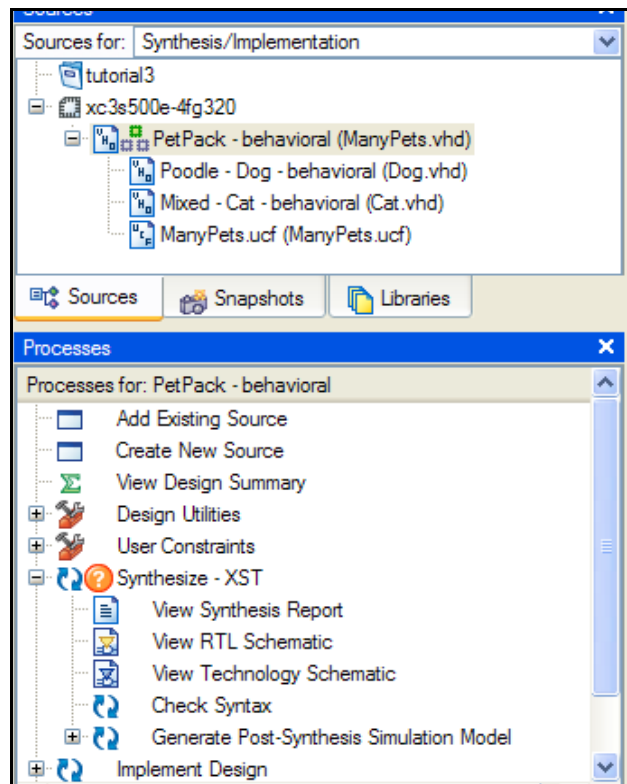


Figure 16. Synthesis

- Ensure the “Sources for:” selection is set for Synthesis/Implementation. Highlight the ManyPets.vhd code, expand the Generate Programming File selection and double click on the iMPACT choice.

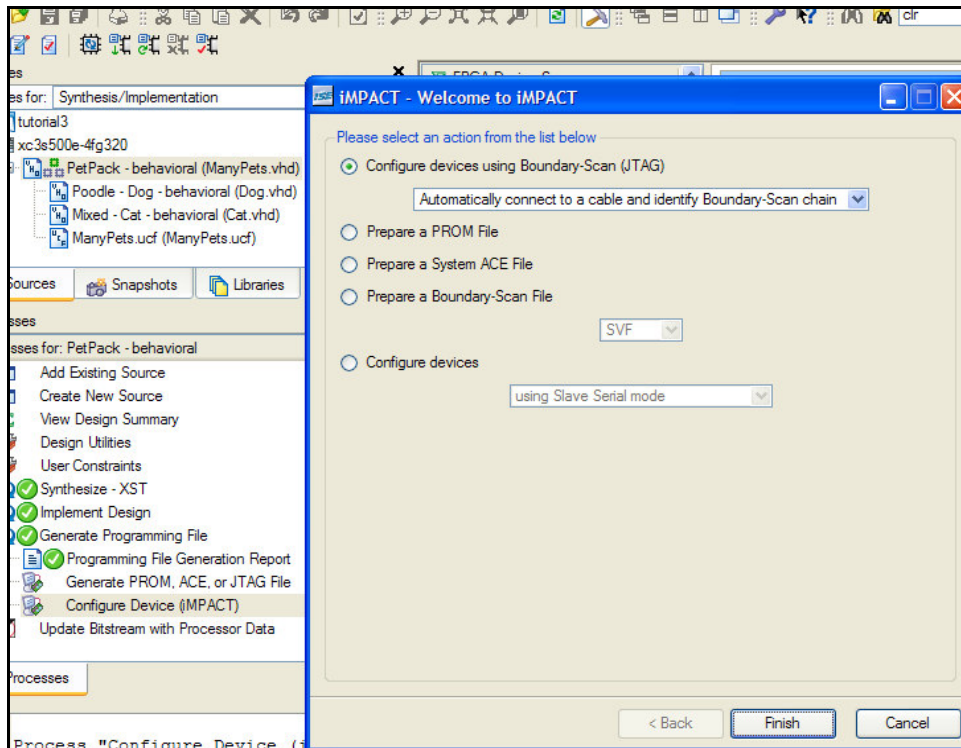


Figure 17. iMPACT

15. Choose the JTAG option.

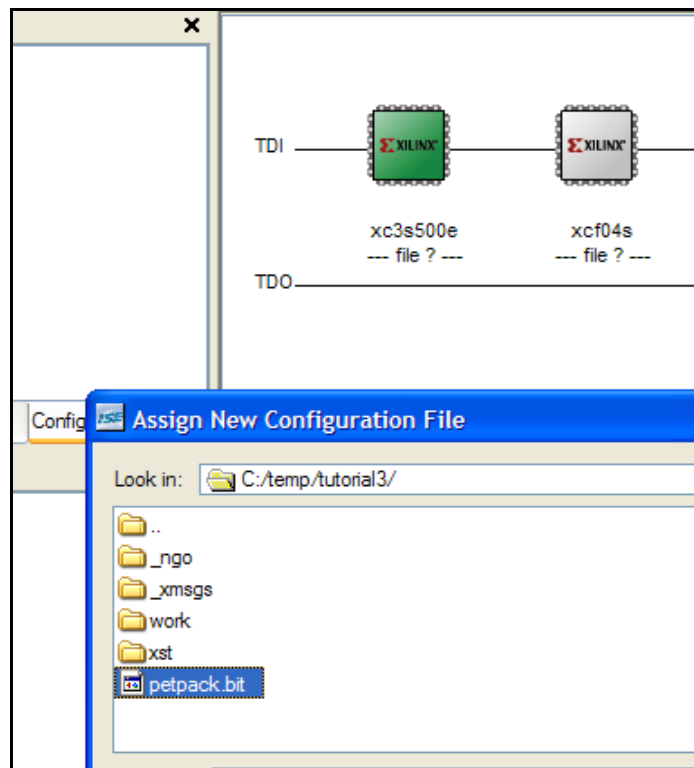


Figure 18. Choosing Programming File

16. When the xc3s500e FPGA device is highlighted, choose petpack.bit and Open. Bypass the other two devices.

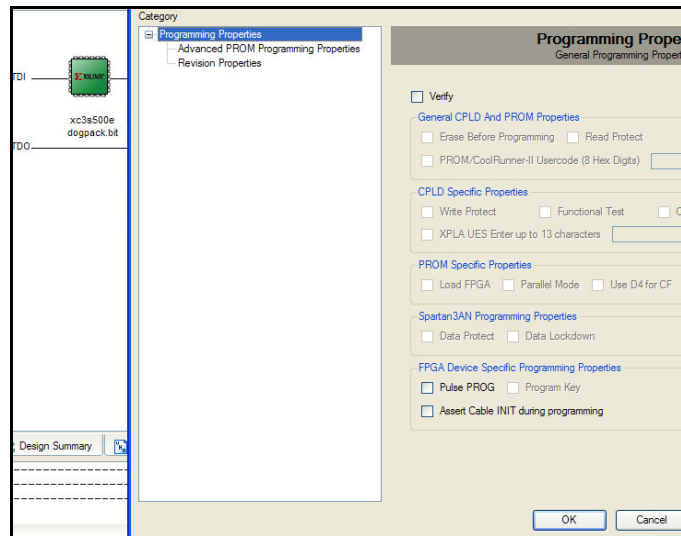


Figure 19. Programming Device

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 and switch 2 and 3 are water and rub for cat. Flipping switches high will result in LEDs being activated.

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;
```

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 Cat is port (

water: in std_logic;
rub: in std_logic;
meow: out std_logic

);
end Cat;

--Defining the modules behavior
Architecture behavioral of Cat is
begin
process (water, rub) begin

meow <= water AND rub;

end process;
end behavioral;
```

Third 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 PetPack is port (

dog1_Pat: in std_logic;
dog1_Feed: in std_logic;
dog1_Bark: out std_logic;
cat1_Water: in std_logic;
cat1_Rub: in std_logic;
cat1_Meow: out std_logic
);
end PetPack;

--Defining the modules behavior
Architecture behavioral of PetPack is

component Dog is port (
pat: in std_logic;
feed: in std_logic;
bark: out std_logic
);
end component Dog;

component Cat is port (
water: in std_logic;
rub: in std_logic;
meow: out std_logic
);
end component Cat;

begin

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

```
Mixed : Cat
port map (
water => cat1_Water,
rub => cat1_rub,
meow => cat1_meow
);
end Behavioral;
```

First Test Bench

--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 entities 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 Test Bench

--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 Cat  
PORT (  
  water : in std_logic;  
  rub : in std_logic;  
  meow : out std_logic  
);  
END COMPONENT;
```

```
--develop signals to stimulate
```

```
SIGNAL water : std_logic := '0';  
SIGNAL rub : std_logic := '0';  
SIGNAL meow : std_logic := '0';
```

```
--UUT means unit under test  
BEGIN
```

```
UUT : Cat  
--map signals on right to entities on the left  
PORT MAP (  
  water => water,  
  rub => rub,  
  meow => meow  
);
```

```
signal_Water: process  
begin  
  water <= NOT water;  
  wait for 1 ns;  
end process;
```

```
signal_Rub: process
begin
rub <= NOT rub;
wait for 2 ns;
end process;

END testbench_arch;
```

Third Test Bench

--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 PetPack  
PORT (  
dog1_Pat: in std_logic;  
dog1_Feed: in std_logic;  
dog1_Bark: out std_logic;  
cat1_Water: in std_logic;  
cat1_Rub: in std_logic;  
cat1_Meow: 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 cat1_Water : std_logic := '0';  
SIGNAL cat1_Rub : std_logic := '0';  
SIGNAL cat1_Meow : std_logic := '0';
```

```
--UUT means unit under test  
BEGIN  
UUT : PetPack  
--map signals on right to entities on the left  
PORT MAP (  
dog1_Pat => dog1_Pat,  
dog1_Feed => dog1_Feed,  
dog1_Bark => dog1_Bark,  
cat1_Water => cat1_Water,  
cat1_Rub => cat1_Rub,  
cat1_Meow => cat1_Meow  
);
```

```
signal_Pat_Water: process
begin
dog1_Pat <= NOT dog1_Pat;
cat1_Water <= NOT cat1_Water;
wait for 1 ns;
end process;
```

```
signal_Feed_Rub: process
begin
dog1_Feed <= NOT dog1_Feed;
cat1_Rub <= NOT cat1_Rub;
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 "cat1_Water" LOC = "H18"; #SW2  
NET "cat1_Rub" LOC = "N17"; #SW3  
NET "cat1_Meow" LOC = "E12"; #LED1
```