ISim Hardware Co-Simulation Tutorial: Accelerating Floating Point Fast Fourier Transform Simulation

UG817 (v 13.2) July 28, 2011

XILINX®

Xilinx is disclosing this user guide, manual, release note, and/or specification (the “Documentation”) to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU “AS-IS” WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© Copyright 2002-2011 Xilinx Inc. All Rights Reserved. XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners. The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

Table of Contents

XILINX®

Chapter 1

Introduction

This tutorial describes how to use ISim Hardware Co-simulation to accelerate the simulation of floating point fast Fourier Transform (FFT) and verify the FFT implementation on a Xilinx® ML605 board.

Digital Signal Processing (DSP) designs are typically very time-consuming to simulate in software due to their data and computation intensiveness. A fast bit-accurate model is often used to speed up the simulation of a DSP function, but it does not provide any cycle accuracy and is not straightforward to integrate with other Register Transfer Level (RTL) modules. A behavioral RTL model provides bit-and-cycle accuracy but is relatively slower to simulate. A structural RTL or gate-level model is even much slower to simulate. Sometimes an IP does not provide a fast bit-accurate model or even a behavioral RTL model, which leaves the slowest structural/gate-level simulation as the only choice. ISim hardware co-simulation provides an additional mean to simulate DSP functions by offloading intensive computations to FPGA. It can bring synthesizable HDL code, synthesized or protected netlists such as IP cores generated by CORE Generator™ into FPGA for co-simulation. That solves the problem on getting a bit-and-cycle accurate simulation models as well as bolstering the simulation performance. For many complex DSP designs, not only does it accelerate the simulation of a design, it verifies the implementation of the design on actual hardware. ISim hardware co-simulation complements RTL, post-synthesis, and post-implementation simulation to complete the verification tool suite.

Prerequisites

- ISE® Design Suite, version 13.2
- Virtex®-6 FPGA ML605 Evaluation Kit
- Design File: rdf0125_fft_sim_tutorial.zip

Tutorial Files

File	Description
fp_fft_top.v	Wrapper that instantiates the floating point FFT core.
fp_fft_tb.v	Top-level test bench that generates test vectors to exercise the FFT core.
FloatingPointFFT.xise	ISE® project for this tutorial.
sim.tcl	Custom simulation command file that measures the ISim simulation time.
fp_fft_tb.wcfg	Custom waveform configuration file.
fp_fft_tb.prj	ISim project file for the command line flow.
full_compile.bat	Windows batch file to fully compile the design for hardware co-simulation with the Fuse command line.

File	Description
full_compile.sh	Linux shell script to fully compile the design for hardware co-simulation with the Fuse command line.
incr_compile.bat	Windows batch file to incrementally compile the test bench for hardware co-simulation with the Fuse command line.
incr_compile.sh	Linux shell script to incrementally compile the test bench for hardware co-simulation with the Fuse command line.
run_isim.bat	Windows batch file to launch the ISim simulation.
run_isim.sh	Linux shell script to launch the ISim simulation.

Note When performing this tutorial, all data files must be copied to your current working directory.

Tutorial

This tutorial describes how to use ISim Hardware Co-simulation to accelerate the simulation of floating point Fast Fourier Transform (FFT) and verify the FFT implementation on a Xilinx® ML605 board.

This tutorial is separated into four sections that contain the steps you need to perform to run an FFT design through ISim hardware co-simulation. Perform the steps in the order that they are presented. These sections are as follows.

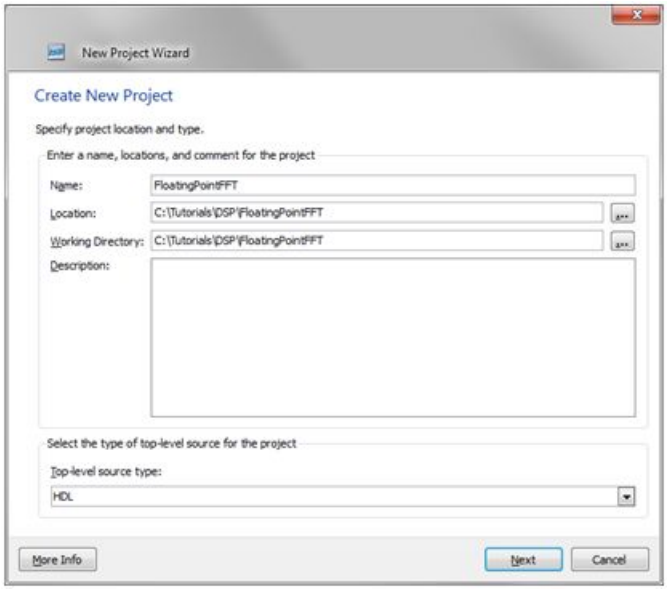
1. Generating a FFT Core in CORE Generator™
2. Creating a Test Bench
3. Compiling the Design for Hardware Co-Simulation
4. Running ISim Hardware Co-Simulation

Step 1: Generating a FFT Core in CORE Generator

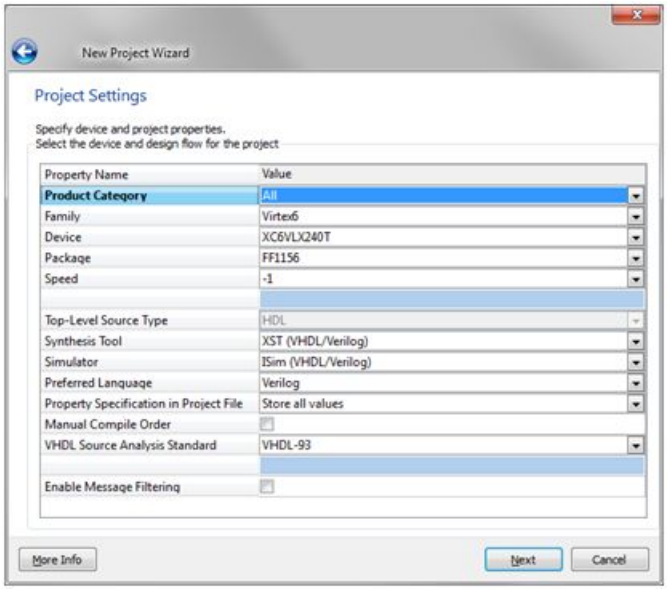
In this tutorial, we will use the Fast Fourier Transform IP core in the CORE Generator tool and create an ISim hardware co-simulation test bench that runs on the Virtex®-6 FPGA ML605 Evaluation Kit.

Note The screen captures in this tutorial are based on the Fast Fourier Transform version 7.1. The CORE Generator GUI might look different in later versions of the tool.

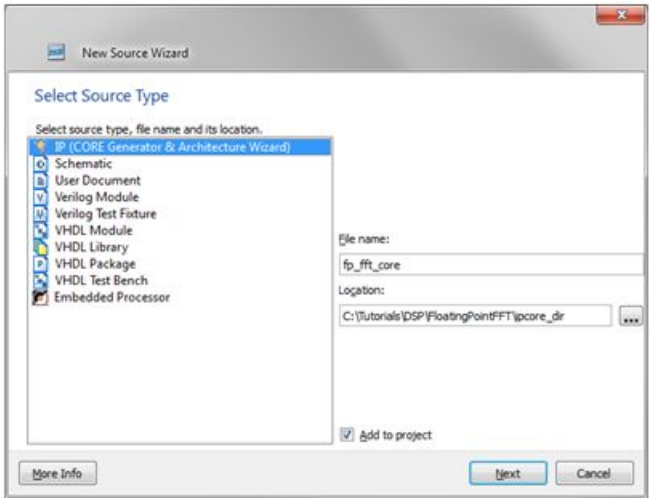
1. Launch the ISE® Project Navigator.
2. Choose **File > New Project** to open the New Project Wizard. Enter a project name (FloatingPointFFT) and location. Click **Next**.



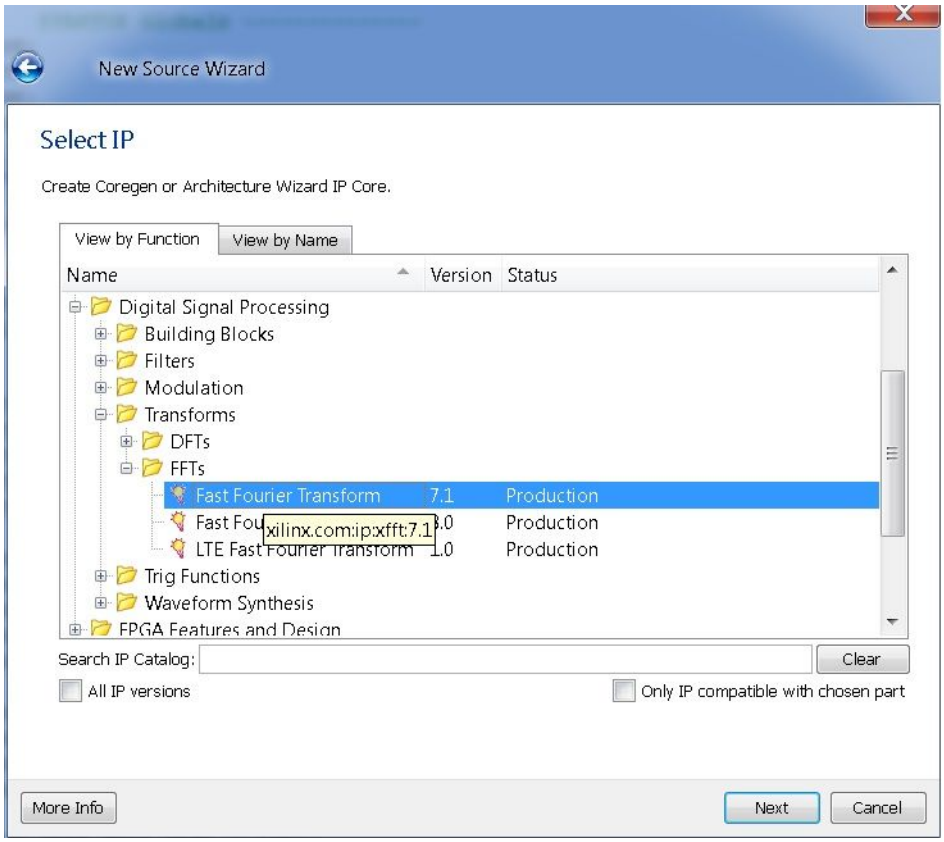
3. On the **Project Settings** page, choose the part for the ML605 board, which is Virtex®-6 device **XC6VLX240T**, package **FF1156**, and speed **-1**. Select **ISim** as the simulator and **Verilog** as the preferred language. Click **Next** and then **Finish** to complete the project creation.



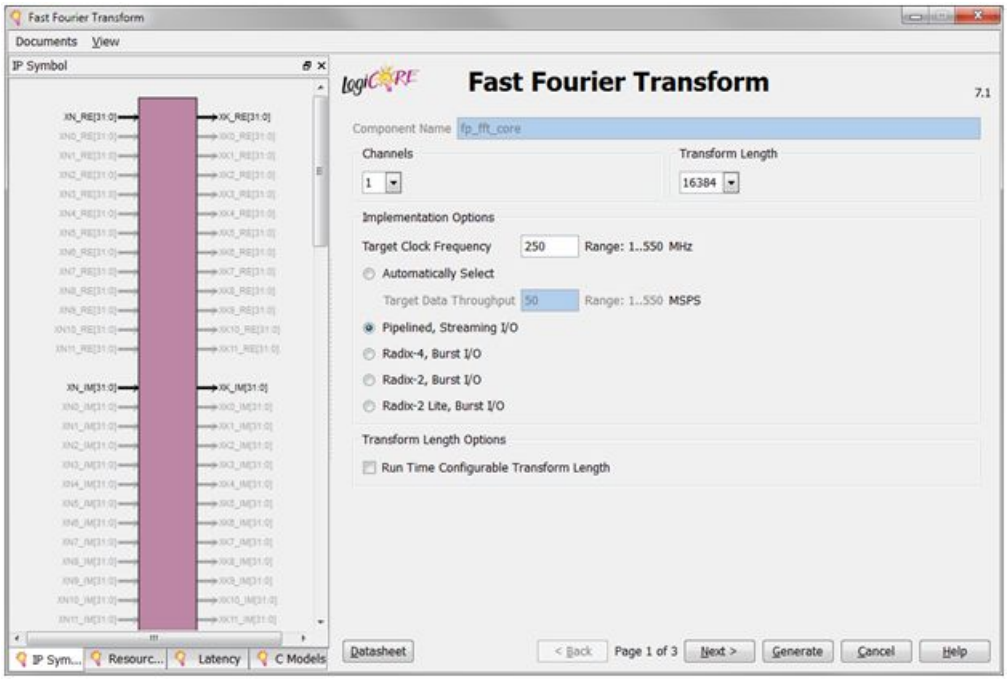
4. Choose **Project > New Source** to open the New Source Wizard. Select **IP (CORE Generator & Architecture Wizard)** and name the IP as **`fp_fft_core`**. Click **Next**.



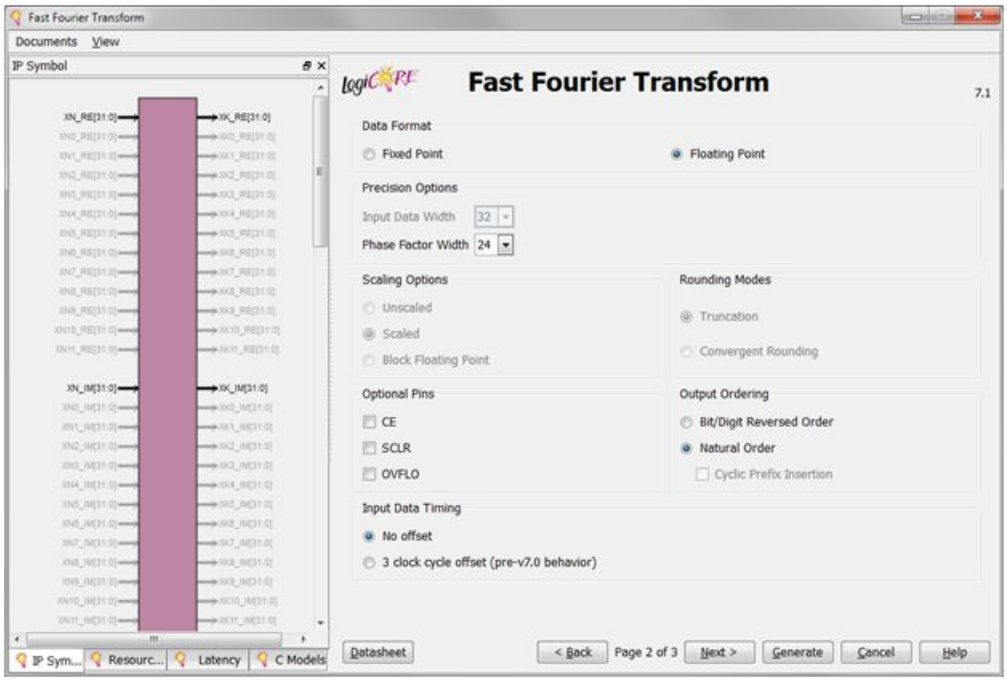
5. Select **Fast Fourier Transform version 7.1** from the IP list. Click **Next** and then **Finish**.



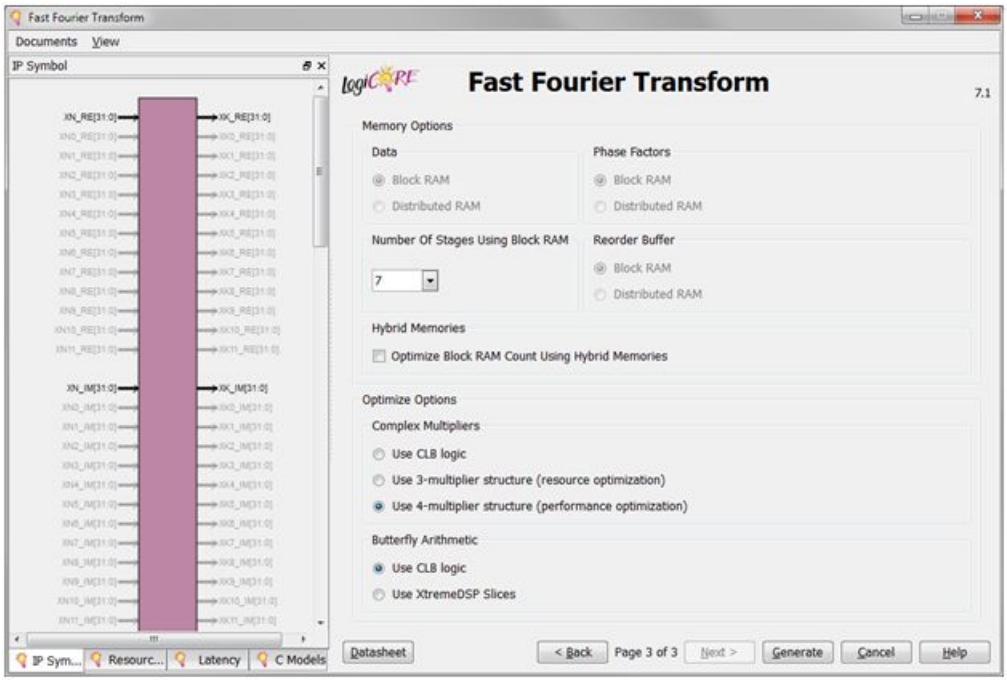
6. After launching the FFT core user interface, set the **Transform Length** to **16384**. Select **Pipelined, Streaming I/O** as the implementation. Click **Next**.



7. Set the Data Format to **Floating Point**, and the Output Ordering to **Natural Order**. Click **Next**.



8. Choose **Use 4-multiplier structure (performance optimization)** for the Complex Multipliers option. Click **Generate** to generate the core.



9. Add a top-level module, **`fp_fft_top`**, that instantiates the generated fp_fft_core IP core. This is required because ISim hardware co-simulation only supports a Hardware Definition Language (HDL) top-level module. You can use the completed `fp_fft_top.v` provided in this tutorial. Choose **Project > Add Source**. Add the `fp_fft_top.v`.

Step 2: Creating a Test Bench

1. Add a Verilog test bench module `fp_fft_tb.v` that generates test vectors to exercise the **`fp_fft_top`** instance. You can use the completed `fp_fft_tb.v` file provided in this tutorial. Choose **Project > Add Source**. Add the `fp_fft_tb.v`.

The test bench has four 32-bit by 16384 arrays `fft_xn_re_data`, `fft_xn_im_data`, `fft_xk_re_data`, `fft_xk_im_data`, for storing the real and imaginary components of the FFT input vector xn and output vector xk.

When the simulation starts, the test bench loads the FFT input vector from a data file `fft_xn_data.txt` into fft_xn_re_data and fft_xn_im_data array. After the FFT computation is done, the test bench stores the values in the fft_xk_re_data and fft_xk_im_data array in a result file `fft_xk_data.txt`. Both `fft_xn_data.txt` and `fft_xk_data.txt` store one data point per line. Each data point compromises two 32-bit hexadecimal values, the real and imaginary part respectively, separated by a space. The hexadecimal values are the binary representation of the floating point numbers using the IEEE 754 standard.

Note The TXT data files must be in the directory from which the simulation is being run.

The Verilog tasks defined in the test bench include:

- **clear_fft_xk_data**

  Fills the `fft_xk_re_data` and `fft_xk_im_data` array with zeros.

- **load_fft_xn_data**

  Load the data samples from the data file, `fft_xn_data.txt`, into the `fft_xn_re_data` and `fft_xn_im_data` array.

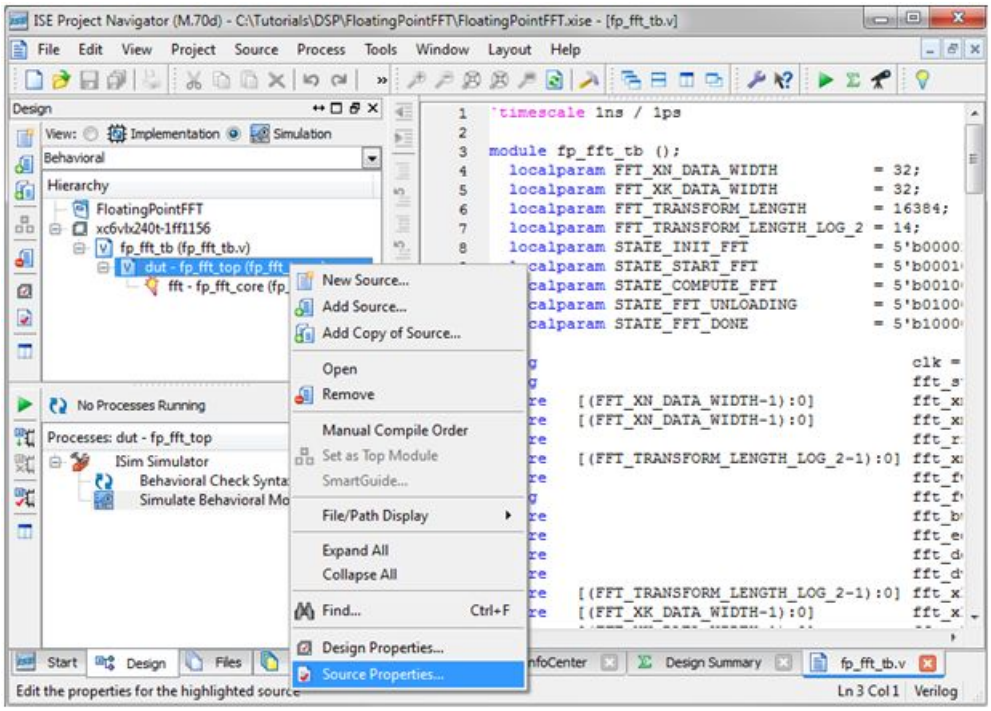
- **save_fft_xk_data**

  Save the data samples in the `fft_xn_re_data` and `fft_xn_im_data` array to the result file, `fft_xk_data.txt`.

Step 3: Compiling the Design for Hardware Co-Simulation

After you create the test bench and the custom constraints file, you can compile the design for hardware co-simulation using the ISim compiler. This can be done in Project Navigator by enabling Hardware Co-Simulation on a selected instance in your design. The selected instance, including its submodules, are co-simulated in hardware during the ISim simulation. Other modules are simulated in software.

1. Switch to the **Simulation View** in Project Navigator. Right-click the **dut – fp_fft_top** instance from the **Pane** of the Design Panel, and click **Source Properties**.



2. Select the Hardware Co-Simulation category. Check the **Enable Hardware Co-Simulation** check box. Set the Clock Port to **clk**. Select **ML605 (JTAG)** as the Target Board for the Hardware Co-Simulation. Leave the Enable Incremental Implementation option unchecked.

   Note The enabled instance for Hardware Co-Simulation is now marked with a special icon.

   The Enable Incremental Implementation option can be used after the design has been compiled for Hardware Co-Simulation. If the instance selected for Hardware Co-Simulation does not change in subsequent runs, you can turn on this option to skip the synthesis, implementation, and bitstream generation for Hardware Co-Simulation. It allows the test-bench or any portion simulated in software to be modified and simulated again quickly.

3. Select the **fp_fft_tb** instance from the Pane of the Design panel. Go to the Pane of the Design panel, right-click on Simulate Behavioral Model and click **Process Properties**.
4. Change the *Property display level* to Advanced. Set the properties for the Simulate Behavioral Model process.
5. Run the Simulate Behavioral Model process for the **fp_fft_tb** instance.

Compiling the Design on the Command Line

You can invoke the ISim compiler through the Fuse command line tool. You must provide Fuse a project file, the design top level module(s), and other optional arguments such as libraries to link in and library search paths. To compile the design for hardware co-simulation, you must provide the following extra arguments:

```
fuse -prj [project file] [top level modules]
     -hwcosim_instance [instance]
     -hwcosim_clock [clock]
     -hwcosim_board [board]
     -hwcosim_constraints [constraints file]
     -hwcosim_incremental [0|1]
```

- `hwcosim_instance`: Specifies the full hierarchical path of the instance to co-simulate in hardware
- `hwcosim_clock`: Specifies the port name of the clock input for the instance.
  - This is the clock in the lock-step portion, that is to be controlled by the test bench.
  - For a design with multiple clocks, specify the fastest clock using this option so that ISim can optimize the simulation. Other clock ports are treated as regular data ports.
- `hwcosim_board`: Specifies the identifier of the hardware board to use for co-simulation.
- `hwcosim_constraints` (optional): Specifies the custom constraints file that provides additional constraints for implementing the instance for hardware co-simulation. We also use the constraints file to specify which ports of the instance are mapped to external I/Os or clocks.
- `hwcosim_incremental` (optional): Specifies whether Fuse should reuse the last generated hardware co-simulation bitstream and skip the implementation flow.

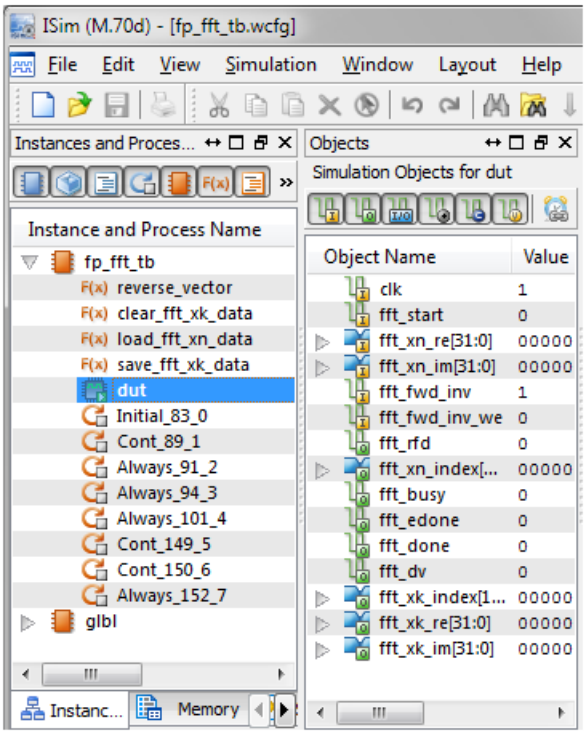
For example, to compile the FFT design for this tutorial, you can run the Fuse command line as follows:

```
fuse -prj fp_fft_tb.prj fp_fft_tb
     –o fp_fft_tb.exe
     -hwcosim_instance /fp_fft_tb/dut
     -hwcosim_clock clk
     -hwcosim_board ml605-jtag
```

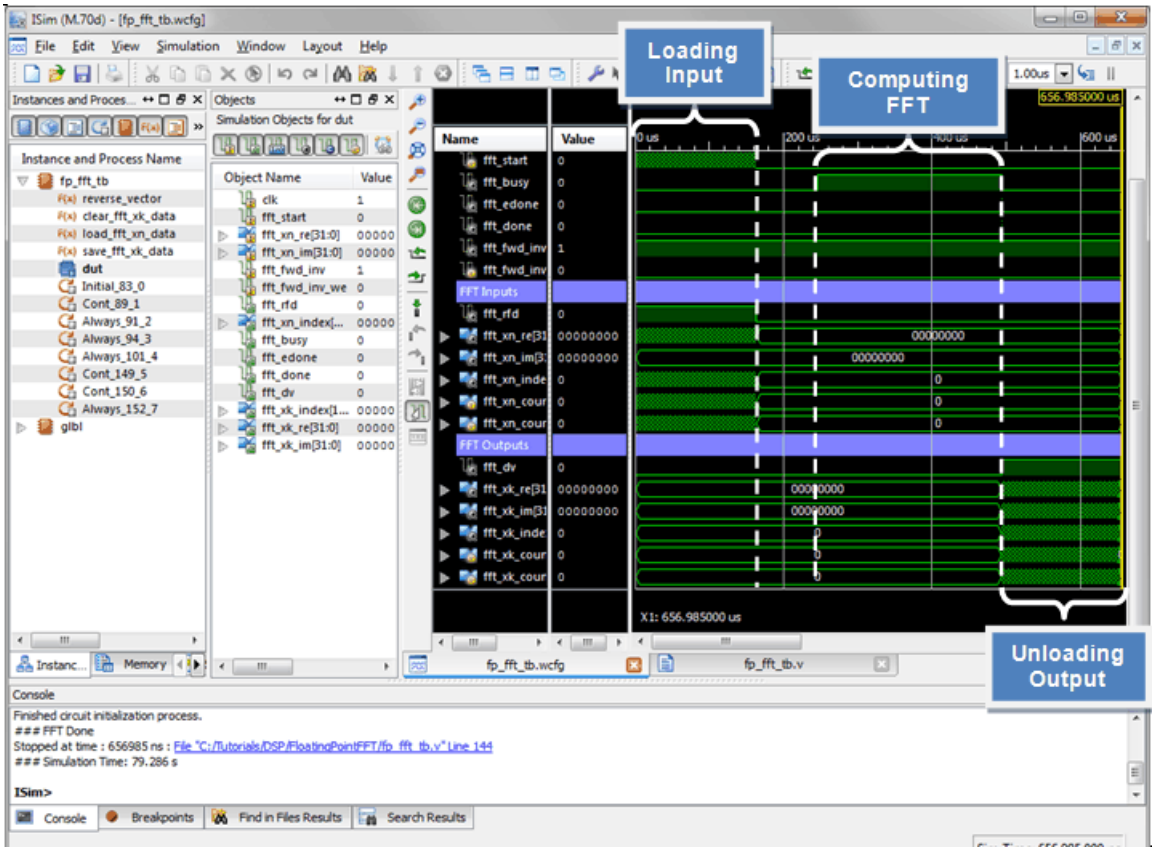
Step 4: Running ISim Hardware Co-Simulation

The simulation executable generated by the compiler runs in the same way in both software simulation and hardware co-simulation flow. Project Navigator automatically launches the simulation executable in GUI mode after the compilation finishes.

In the Instances and Processes view, the instance selected for hardware co-simulation is indicated with a special icon  . As the instance runs in hardware, you cannot expand it to see its internal signals and submodules.



Before the simulation starts, ISim programs the FPGA with the bitstream file generated for hardware co-simulation. You might notice the message in the ISim console window: `"Downloading bitstream, please wait till status is READY"`. After the FPGA is configured, the console shows `"Bitstream download is complete. READY for simulation."` From this point, you can run the simulation and interact with the ISim GUI the same way you do in the software simulation flow.



XILINX®

Chapter 3

Benchmark

The following table compares the performance[1] between ISim simulation and ISim hardware co-simulation.

- The FFT design takes about 65700 simulation clock cycles to finish.
- The compilation time measures the time taken to compile the FFT design into a simulation executable using the ISim compiler (Fuse).
- ISim hardware co-simulation takes more time for compilation as it runs through synthesis, implementation, and bitstream generation for the portion of design under co-simulation. As the number of simulation cycles increases, the performance gain in simulation using ISim hardware co-simulation becomes more substantial and amortizes the overhead in compilation.

As a good practice, you can first simulate your design in software for a small number of cycles to verify a few test cases. Then you switch to ISim hardware co-simulation for a longer simulation with more comprehensive test cases.

Using the **Enable Incremental Implementation** option, you can reuse the previously generated bitstream to save the compilation time if the portion of design under co-simulation has not changed. The compilation time could be even faster than a regular ISim compilation because only a subset of the design is recompiled. You can thus continuously modify the test bench or other portions of design that are simulated in software and achieve more design turns per day.

Simulation performance varies on different hardware co-simulation interfaces. The JTAG co-simulation interface is readily available on almost any FPGA board with a JTAG port. It consumes fewer resources on FPGA for implementing the hardware co-simulation interface; however, its latency and throughput are worse than other interfaces like Ethernet.

The point-to-point Ethernet co-simulation interface supports an Ethernet connection of 10/100/1000 Gbps and achieves a much higher throughput than JTAG; however, it consumes more resources on FPGA for implementing the hardware co-simulation interface and is not supported on all boards.

1. The benchmark was performed using ISE® 13.2 on a machine with a Core i7 2.8GHz CPU, 8GB RAM, and running a 64-bit Windows OS.

Comparison on Simulation Performance

Note Speed-ups relative to ISim simulation are in parenthesis

	Compilation Time	Simulation Time	Total Time
ISim simulation	49s	1383s	1432s
ISim hardware co-simulation on ML605 through JTAG[2]	685s (0.07x)	90s (15x)	775s (1.8x)
ISim hardware co-simulation on ML605 through JTAG[2] (Enable Incremental Implementation)	5s (10x)	90s (15x)	95s (15x)
ISim hardware co-simulation on ML605 through point-to-point Ethernet[3]	887s (0.05x)	5s (277x)	892s (1.6x)
ISim hardware co-simulation on ML605 through point-to-point Ethernet[3] (Enable Incremental Implementation)	5s (10x)	5s (277x)	10s (143x)

2. The JTAG co-simulation interface uses a Platform Cable USB with the speed configured to 12 MHz.
3. The point-to-point Ethernet co-simulation interface uses a Gigabit Ethernet connection directly connecting the PC and the ML605 board.

Appendix

Additional Resources

- **Xilinx Glossary** - http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- **Xilinx Documentation** - http://www.xilinx.com/support/documentation
- **Xilinx Support** - http://www.xilinx.com/support