Panagiotis Mousouliotis – Aristotle University of Thessaloniki

In-Circuit Verification using LogiCORE IP, ChipScope Pro and Virtual Input/Output (VIO) core

Project 2

Design and In-Circuit Verification of a 2:1 8-bit Multiplexer

LogiCORE IP

Intellectual Property (IP) refers to preconfigured logic functions that can be used in a design. Xilinx provides a wide selection of IP, that is optimized for Xilinx FPGAs.

These can include functions delivered through the Xilinx CORE Generator software, through the Xilinx Architecture Wizard, as standalone archives, from third parties, through Xilinx Platform Studio (XPS), or through System Generator.

Xilinx and its partner companies produce IP, called LogiCORE products, ranging in complexity from simple arithmetic operators and delay elements to complex system-level building blocks, such as Digital Signal Processing (DSP) filters and transforms [1].

Xilinx CORE Generator

The CORE Generator system accelerates design time by providing access to highly parameterized Intellectual Properties (IP) for Xilinx FPGAs and is included in the ISE Design Suite [1].

The CORE Generator tool provides a catalog of architecture specific, domain-specific (embedded, connectivity, and DSP) and market specific IP (Automotive, Consumer, Mil/Aero, Communications, AVB, and others).

These user-customizable IP functions range in complexity from commonly used functions, such as memories and FIFOs, to complex, system-level building blocks, such as filters and transforms.

Using these IP functions can save days to months of design time. These optimized IP allow FPGA designers to focus efforts on building designs more quickly.

Xilinx ChipScope Pro tool

ChipScope Pro tool is used for in-circuit design debug/verification. ChipScope Pro tool design flow for in-circuit verification (using Project Navigator) consists of the following steps [1] :

1. ChipScope Pro cores insertion in the design using the CORE Generator software or ChipScope Pro Core Inserter.
2. Design implementation and device configuration.
3. Design analysis in the ChipScope Pro Analyzer.

The ChipScope Pro tool allows the designer to embed the following cores within the design, which assist with on-chip debugging: integrated logic analyzer (ILA), integrated bus analyzer (IBA), and virtual input/output (VIO) low-profile software cores.

In this project the VIO core will be used. The LogiCORE IP ChipScope Pro Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time [2].

Design and In-Circuit Verification of a 2:1 8-bit Multiplexer

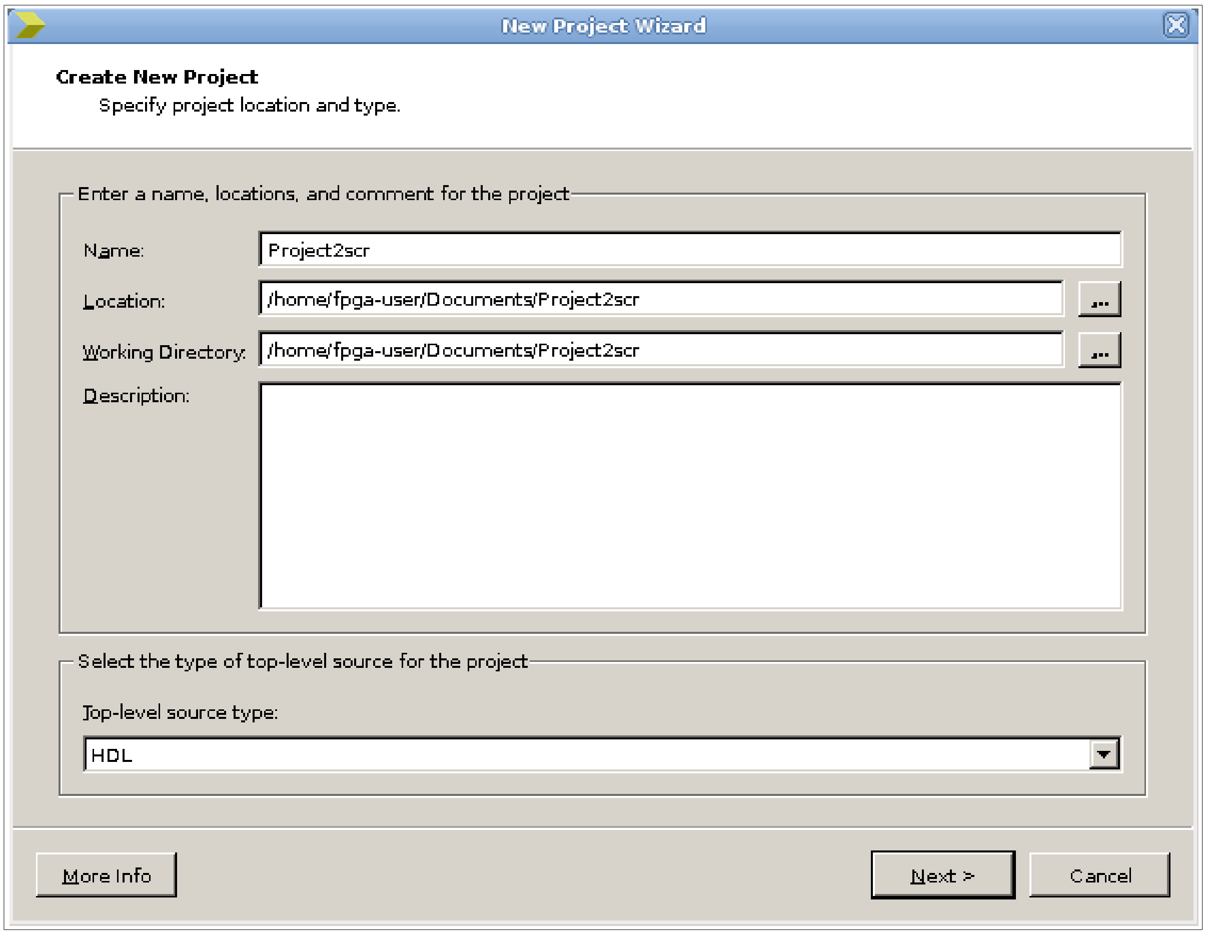
The following pages will demonstrate how to implement and verify in-circuit a 2:1 8-bit multiplexer using Xilinx ISE 13.3, VHDL and CORE Generator system as design entry, Xilinx Evaluation boards and ChipScope Pro tool for in-circuit verification.

1. Open Xilinx ISE :

Double click the ISE desktop icon. Alternatively open a new terminal and give the command "`./.bin/ise`" (without the quotes). For remote users, open a new terminal and give the following commands "`export DISPLAY=:1`" and "`./.remote/ise`" (without the quotes).

2. Create a new Xilinx ISE Project :

In the ISE Project Navigator window, select `File->New Project...`. In the New Project Wizard window, type `Project2scr` in the Name field and `/home/fpga-user/Documents/Project2scr` in the Location field and click Next (see **Figure 1**).


New Project Wizard

Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: Project2scr

Location: /home/fpga-user/Documents/Project2scr

Working Directory: /home/fpga-user/Documents/Project2scr

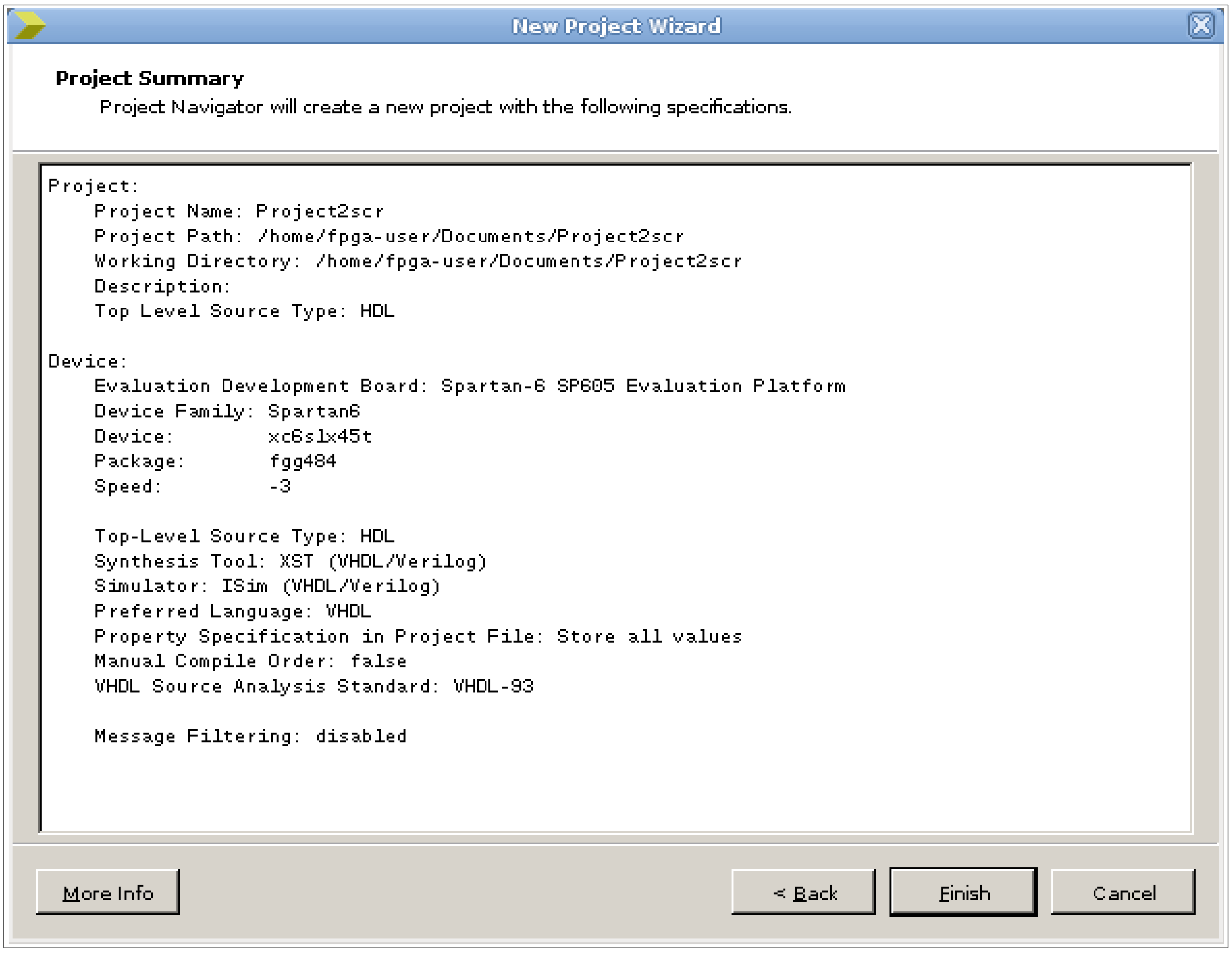
Description:

Select the type of top-level source for the project

Top-level source type: HDL

Figure 1

In the next New Project Wizard window, select the target Xilinx FPGA Evaluation board (e.g. the Spartan-6 SP605 Evaluation Platform) in the Evaluation Development Board field, select VHDL in the Preferred Language field and click Next. The last New Project Wizard window must be similar to the one shown in **Figure 2**. Click Finish to create the project `Project2scr` or navigate to the previous windows using Back to make corrections.


New Project Wizard
Project Summary
Project Navigator will create a new project with the following specifications.
More Info
< Back
Finish
Cancel

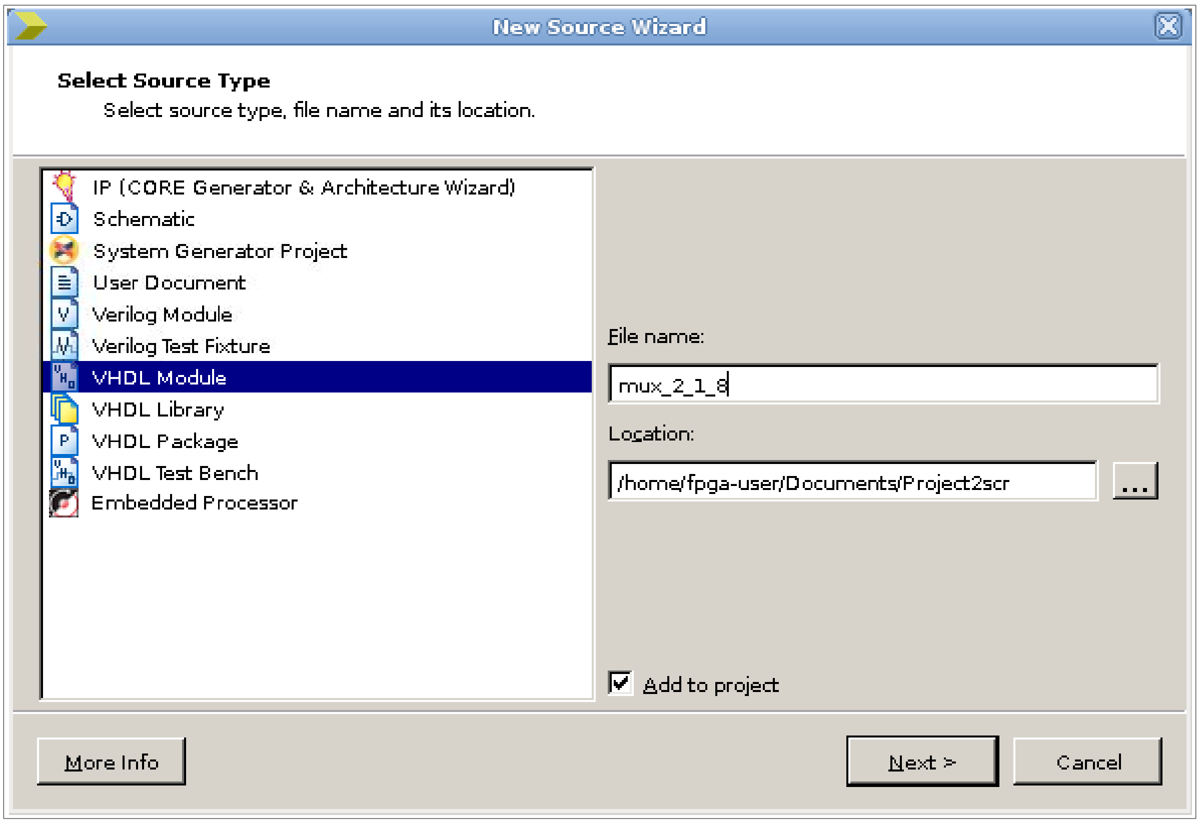
```
Project:
    Project Name: Project2scr
    Project Path: /home/fpga-user/Documents/Project2scr
    Working Directory: /home/fpga-user/Documents/Project2scr
    Description:
    Top Level Source Type: HDL

Device:
    Evaluation Development Board: Spartan-6 SP605 Evaluation Platform
    Device Family: Spartan6
    Device:        xc6slx45t
    Package:       fgg484
    Speed:         -3

    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: false
    VHDL Source Analysis Standard: VHDL-93

    Message Filtering: disabled
```

Figure 2

3. Add New Source in the Project2scr project :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `mux_2_1_8` in the File Name field, select VHDL Module and click Next (see **Figure 3**).

The mux will have two 8-bit inputs (data inputs A and B), one 1-bit input (select input SEL) and one 8-bit output (data output MUX_OUT). So the next New Source Wizard window must be completed as shown in **Figure 4**.



New Source Wizard

Select Source Type

Select source type, file name and its location.

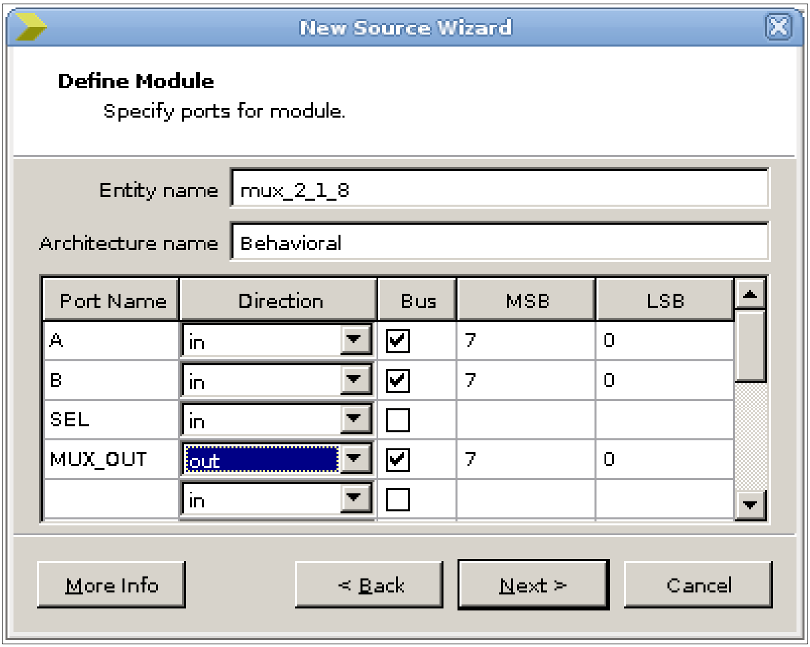
- IP (CORE Generator & Architecture Wizard)
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: mux_2_1_8

Location: /home/fpga-user/Documents/Project2scr

☑ Add to project

Figure 3



New Source Wizard

Define Module

Specify ports for module.

Entity name: mux_2_1_8

Architecture name: Behavioral

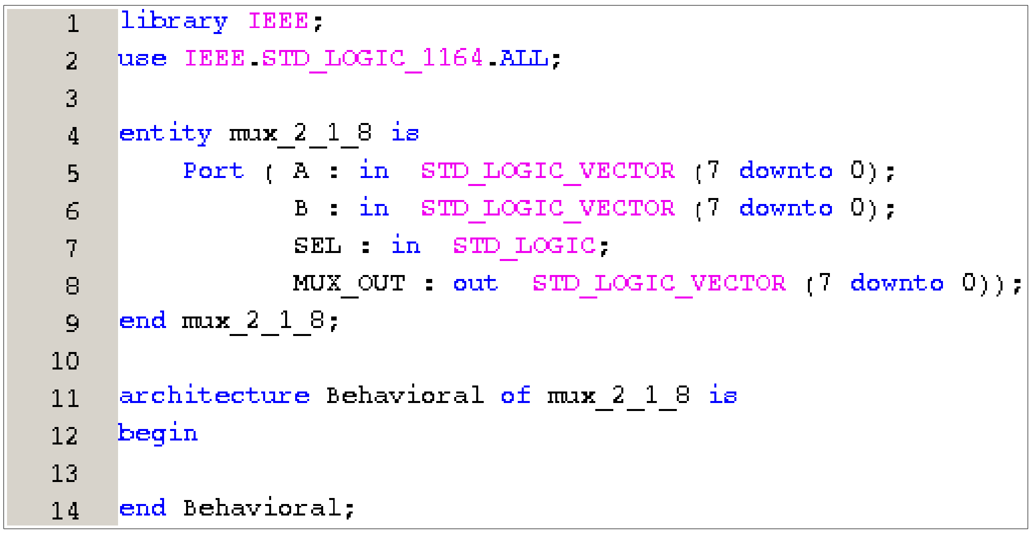
Port Name	Direction	Bus	MSB	LSB
A	in	☑	7	0
B	in	☑	7	0
SEL	in	☐		
MUX_OUT	out	☑	7	0
	in	☐		

Figure 4

Clicking Next and Finish, the file `mux_2_1_8.vhd` is created.

4. Define the behavior of the `mux_2_1_8` VHDL module :

Remove the green highlighted lines, the comments, and edit the `mux_2_1_8.vhd` file to look like the one shown in **Figure 5**.

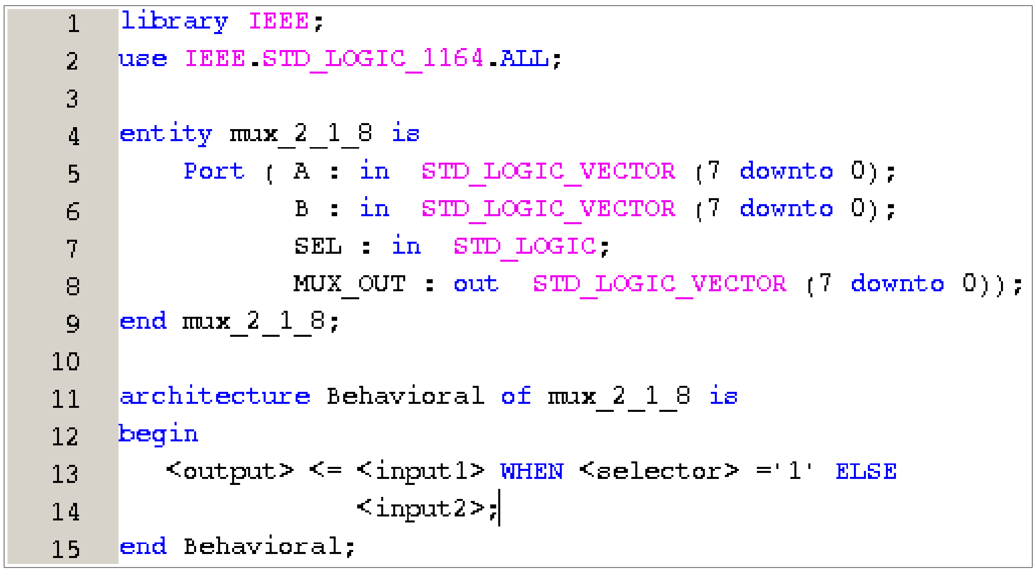


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_2_1_8 is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL : in  STD_LOGIC;
           MUX_OUT : out  STD_LOGIC_VECTOR (7 downto 0));
end mux_2_1_8;

architecture Behavioral of mux_2_1_8 is
begin

end Behavioral;
```

Figure 5

In **Figure 5** ports `A`, `B` and `MUX_OUT` are of type `STD_LOGIC_VECTOR`. The `STD_LOGIC_VECTOR` type represents an array of `STD_LOGIC` objects [3]. The `(7 downto 0)` expression that follows `STD_LOGIC_VECTOR` type indicates array's length (8 bits), the most significant bit (7) and the least significant bit (0) – see also **Figure 4**. If the expression following `STD_LOGIC_VECTOR` type was `(0 to 7)`, the most significant bit would be bit 0 and the least significant bit would be bit 7.

In the ISE Project Navigator window, select `Edit -> Language Templates...`. In the subwindow that opens expand VHDL, Synthesis Constructs, Coding Examples, Multiplexers and select 2-to-1 (assign). Copy the expression on the right and paste it in the architecture body of `mux_2_1_8` module (see **Figure 6**).

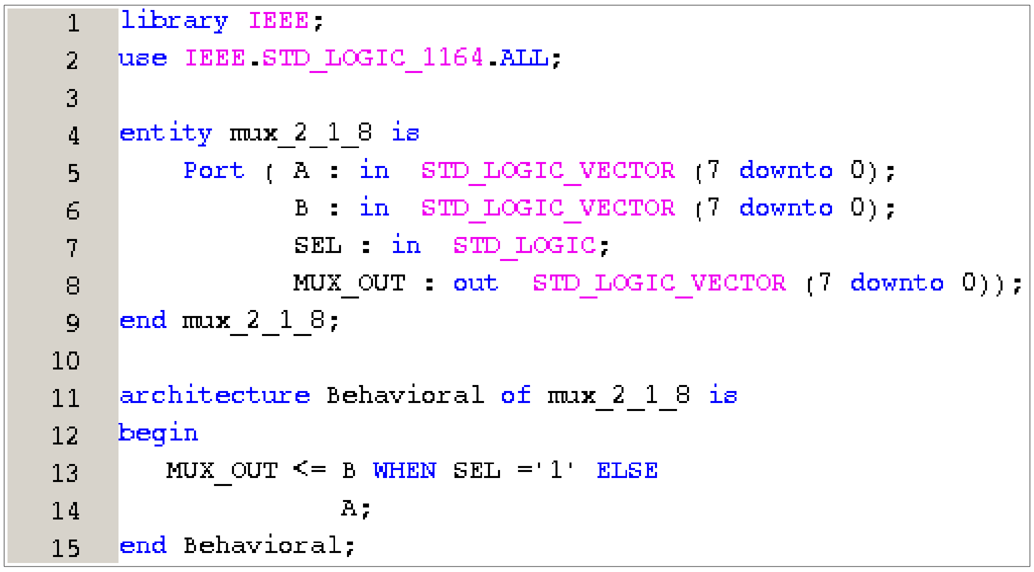


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_2_1_8 is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL : in  STD_LOGIC;
           MUX_OUT : out  STD_LOGIC_VECTOR (7 downto 0));
end mux_2_1_8;

architecture Behavioral of mux_2_1_8 is
begin
   <output> <= <input1> WHEN <selector> ='1' ELSE
               <input2>;
end Behavioral;
```

Figure 6

Replace `<output>` with `MUX_OUT`, `<input1>` with `B`, `<selector>` with `SEL` and `<input2>` with `A` (see **Figure 7**). Save the changes (`File → Save` or use the `Ctrl + S` key combination).

Figure 7 shows the `mux_2_1_8` module after the above-mentioned replacements. The lines `13-14` describe a conditional signal assignment. When the select port `SEL` is `'1'` (condition), `MUX_OUT` takes input's `B` value, else input's `A` value. This behavior describes a multiplexer.



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

entity mux_2_1_8 is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL : in  STD_LOGIC;
           MUX_OUT : out  STD_LOGIC_VECTOR (7 downto 0));
end mux_2_1_8;

architecture Behavioral of mux_2_1_8 is
begin
   MUX_OUT <= B WHEN SEL ='1'  ELSE
              A;
end Behavioral;
```

Figure 7

Module `mux_2_1_8` is so simple that behavioral simulation can be skipped.

5. Generate ICON and VIO cores :

In the ISE Project Navigator window, select Implementation, select the `mux_2_1_8` module, expand Synthesize – XST and double-click View RTL Schematic to view the RTL schematic of the synthesized `mux_2_1_8` module. In the Set RTL/Tech Viewer Startup Mode select Start with a Schematic of the top-level block and click OK. Double-click on the synthesized `mux_2_1_8` module (black-box-like) graphic to view the schematic shown in **Figure 8**.

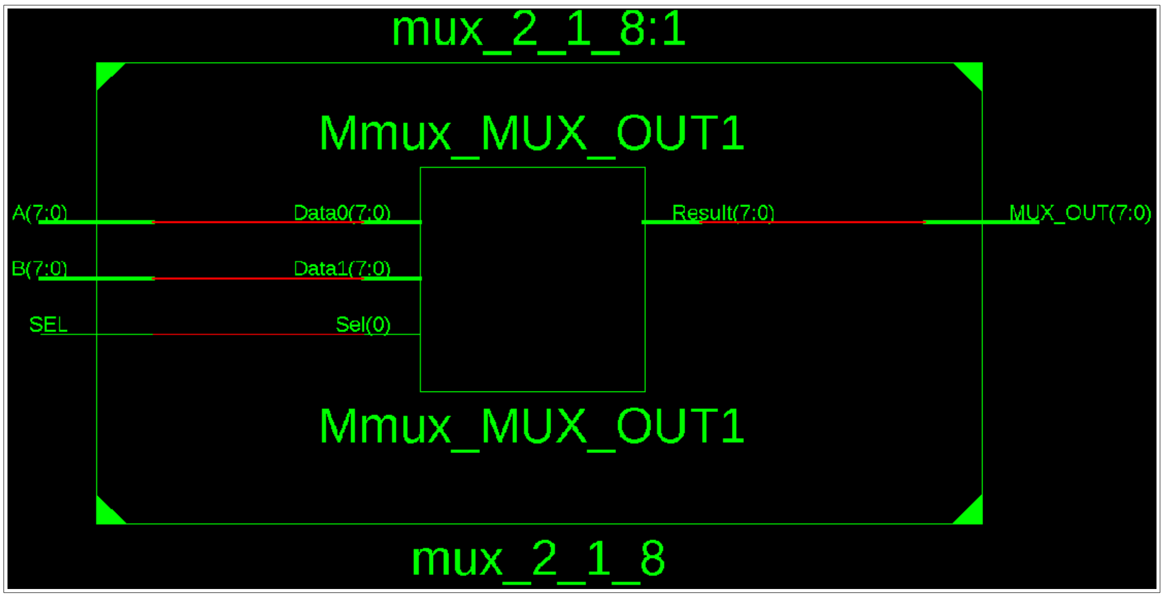
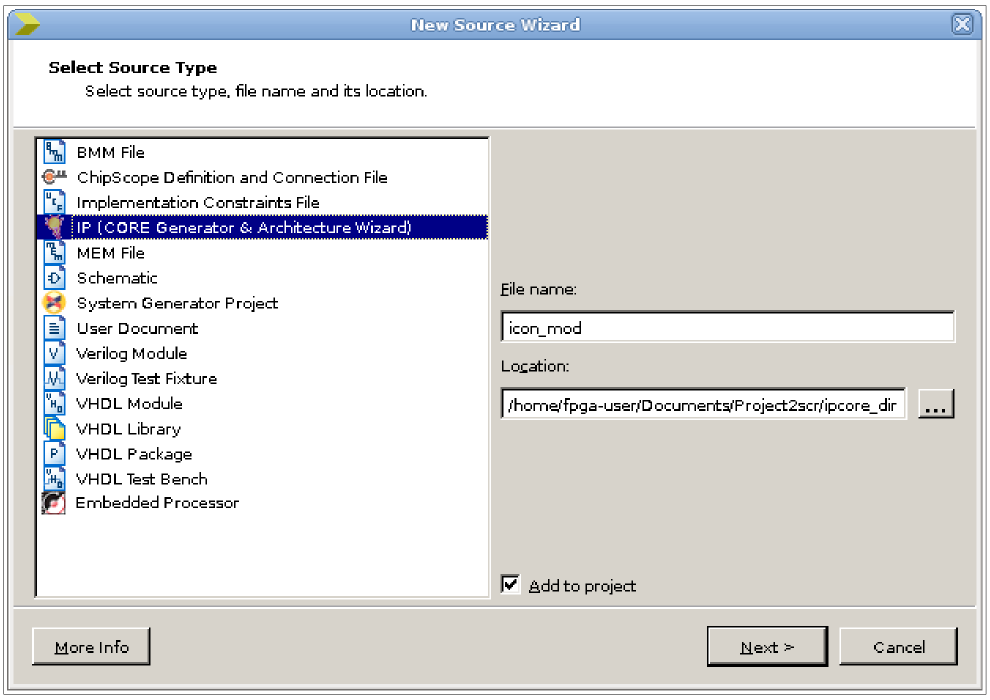


Figure 8

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window, type `icon_mod` in the File Name field, select IP (CORE Generator & Architecture Wizard) and click Next (see **Figure 9**).


New Source Wizard

Select Source Type
Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

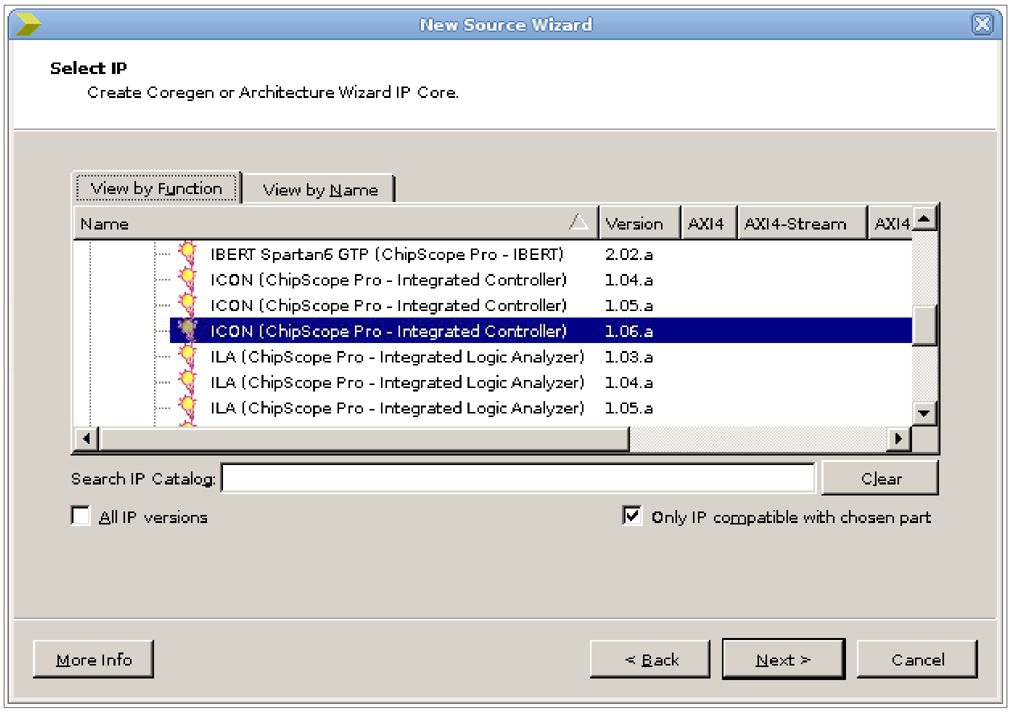
File name: icon_mod

Location: /home/fpga-user/Documents/Project2scr/ipcore_dir

☑ Add to project

Figure 9

In the New Source Wizard window (Select IP), check Only IP compatible with chosen part, expand Debug & Verification, expand ChipScope Pro, select ICON (ChipScope Pro – Integrated Controller) version 1.06.a, click Next and Finish (see **Figure 10**).



Name	Version	AXI4	AXI4-Stream	AXI4
IBERT Spartan6 GTP (ChipScope Pro - IBERT)	2.02.a			
ICON (ChipScope Pro - Integrated Controller)	1.04.a			
ICON (ChipScope Pro - Integrated Controller)	1.05.a			
ICON (ChipScope Pro - Integrated Controller)	1.06.a			
ILA (ChipScope Pro - Integrated Logic Analyzer)	1.03.a			
ILA (ChipScope Pro - Integrated Logic Analyzer)	1.04.a			
ILA (ChipScope Pro - Integrated Logic Analyzer)	1.05.a			

Figure 10

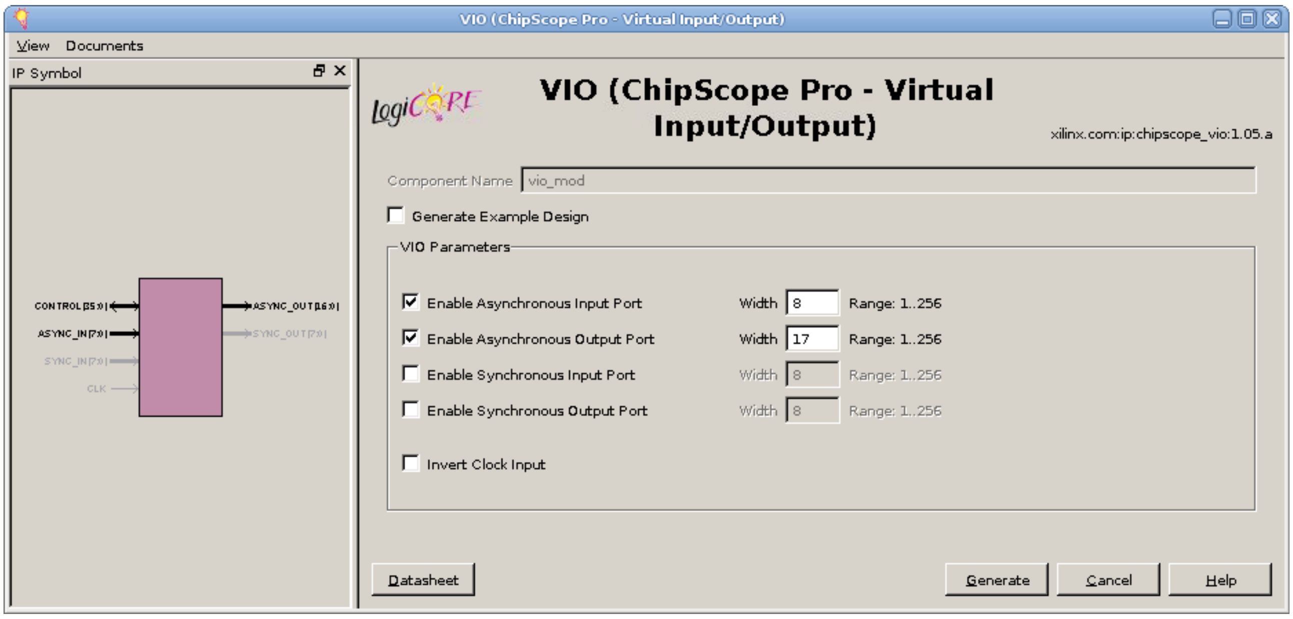
In the ICON (ChipScope Pro - Integrated Controller) window click Generate.

In order to use any ChipScope Pro core an ICON core is needed. The LogiCORE IP ChipScope Pro Integrated CONtroller core (ICON) provides an interface between the JTAG Boundary Scan (BSCAN) component of the FPGA and the ChipScope Pro cores (ILA, VIO, ATC2, AXI) [4]. These debug cores are directly attached to the ICON core control ports. Once generated, the ICON core is easily instantiated and connected to these cores using standard VHDL syntax.

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `vio_mod` in the File Name field, select IP (CORE Generator & Architecture Wizard) and click Next.

In the New Source Wizard window (Select IP), check Only IP compatible with chosen part, expand Debug & Verification, expand ChipScope Pro, select VIO (ChipScope Pro – Virtual Input/Output) version 1.05.a, click Next and Finish.

In the VIO (ChipScope Pro - Virtual Input/Output) window, check Enable Asynchronous Input Port (Width 8), check Enable Asynchronous Output Port (Width 17) and click Generate (see **Figure 11**). The VIO output port will feed the `mux_2_1_8` module input ports (8-bit A + 8-bit B + 1-bit SEL = 17 bits) and the VIO input port will read the `mux_2_1_8` module output port (8-bit MUX_OUT).



VIO (ChipScope Pro - Virtual Input/Output)

xilinx.com:ip:chipscope_vio:1.05.a

Component Name: vio_mod

☐ Generate Example Design

VIO Parameters

☑ Enable Asynchronous Input Port	Width 8	Range: 1..256
☑ Enable Asynchronous Output Port	Width 17	Range: 1..256
☐ Enable Synchronous Input Port	Width 8	Range: 1..256
☐ Enable Synchronous Output Port	Width 8	Range: 1..256

☐ Invert Clock Input

Figure 11

The `mux_2_1_8` module and the two cores (ICON and VIO) will be used as components in a single VHDL module (`mux_vio` module – top-module). The implementation of `mux_vio` module will be used in ChipScope Pro Analyzer for `mux_2_1_8` module verification.

6. Create and Define the structure of the `mux_vio` VHDL module :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `mux_vio` in the File Name field, select VHDL Module and click Next. In the next New Source Wizard window (Define Module) change the Architecture name field from `Behavioral` to `Structural`, click Next and Finish.

Module `mux_vio` doesn't have any ports (no ports were defined in module creation) because no FPGA general purpose I/O ports will be used for `mux_2_1_8` module verification. JTAG Boundary Scan (BSCAN) dedicated FPGA ports will be used (through the ICON core – the ICON core provides an interface between the JTAG Boundary Scan (BSCAN) component of the FPGA and the VIO core).

In the ISE Project Navigator window, select Implementation, select the `icon_mod` core, expand CORE Generator and double-click View HDL Instantiation Template (see **Figure 12**).

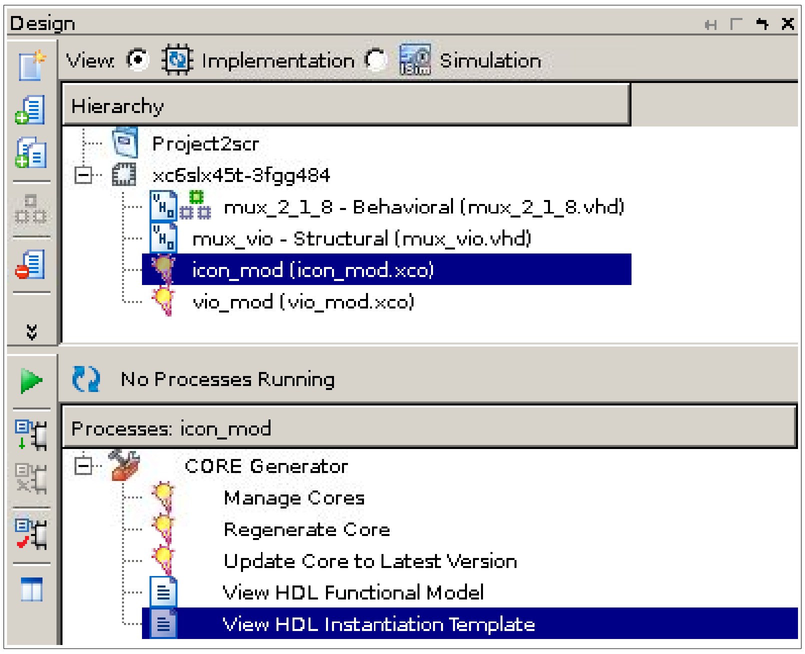
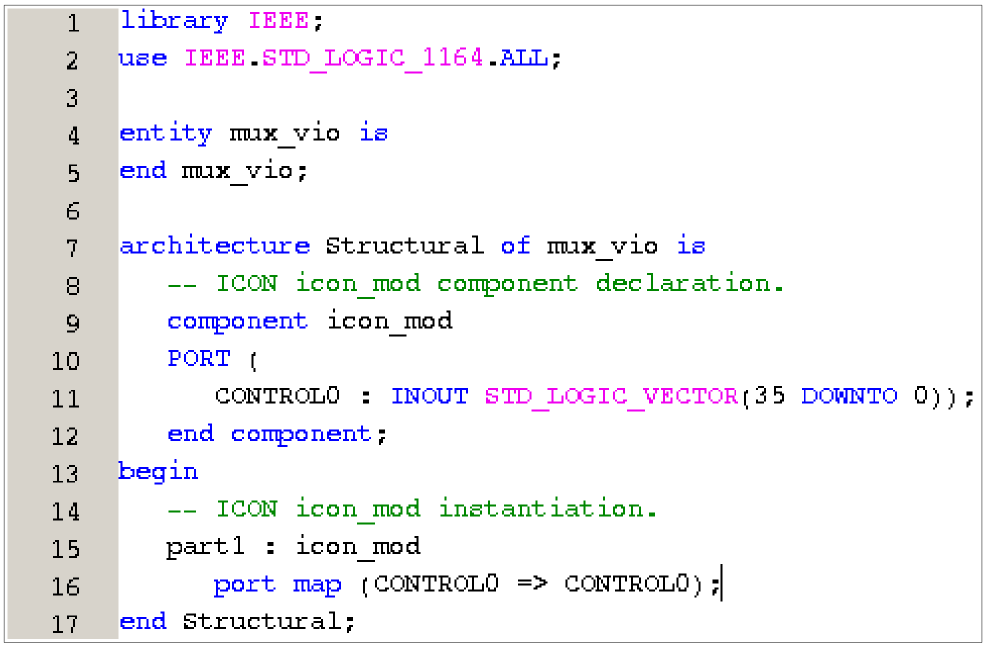

Design
View: Implementation Simulation
Hierarchy
Project2scr
xc6slx45t-3fgg484
mux_2_1_8 - Behavioral (mux_2_1_8.vhd)
mux_vio - Structural (mux_vio.vhd)
icon_mod (icon_mod.xco)
vio_mod (vio_mod.xco)
No Processes Running
Processes: icon_mod
CORE Generator
Manage Cores
Regenerate Core
Update Core to Latest Version
View HDL Functional Model
View HDL Instantiation Template

Figure 12

Copy the component declaration of the `icon_mod` core in the `mux_vio` module's architecture, before the word `begin`. Copy the instance declaration of the `icon_mod` core in the `mux_vio` module's architecture body and name the instance `part1` (see **Figure 13**).

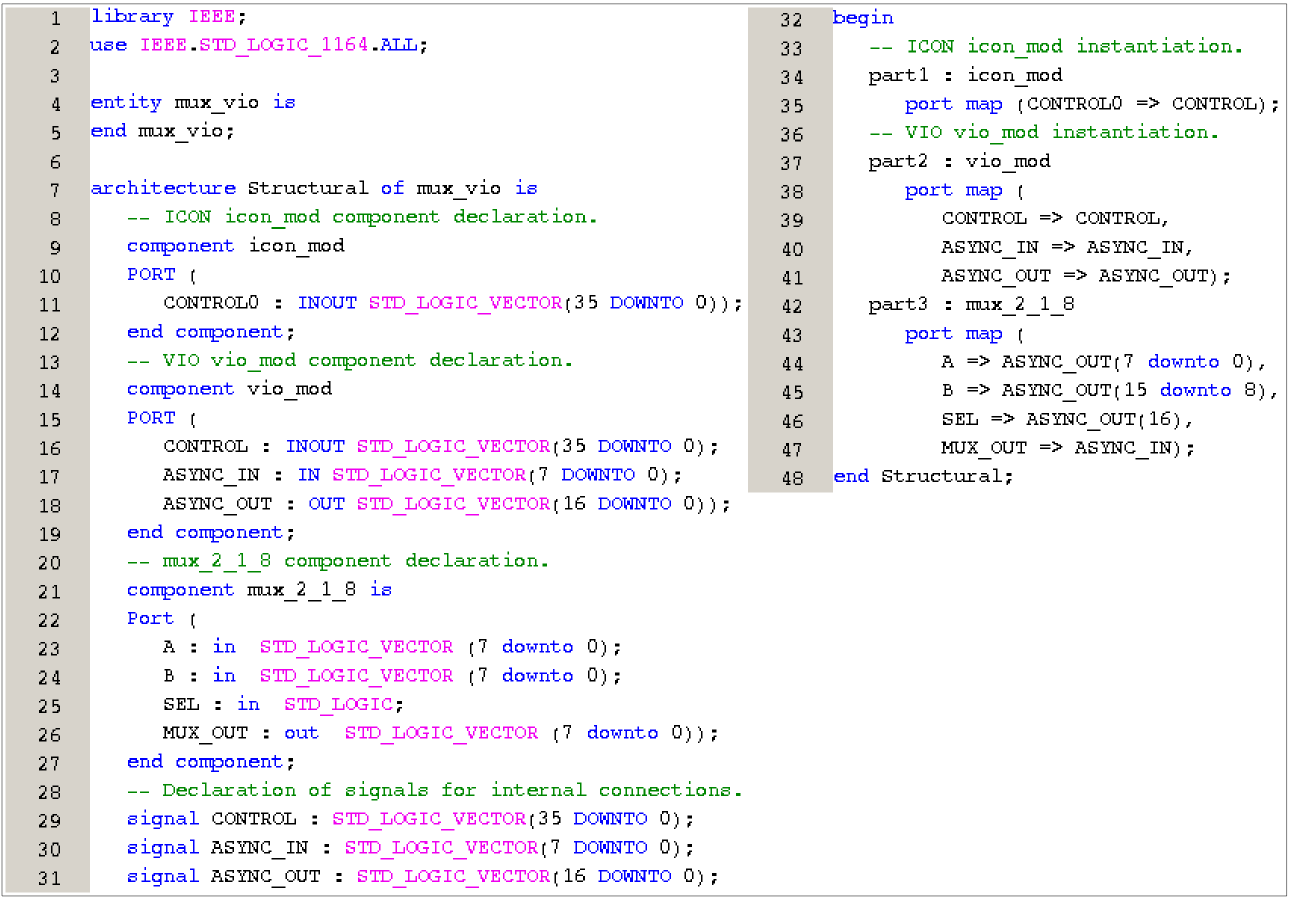
Follow the above-mentioned steps for `vio_mod` core instantiation in the `mux_vio` module.

Copy the entity declaration of the `mux_2_1_8` module to the `mux_vio` module's architecture, before the word `begin` and edit it (in the `mux_vio` module) to form the component declaration of the `mux_2_1_8` module. **Figure 14** shows how to instantiate `mux_2_1_8` in `mux_vio` module's architecture body and how to connect the cores and the module `mux_2_1_8` to complete `mux_vio` module's description.



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

entity mux_vio is
end mux_vio;

architecture Structural of mux_vio is
    -- ICON icon_mod component declaration.
    component icon_mod
    PORT (
        CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
    end component;
begin
    -- ICON icon_mod instantiation.
    part1 : icon_mod
        port map (CONTROL0 => CONTROL0);
end Structural;
```

Figure 13



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

entity mux_vio is
end mux_vio;

architecture Structural of mux_vio is
   -- ICON icon_mod component declaration.
   component icon_mod
   PORT (
      CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
   end component;
   -- VIO vio_mod component declaration.
   component vio_mod
   PORT (
      CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
      ASYNC_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      ASYNC_OUT : OUT STD_LOGIC_VECTOR(16 DOWNTO 0));
   end component;
   -- mux_2_1_8 component declaration.
   component mux_2_1_8 is
   Port (
      A : in  STD_LOGIC_VECTOR (7 downto 0);
      B : in  STD_LOGIC_VECTOR (7 downto 0);
      SEL : in  STD_LOGIC;
      MUX_OUT : out  STD_LOGIC_VECTOR (7 downto 0));
   end component;
   -- Declaration of signals for internal connections.
   signal CONTROL : STD_LOGIC_VECTOR(35 DOWNTO 0);
   signal ASYNC_IN : STD_LOGIC_VECTOR(7 DOWNTO 0);
   signal ASYNC_OUT : STD_LOGIC_VECTOR(16 DOWNTO 0);
begin
   -- ICON icon_mod instantiation.
   part1 : icon_mod
      port map (CONTROL0 => CONTROL);
   -- VIO vio_mod instantiation.
   part2 : vio_mod
      port map (
         CONTROL => CONTROL,
         ASYNC_IN => ASYNC_IN,
         ASYNC_OUT => ASYNC_OUT);
   part3 : mux_2_1_8
      port map (
         A => ASYNC_OUT(7 downto 0),
         B => ASYNC_OUT(15 downto 8),
         SEL => ASYNC_OUT(16),
         MUX_OUT => ASYNC_IN);
end Structural;
```

Figure 14

In the ISE Project Navigator window, select Implementation, right-click on `mux_vio` module and select `Select as Top Module`, select the `mux_vio` module, expand

Synthesize – XST and double-click Check Syntax to ensure `mux_vio` module's correctness.

7. Synthesize and Implement the `mux_vio` design :

In the ISE Project Navigator window select Implementation, select the `mux_vio` module, expand Synthesize – XST and double-click View RTL Schematic to view the RTL schematic of the synthesized `mux_vio` module. In the Set RTL/Tech Viewer Startup Mode select Start with a Schematic of the top-level block and click OK. Double-click on the synthesized `mux_vio` module (black-box-like) graphic to view the schematic shown in **Figure 15**.

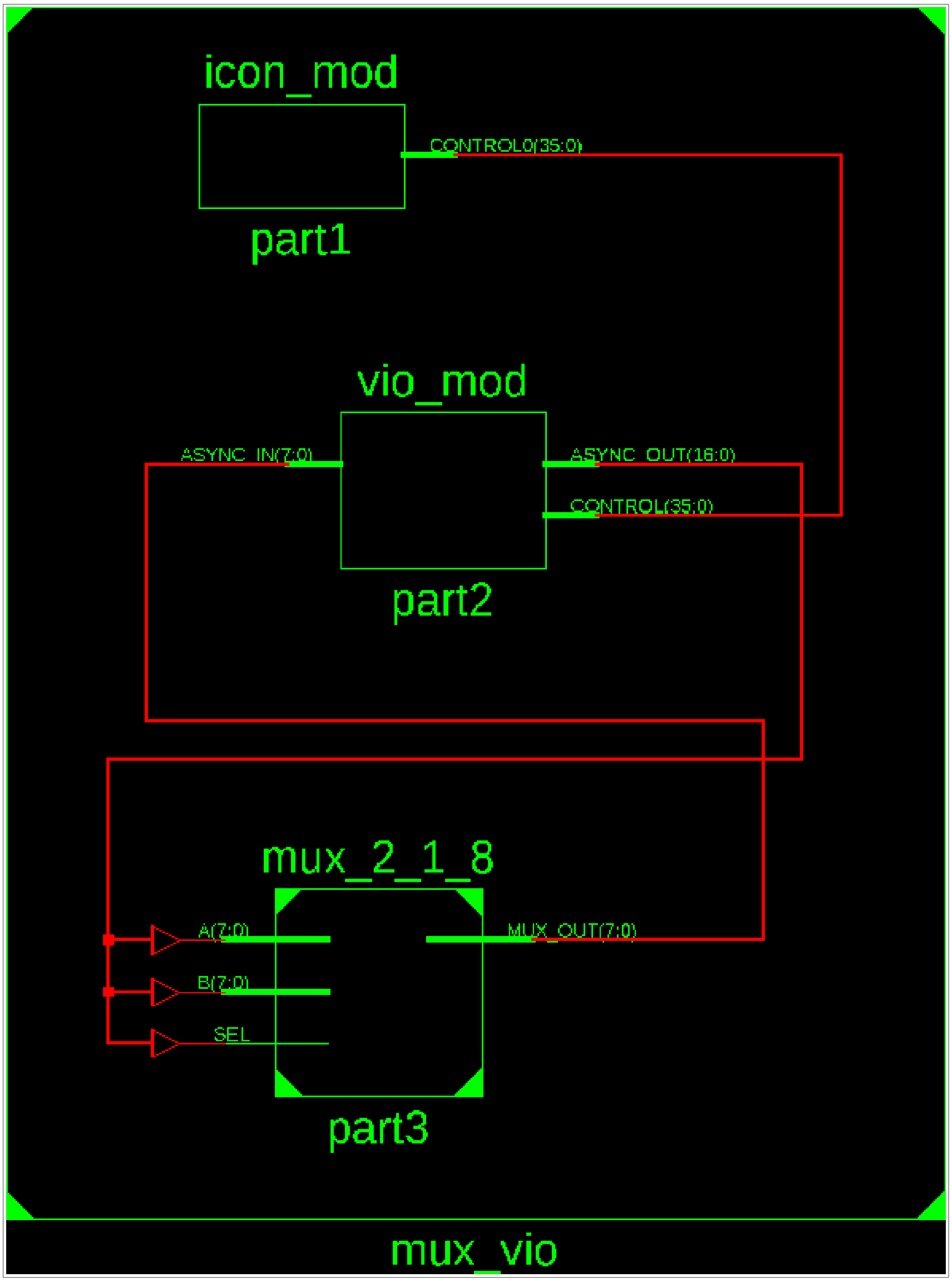


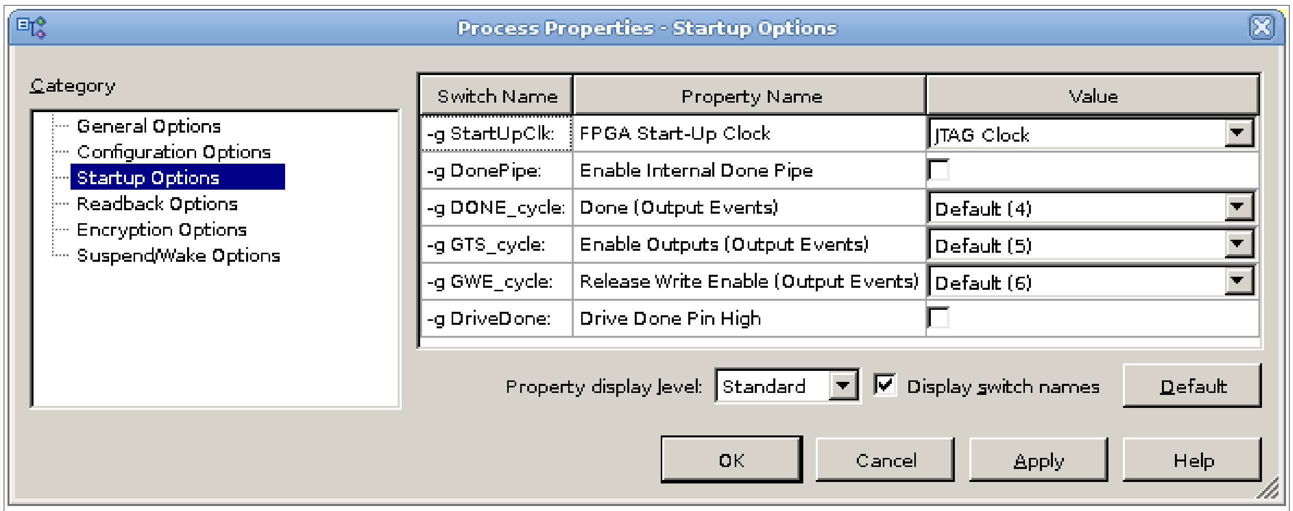
Figure 15

To implement the `mux_vio` module, in the ISE Project Navigator window select Implementation, select the `mux_vio` module and double-click Implement Design.

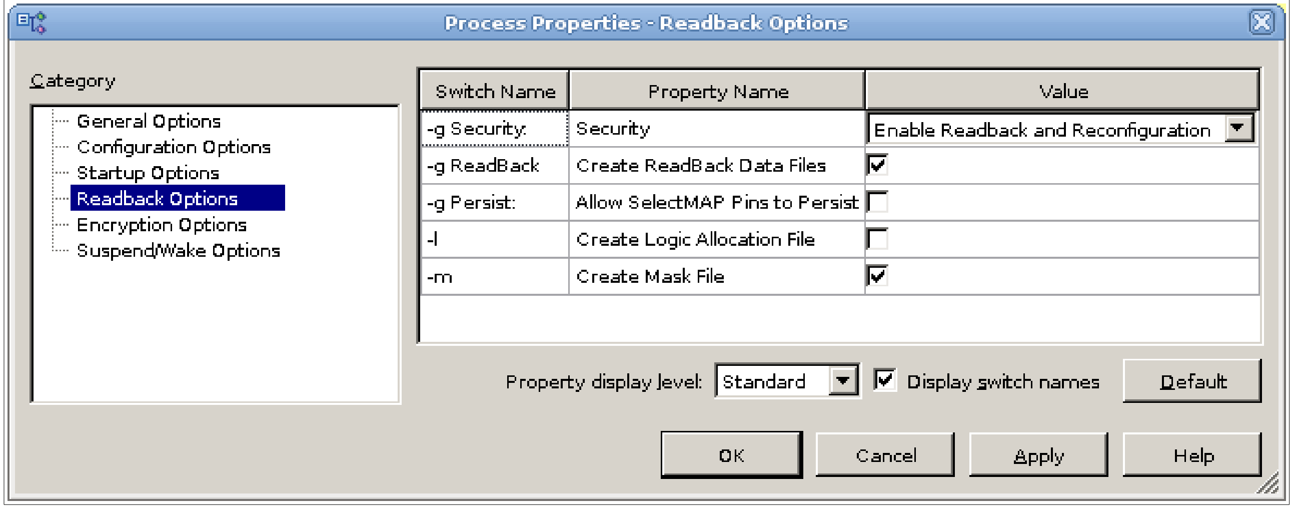
The `icon_mod` and `vio_mod` cores use the FPGA resources. Most of the FPGA resources used for the `mux_vio` module implementation are used by the `icon_mod` and `vio_mod` cores.

8. Generate Programming (Configuration) File :

In the ISE Project Navigator window select Implementation, select the `mux_vio` module, right-click Generate Programming File and select Process Properties. In the Process Properties – Startup Options window select Startup Options category and change FPGA Start-Up Clock property to JTAG Clock, then select Readback Options category and check Create Readback Data files and Create Mask File properties (see **Figure 16**). Click OK to close the Process Properties – Startup Options window. The above changes have been made in order to program the FPGA and verify FPGA programming using JTAG.



Switch Name	Property Name	Value
-g StartUpClk:	FPGA Start-Up Clock	JTAG Clock
-g DonePipe:	Enable Internal Done Pipe	☐
-g DONE_cycle:	Done (Output Events)	Default (4)
-g GTS_cycle:	Enable Outputs (Output Events)	Default (5)
-g GWE_cycle:	Release Write Enable (Output Events)	Default (6)
-g DriveDone:	Drive Done Pin High	☐



Switch Name	Property Name	Value
-g Security:	Security	Enable Readback and Reconfiguration
-g ReadBack	Create ReadBack Data Files	✓
-g Persist:	Allow SelectMAP Pins to Persist	☐
-l	Create Logic Allocation File	☐
-m	Create Mask File	✓

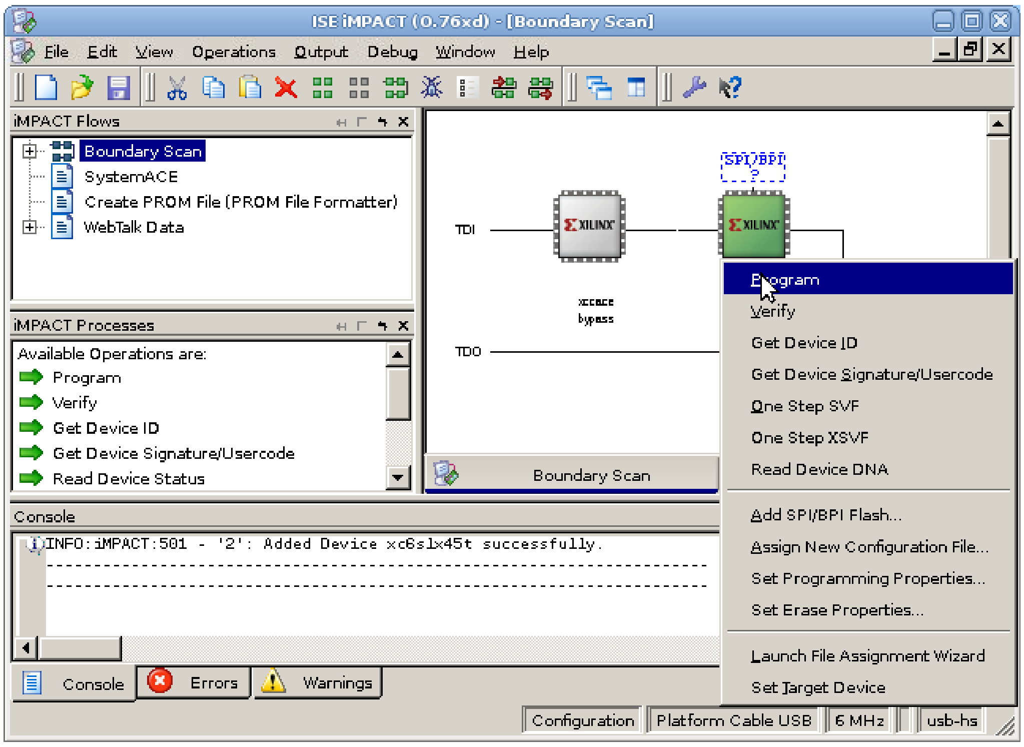
Figure 16

Double-click Generate Programming File to complete programming file generation.

9. Configure the FPGA :

In the ISE Project Navigator window select Implementation, select the `mux_vio` module, double-click Configure Target Device and click OK to open Impact. Impact software tool is used for FPGA configuration. In the ISE iMPACT window double-click Boundary Scan.

Right-click on Right click to Add Device or Initialize JTAG chain and select Initialize Chain. Click Yes to the Auto Assign Configuration Files Query Dialog, select Bypass for the xccace device, navigate to `/home/fpga-user/Documents/Project2scr`, select `mux_vio.bit` file and click Open for the xc6slx45t Spartan-6 FPGA. Click No to the Attach SPI or BPI PROM window. Select Device 2 in the Device Programming Properties window and check the Verify property. Right-click on the xc6slx45t Spartan-6 FPGA and select Program (see **Figure 17**).



```
INFO:iMPACT:501 - '2': Added Device xc6slx45t successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
```

Figure 17

After the Program Succeeded message close the ISE iMPACT window (click No to iMPACT-Save Project window).

10. Analyze design using ChipScope :

In the ISE Project Navigator window select Implementation, select the `mux_vio` module and double-click Analyze Design Using ChipScope. Click the Open Cable/Search JTAG Chain icon under the File menu in ChipScope Pro Analyzer window (see **Figure 18**) and click OK to the ChipScope Pro Analyzer (JTAG Chain Device Order) window.

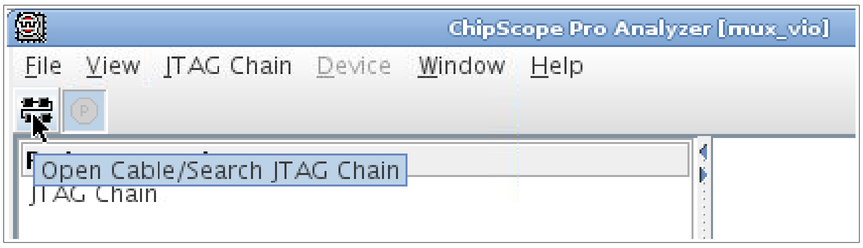

ChipScope Pro Analyzer [mux_vio]
File View JTAG Chain Device Window Help
Open Cable/Search JTAG Chain
JTAG Chain

Figure 18

Double-click the VIO Console (see **Figure 19**) to open it.

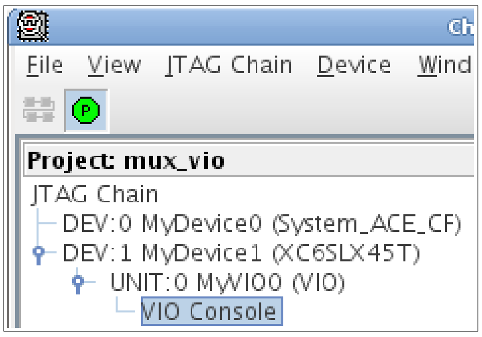
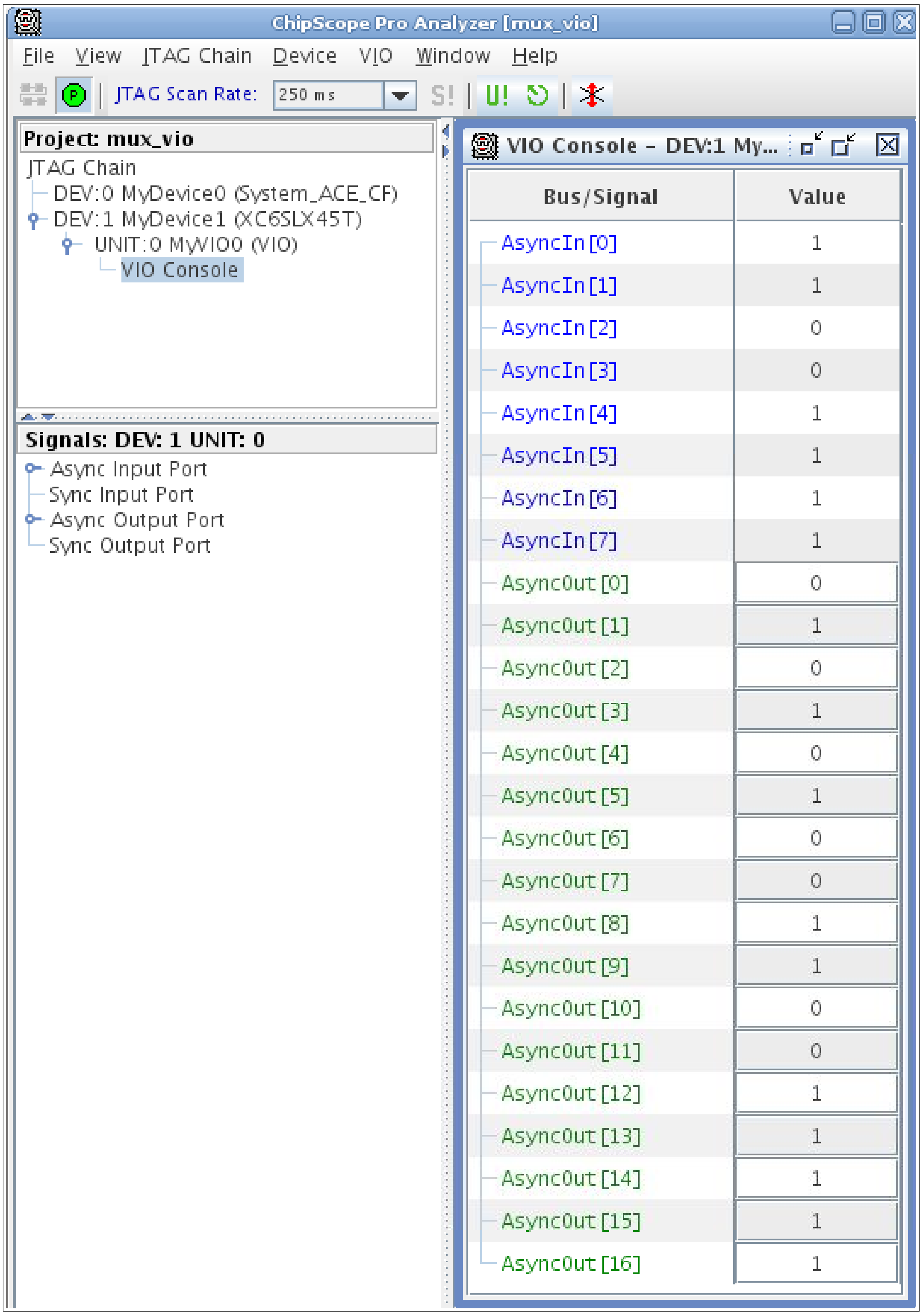

Ch
File View JTAG Chain Device Wind
Project: mux_vio
JTAG Chain
DEV:0 MyDevice0 (System_ACE_CF)
DEV:1 MyDevice1 (XC6SLX45T)
UNIT:0 MyVIO0 (VIO)
VIO Console

Figure 19

Change the bits to `AsyncOut` inputs and observe the `AsyncIn` outputs (see **Figure 20**). `AsyncOut(7 downto 0)` is connected to `mux_2_1_8` module's `A` input, `AsyncOut(15 downto 8)` is connected to `mux_2_1_8` module's `B` input, `AsyncOut(16)` is connected to `mux_2_1_8` module's `SEL` input and `AsyncIn(7 downto 0)` is connected to `mux_2_1_8` module's `MUX_OUT` output.

After verifying `mux_2_1_8` module's correct behavior, close all the windows.


ChipScope Pro Analyzer [mux_vio]
File View JTAG Chain Device VIO Window Help
JTAG Scan Rate: 250 ms

Project: mux_vio

JTAG Chain
- DEV:0 MyDevice0 (System_ACE_CF)
- DEV:1 MyDevice1 (XC6SLX45T)
  - UNIT:0 MyVIO0 (VIO)
    - VIO Console

Signals: DEV: 1 UNIT: 0
- Async Input Port
- Sync Input Port
- Async Output Port
- Sync Output Port

VIO Console – DEV:1 My...

Bus/Signal	Value
AsyncIn [0]	1
AsyncIn [1]	1
AsyncIn [2]	0
AsyncIn [3]	0
AsyncIn [4]	1
AsyncIn [5]	1
AsyncIn [6]	1
AsyncIn [7]	1
AsyncOut [0]	0
AsyncOut [1]	1
AsyncOut [2]	0
AsyncOut [3]	1
AsyncOut [4]	0
AsyncOut [5]	1
AsyncOut [6]	0
AsyncOut [7]	0
AsyncOut [8]	1
AsyncOut [9]	1
AsyncOut [10]	0
AsyncOut [11]	0
AsyncOut [12]	1
AsyncOut [13]	1
AsyncOut [14]	1
AsyncOut [15]	1
AsyncOut [16]	1

Figure 20

Conclution

Using ChipScope Pro tool and the related cores the designer can debug/verify a design in-circuit without having physical access to the FPGA board (just using remote access) or without using FPGA's ports and board components (push buttons and leds) that the board might not even have.

References

[1] Xilinx ISE 13.3 Help

[2] Xilinx Corporation, LogiCORE IP ChipScope Pro Virtual Input/Output (VIO) (1.04a), 2011, http://www.xilinx.com

[3] Stephen Brown and Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd Edition, McGraw-Hill, 2009

[4] Xilinx Corporation, LogiCORE IP ChipScope Pro Integrated Controller (ICON) (v1.05a), 2011, http://www.xilinx.com