XAPP739 (v1.0) September 23, 2011

# AXI Multi-Ported Memory Controller

Author: Khang Dao and Dylan Buli

## Summary

A multi-ported memory controller (MPMC) is used in applications where multiple devices share a common memory controller. This is a common requirement in many video, embedded, and communications applications where data from multiple sources move through a common memory device, typically DDR3 SDRAM memory.

This application note demonstrates how to create a basic DDR3 MPMC design using the ISE® Design Suite Logic Edition tools, including Project Navigator (ProjNav) and the CORE Generator™ tool. The MPMC is created by combining the Memory Interface Generator (MIG) IP block and the AXI Interconnect IP block, both provided in the ISE Design Suite Logic Edition.

AXI is a standardized IP interface protocol based on the ARM® AMBA4® AXI specification. The AXI interfaces used in this example design consist of AXI4, AXI4-Lite, and AXI4-Stream interfaces as described in the AMBA4 specification. These interfaces provide a common IP interface protocol framework for building the system.

## Overview

The example design in this application note is a full working hardware system on the Virtex®-6 FPGA ML605 evaluation platform board. This ML605 design implements a simple video system where data from a video test pattern generator (TPG) is looped in and out of memory multiple times before being sent to the digital visual interface (DVI) display on the board. The DDR3 memory therefore acts as a multi-ported memory being shared by multiple video frame buffers. The video frame buffers are controlled by two AXI Video Direct Memory Access (AXI VDMA) IP cores. Each AXI VDMA takes AXI4-Stream data carrying video information and moves the data to or from memory over AXI4 interfaces. The AXI Interconnect acts as an arbitrated switch to multiplex AXI4 transactions to the shared MIG memory controller, thus creating an AXI MPMC system. A clock generator block supplies clocks throughout the system, and AXI4-Lite master blocks generate the necessary configuration commands to set up the AXI VDMA after reset. Figure 1 provides a block diagram of the system and illustrates the basic data flow in the system.

***Note:*** The Xilinx Platform Studio (XPS) tools might provide a higher level of automation than the ISE Design Suite Embedded Edition (EDK) when building the equivalent system as described in this application note.

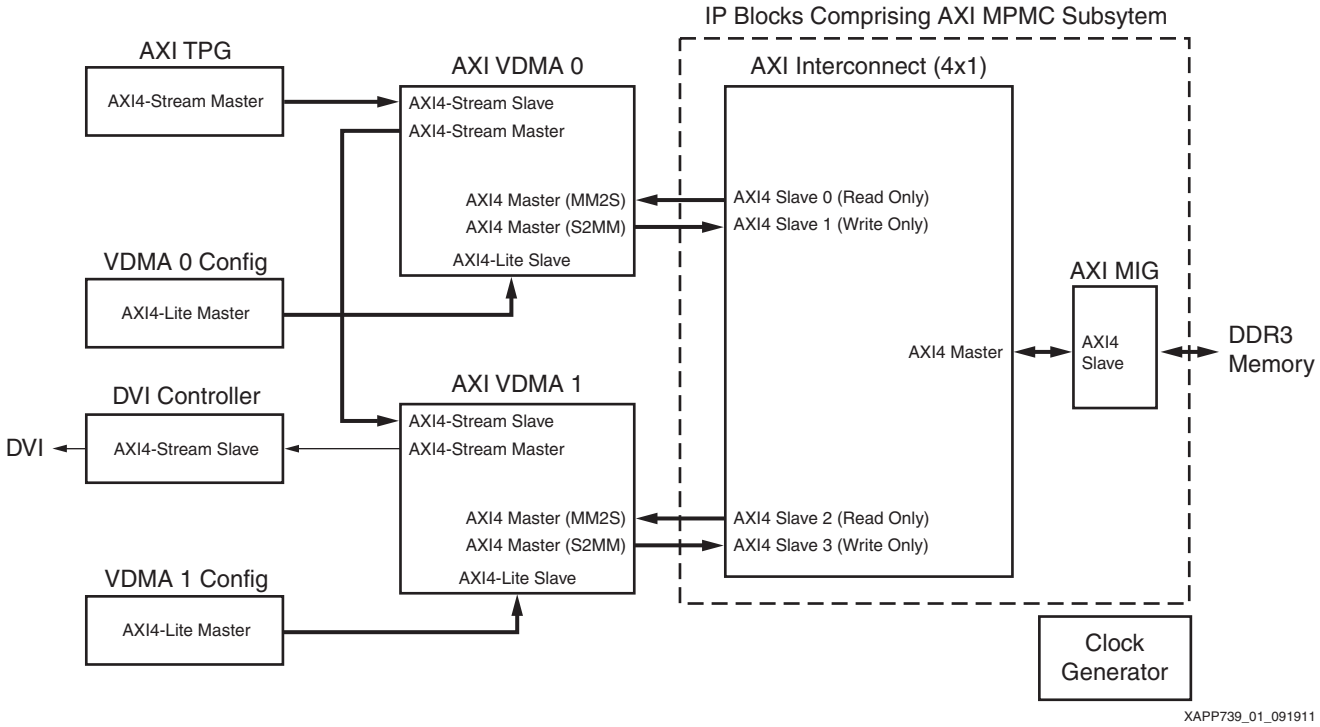



*Figure 1:* **Overview of AXI MPMC System (Project Navigator)**

# Quick Start

This section provides the steps to build the design starting from the complete project fileset and also shows how to run the demonstration design on the ML605 board.

***Note:*** Instructions to build the design from a new ProjNav project are covered in Creating the AXI MPMC Design from a New ProjNav Project, page 11.

## Steps to Open and Rebuild the Design

***Note:*** These steps are written using a Linux environment. For PC users, corresponding directory path separators and tool invocation commands might be needed, but generally, the GUI commands are the same.

1. Install the ISE Design Suite 13.2 (requires Logic Edition at a minimum)
2. Unzip the application note reference design files into a local folder (referred to as `<design_dir>`)
3. Open Project Navigator by selecting **Start** > **Xilinx ISE Design Suite 13.2** > **Project Navigator** in a PC or the "ise" command in Linux.

4. In ProjNav, select **File** > **Open Project** (Figure 2).

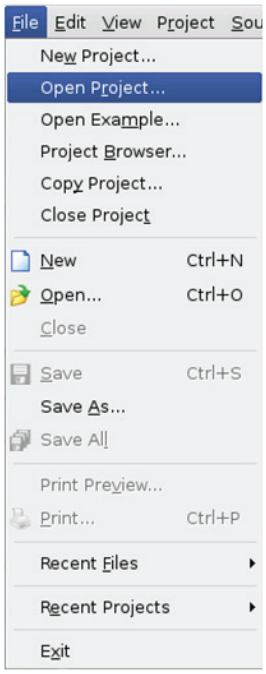



*Figure 2:* **Opening a Project in Project Navigator**

5. Select `<design_dir>/projnav/ml605_mpmc_reference_projnav.xise` and click **Open** (Figure 3).

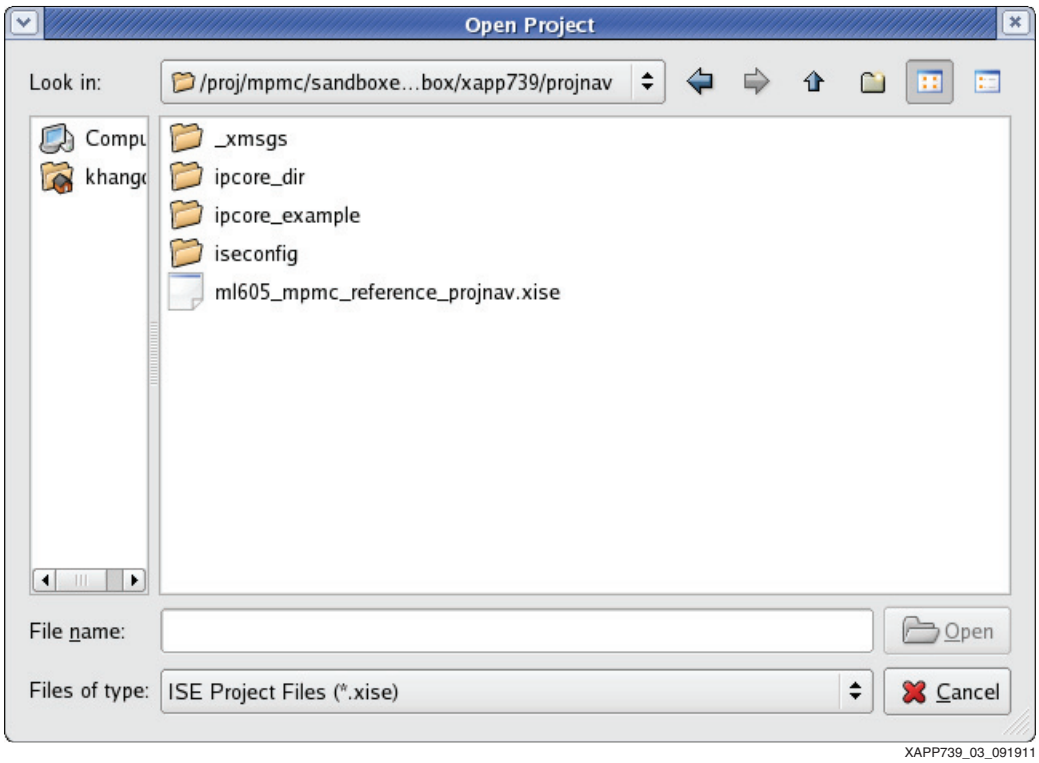



*Figure 3:* **Open Project Menu**

6. To not rebuild the design but only run the pre-generated bitstream in hardware, skip to step 8. Otherwise, to rebuild the design, continue with these instructions:
   a. If this is the first time the design is run, the IP cores in the project need to be regenerated. To rebuild the IP cores, select **xc6vlx240t-1ff156** in the Hierarchy pane and double-click **Design Utilities** > **Regenerate All Cores** (Figure 4).

      ***Note:*** If the cores are not regenerated and the design is implemented, Project Navigator prompts the user to regenerate each core (Click **Yes** for each IP core).

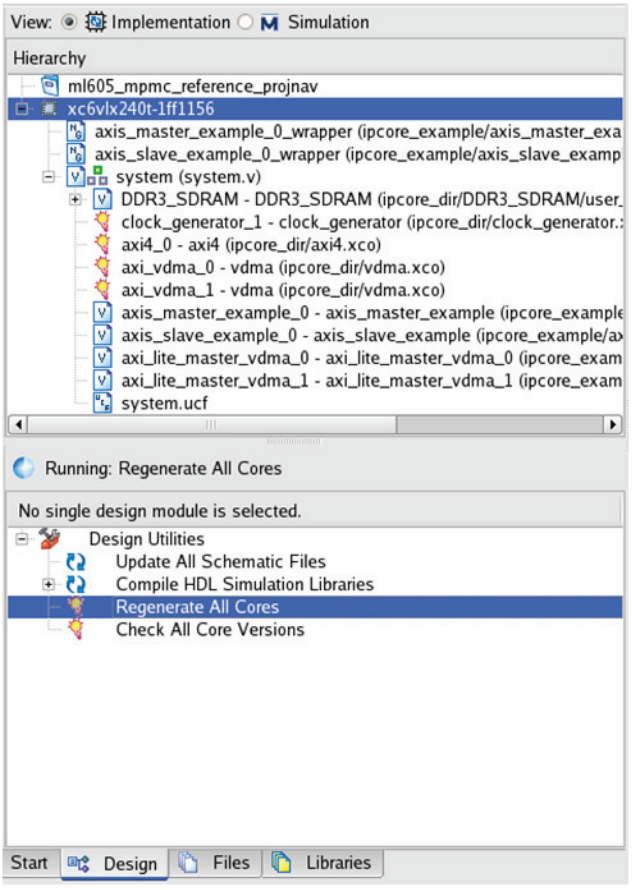



*Figure 4:* **Regenerating All Cores**

7. Build the design to a bitstream using **Process** > **Implement Top Module** (Figure 5). This step builds the design to a bitstream and might take some time to complete.

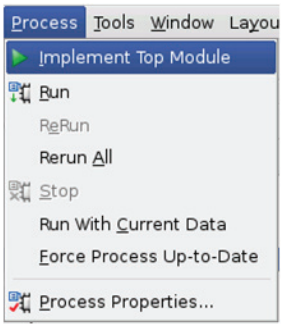



*Figure 5:* **Implementing a Design in Project Navigator**

If an error in synthesis occurs with an output message that says `unknown module <axi_vdma>`, the workaround for it is:

a. Close Project Navigator.

b. Edit `<design_dir>/projnav/ml605_mpmc_reference.xise`.

c. Find this section of text:

```
<file xil_pn:name="ipcore_dir/vdma.xise"
xil_pn:type="FILE_COREGENISE">
<association xil_pn:name="Implementation" xil_pn:seqID="237"/>
</file>
```

d. Move the above section of text to the top of the file just below the line `<files>`.

***Note:*** The `seqID` value might be different.

e. Save the text file.

f. Reopen the text file in Project Navigator.

g. Select **Project** > **Cleanup Project Files**.

h. Rebuild the project again. The `unknown module <axi_vdma>` error message should be resolved.

## ML605 Board Setup

8. The reference design runs on the ML605 board shown in Figure 6.

***Note:*** Not all ML605 features shown in Figure 6 are referenced or used in this document.

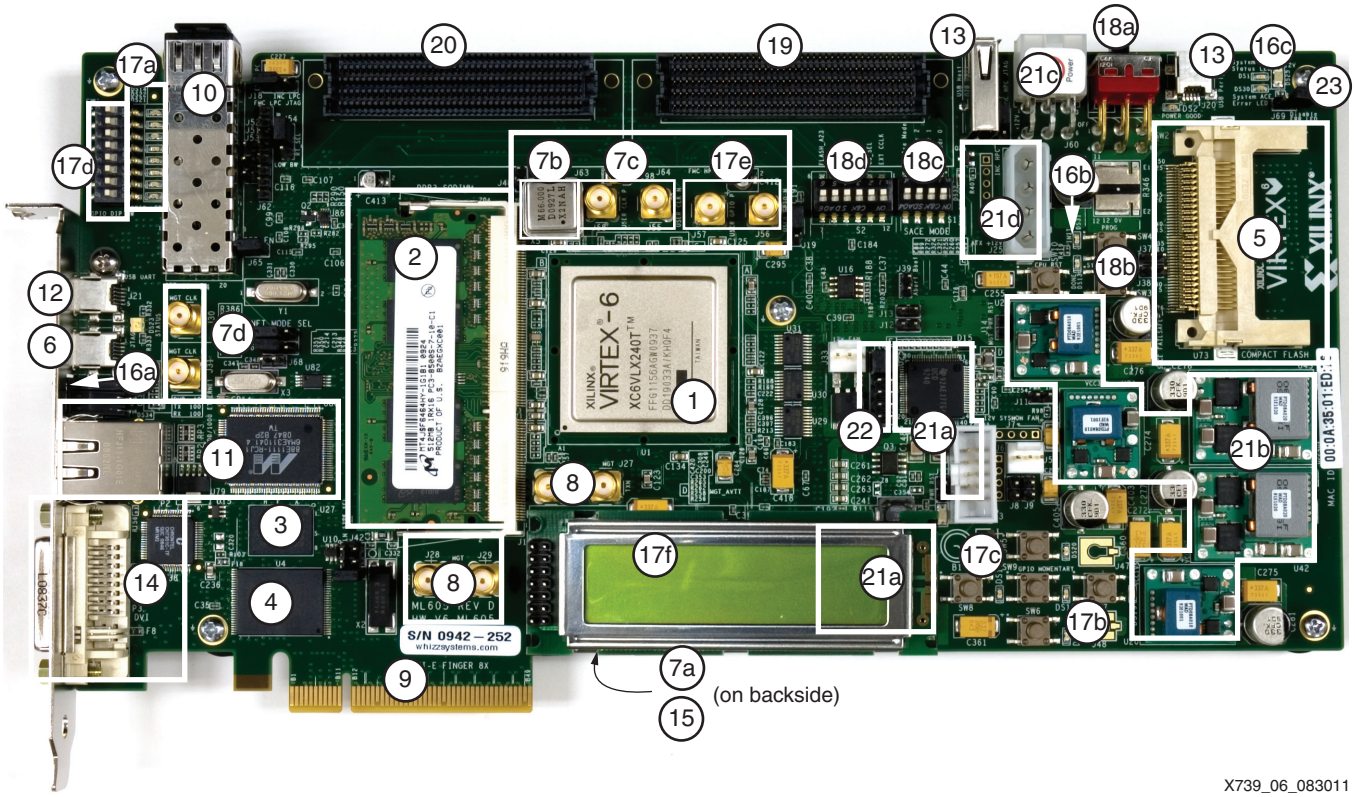



*Figure 6:* **ML605 Board Photo**

9. Connect the USB cable from the host PC to the USB JTAG port (6 in Figure 6) of the ML605 board to a USB cable (provided with the board) and connect it to the host PC. Ensure that the appropriate device drivers are installed.

10. Connect the ML605 DVI connector to a video monitor capable of displaying 1280x720p, 60 Hz video signal (14 in Figure 6).
11. Connect the power supply cable to the ML605 board (21c in Figure 6).
12. Turn on the power to the ML605 board (18a in Figure 6).

## Download and Run Bitstream

If you have not rebuilt the design and are programming the ML605 using the pre-generated bitstream included in the application note files, perform the following step.

***Note:*** Do not perform this step if the design has already been rebuilt.

13. In the Processes window pane, right-click **Generate Programming File** and select **Force Process Up-to-Date** (Figure 7). This puts a green check mark next to Generate Programming File and allows iMPACT to be run without building the design.

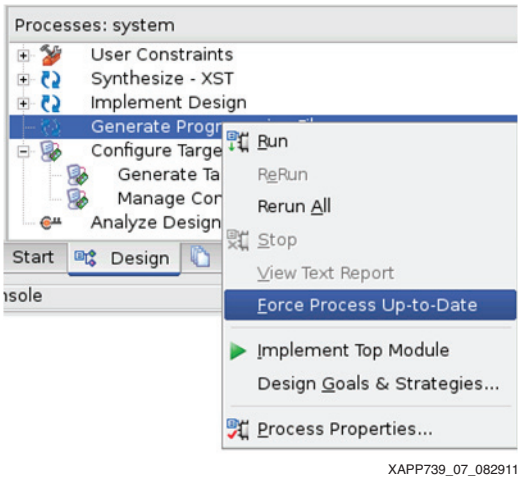



*Figure 7:* **Forcing Generate Programming File Step to be Up-to-Date**

14. In Project Navigator, go to the Processes window pane, right-click **Configure Target Device** and select **Run** to launch the bitstream download tool iMPACT (Figure 8).

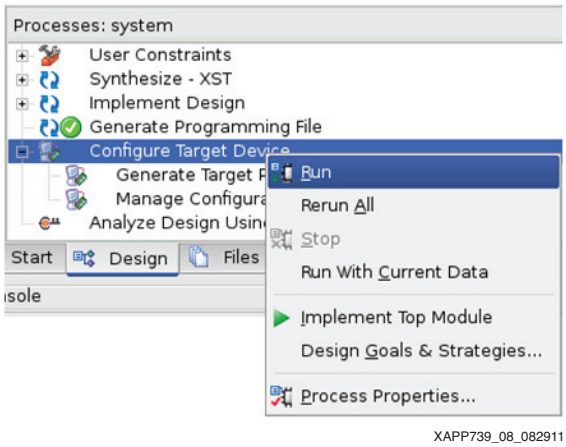



*Figure 8:* **Launching iMPACT from Project Navigator**

15. Click **OK** in the Warning window (Figure 9).

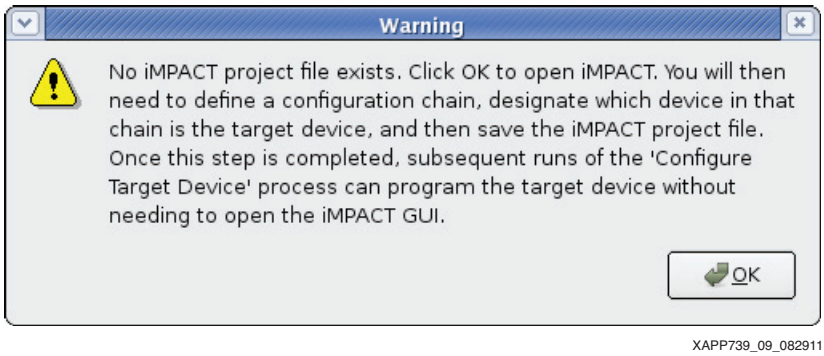



*Figure 9:* **iMPACT Warning Message**

16. The iMPACT tool is launched. Select **Edit** > **Launch Wizard** (Figure 10).

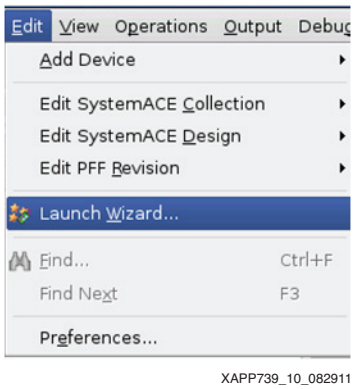



*Figure 10:* **Launching iMPACT Wizard**

17. Click **OK** to start the JTAG scan of the ML605 board (Figure 11).

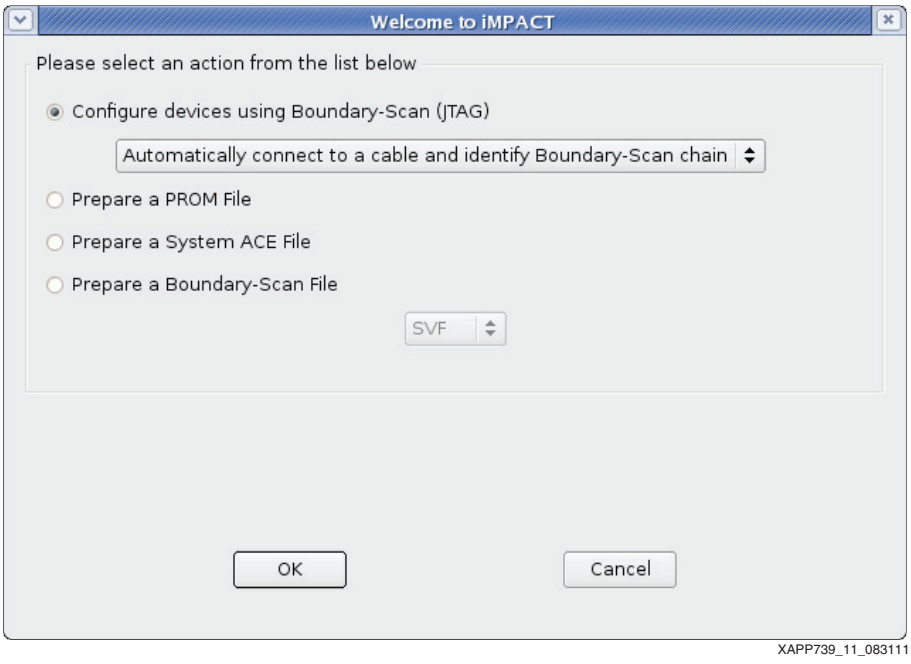



*Figure 11:* **iMPACT Wizard Welcome Menu**

18. Click **Yes** to auto assign configuration files (Figure 12).

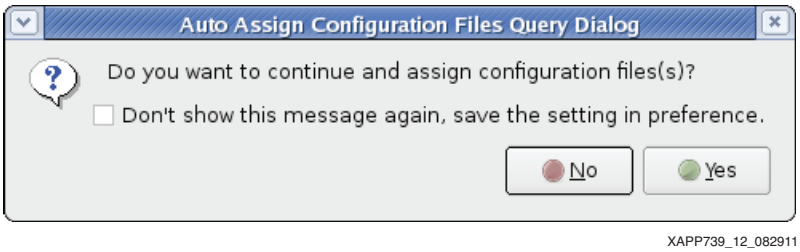



*Figure 12:* **Auto Assign Configuration Files Query Dialog**

19. For the first device in the JTAG chain, the SystemAce™ interface, click **Bypass** (Figure 13).

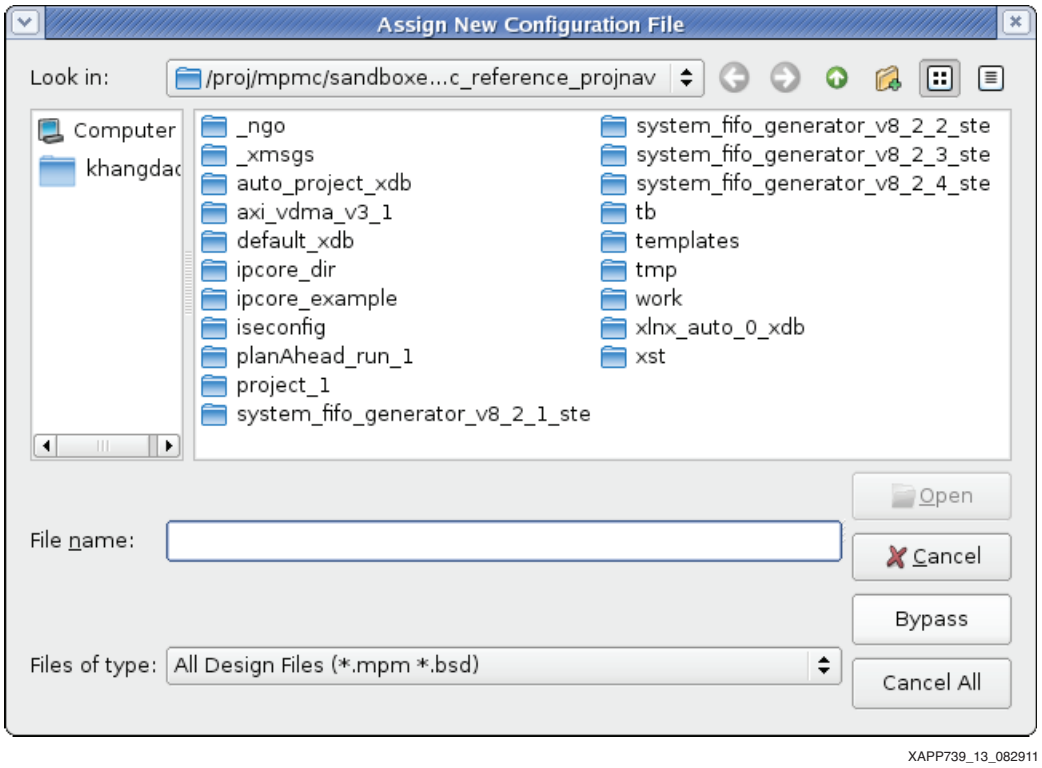



*Figure 13:* **Assign New Configuration File (SystemAce Interface)**

20. For the XC6VLX240T device, select **system.bit** and click **Open** to load `<design_dir>/projnav/system.bit` to configure the FPGA (Figure 14).

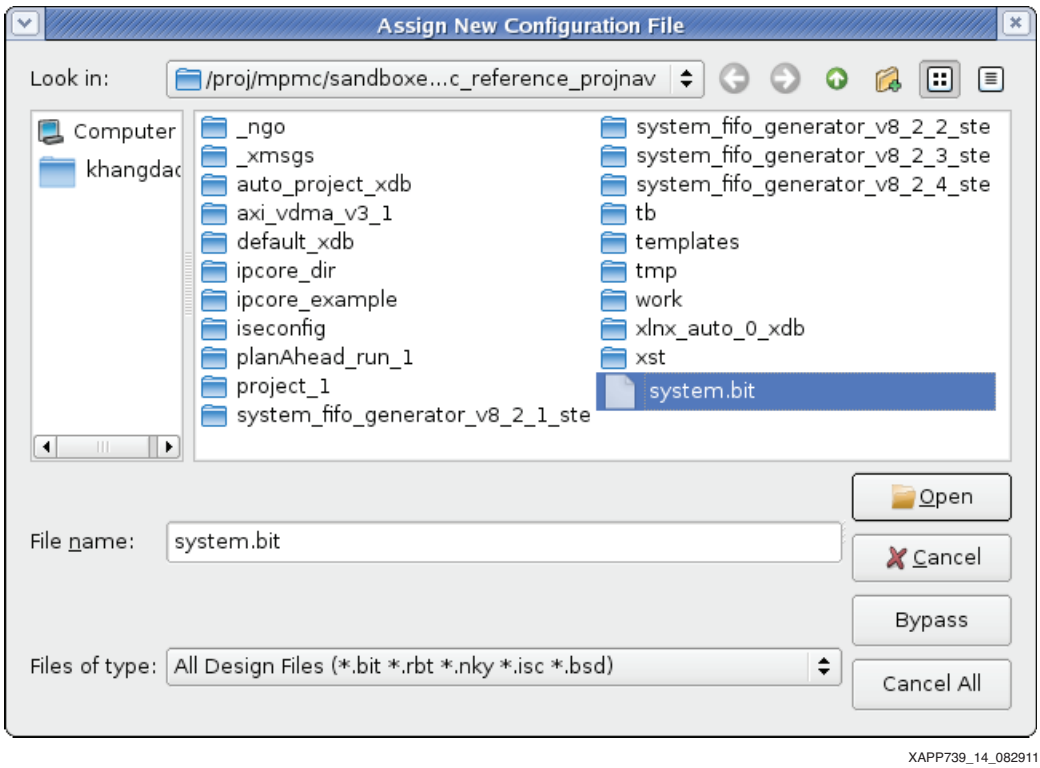



*Figure 14:* **Assign New Configuration File (XC6VLX240T Device)**

21. For SPI or BPI PROM options, click **No** (Figure 15).

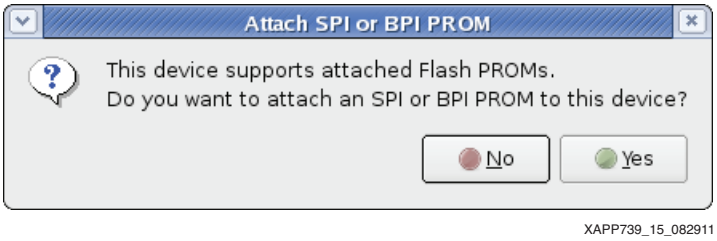



*Figure 15:* **Attach SPI or BPI PROM Query Dialog**

22. In Device Programming Properties, click **OK** (Figure 16).

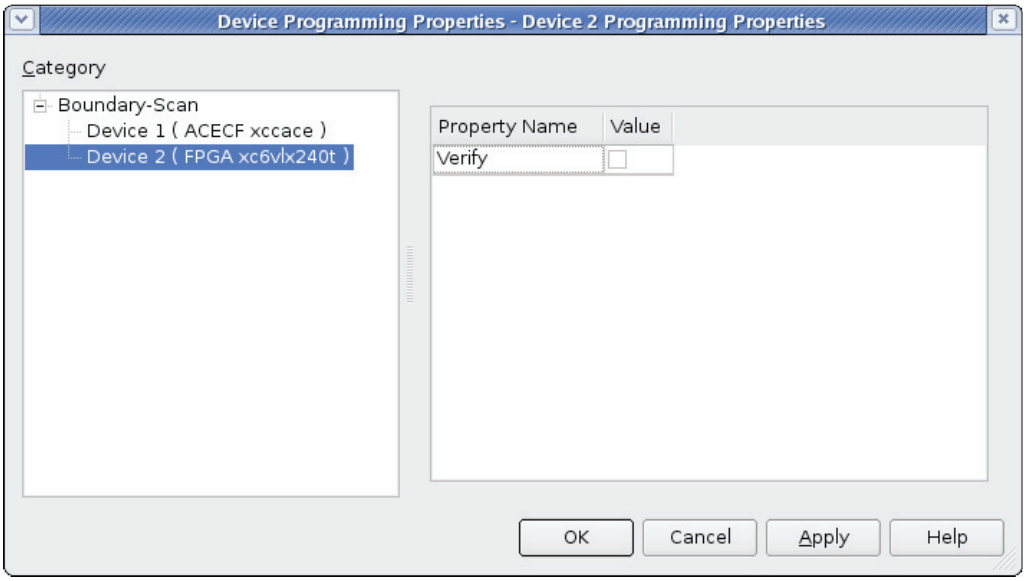



*Figure 16:* **Device Programming Properties Menu**

23. Program the FPGA. Right click the **xc6vlx240t** device and select **Program** (Figure 17).

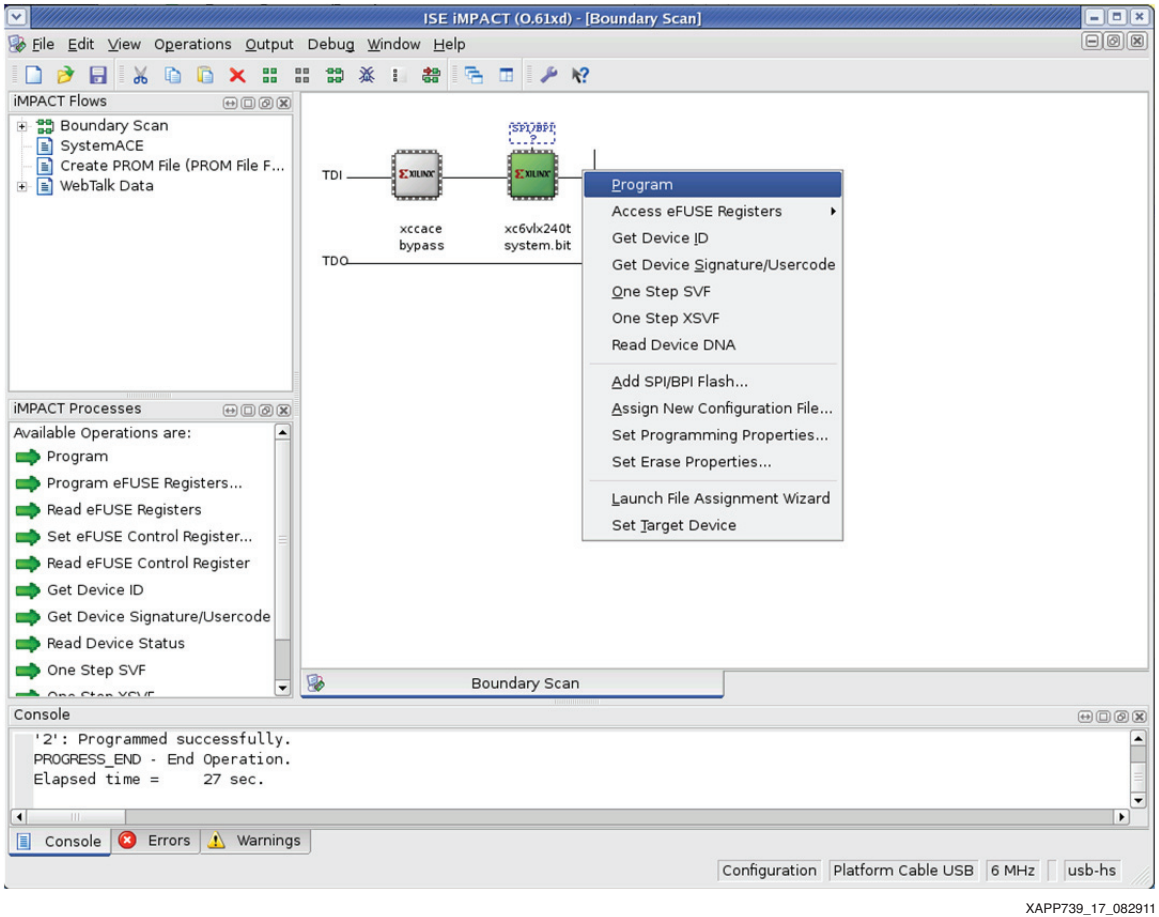



*Figure 17:* **Program FPGA Command in iMPACT**

24. A Program Succeeded message is displayed after programming completes. After the design is downloaded, the video monitor shows a number of white circular ripple patterns that slowly move outward. This demonstrates a live AXI MPMC system in hardware moving multiple frames of video data through DDR3 memory controlled by two AXI VDMA blocks.

# Creating the AXI MPMC Design from a New ProjNav Project

This section provides the steps to build the design starting from a new project. It describes how to use Project Navigator to add/configure IP blocks to the project, connect them together into a system, and implement the design to a bitstream.

## Start a New ProjNav Project and Set the Project Options

1. Install ISE Design Suite 13.2 (requires Logic Edition at a minimum).
2. Unzip the reference design files into a local folder (referred to as `<design_dir>`). This folder contains source files needed in the design creation process.
3. Open Project Navigator by selecting **Start** > **Xilinx ISE Design Suite 13.2** > **Project Navigator** in a PC or the "ise" command in Linux.
4. Create a new Project by selecting **File** > **New Project** (Figure 18).

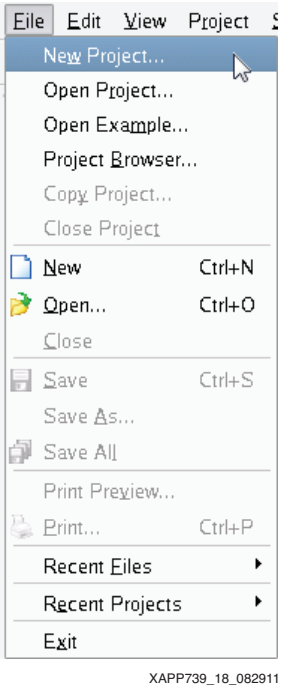



*Figure 18:* **Creating a New Project in Project Navigator**

5. Create a new project called **`ml605_mpmc_reference`** in a new directory (referred to as `<user_dir>`), then click **Next** (Figure 19).

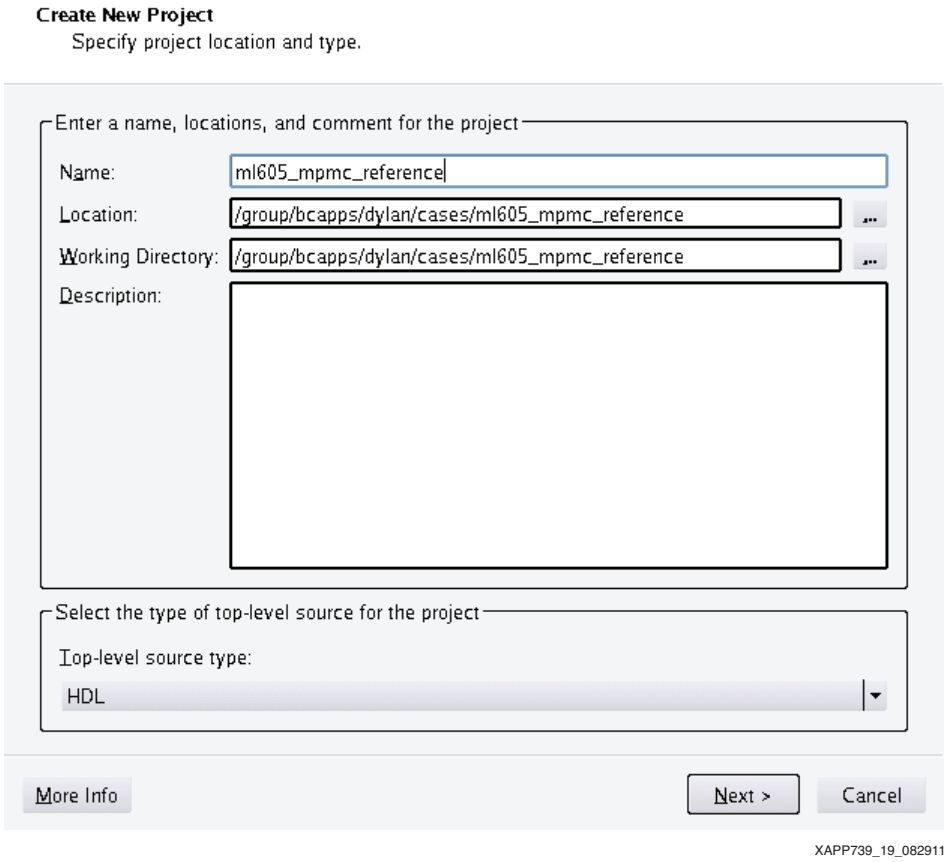



*Figure 19:* **Create New Project Menu**

6. Enter project settings as shown in Figure 20 and click **Next**.

   ***Note:*** Selecting the ML605 board causes FPGA device information to be entered automatically. The instructions in this application notes use Verilog as the preferred language.

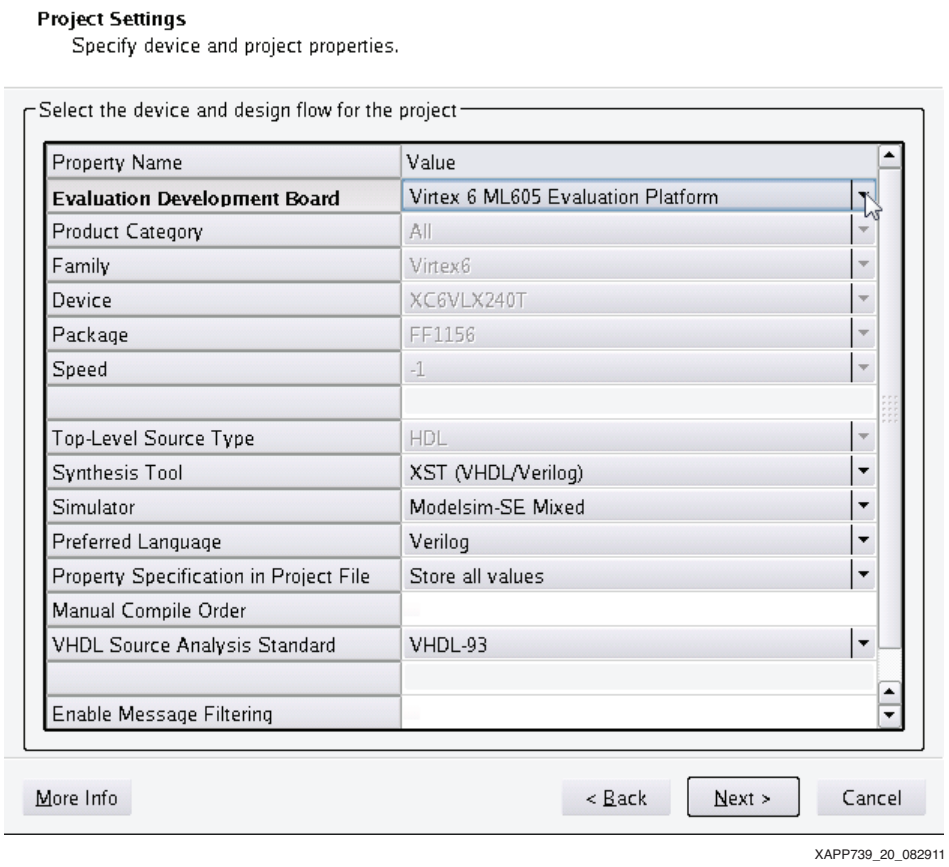



*Figure 20:* **Project Settings Menu**

7. Review the project settings and click **Finish** (Figure 21). This step returns to Project Navigator with a new project opened.

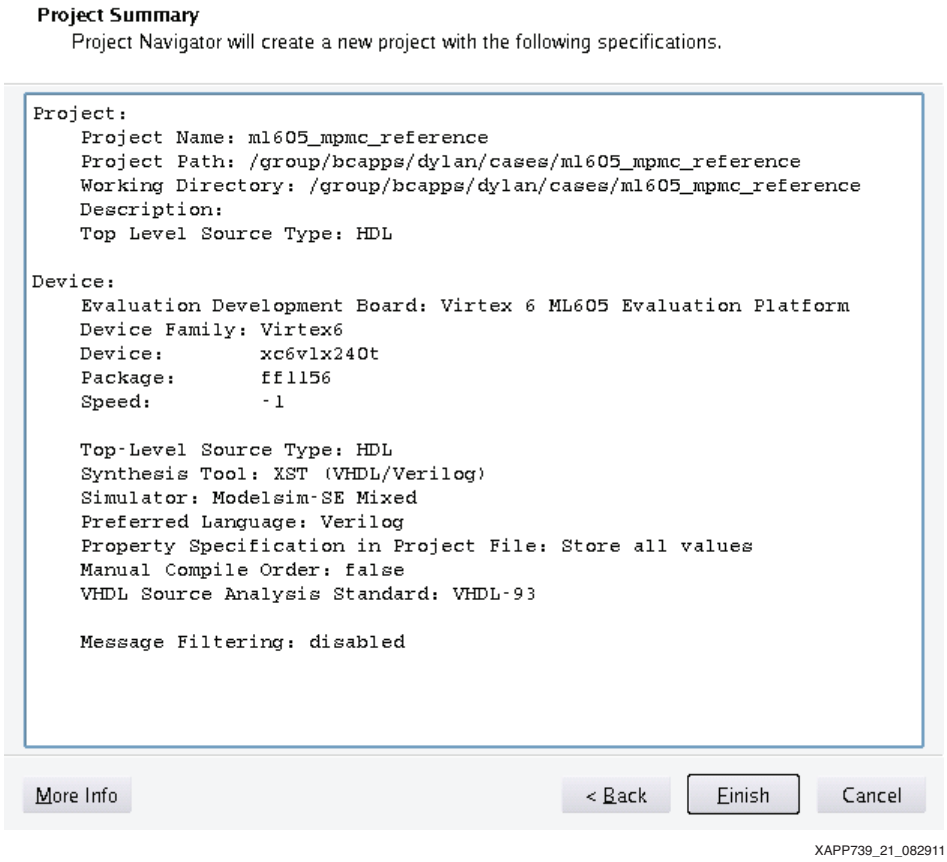



*Figure 21:* **Project Summary Menu**

8. Copy the directory `<design_dir>/projnav/ipcore_example` to `<user_dir>` so that `<user_dir>/ipcore_example` is created. This directory contains additional source files for later use.

## Adding the Memory Controller (MIG IP Core) to the Design

This section describes the steps to add the AXI-based memory controller (MIG IP core) to the project, configured for the ML605 board. Detailed information about the MIG IP core for Virtex-6 FPGAs is described in *Virtex-6 FPGA Memory Interface Solutions User Guide* [Ref 1].

### Generating MIG IP Core

9. Click **Project** > **New Source** to add a new IP core to the system (Figure 22).

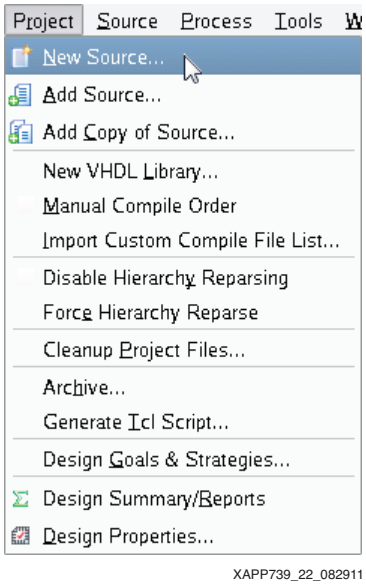



*Figure 22:* **Creating a New Source in Project Navigator**

10. For the source type, select **IP** and enter the file name **DDR3_SDRAM**. The location should be at the default `<user_dir>/ipcore_dir`. Click **Next**.

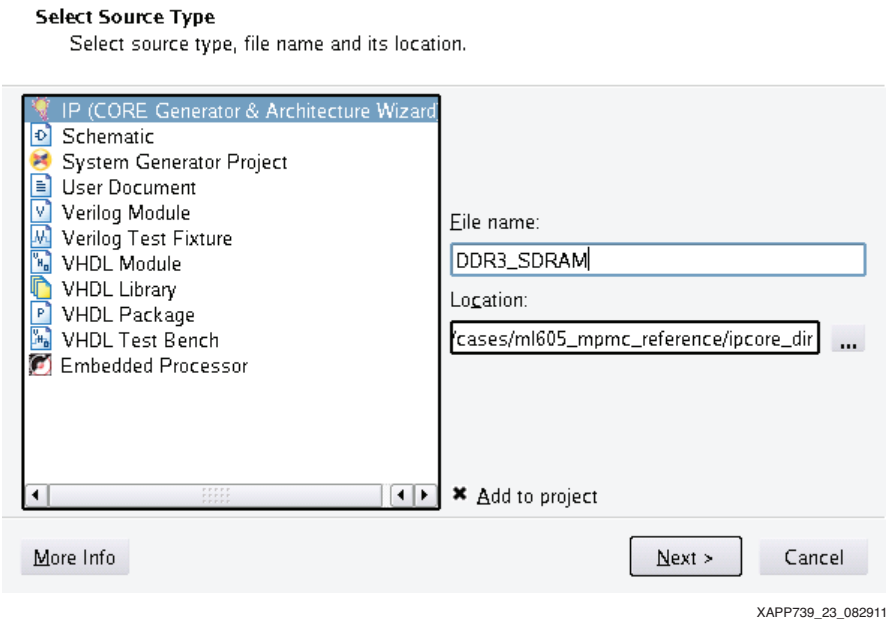



*Figure 23:* **New Source Wizard - Source Type Menu**

11. In the Select IP window, click the **View by Function** tab, then click **Memories & Storage Elements** > **Memory Interface Generators** > **MIG Virtex6 and Spartan6** (Version 3.8) from the IP catalog. Click **Next** (Figure 24).

***Note:*** The version number might not appear in the GUI.

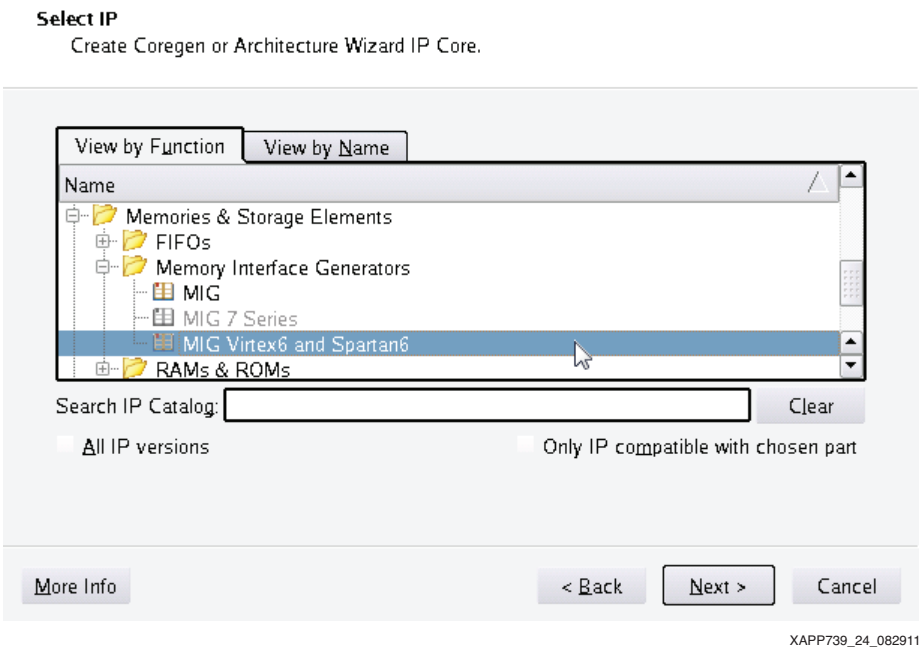



*Figure 24:* **New Source Wizard - Select IP Menu**

12. In the Summary window, review the settings and click **Finish**. This creates a CORE Generator tool project for the MIG IP and opens the MIG IP configuration GUI (Figure 25).

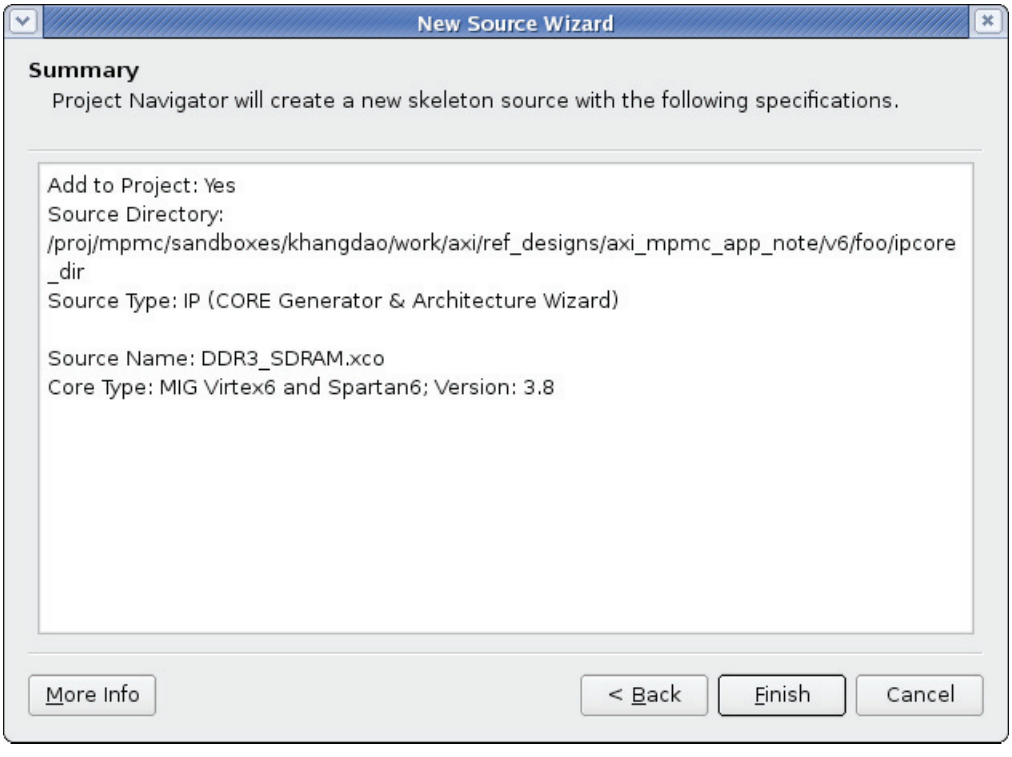



*Figure 25:* **New Source Wizard - Summary Menu**

13. In the MIG GUI, review the core options and click **Next** (Figure 26).

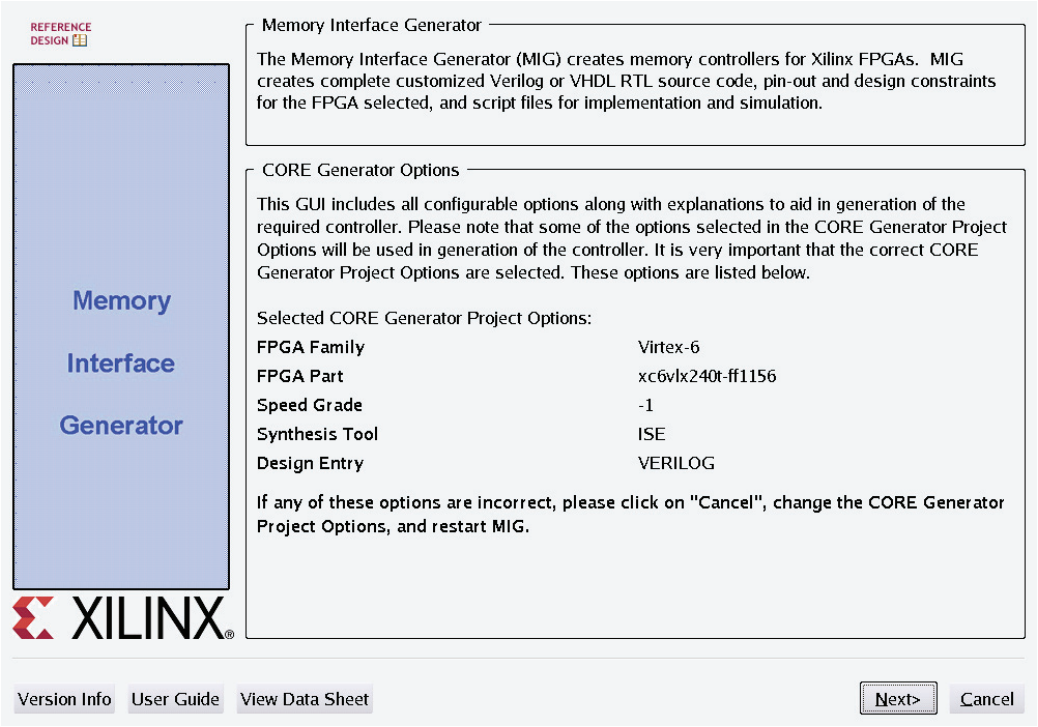
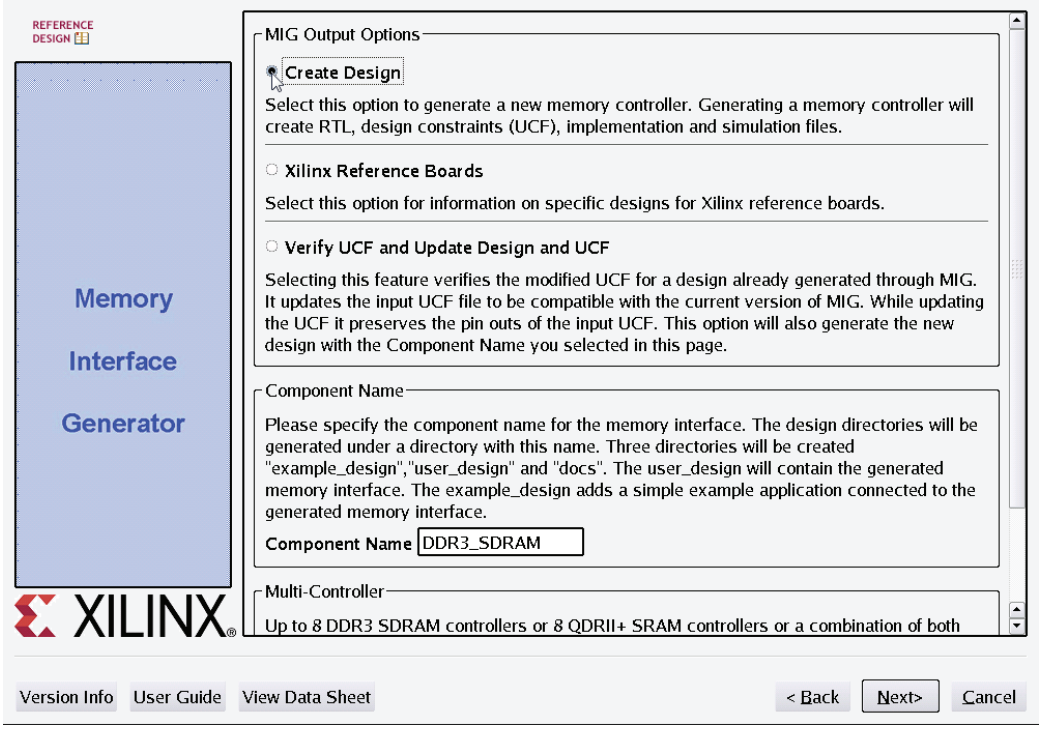



*Figure 26:* **Virtex-6 FPGA Memory Interface Generator Front Page**

14. Select **Create Design** and click **Next** (Figure 27). For Number of Controllers, use the default value of **1**.

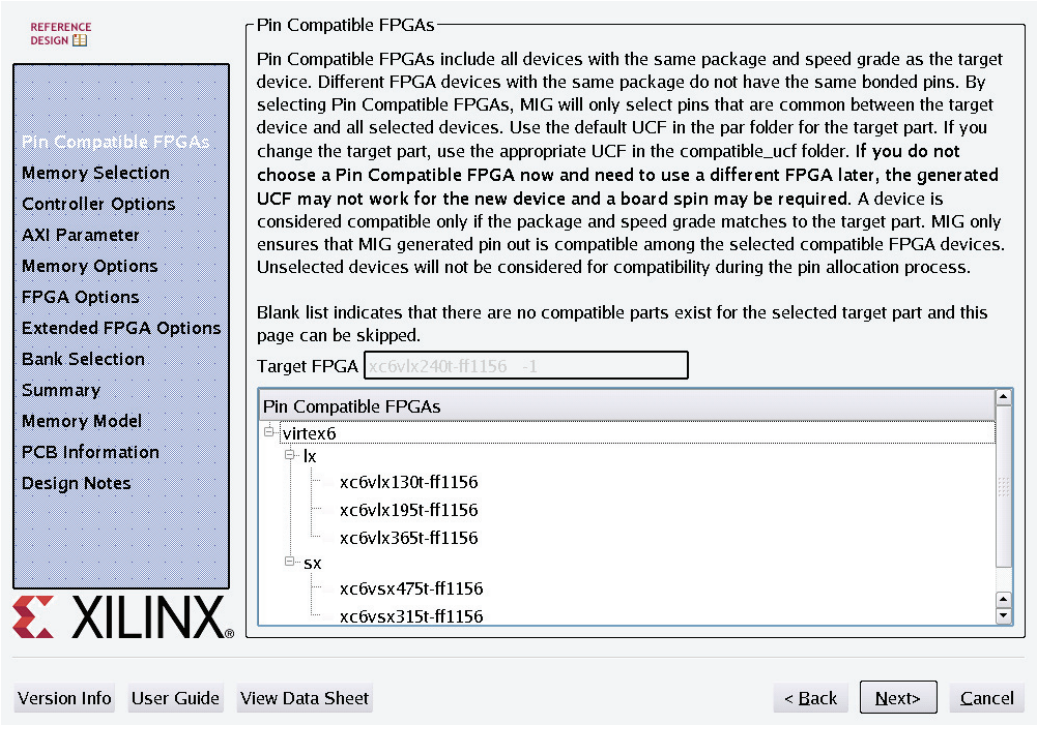
***Note:*** The Xilinx Reference Boards option is not supported for the ML605 board. The component name is also preloaded with the IP file name previously set by the user.




XAPP739_27_082911

*Figure 27:* **MIG Output Options**

15. For the Pin Compatible FPGAs settings, click **Next**. This page is normally used for a new board pinout definition that is desired to be compatible across multiple devices in the same package (Figure 28).




XAPP739_28_082911

*Figure 28:* **MIG Pin-Compatible Virtex-6 FPGAs**

16. For the Memory Selection settings, select **DDR3 SDRAM** and **Enable AXI Interface**, then click **Next**. This creates a memory controller for the DDR3 memory on the ML605 board that has an AXI interface to connect to the AXI Interconnect IP (Figure 29).

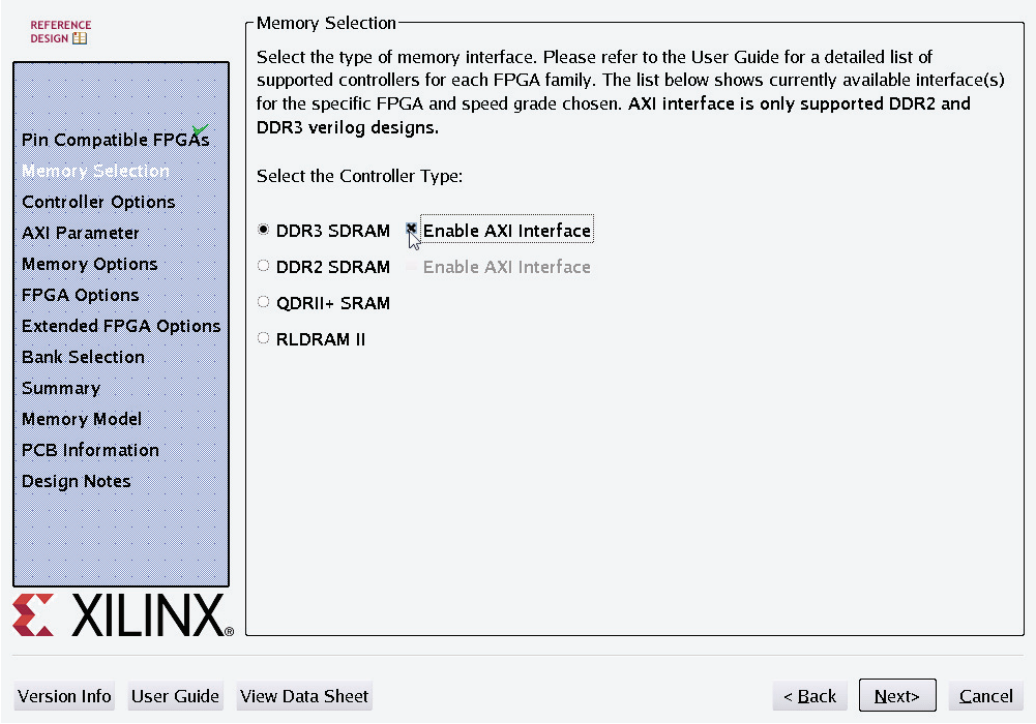



*Figure 29:* **MIG Memory Type and Controller Selection**

17. For the Controller Options settings, set the clock frequency to **2500 ps** to set the design for a 400 MHz memory clock, which is the maximum clock frequency supported for the device and speed grade of FPGA on the ML605 board. Set the memory information to match the ML605 board as follows:
    - Memory Type: **SODIMM**
    - Memory Part: **MT4JSF6464HY-1G1**
    - Data Width: **64**
    - ECC: **Disabled**
    - Data Mask: **Checked**
    - Ordering: **Normal**

    Click **Next** (Figure 30).

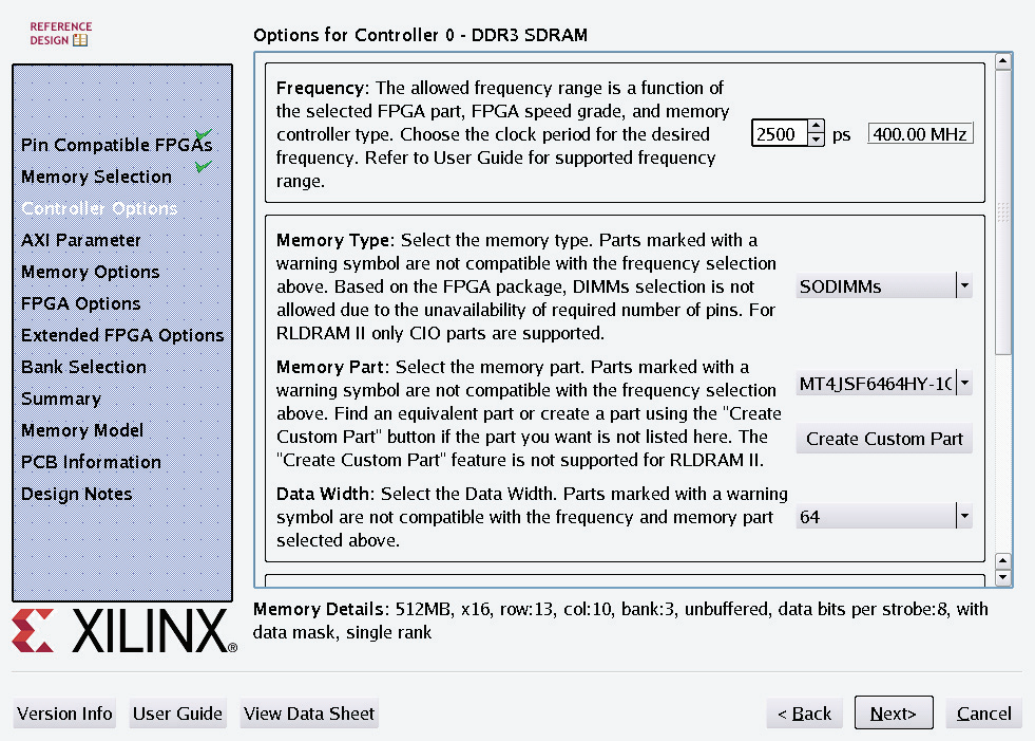



*Figure 30:* **MIG Controller Options Page**

18. For AXI Parameter settings, select the following parameters (Figure 31):
    - AXI Data Width: **256** bits.

      This sets the AXI Interface data width to match the native data width of the MIG data paths (4 × 64-bit memory data width = 256-bit AXI data width).
    - AXI Supports Narrow Burst: **0**.

      This disables support for AXI narrow burst transactions to reduce area and latency. Narrow burst transactions generally occur when a master requests a burst data width narrower than its full data width. Single clock cycle transactions are not affected by this optimization. Narrow burst transactions are generally unused by Xilinx IP, as described in the *AXI Reference Guide* [Ref 2].
    - AXI ID Width: **4**.

      This value is set depending on the network of masters connected to this memory controller. The value is generally a system-level parameter described in the ARM AXI4 documentation [Ref 7] and is determined by the AXI Interconnect or IP block connected to the memory controller. In this design, a setting of **4** matches the default configuration used by AXI Interconnect, with four masters that do not issue their own ID signals. However, for a custom user system, this value might need to be adjusted to meet the requirements of the AXI system it is connected to, especially if the number of devices or AXI Interconnect configuration is changed significantly.

    Click **Next.**

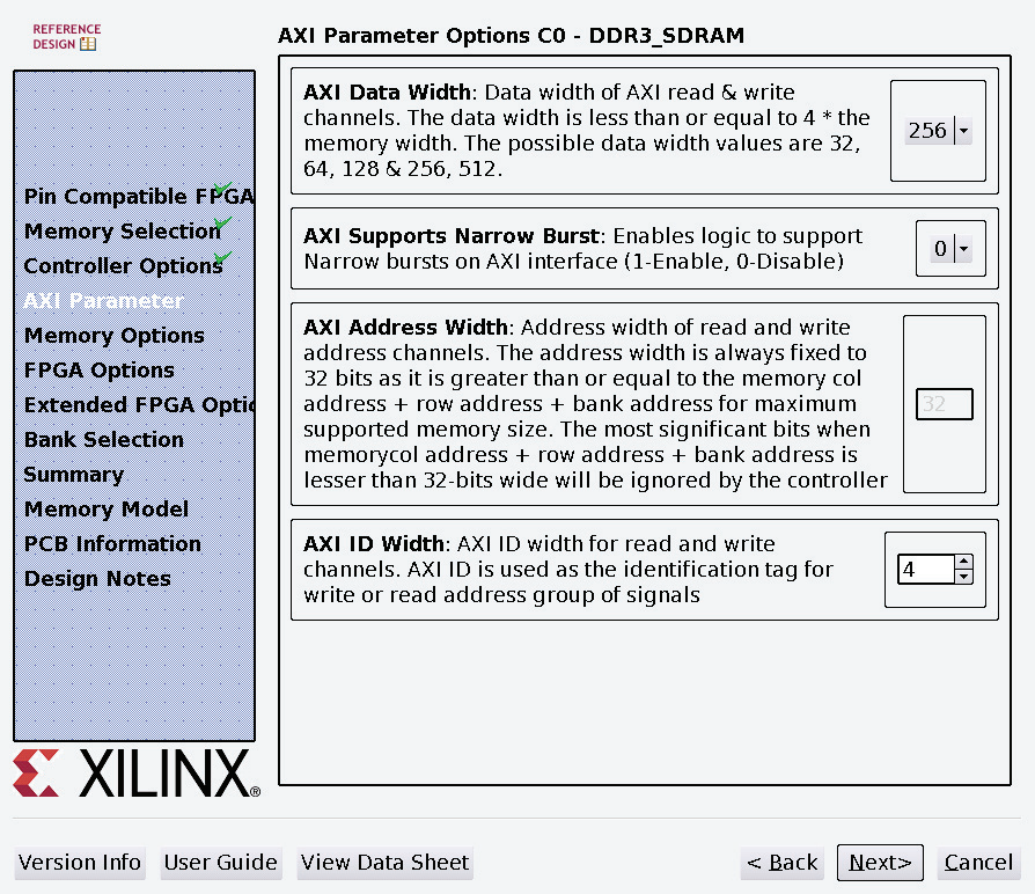



*Figure 31:* **MIG AXI Parameter Options**

19. For the Memory Options settings, select the following (Figure 32):
    - Burst Length: **8-Fixed**.

      This is a recommended general setting given that the masters in the system are video devices generally using longer bursts.
    - Read Burst Type: **Sequential**.

      This is a recommended general setting given that the masters in the system are video devices generally using sequential bursts.
    - Output Driver Impedance Control: **RZQ/7**

      This matches the design requirements of the ML605 board.
    - RTT/ODT: **RZQ/4**

      This matches the design requirements of the ML605 board.
    - Memory Address Mapping Selection: (Either setting can be used.)

      Because the video devices in the system have long sequential access patterns, similar results are likely to be observed. **Bank/Row/Column** mapping is chosen because, in theory, it might allow multiple masters to access different banks, potentially keeping them open longer.

    Click **Next**.

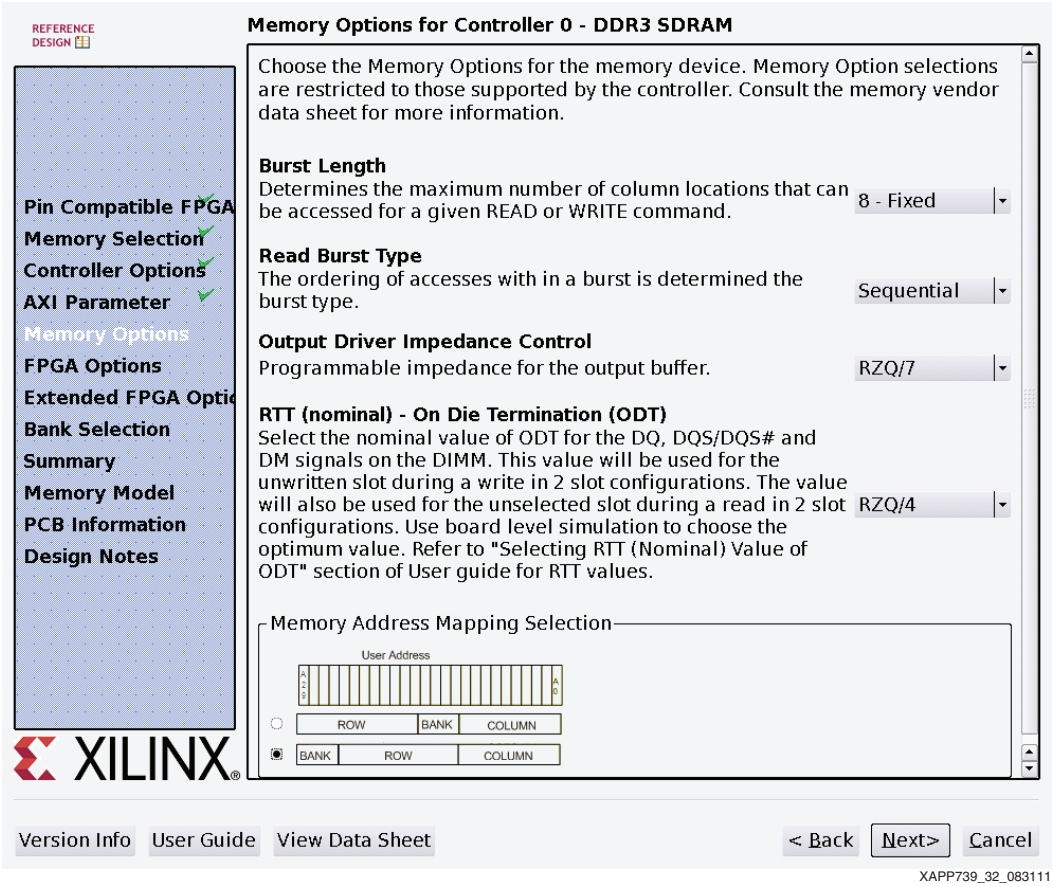
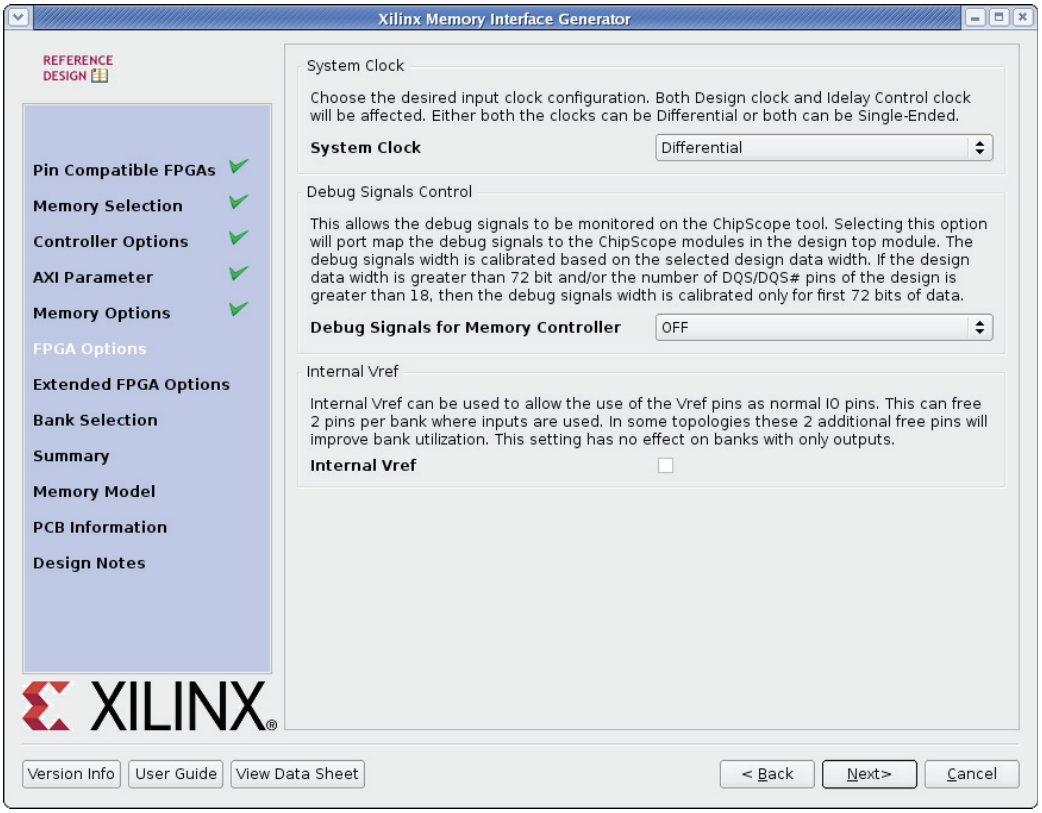



*Figure 32:* **MIG Memory Mode Options**

20. For the FPGA Options settings, use the default options:
    - System Clock: **Differential**
    - Debug Signals for Memory Controller: **OFF**
    - Internal Vref: (Box unchecked).

    This matches the normal configuration of the ML605 board. Click **Next** (Figure 33).




XAPP739_33_082911

*Figure 33:* **MIG FPGA Options**

21. For the Pin/Bank Selection Mode settings, select **Fixed Pin Out**, then click **Next**. This allows the user to enter predefined pinout information for an existing board like the ML605 board (Figure 34).

    ***Note:*** The other option is for a new board design in which the user wants the tool to choose a pinout.

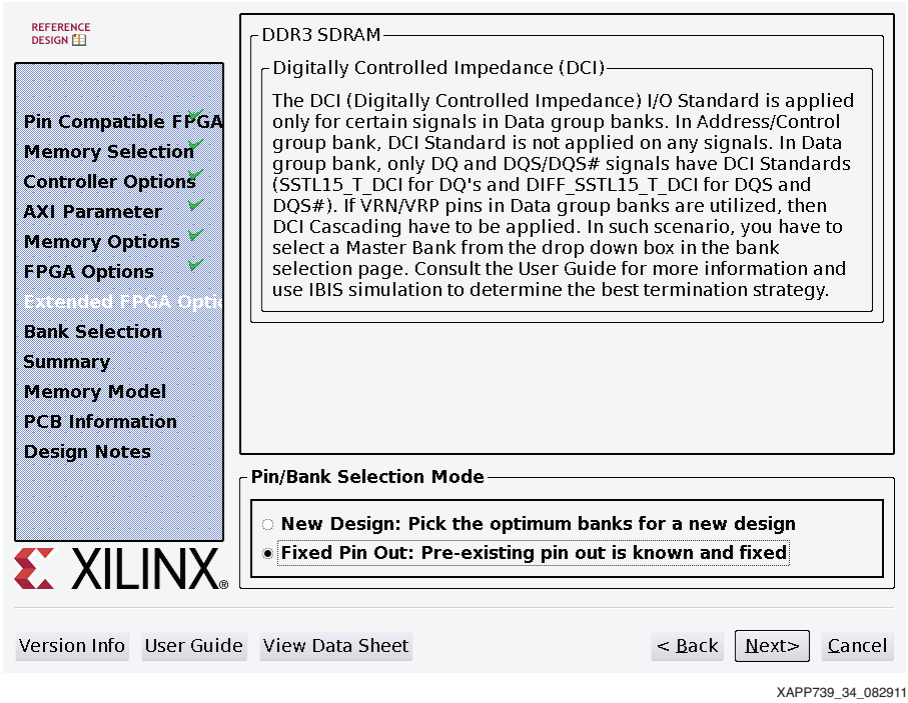



*Figure 34:* **MIG Extended FPGA Options**

22. For the Pin Selection settings, the user would normally use the pull-down menus in the GUI to enter the pinout information for I/Os required by the memory interface. For this design, click **Read UCF** and enter the name of a file pre-generated for the ML605 board to save time from entering each pin location, resulting in Figure 35.

    ***Note:*** To save time, the MIG GUI has an option to read a UCF file to import the pinout information. The format of this file is described in the MIG documentation.

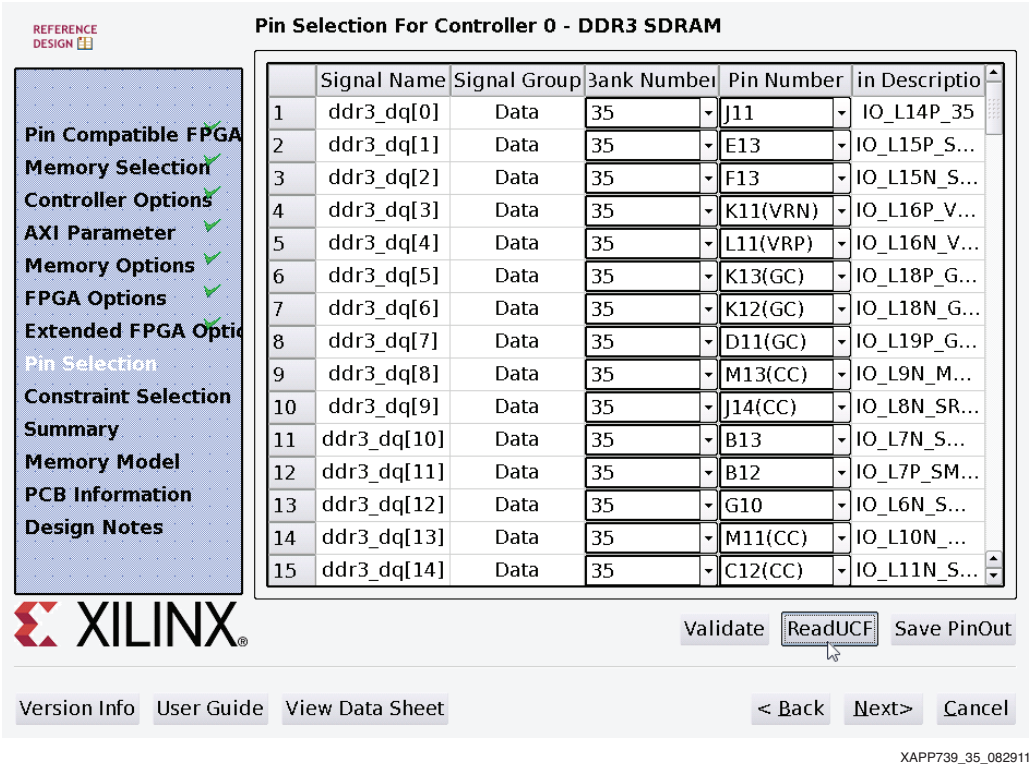



*Figure 35:* **MIG Pin Selection Using an Existing Pinout**

23. Browse and select the file in `<user_dir>/ipcore_example/mig.ucf`, then click **Open** to load the ML605 pinout information file. This returns the user to the Pin Selection menu. Click **Next** (Figure 36).

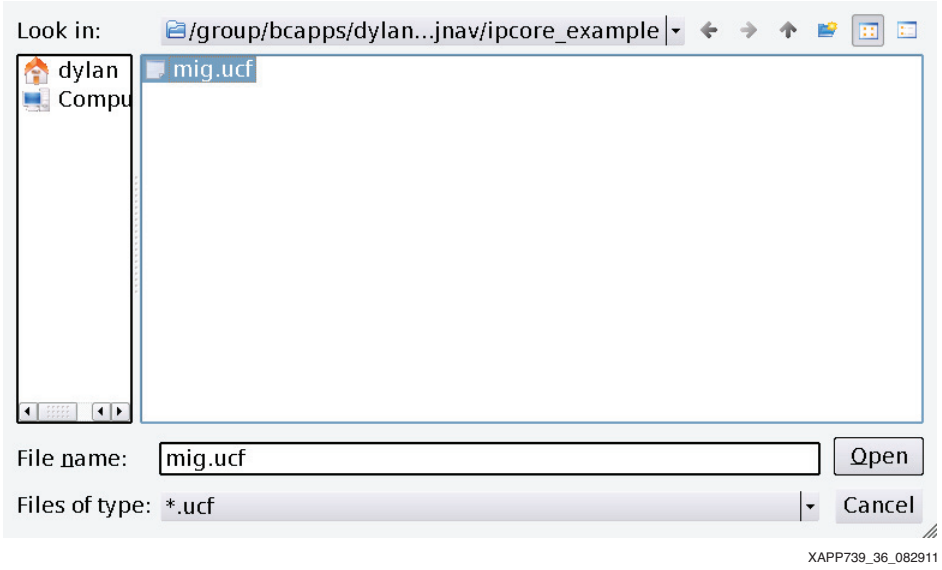



*Figure 36:* **MIG Pin Selection Read UCF File Browser**

24. In the Constraint Selection window, the BUFIO and Master Bank locations must be set for the ML605 board. The *Virtex-6 FPGA Memory Interface Solutions User Guide* [Ref 1] describes how these values are determined for a custom board based on the desired pinout. For the ML605 board, the Master Bank Inner Left IOs setting should be **26** and the

Master Bank Inner Right IOs setting should be **36**. To save time, click **Read UCF** to load the pre-generated UCF file again for the ML605 board.

***Note:*** It is also possible to skip the Read UCF step and simply enter the BUFIO information, as shown complete in Figure 37.

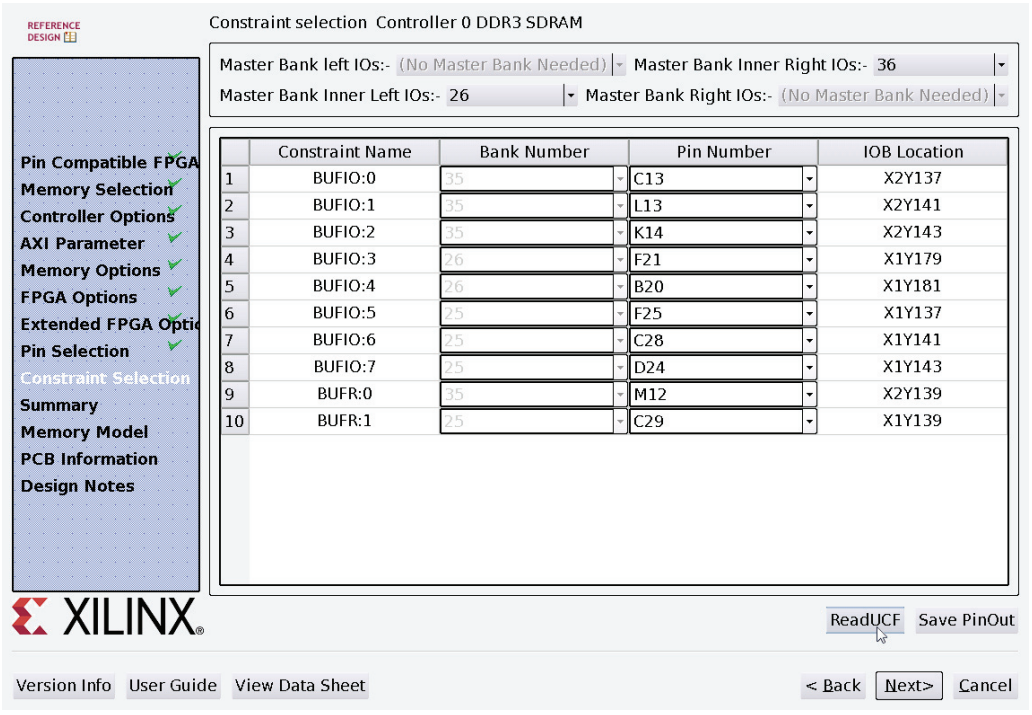



*Figure 37:* **MIG Constraint Selection Menu**

25. To load BUFIO information instead of entering the data manually, click **Read UCF** and select the file again in `<user_dir>/ipcore_example/mig.ucf`, then click **Open** to load the ML605 pinout information file. This returns to the Constraint Selection window. Click **Next** (Figure 38).

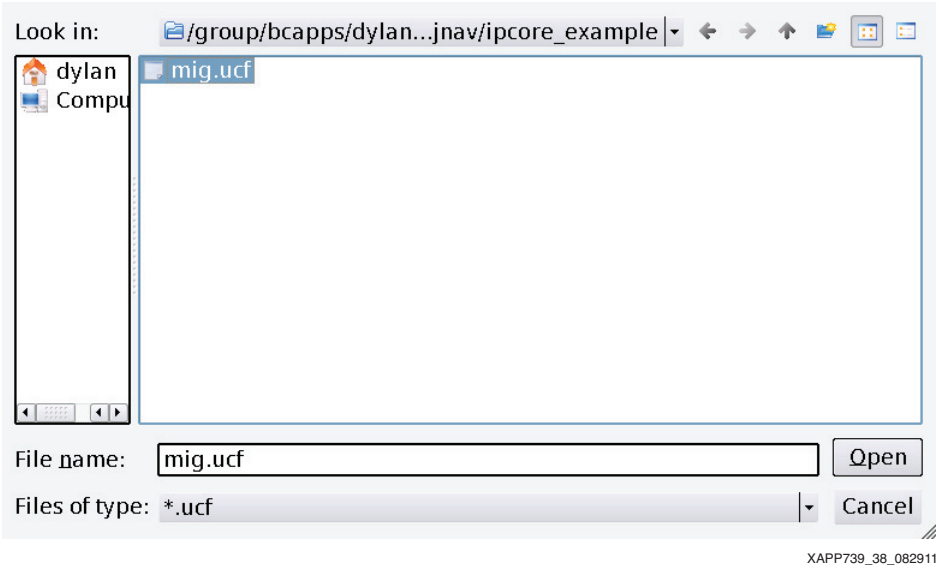



*Figure 38:* **MIG Constraint Selection Read UCF File Browser**

26. Review the information in the Summary page and click **Next** (Figure 39).

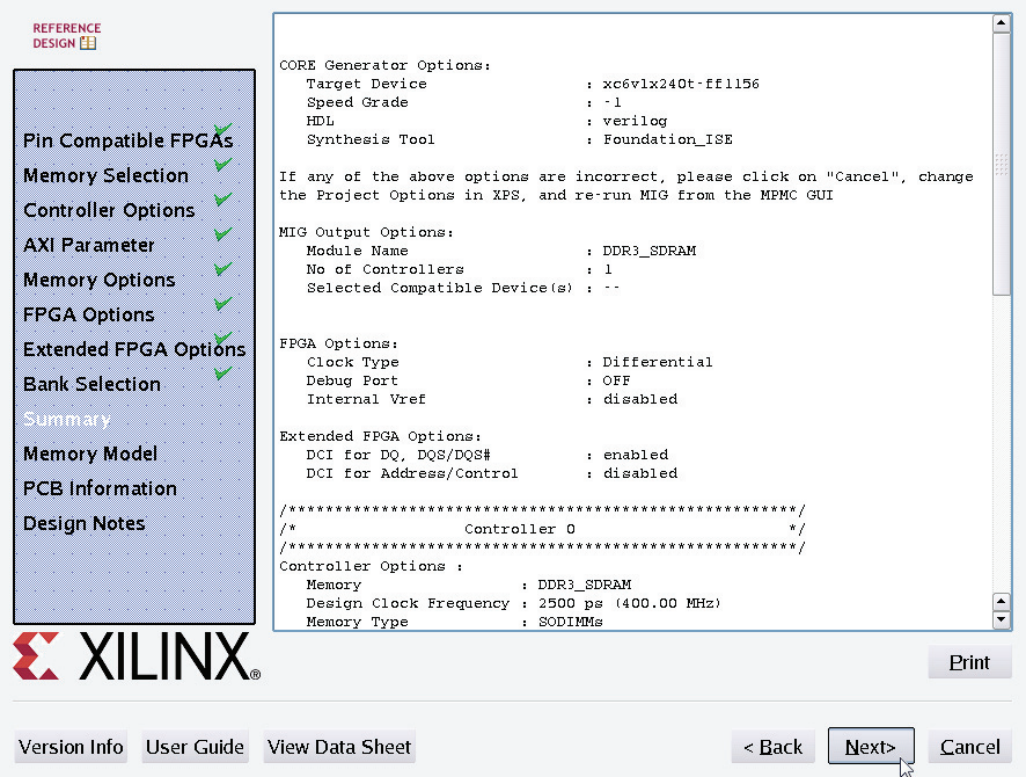



*Figure 39:* **MIG Summary**

27. Review the license agreement, then click **Accept** and **Next** to have the memory simulation model delivered from the CORE Generator tool to be used to create a simulation testbench (Figure 40).

    ***Note:*** A memory simulation model suitable for system simulation is generated if the license is accepted. If the license agreement is declined, a separate model must be obtained to perform any simulation testbench creation steps.

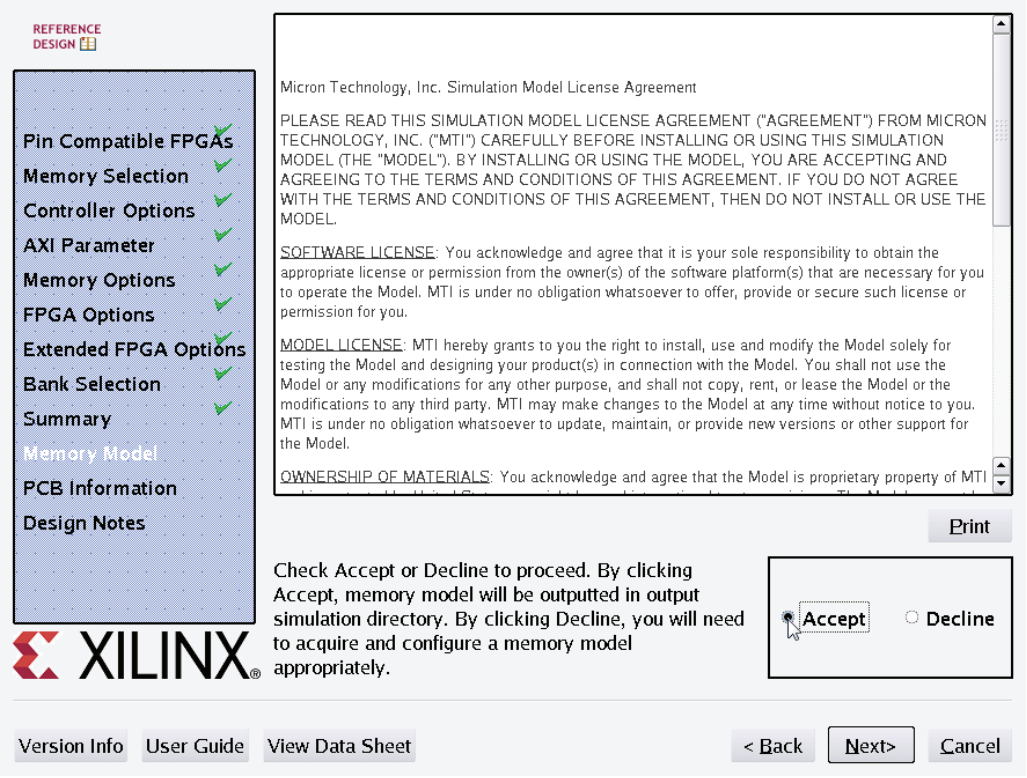



*Figure 40:* **MIG Micron Model License Agreement**

28. Review the PCB design information and click **Next** (Figure 41).

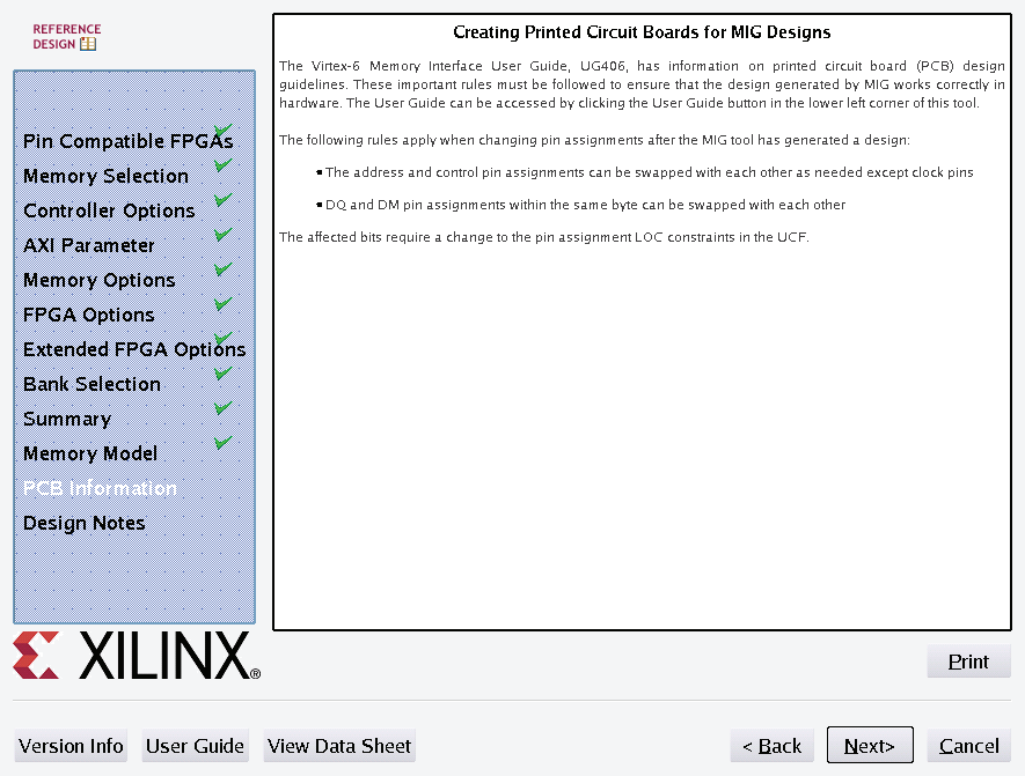



*Figure 41:* **MIG PCB Information**

29. Review the Design Notes and Click **Generate**. This generates the MIG IP files and returns the user to ProjNav (Figure 42).

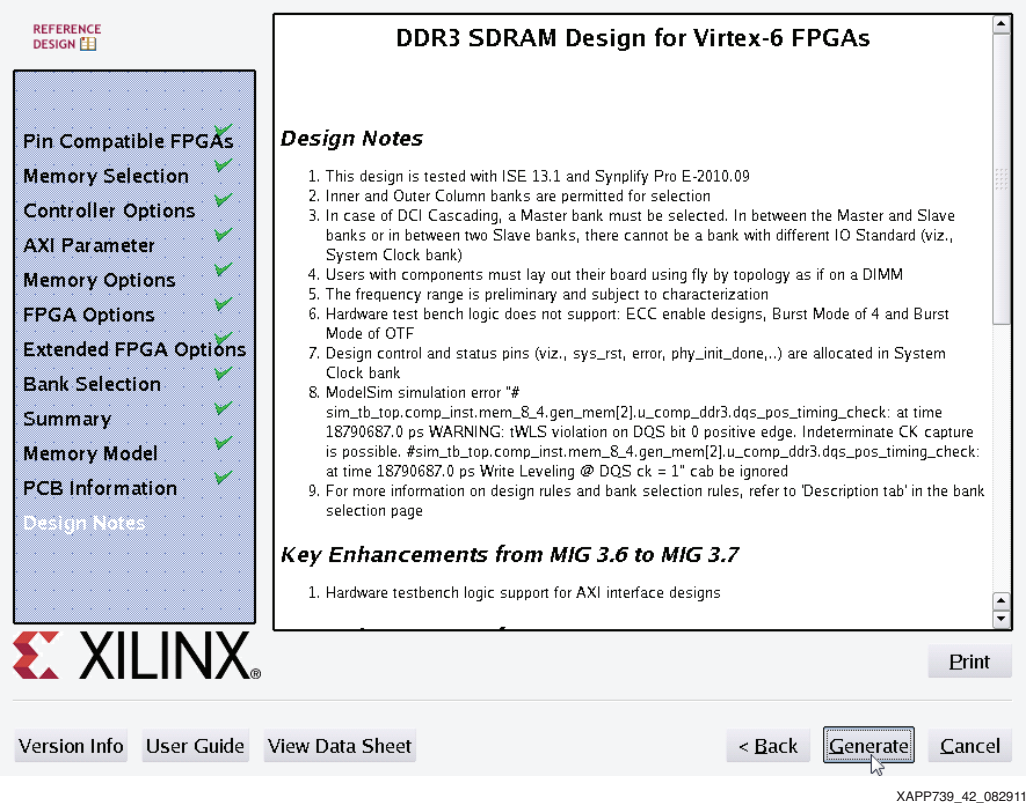



*Figure 42:* **MIG Design Notes**

## Incorporating MIG and the Top-Level into the Design

30. In the Hierarchy window pane of Project Navigator, select **DDR3_SDRAM**. Then, in the lower Processes window pane, expand the CORE Generator tool section and double-click **View HDL Instantiation Template**. This opens the `DDR_SDRAM.veo` file. Save this file as `<user_dir>/system.v` to use as the basis for creating the top-level HDL file for the AXI MPMC design (Figure 43).

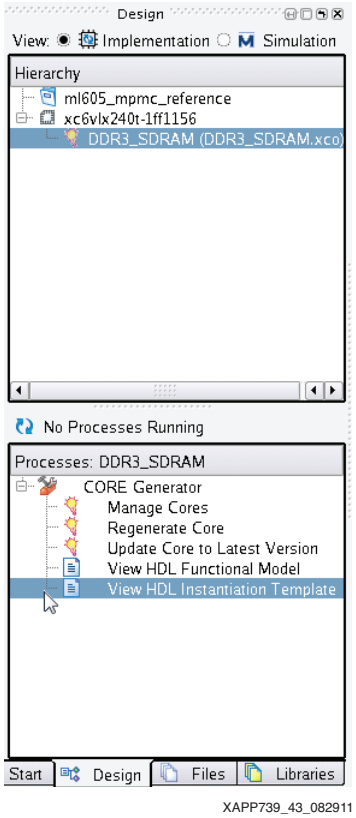



*Figure 43:* **Viewing a CORE Generator IP Instantiation Template From Project Navigator**

31. Edit `<user_dir>/system.v` to make the I/O connection from the MIG IP to the top-level ports. Define the top-level I/O ports and connect them to the MIG IP as shown in Table 1.

    ***Note:*** To save time, the completed `<design_dir>/projnav/system.v` can be used instead.

*Table 1:* **Connections in `system.v` from the MIG DDR3_SDRAM Instance to the Top-Level Ports**

| DDR3_SDRAM Instance Port Name | Top-Level Port Name in `system.v` |
|---|---|
| clk_ref_p | clk_ref_p (output) |
| clk_ref_n | clk_ref_n (output) |
| ddr3_dq | ddr3_dq (inout) |
| ddr3_addr | ddr3_addr (output) |
| ddr3_ba | ddr3_ba (output) |
| ddr3_ras_n | ddr3_ras_n (output) |
| ddr3_cas_n | ddr3_cas_n (output) |
| ddr3_we_n | ddr3_we_n (output) |
| ddr3_reset_n | ddr3_reset_n (output) |
| ddr3_cs_n | ddr3_cs_n (output) |
| ddr3_odt | ddr3_odt (output) |
| ddr3_cke | ddr3_cke (output) |
| ddr3_dm | ddr3_dm (output) |
| ddr3_dqs_p | ddr3_dqs_p (inout) |

*Table 1:* **Connections in `system.v` from the MIG DDR3_SDRAM Instance to the Top-Level Ports *(Cont'd)***

| DDR3_SDRAM Instance Port Name | Top-Level Port Name in `system.v` |
|---|---|
| ddr3_dqs_n | ddr3_dqs_n (inout) |
| ddr3_ck_p | ddr3_ck_p (input) |
| ddr3_ck_n | ddr3_ck_n (input) |
| sys_rst | sys_rst (input) |

32. In Project Navigator, select **Project** > **Add Source**, choose `<user_dir>/system.v`, and click **Open** to add it to the project (Figure 44).

    ***Note:*** To save time, the completed `<design_dir>/projnav/system.v` can be used instead.

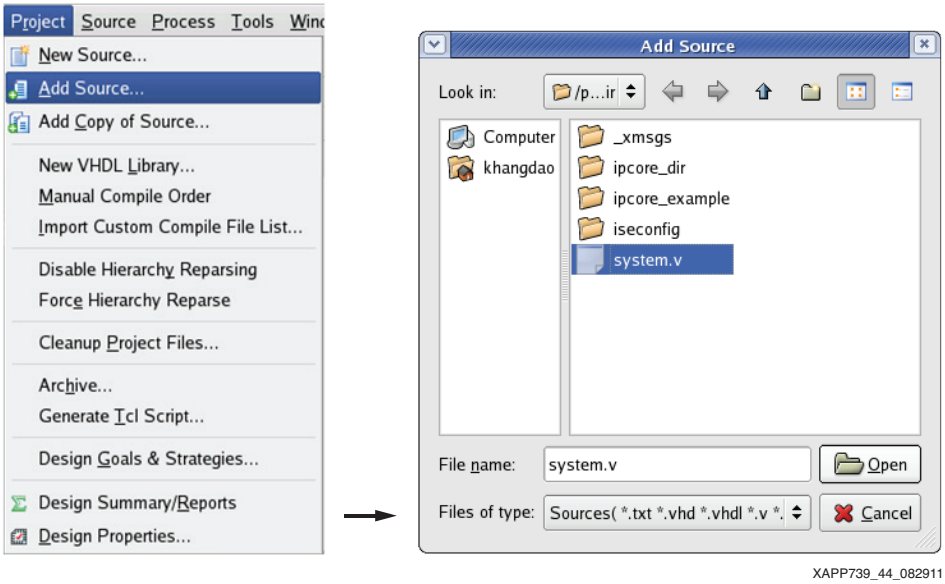



*Figure 44:* **Adding Source Files to Project Navigator**

33. For the Adding Source Files menu, click **OK** (Figure 45).

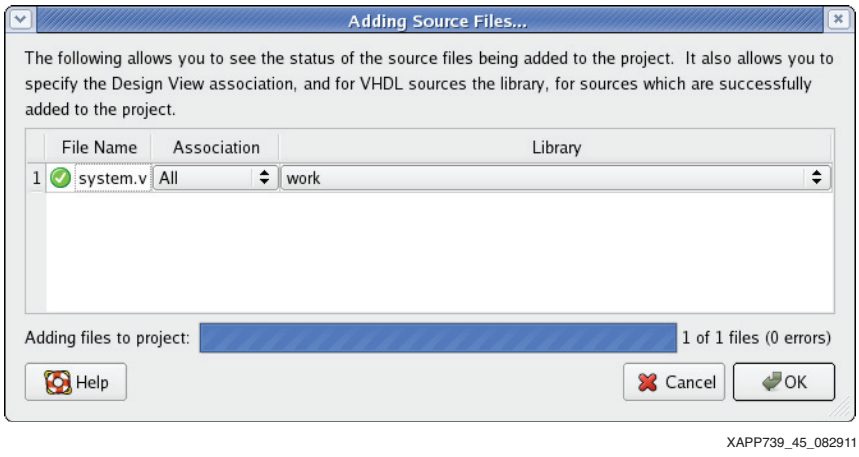



*Figure 45:* **Adding Source File Associations**

Due to some conflicting Verilog module names between MIG and the AXI Interconnect in the ISE Design Suite 13.2, the following steps are needed to work around the naming conflict. The

MIG XCO core source must removed and the non-conflicting Verilog files are added to the project instead.

34. In the Hierarchy pane, right-click and remove the `DDR3_SDRAM.xco` file. When prompted to confirm remove, click **Yes** (Figure 46).

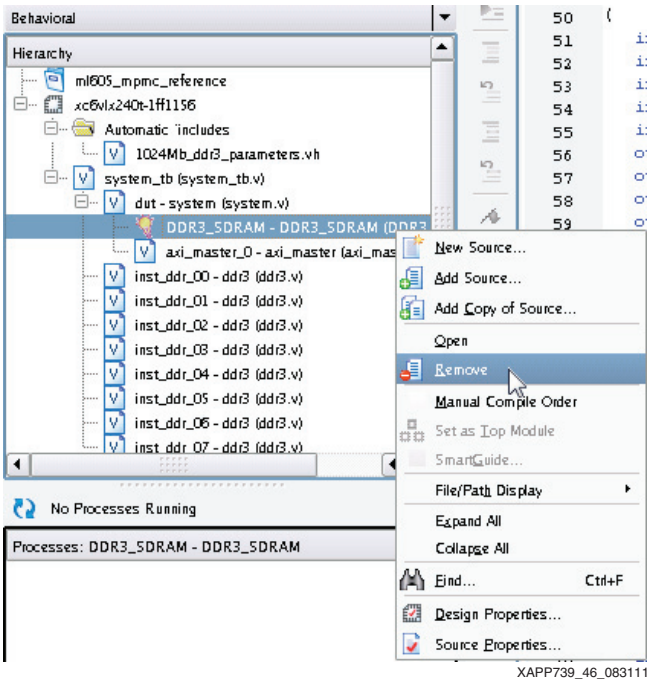



*Figure 46:* **Removing a MIG XCO file from Project Navigator**

35. Click **Project** > **Add Source** to add back only the necessary MIG Verilog source files (Figure 47).

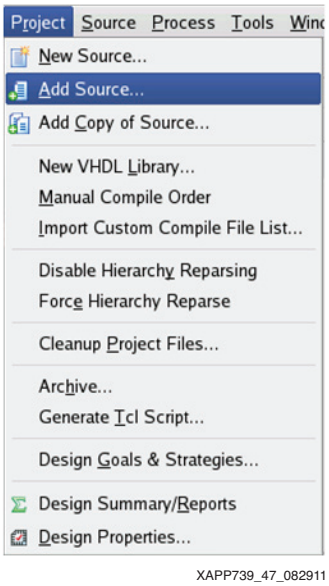
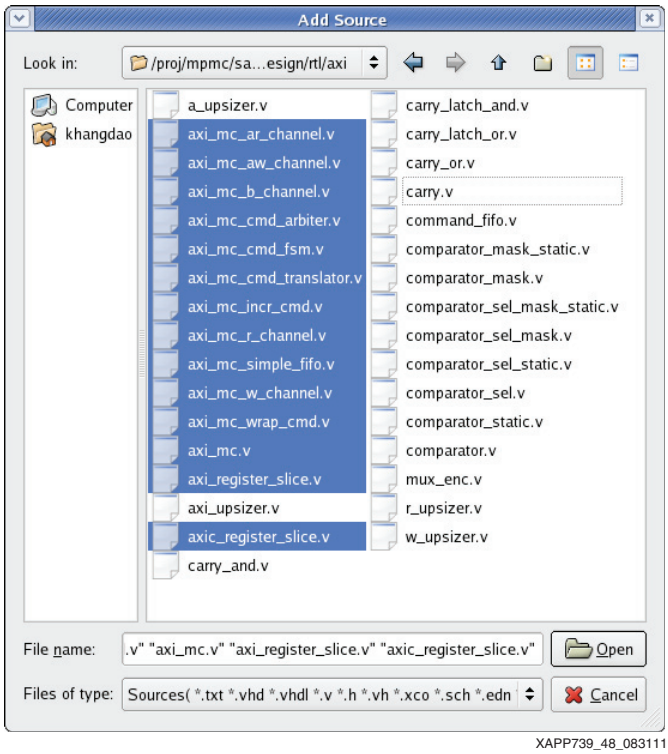



*Figure 47:* **Adding Source Files to Project Navigator**

36. Navigate to the `<user_dir>/ipcore_dir/DDR3_SDRAM/user_design/rtl/axi` directory. Add only the files shown in Figure 48 to prevent duplicate module conflicts.

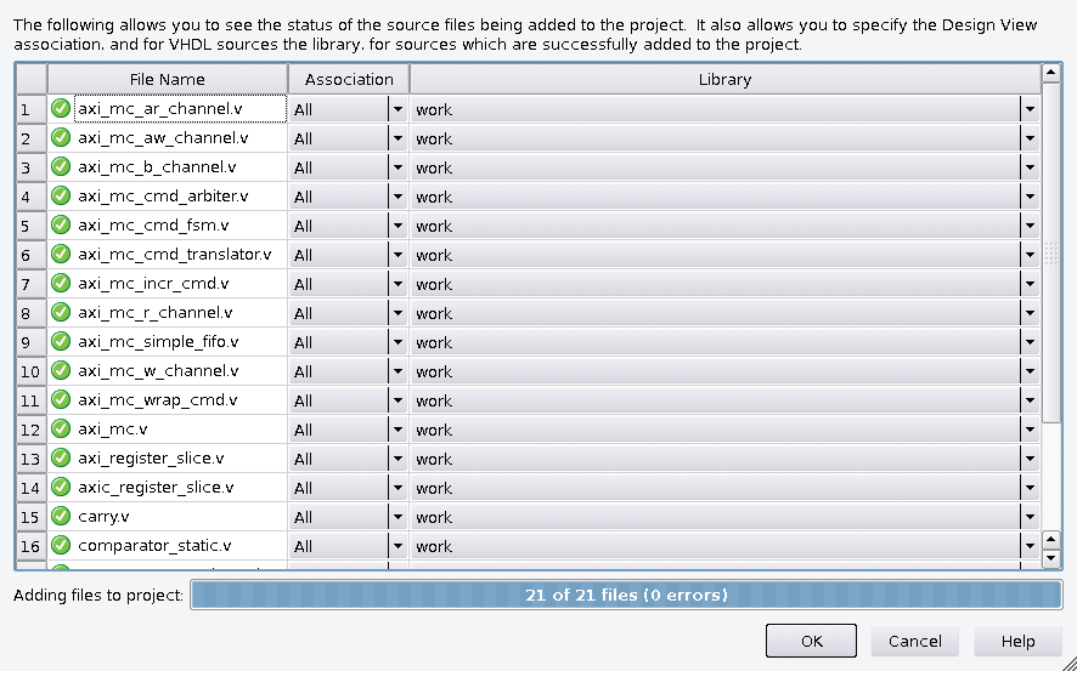
    ***Note:*** Use **Shift-Select** to highlight a range of files and **Control-Select** to select additional files.




XAPP739_48_083111

*Figure 48:* **Browser to Choose Source Files to Add to Project Navigator**

37. Click **OK** to add these Verilog files with Association column value set to All, which incorporates them to both Implementation and Simulation flows (Figure 49).




XAPP739_49_083111

*Figure 49:* **New Verilog Source File Association Selection**

38. Repeat step 35 to step 37 to add the rest of the MIG Verilog files inside the `controller`, `ecc`, `ip_top` (all except `clk_ibuf.v`, whose instantiation will be removed in step 41b), `phy`, and `ui` subdirectories under `<user_dir>/ipcore_dir/DDR3_SDRAM/user_design/rtl/`.
39. The DDR3_SDRAM instance should now be under `system.v` in the Project Navigator Hierarchy pane. Any additional unused MIG Verilog files (not under the DDR3_SDRAM hierarchy) might still exist and can be safely removed from the project. An example is shown in Figure 50.

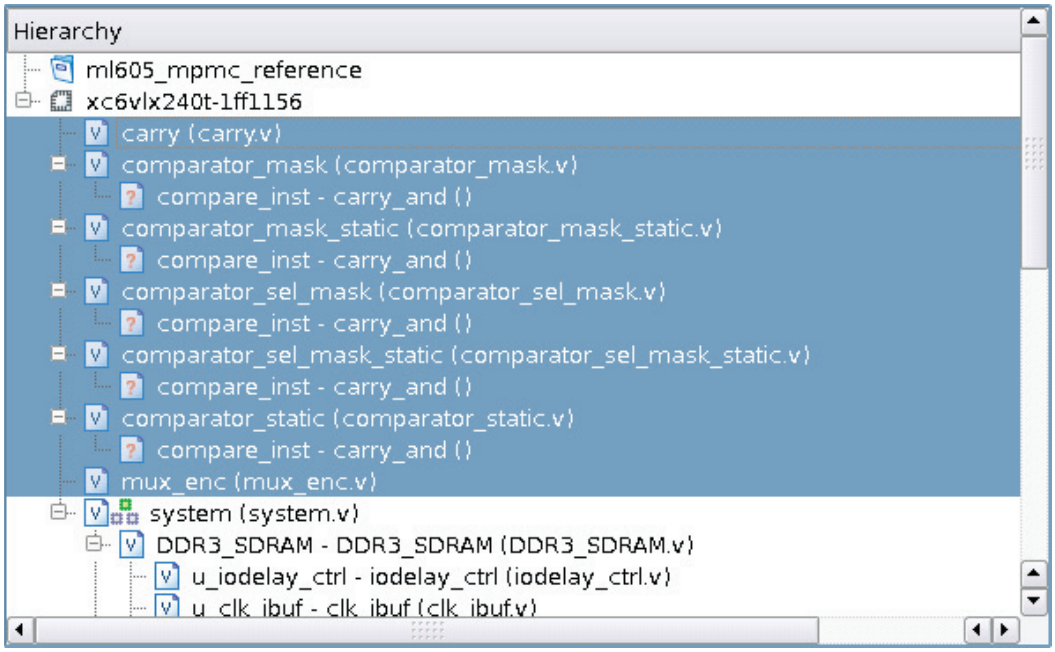



*Figure 50:* **Removing Unnecessary MIG Files Using Project Navigator**

## MIG ML605 Adjustments

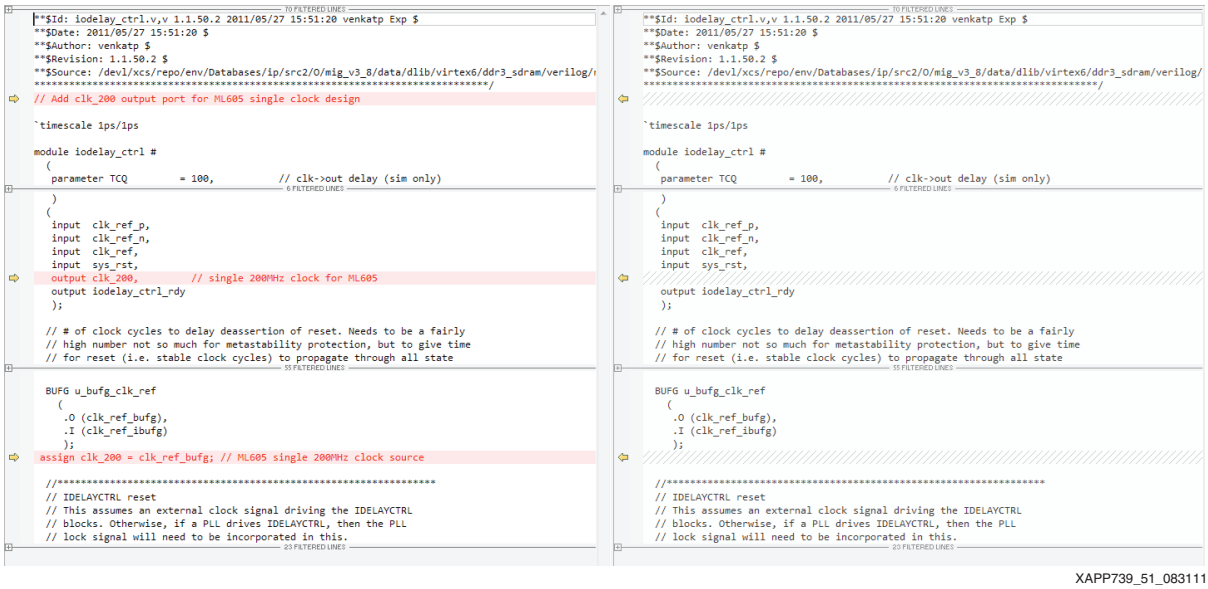
The normal MIG flow in the CORE Generator tool delivers its own clock generation module as part of the example design it creates. This AXI MPMC design requires additional clock connections and features than the default 13.2 version generated by the MIG design of the CORE Generator tool. Therefore, the following steps are needed to manually edit the clocking logic generated by the MIG tool to suit the ML605 AXI MPMC system's needs.

> ***Caution!*** Regeneration of the MIG core overwrites these changes. Consider backing up the modified files to another location.

The following step 40 through step 43 involve edits to some MIG files. To save time, copy the corresponding files from `<design_dir>/projnav/ipcore_dir/DDR3_SDRAM/user_design/rtl/ip_top` directory that have already been edited.

40. Edit the `<user_dir>/ipcore_dir/DDR3_SDRAM/user_design/rtl/ip_top/iodelay_ctrl.v` file to include an additional output port for a clk_200 signal. The changes are summarized in the left pane (Figure 51).

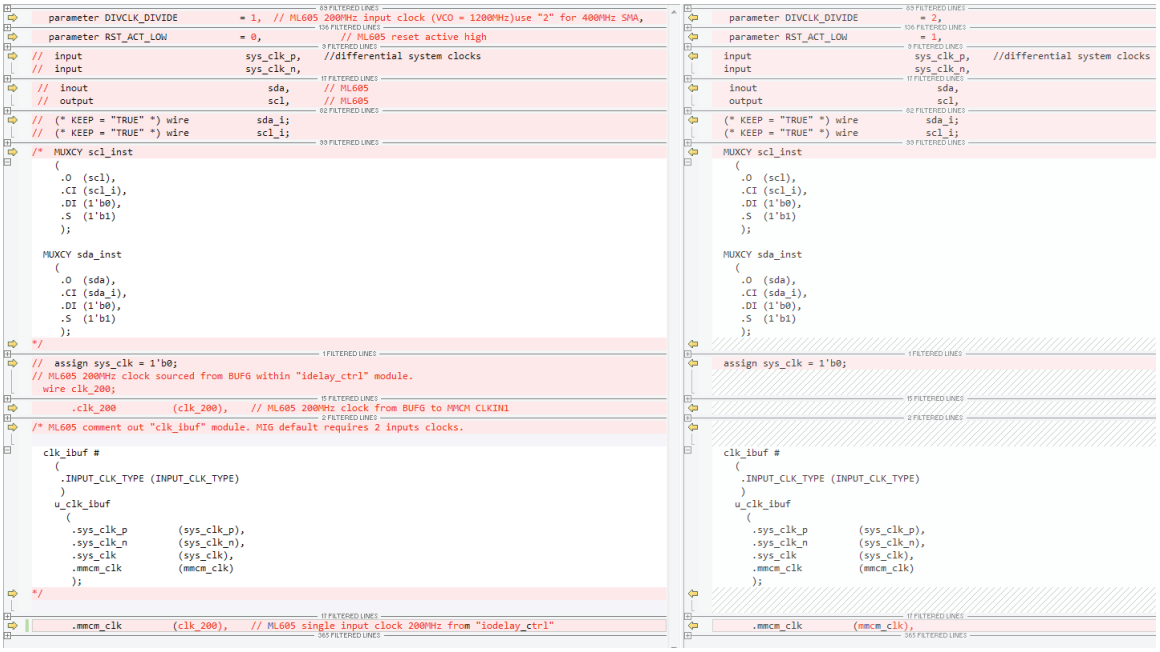
    ***Note:*** **File** > **Open** can be used to invoke Project Navigator's text editor.




XAPP739_51_083111

*Figure 51:* **Changes to MIG `iodelay_ctrl.v` File**

41. Edit the `<user_dir>/ipcore_dir/DDR3_SDRAM/user_design/rtl/ip_top/DDR3_SDRAM.v` file in the following steps. The ML605 board only has a single 200 MHz reference clock input, and an active-Low reset input that must be changed from the default MIG output.
    a. Change the DIVCLK_DIVIDE parameter to 1 and RST_ACT_LOW to 0.
    b. Remove the sys_clk_p/sys_clk_n ports and its associated `clk_ibuf` module.
    c. Remove the sda/scl SPD EEPROM ports and their associated logic.
    d. Connect the mmcm_clk port of the u_infrastructure instance to the recently created clk_200 signal from the `iodelay_ctrl.v` file.
42. The resulting `DDR3_SDRAM.v` file changes should resemble the left pane in a difference report (Figure 52).




XAPP739_52_083111

*Figure 52:* **Changes to MIG `DDR3_SDRAM.v` file**

### Creating Design Constraints

The majority of constraints in the design come from the generated MIG core. This section describes how to modify the MIG constraints and add any needed constraints specific to this design and the ML605 development board. To save time, the completed constraints file can be copied from `<design_dir>/projnav/system.ucf` to `<user_dir>/system.ucf`.

43. Copy the `<user_dir>/ipcore_dir/DDR3_SDRAM/user_design/par/DDR3_SDRAM.ucf` as `<user_dir>/system.ucf` and modify it using a text editor like **File** > **Open in Project Navigator**:
    a. Add a timing ignore (TIG) constraint on the initialization done signal from the MIG tool:

```
NET "DDR3_SDRAM/u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_init/dfi_init_complete" TIG;
```

    b. Remove any sys_clk_p, sys_clk_n, sda, scl, and phy_init_done related constraints because these ports are not used.
    c. Remove the sys_clk_p and TNM_sys_clk PERIOD constraint because sys_clk_p is not used due to earlier modifications.
    d. Modify the INST hierarchal paths to include the DDR_SDRAM instantiation in `system.v` by performing a search and replace of all instances of u_memc_ui_top with DDR3_SDRAM/u_memc_ui_top (except from step a above).
    e. Add a DIFF_TERM = “TRUE” differential termination constraint to clk_ref_p/clk_ref_n.

```
NET  "clk_ref_p"             IOSTANDARD = LVDS_25 | DIFF_TERM = "TRUE";
NET  "clk_ref_n"             IOSTANDARD = LVDS_25 | DIFF_TERM = "TRUE";
```

    f. Change the sys_rst IOSTANDARD to SSTL_15, and add a TIG constraint:

```
NET  "sys_rst"               IOSTANDARD = SSTL15 | TIG;
```

    g. Add these I/O location (LOC) constraints for the video and video I2C pinout:

```
NET dvi_out_reset_n   LOC = AP17;
NET dvi_out_de        LOC = AD16;
NET dvi_out_vsync     LOC = AD15;
NET dvi_out_hsync     LOC = AN17;
NET dvi_out_clk_p     LOC = AC18;
NET dvi_out_clk_n     LOC = AC17;
NET dvi_out_data(0)   LOC = AJ19;
NET dvi_out_data(1)   LOC = AH19;
NET dvi_out_data(2)   LOC = AM17;
NET dvi_out_data(3)   LOC = AM16;
NET dvi_out_data(4)   LOC = AD17;
NET dvi_out_data(5)   LOC = AE17;
NET dvi_out_data(6)   LOC = AK18;
NET dvi_out_data(7)   LOC = AK17;
NET dvi_out_data(8)   LOC = AE18;
NET dvi_out_data(9)   LOC = AF18;
NET dvi_out_data(10)  LOC = AL16;
NET dvi_out_data(11)  LOC = AK16;
NET video_out_scl     LOC = AN10;
NET video_out_sda     LOC = AP10;
```

44. Verify the changes against the ensuing difference report with the modified `system.ucf` on the left pane (step 53).

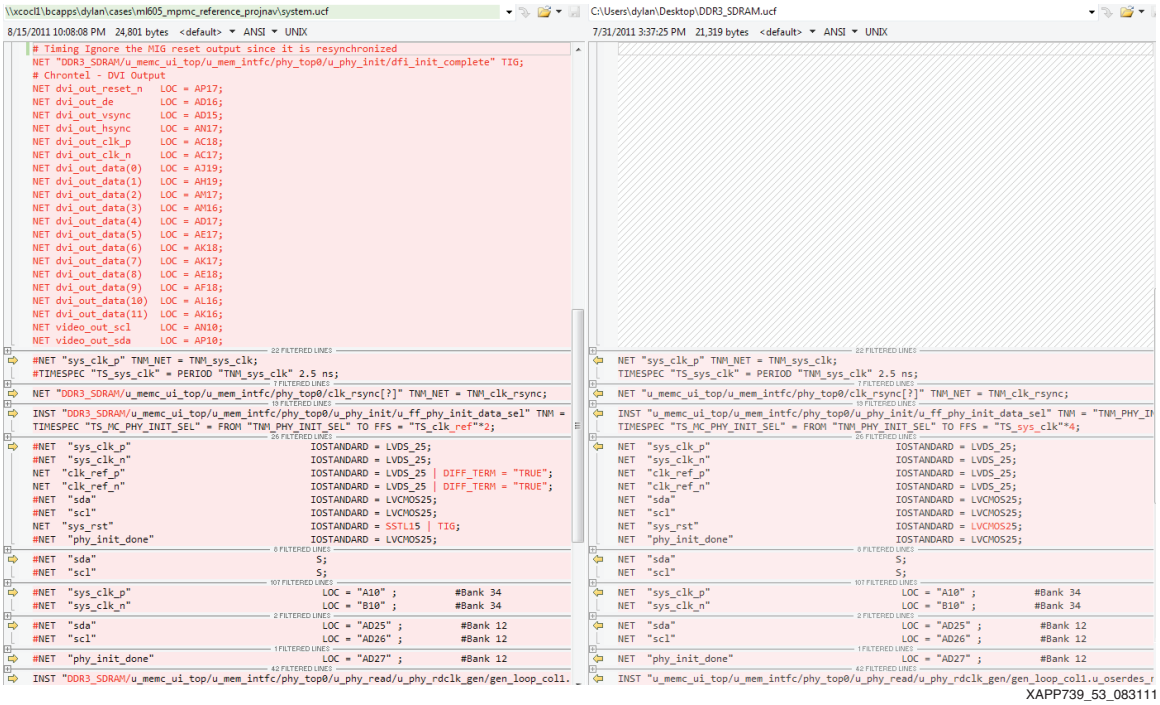



*Figure 53:* **Changes to `system.ucf` File**

45. Add `system.ucf` to the project:
    a. Select **Project** > **Add Source**.
    b. Browse and select `<user_dir>/system.ucf` and click **Open**.
    c. Associate the `system.ucf` file for Implementation and click **OK**.

## Adding an AXI Interconnect to the Design

This section describes the steps to add the AXI Interconnect IP block to the project. Detailed information about the AXI Interconnect IP core is described in *LogiCORE IP AXI Interconnect Datasheet* [Ref 3]. The configuration of the AXI Interconnect and the process of optimizing AXI systems is described in the “AXI System Optimization: Tips and Hints” chapter of the *AXI Reference Guide* [Ref 2].

46. Click **Project** > **New Source** to add a new IP core to the system (Figure 54).

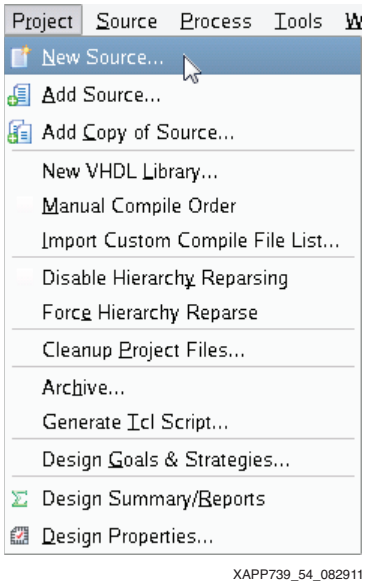



*Figure 54:* **Creating a New Source in Project Navigator**

47. For the Source Type, select **IP**, then enter the file name **AXI4** and the location should default to `<user_dir>/ipcore_dir`. Click **Next** (Figure 55).

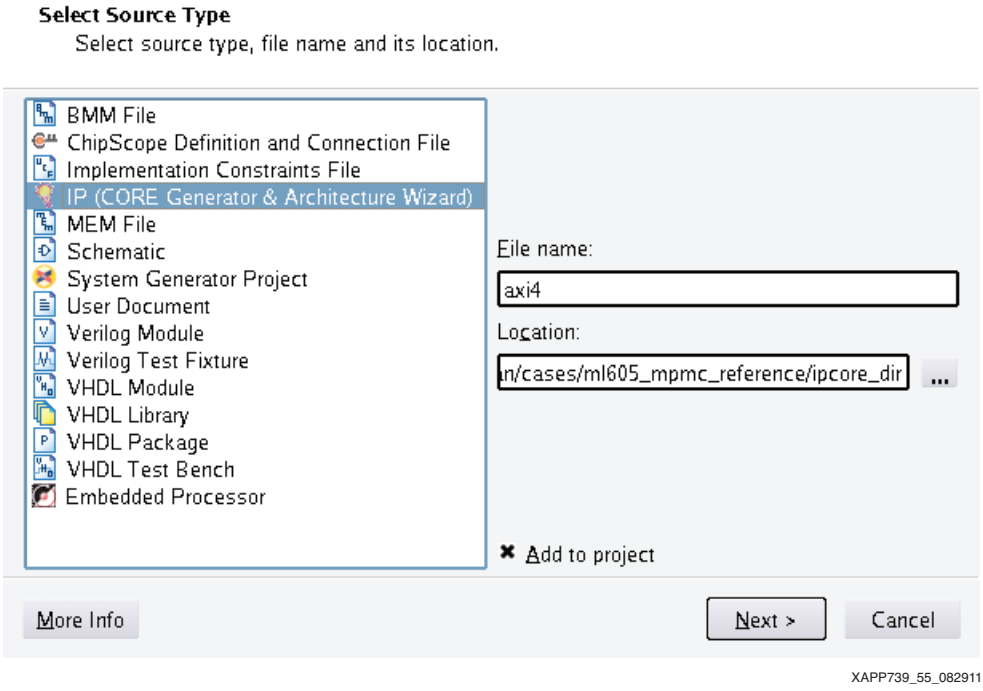



*Figure 55:* **New Source Wizard - Source Type Menu**

48. In the Select IP window, click the **View by Function** tab, then select **AXI Infrastructure** > **AXI Interconnect** (Version 1.03.a) from the IP catalog. Click **Next** (Figure 56).

    ***Note:*** The version number might not appear in the GUI.

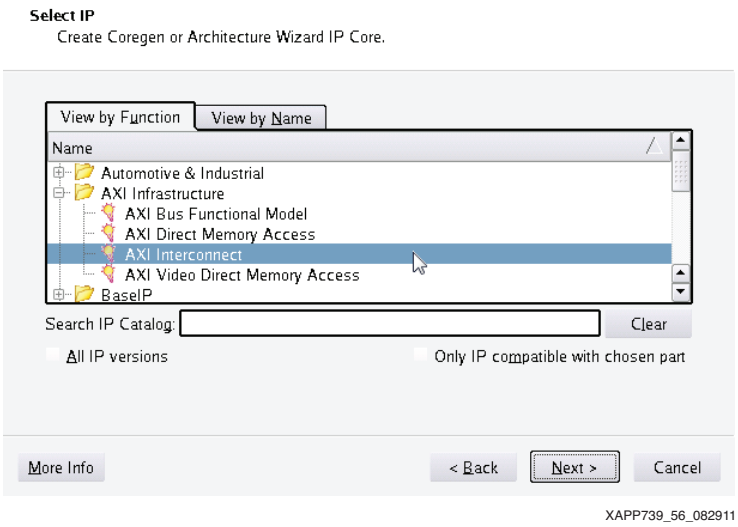



*Figure 56:* **New Source Wizard - Select IP Menu**

49. In the Summary window, review the settings and click **Finish**. This creates a CORE Generator tool project for the AXI Interconnect and opens the IP configuration GUI (Figure 57).

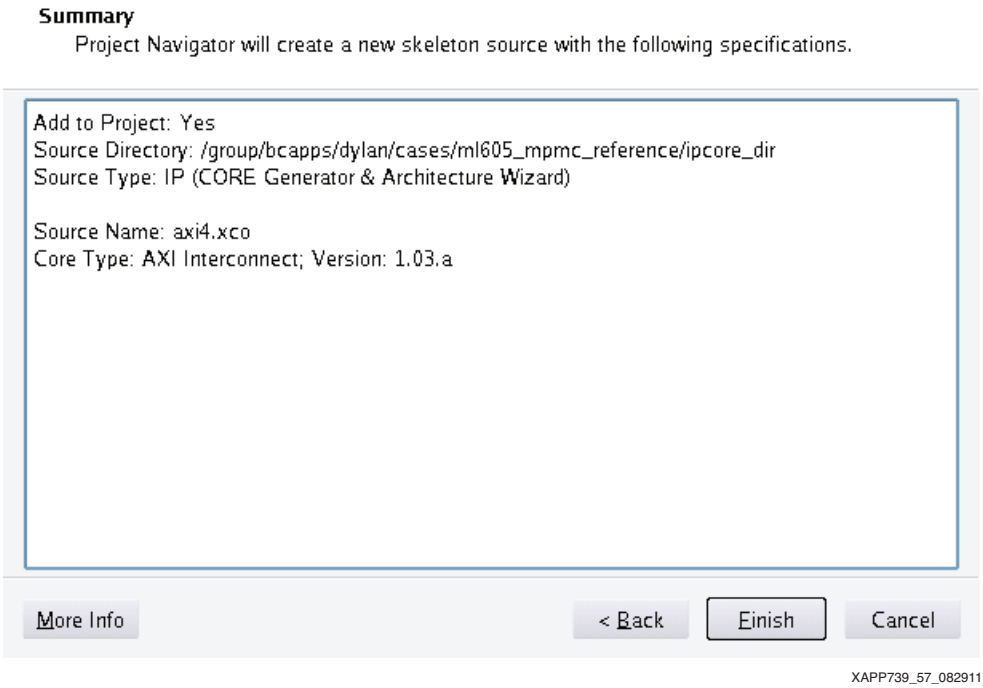



*Figure 57:* **New Source Wizard - Summary Menu**

50. In the AXI Interconnect IP configuration GUI (Figure 58), configure the IP Core to support connections from four master endpoint IP cores to one slave. Each AXI_VDMA contains a separate read-only and write-only interface, resulting in a total of four AXI masters between both AXI_VDMAs. The following settings should be used for the AXI Interconnect:

- Number of Slave Interfaces: **4**.

  This allows four AXI master endpoints to be connected to the Interconnect.
- Slave Interface Thread ID Width: **0**.

  This is because the AXI_VDMA master does not generate IDs. Four slave interfaces that do not use the ID fields result in a master interface ID width of 4. These ID fields are used to route responses back to the appropriate AXI Interconnect slave interface and the AXI DMA connected to it.
- Address Width: **32**.

  This matches the MIG tools address width setting.
- Interconnect Internal Data Width: **256**.

  This matches the MIG tool's AXI Interface data width.

  ***Note:*** Setting this value lower than 256 bits causes a throughput bottleneck between the MIG tool and the AXI Interconnect. Setting this value above 256 bits increases area unnecessarily.
- Period of Interconnect_ACLK: **5000 ps**.

  This sets the AXI Interconnect to operate at 200 MHz, the same clock frequency as the MIG tool's AXI Interface.

  ***Note:*** The AXI Interconnect is configured to match the 256 bit x 200 MHz AXI Interface of the AXI MIG it connects to. This optimizes the performance of the system and reduces the need for width and clock conversion.

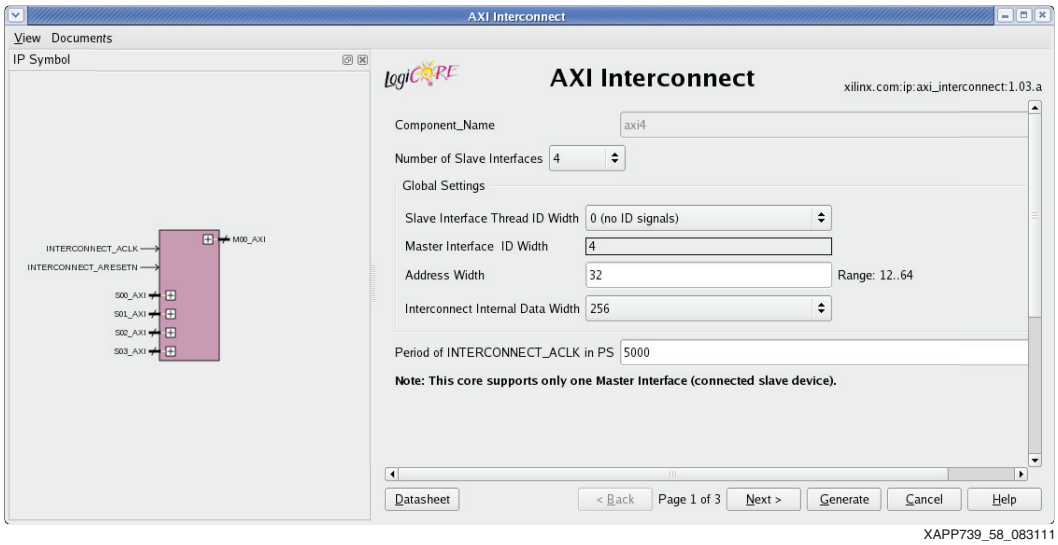



*Figure 58:* **AXI Interconnect Global Settings**

51. In Page 2 of the AXI Interconnect IP Configuration GUI, use the following settings (Figure 59):
    - Master Interface Data Width: **256 bits**.

      This matches the MIG slave AXI interface width.
    - Slave Interface Data Width: **32 bits**.

      This matches each AXI_VDMA master AXI interface width.
    - Slave Interface 0-3, Is Aclk Async: **Checked**.

      Clock converters are required on the slave interfaces (enable corresponding check boxes) to the AXI VDMA masters because they run at 75 MHz, whereas the AXI Interconnect runs at 200 MHz.
    - Register Slices are not enabled because the system is relatively simple, but if timing is not met, the pipelining of register slices might be required.

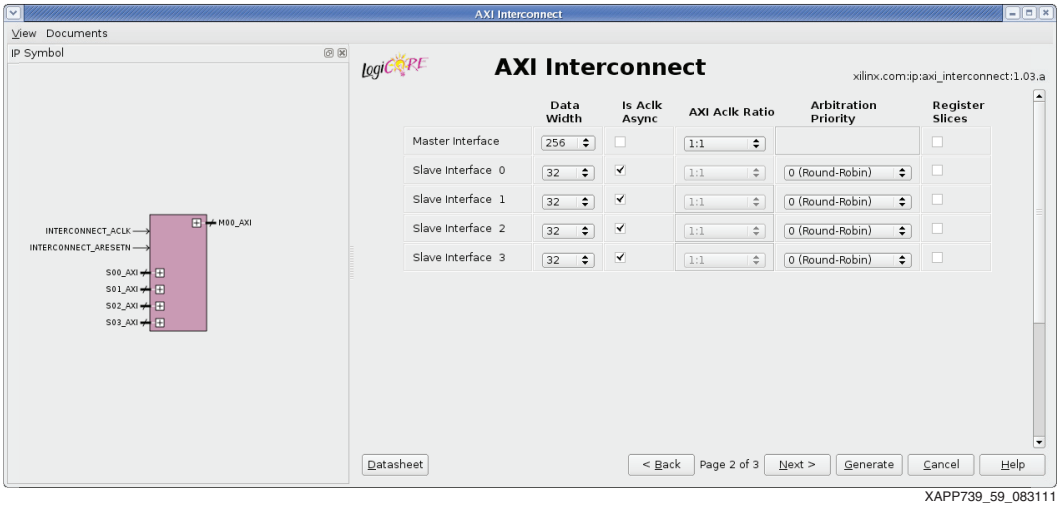
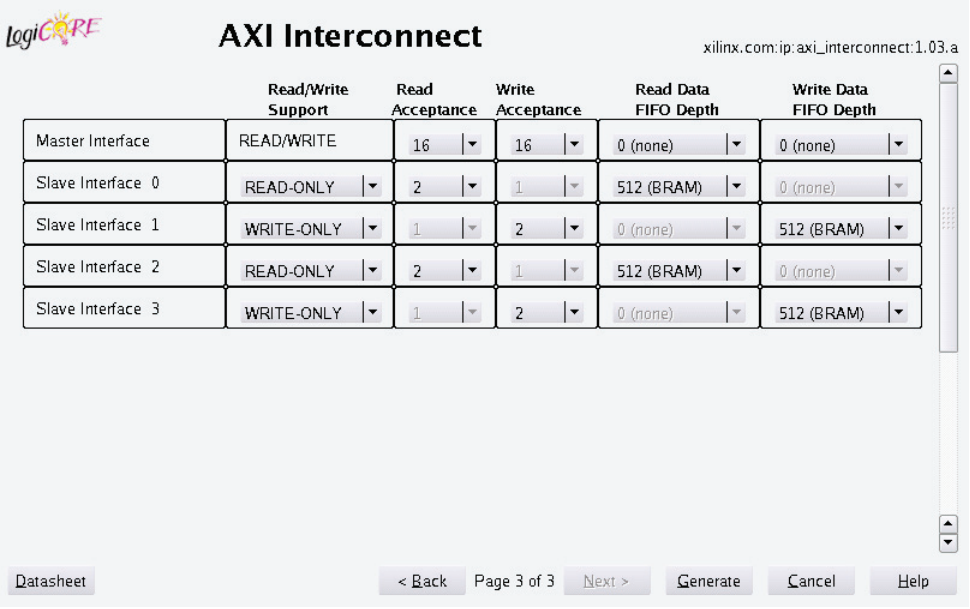



*Figure 59:* **AXI Interconnect Settings**

52. In Page 3 of the AXI Interconnect IP Configuration GUI, use the following settings (Figure 60):
    - Slave Interfaces 0 and 2 should be read-only to connect to the MM2S AXI interfaces of the AXI_VDMA.
    - Slave Interfaces 1 and 3 should be write-only to connect to the S2MM AXI interfaces of the AXI_VDMA.
    - As described in the AXI Optimization chapter of the *AXI Reference Guide* [Ref 2], the AXI_VDMA uses a relatively long burst so that an acceptance value of 2 minimizes area while allowing support for pipelined transactions. On the master interface to the memory controller, a higher acceptance value of 16 allows it to pipeline as many transactions as possible to the memory controller to maximize throughput.
    - As described in the AXI Optimization chapter of the *AXI Reference Guide* [Ref 2], block RAM FIFOs on the slave interfaces should be enabled to improve throughput and reduce performance bottlenecks due to the AXI masters running at a lower throughput than the AXI MIG.

    Click **Generate**.

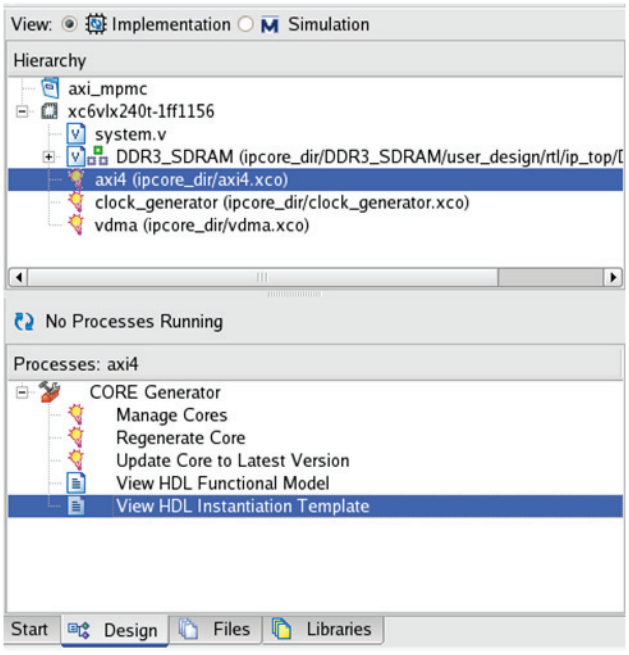
    ***Note:*** After clicking **Generate**, it takes several minutes to generate the core before returning to an idle GUI in Project Navigator.




XAPP739_60_082911

*Figure 60:* **AXI Interconnect Settings Continued**

53. Select the **axi4** instance in the Hierarchy pane of Project Navigator, and in the Processes window pane, double-click **CORE Generator** and then click **View HDL Instantiation Template** (Figure 61).




XAPP739_61_082911

*Figure 61:* **Viewing the HDL Instantiation Template for the AXI Interconnect IP Core**

54. Add the resulting template to the `<user_dir>/system.v` file. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead. If it is desired to edit `system.v` to make the connections between the MIG IP core and the AXI Interconnect, edit the `<user_dir>/system.v` file. Define HDL wires as needed and connect them as shown in Table 2.

*Table 2:* **Connections in `system.v` with AXI Interconnect Added**

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| AXI Interconnect | M00_AXI_* | MIG | s_axi_* |
| MIG | ui_clk (200 MHz) | AXI Interconnect | INTERCONNECT_ACLK |

## Adding Clocking Wizard to the Design

The AXI MIG core contains its own clock generation and reset logic, leaving only the 75 MHz video clock to be generated with the CORE Generator tool's Clocking Wizard. The MIG tool provides an output 200 MHz clock via its ui_clk port that can then be used as the Clocking Wizard input clock.

55. Go to **Project** > **New Source...** and select **IP (CORE Generator & Architecture Wizard)** with a name of **`clock_generator`** (Figure 62).

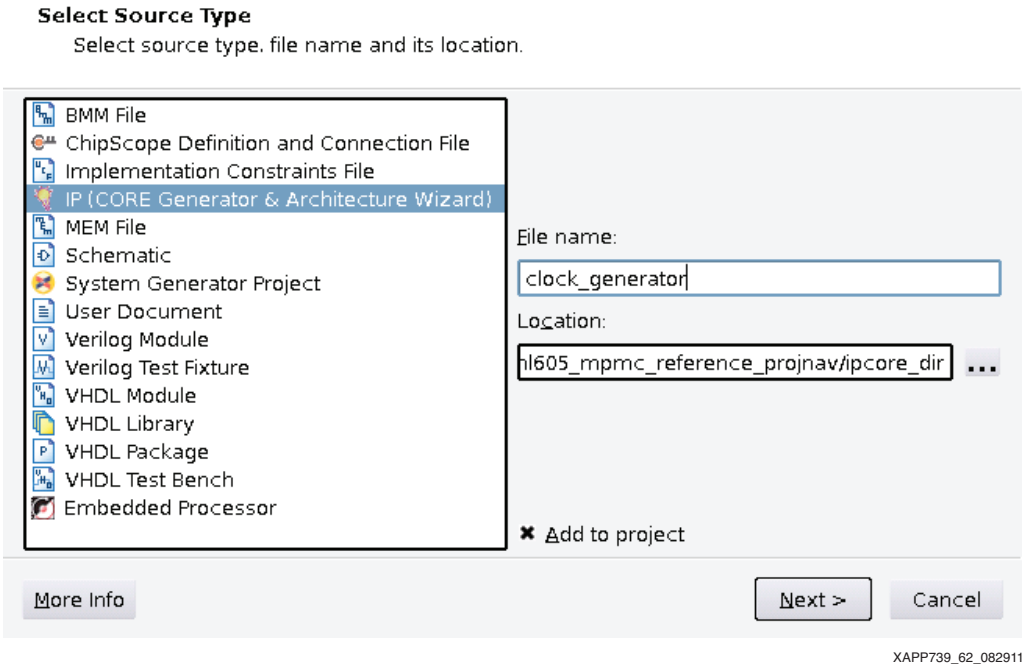



*Figure 62:* **New Source Wizard - Source Type Menu**

56. Click the **View by Function** tab, then select **FPGA Features and Design** > **Clocking** > **Clocking Wizard** (Version 3.2). Click **Next** to advance to the summary screen (Figure 63).

    ***Note:*** The version number might not appear in the GUI.

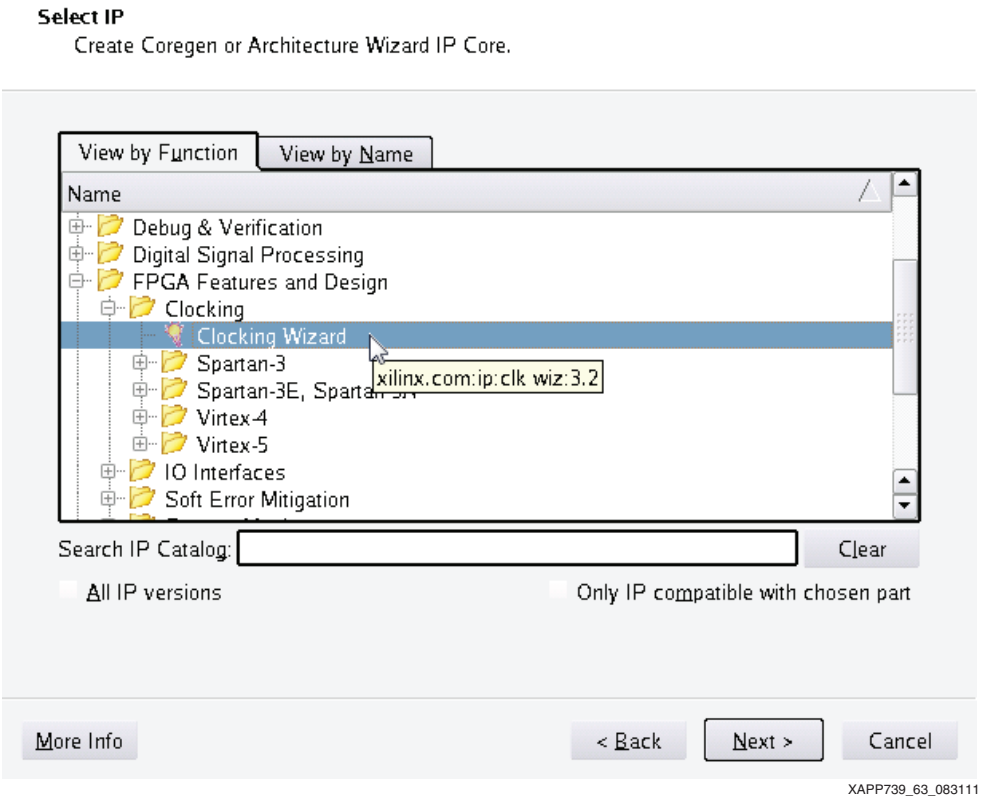



*Figure 63:* **New Source Wizard - Select IP Menu**

57. Click **Finish** to continue to the Clocking Wizard configurator GUI screen (Figure 64).

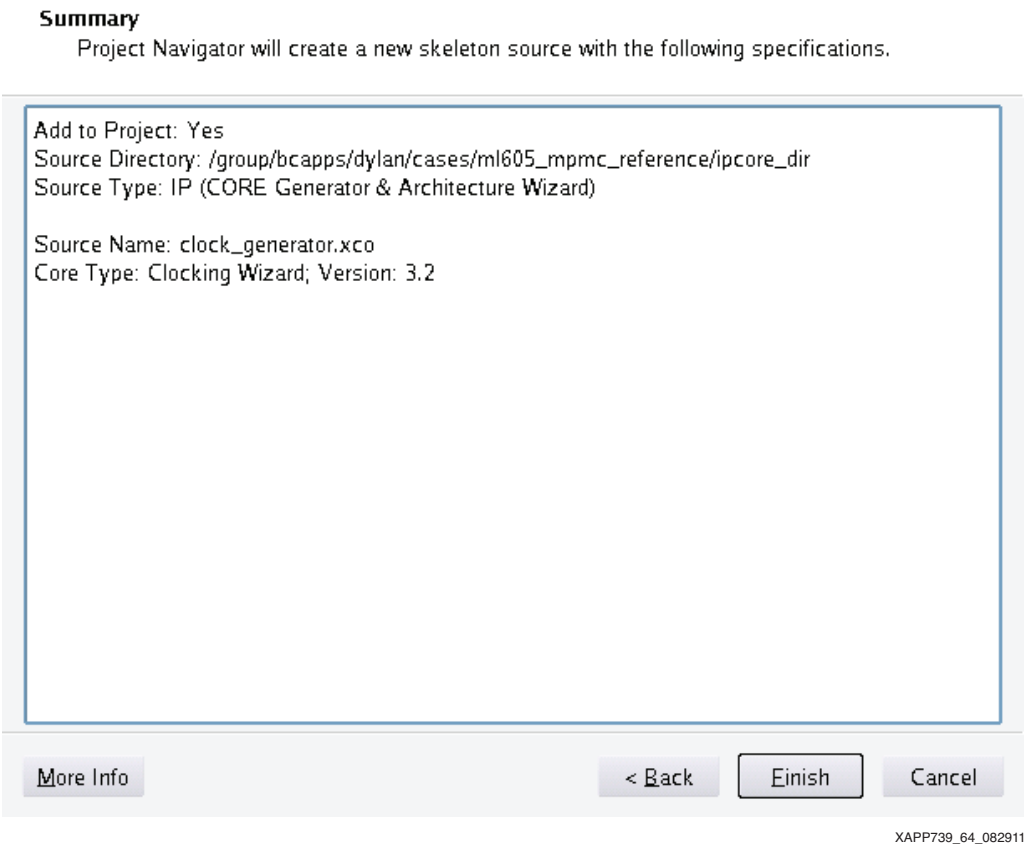
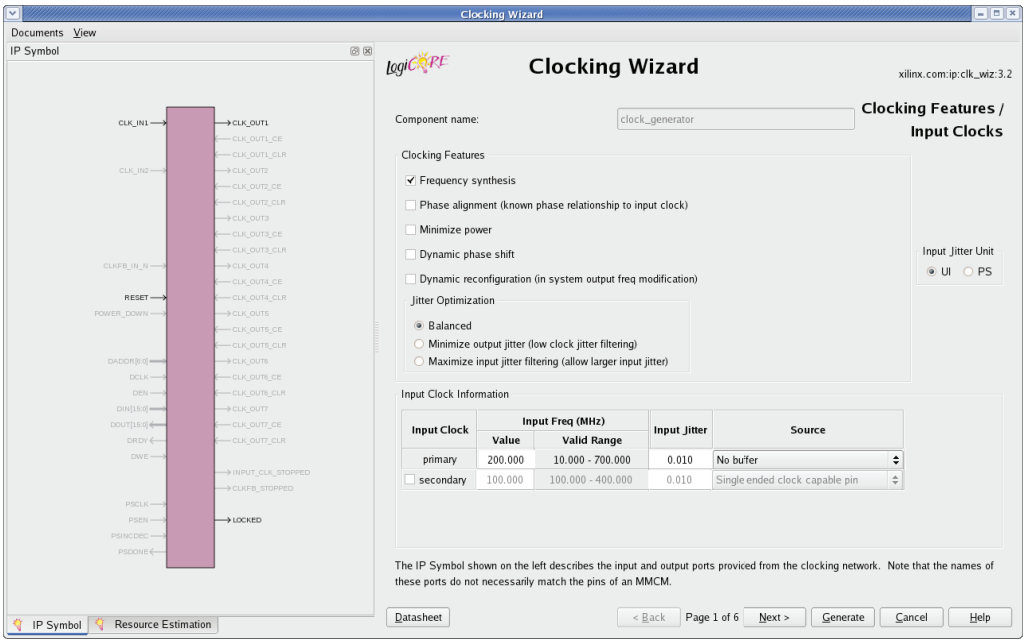



*Figure 64:* **New Source Wizard - Summary Menu**

58. In the Clocking Features/Input Clocks page, make these settings (Figure 65):
    - Select **Frequency Synthesis**.
    - For primary input clock frequency, use **200.000 MHz**.
    - In the source pull-down menu, select **No buffer**.
    - Uncheck **Phase Alignment**.

    Click **Next**.

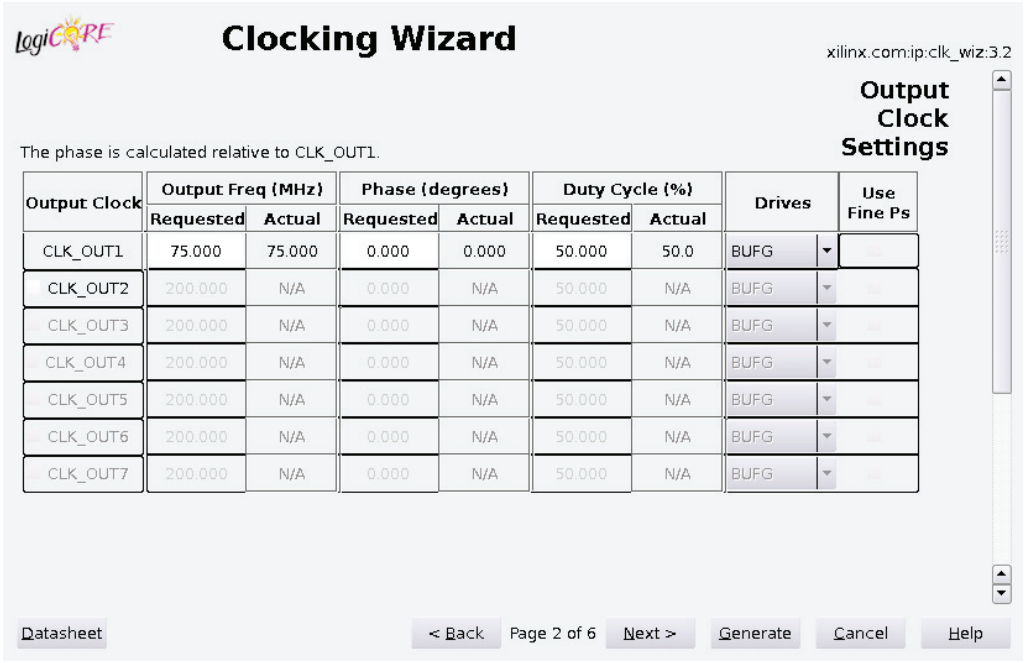



XAPP739_65_082911

*Figure 65:* **Clocking Wizard Features and Input Clock Settings**

59. In the Output Clock Settings page, make these settings (Figure 66):
    - For the CLK_OUT1 output frequency, select **75 MHz**.
    - In the Drives pull-down menu, select the **BUFG** global clock buffer.

    Click **Next**.




XAPP739_66_082911

*Figure 66:* **Clocking Wizard Output Clock Settings**

60. Enable the RESET and LOCKED signals, then click **Next** (Figure 67).

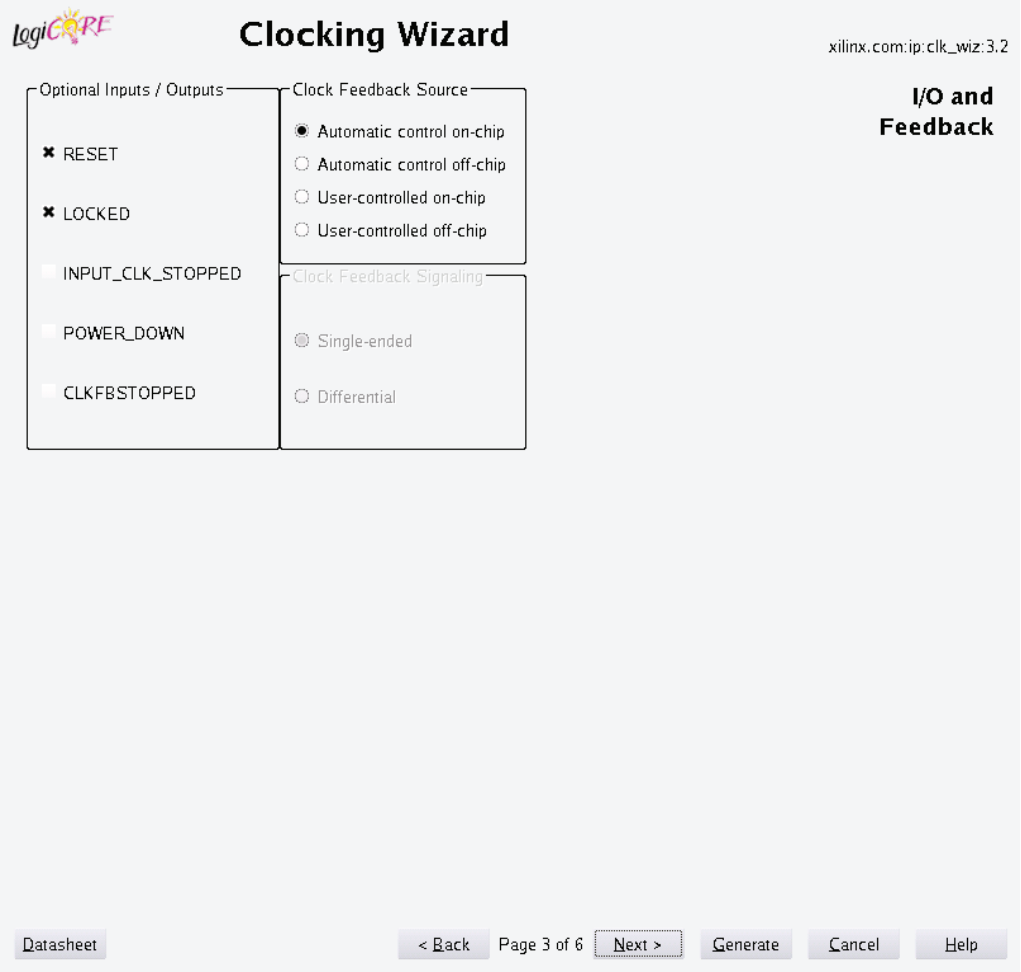



*Figure 67:* **Clocking Wizard I/O and Feedback Settings**

61. Review the MMCM_ADV Settings page and click **Next** (Figure 68).

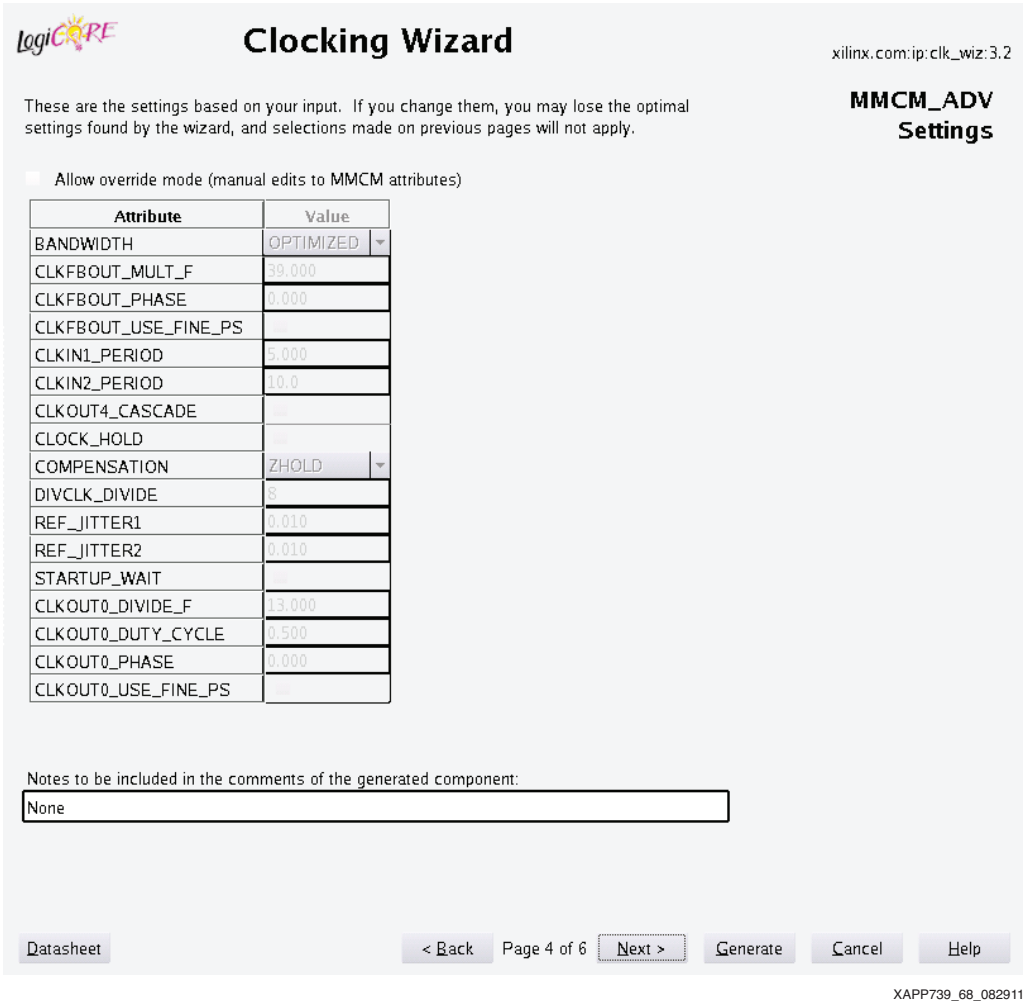



*Figure 68:* **Clocking Wizard MMCM_ADV Settings**

62. In the Clock Summary, Port Naming page, enter **`CLK_IN1`** as the primary input clock name and click **Next** (Figure 69).

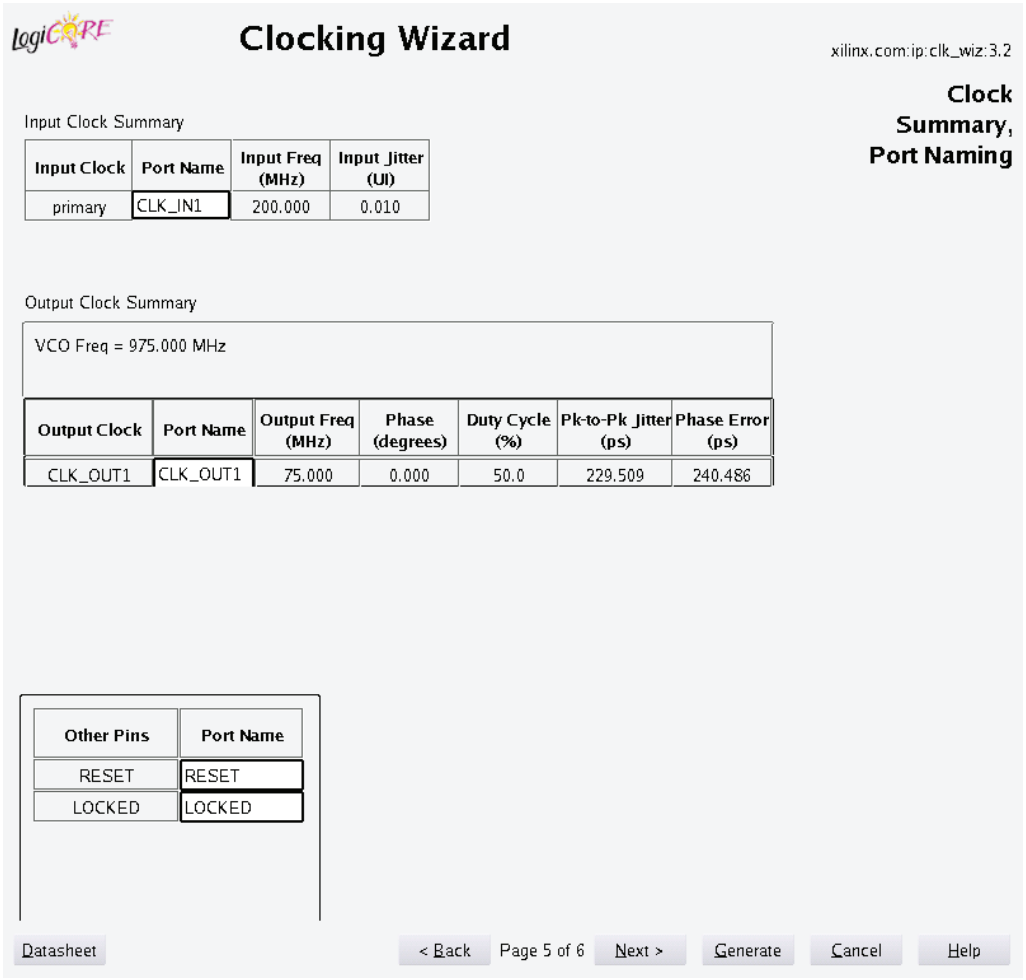



*Figure 69:* **Clocking Wizard Summary and Port Naming Settings**

63. Review the settings and output files in the Core Summary, and complete the wizard by clicking **Generate** (Figure 70).

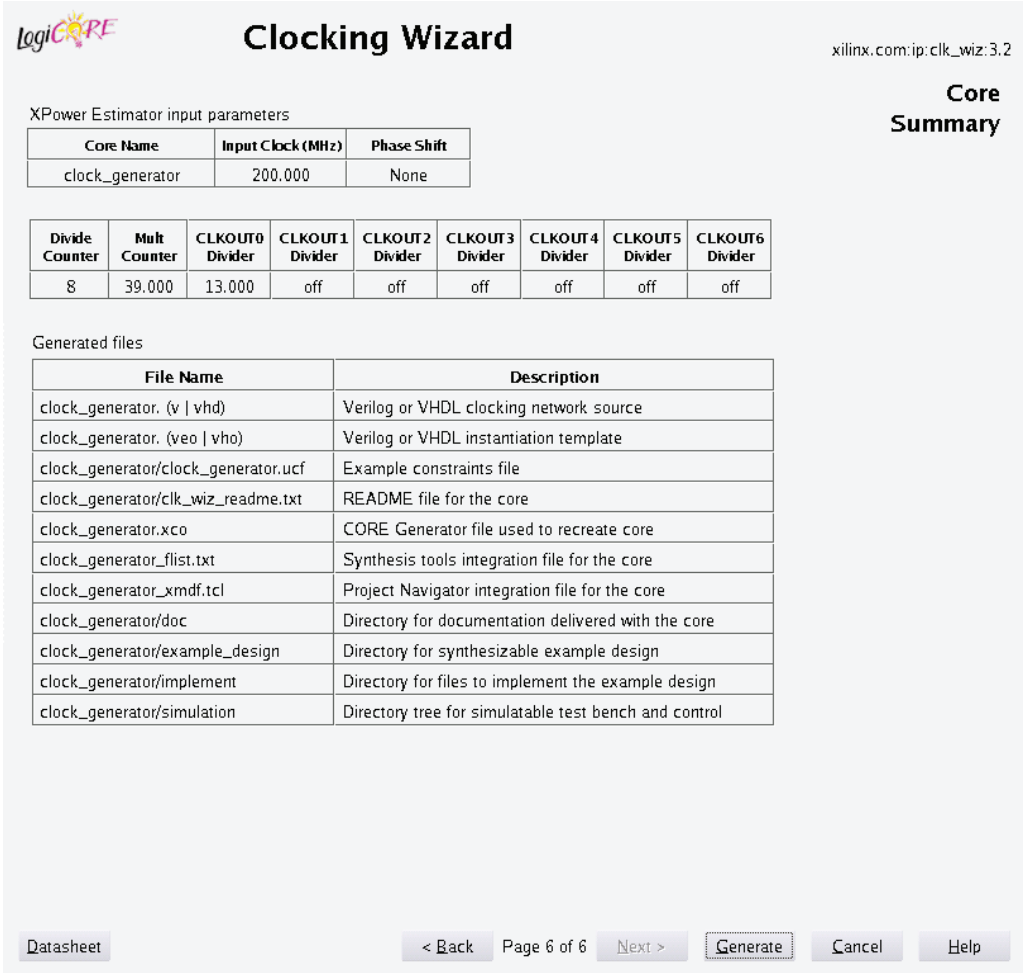



*Figure 70:* **Clocking Wizard Core Summary**

64. Select the clock_generator instance in the Hierarchy pane of Project Navigator and in the Processes window pane, double-click **CORE Generator** > **View HDL Instantiation Template** (Figure 71).

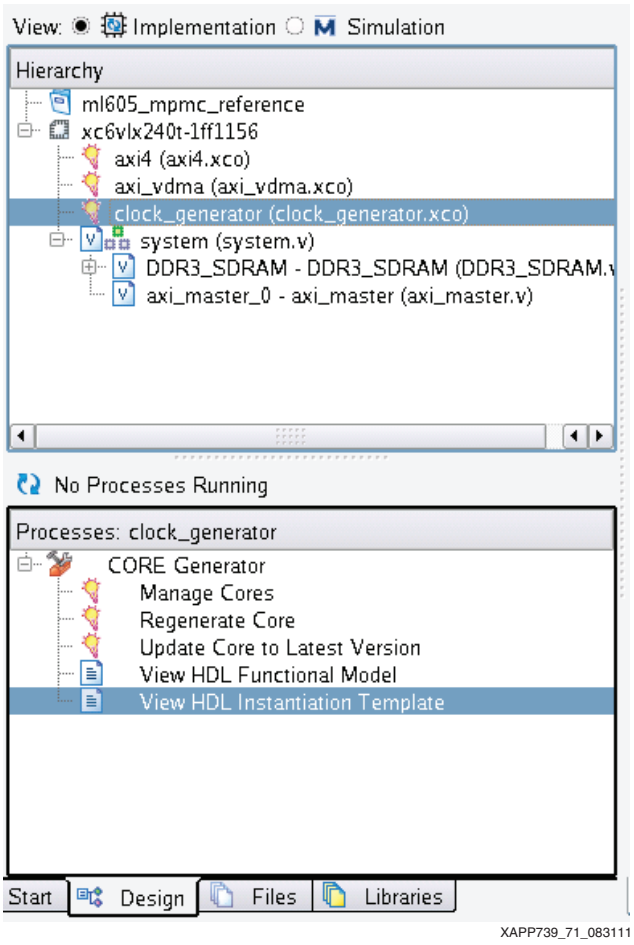



*Figure 71:* **Viewing a CORE Generator IP Instantiation Template From Project Navigator**

65. Add the resulting template to the `<user_dir>/system.v` file. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead. To edit `system.v` to make the connections to the clock_generator, edit the `<user_dir>/system.v` file. Define HDL wires and connect the ports shown in Table 3.

*Table 3:* **Connections in `system.v` with Clock Generator Added**

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| MIG | ui_clk (200 MHz) | Clock Generator | CLK_IN1 |
| MIG | ui_clk_sync_rst | Clock Generator | RESET |
| Clock Generator | LOCKED | AXI Interconnect | INTERCONNECT_ARESETN |
| Clock Generator | CLK_OUT1 (75 MHz) | AXI Interconnect | S0*_AXI_ACLK |

**Notes:**

The LOCKED output of clock_generator drives the INTERCONNECT_ARESETN input of AXI Interconnect to ensure that the interconnect does not come out of reset until all of the mixed-mode clock manager (MMCM)-generated clocks are locked.

## Adding AXI VDMA to the Design

This section describes the generation of both AXI_VDMA cores into the design. VDMA_0 receives video data from the master_example core, and places it In DDR3 memory. VDMA_0 reads data back out and loops the video data over AXI4-Stream channels to VDMA_1. VDMA_1 writes the data to memory, reads it back out, and presents it to the axi_stream_slave_example block for display. Both AXI VDMA cores are configured identically in this design.

66. Go to **Project** > **New Source…** and select **IP (CORE Generator & Architecture Wizard)** with a name of **`vdma`**. Click **Next** (Figure 72).

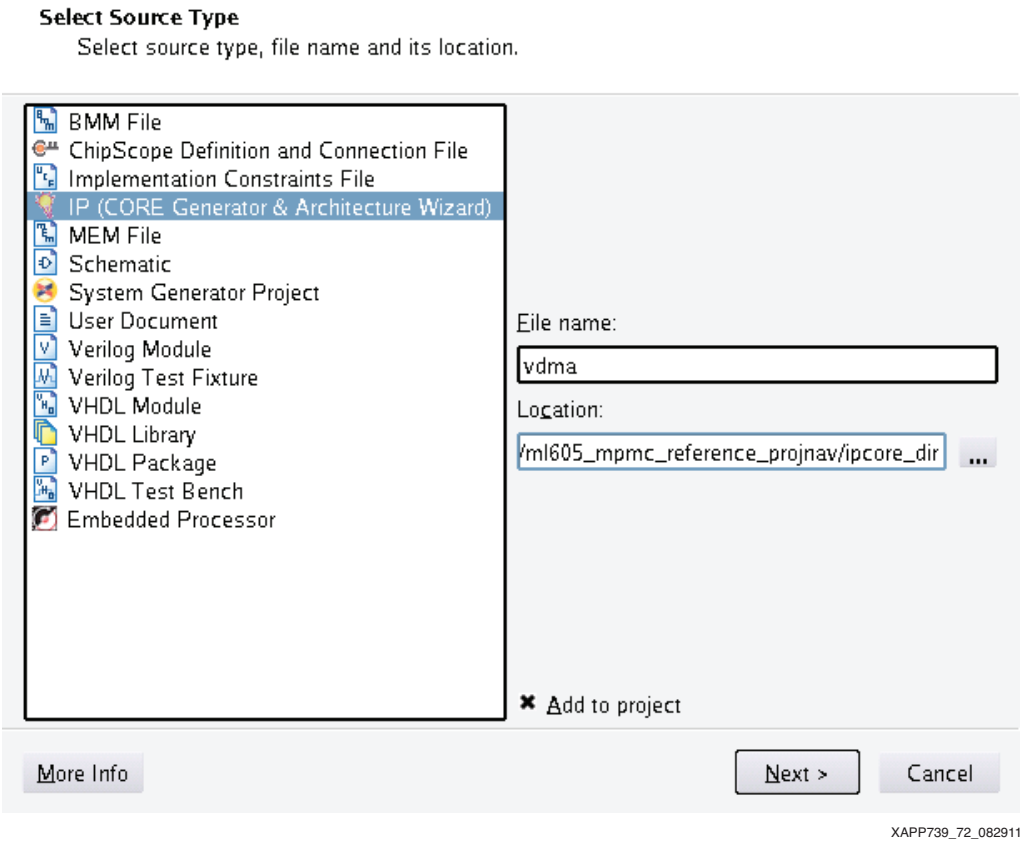



*Figure 72:* **New Source Wizard - Source Type Menu**

67. Click the **View by Function** tab, then select **AXI Infrastructure** > **AXI Video Direct Memory Access** (Version 3.1) and click **Next** (Figure 73).

    ***Note:*** The version number might not appear in the GUI.

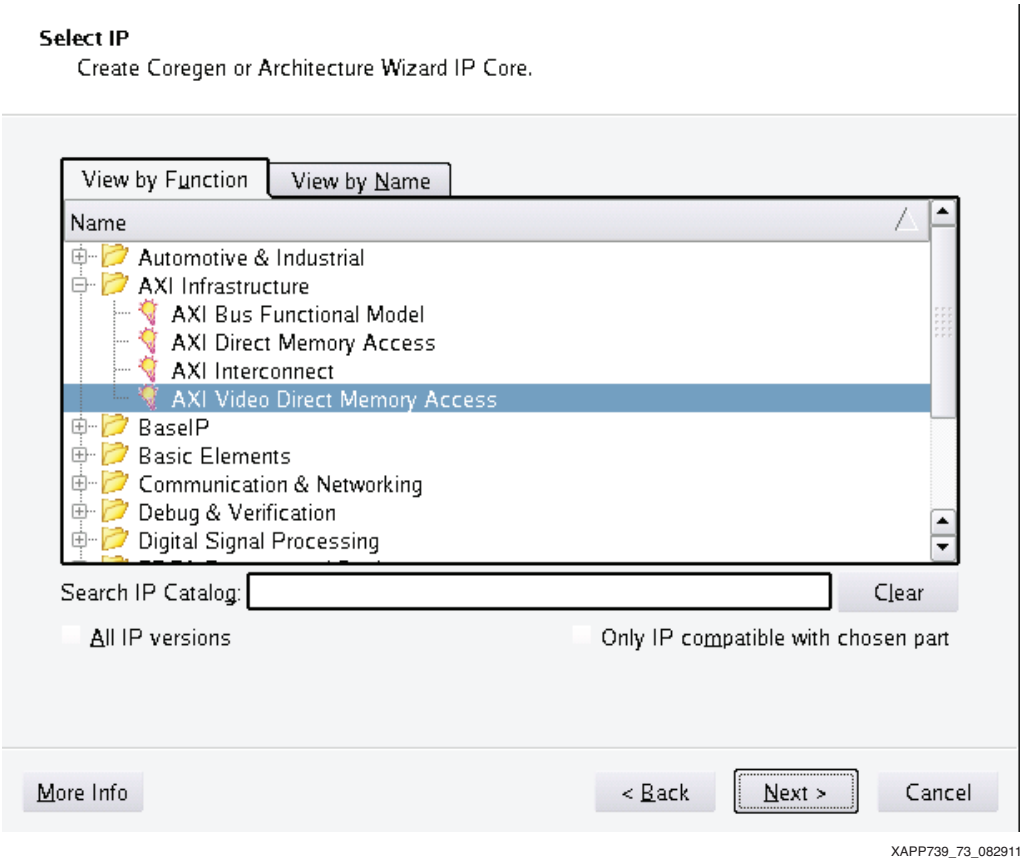



*Figure 73:* **New Source Wizard - Select IP Menu**

68. Review the Summary page and click **Finish** (Figure 74).

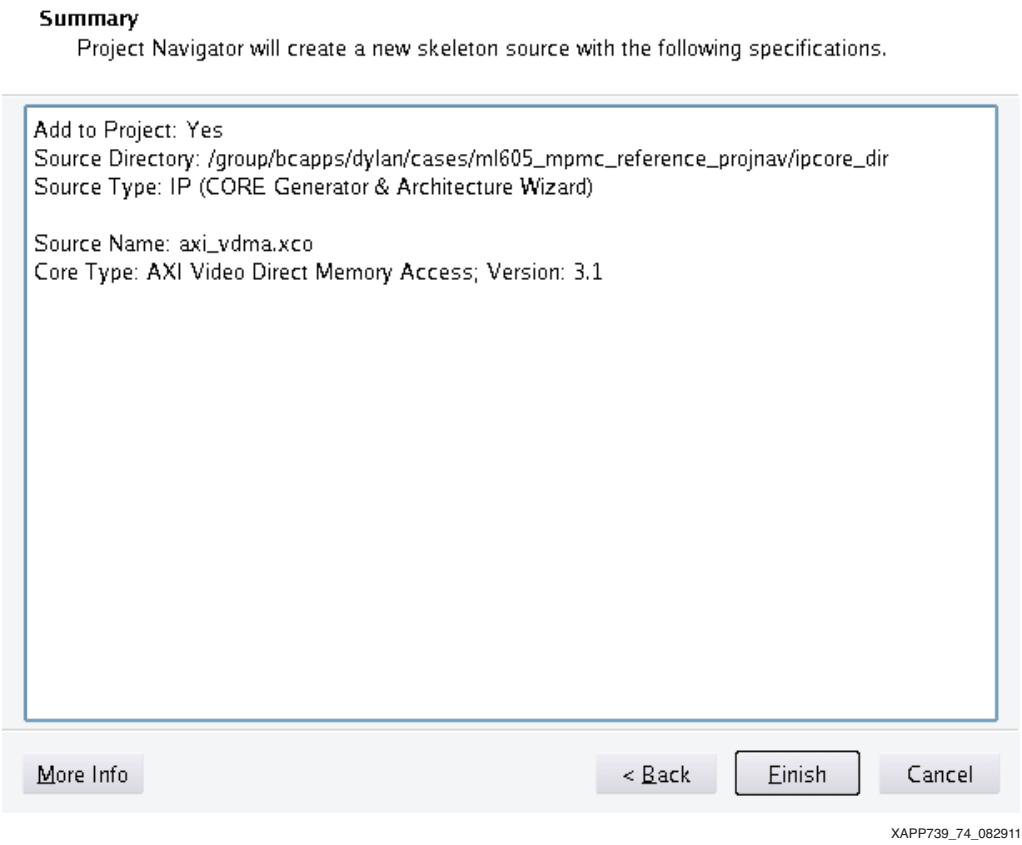
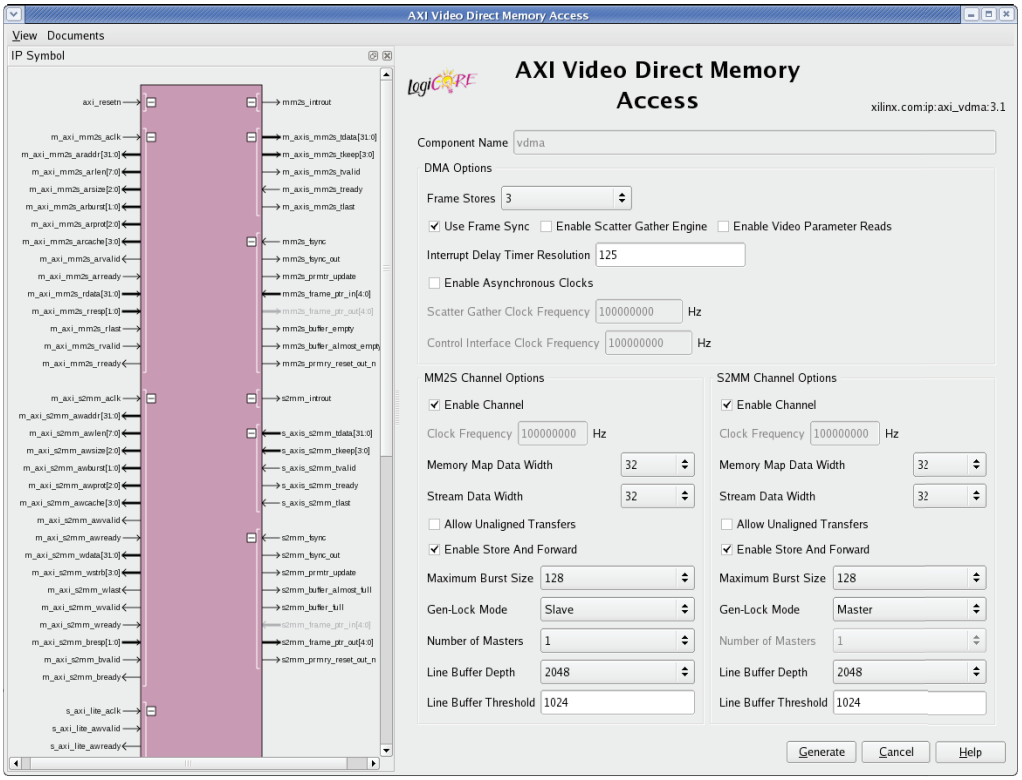



*Figure 74:* **New Source Wizard - Summary Menu**

## Configuring the VDMA

The VDMA will be configured to act as a video frame buffer with settings to enable reasonable performance. The following information describes how to configure AXI VDMA.

69. Make the settings shown in Figure 75 and as described in the list that follows:




XAPP739_75_082911

*Figure 75:* **VDMA IP Configuration GUI**

a. Set Frame Stores to **3**.

   This sets the maximum number of frame buffer ranges needed and generates associated configuration registers for each. For this design, three frame buffers provide sufficient buffering to allow the MM2S channel to read data and the S2MM channel to write data without the channels overwriting or reading stale data.

b. Uncheck **Enable Scatter Gather Engine**.

   Scatter/gather operation generally requires a processor or additional state machine to provide and maintain buffer descriptors. In this design, a simple AXI4 Lite Master configures the VDMAs directly through the AXI4 Lite Slave interface. After the video transfer parameters of hsize, frame delay, stride, and start addresses are written, vsize is written to start the video transfers. Transfers continue indefinitely with frame synchronization signals keeping each interface from overwriting or reading stale data.

c. Uncheck **Enable Video Parameter Reads**.

   A read multiplexer for the video transfer parameters vsize, hsize, frame delay, stride, and start addresses allows reading of the registers via the AXI4 Lite slave interface. This design does not read VDMA registers, so this feature can be disabled to reduce FPGA resource utilization.

d. Check **Use Frame Sync**.

   This feature enables the VDMA frame rates to be synchronized to the video endpoint IP they are connected to.

e. Set Interrupt Delay Timer Resolution to **125**.

   This sets the resolution of the delay timer for delay interrupts. This design does not utilize VDMA interrupts, and therefore, this setting is left at the default value.

f. Uncheck **Enable Asynchronous Clocks**.

All functions of the VDMAs in this design run off the same 75 MHz clock. The asynchronous clock crossing to the 200 MHz DDR3 AXI interface occurs in the AXI Interconnect. Running the VDMAs in a synchronous mode reduces FPGA resource utilization of the VDMA.

g. Check **Enable Channel** for both the MM2S and S2MM. These stand for Memory-Mapped (AXI4 reads) to AXI4-Stream, and AXI4-Stream to Memory-Mapped (AXI4 writes), respectively.

h. For both MM2S and S2MM interfaces, set the Memory Map Data Width to **32**.

   This setting matches the video endpoint datapath widths.

i. For both channels, uncheck **Allow Unaligned Transfers**.

   In this design, all video lines and frames are aligned to 32-bit word boundaries, so the data realignment engines in the VDMA are not needed. Leaving the realignment engines disabled reduces FPGA resource utilization.

j. For both channels, check **Enable Store and Forward**.

   Enabling Store and Forward buffering configures the VDMAs to only post reads or writes on AXI4 that can be immediately completed. This prevents the VDMAs from unnecessarily blocking accesses on AXI4 for extended periods of time.

k. Set Maximum Burst Size to **128**.

   This sets a somewhat large AXI4 burst size for the VDMAs. A setting of 128 provides higher bandwidth to prevent video data over runs or under runs. Choosing a large Maximum Burst Size also improves DDR3 memory utilization. The DDR3 memory has a 256-bit wide AXI4 slave interface that requires long bursts to be efficient. However, using a large burst length can increase latency of other masters in the system. This design trade-off is discussed in the *AXI Reference Guide* [Ref 2].

l. Gen-Lock is used to maintain frame level synchronization between the MM2S and S2MM channels, preventing each interface from overwriting or reading stale data. In this design, the S2MM channel of each VDMA is configured to be the GenLock **Master** and the MM2S channels are configured to be the GenLock **Slave** with Number of Masters set to **1**.

m. Set Line Buffer Depth to **2048**.

   This enables the VDMA's Line Buffers and sets the depth to 2048 bytes. The line buffers allow some elasticity between the AXI4-Stream and associated AXI4 interfaces, providing sufficient buffering to prevent push-back from the VDMA on the AXI4-Stream.

n. Set Line Buffer Threshold to **1024**.

   The line buffer threshold is the watermark setting for the MM2S almost empty flag and the S2MM almost full flag. These are sometimes utilized by video IP to signal when enough data is buffered on MM2S to begin receiving and when there is enough buffer space available on S2MM to begin transmitting. For this design, the almost full and almost empty flags are not utilized, so an arbitrary setting of 1024 was chosen.

Then click **Generate**.

70. Select the **axi_vdma** instance in the Hierarchy pane of Project Navigator. In the Processes window pane, double-click **CORE Generator** > **View HDL Instantiation Template** (Figure 76).

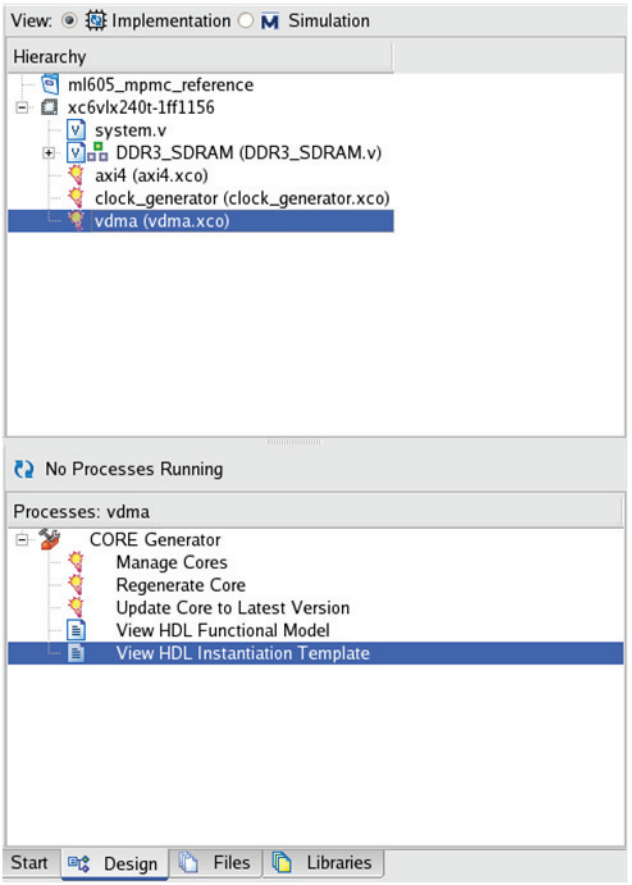



*Figure 76:* **Viewing a CORE Generator IP Instantiation Template From Project Navigator (Before `system.v` is Updated)**

71. Add the resulting Verilog template to the `<user_dir>/system.v` file twice as axi_vdma_0 and axi_vdma_1 instances. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead.

    ***Note:*** After `system.v` is updated and saved to instantiate the axi_vdma twice, the Hierarchy pane of Project Navigator shows both instances of axi_vdma_0 and axi_vdma_1 with the same `vdma.xco` file name associated with them. To edit `system.v` to make the connections to the VDMAs, edit the `<user_dir>/system.v` file. Define HDL wires and connect them as shown in Table 4.

*Table 4:* **Connections in `system.v` with VDMAs Added**

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| AXI_VDMA_0 | m_axi_mm2s_* | AXI Interconnect | S00_AXI_* (read-only) |
| AXI_VDMA_1 | m_axi_mm2s_* | AXI Interconnect | S02_AXI_* (read-only) |
| AXI_VDMA_0 | m_axi_s2mm_* | AXI Interconnect | S01_AXI_* (write-only) |
| AXI_VDMA_1 | m_axi_s2mm_* | AXI Interconnect | S03_AXI_* (write-only) |
| AXI_VDMA_0 | s_axis_mm2s_* | AXI_VDMA_1 | s_axis_s2mm_* |
| Clock Generator | CLK_OUT1 (75 MHz) | AXI_VDMA_* | *aclk |

*Table 4:* **Connections in `system.v` with VDMAs Added *(Cont'd)***

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| AXI Interconnect | S00_AXI_ARESET_OUT_N | AXI_VDMA_0 | axi_resetn |
| AXI Interconnect | S02_AXI_ARESET_OUT_N | AXI_VDMA_1 | axi_resetn |

**Notes:**

Unused AXI Interconnect "S0*_AXI_*" ports can be left unconnected because they correspond to the unused AXI read or write direction in the ports set to be read-only or write-only.

## Adding AXI4-Lite Masters to Configure AXI VDMAs

Each AXI VDMA must be configured at start-up to act as a circular frame buffer on 1280x720p video frames. Each AXI VDMA has an AXI4-Lite slave interface to configure its control registers. In this system, two simple write-only AXI4-Lite masters are instantiated. Each AXI4-Lite master has an internal state machine that executes a fixed sequence of write commands to properly configure the AXI VDMA blocks. A separate AXI4-Lite master IP block is connected directly to each AXI VDMA control interface.

72. Click **Project** > **Add Source** to add the source code for the axi_lite masters to the system (Figure 77).

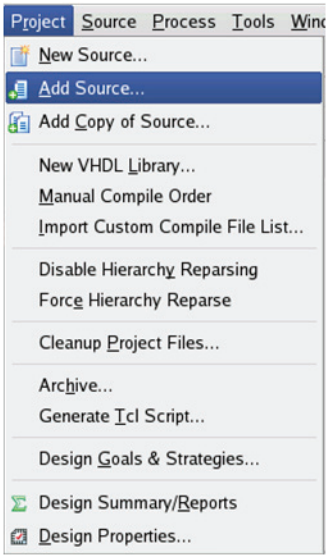
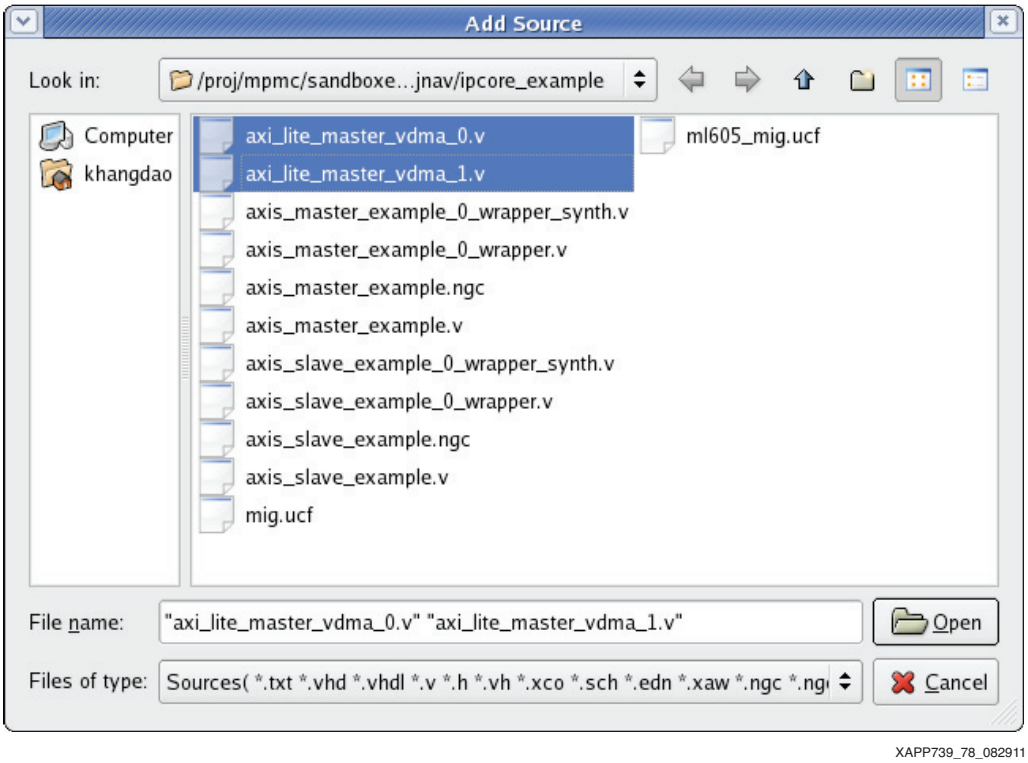



*Figure 77:* **Adding Source Files to Project Navigator**

73. In the Add Source window (Figure 78), browse to the `<user_dir>/ipcore_example/` directory and add the files `axi_lite_master_vdma_0.v` and `axi_lite_master_vdma_1.v`. Then click **Open**.




XAPP739_78_082911

*Figure 78:* **Browser to Add source Files**

74. In the Association pull-down menu, select **All**. Under Library, select **work** and click **OK**. This associates the source code for simulation and synthesis. These files can be browsed in Project Navigator by double clicking them to view the source code for an example AXI4-Lite master. The source code also describes which AXI VDMA register accesses are generated to configure the AXI VDMA control registers for proper operation (Figure 79).

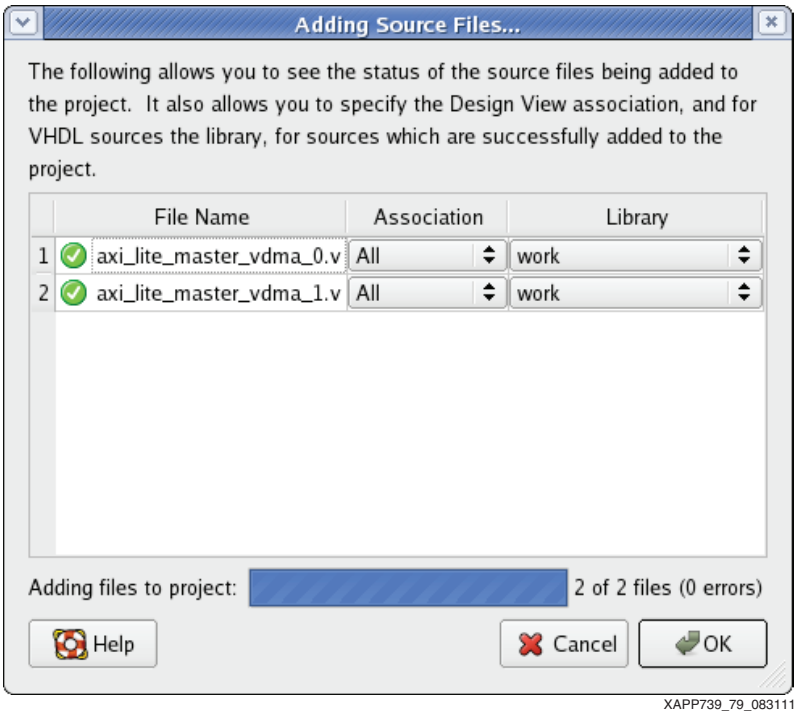



*Figure 79:* **New Source File Association Selection**

75. In the Hierarchy window pane of Project Navigator, select **axi_lite_master_vdma_0**. Then, in the Processes window pane, double-click **Design Utilities** > **View HDL Instantiation Template** (Figure 80).

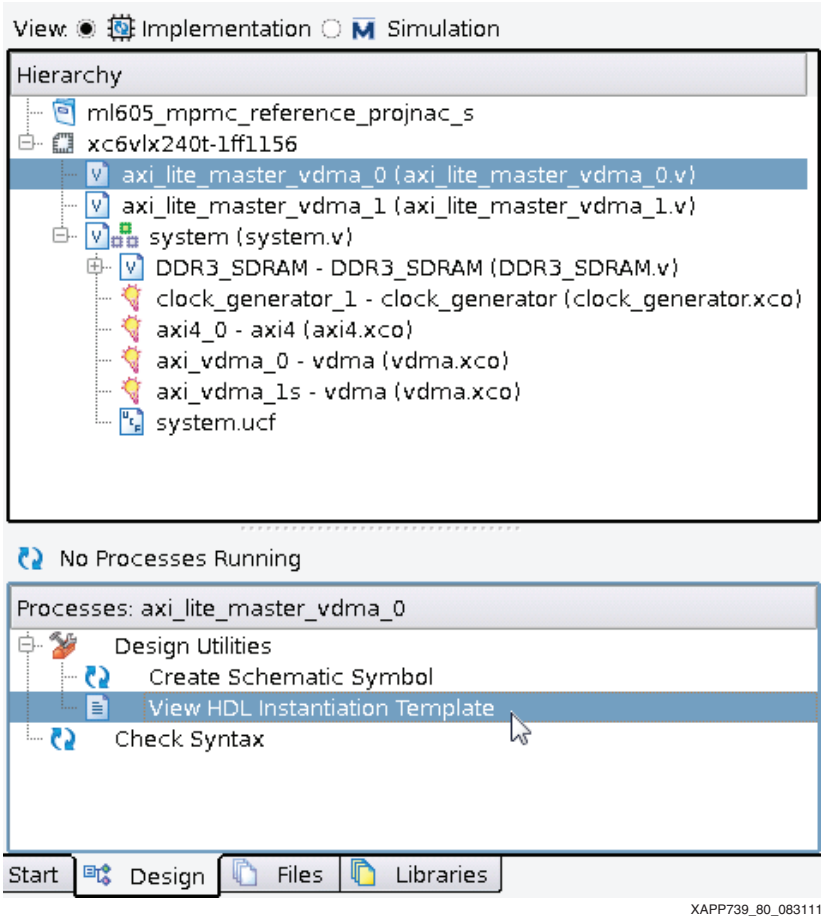



*Figure 80:* **Viewing a CORE Generator IP Instantiation Template From Project Navigator (axi_lite_master_vdma_0)**

76. In the Project Navigator editor window, copy the instantiation template for axi_lite_master_vdma_0 and place it into the `<user_dir>/system.v` file. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead.

77. Repeat the process for the other AXI4-Lite master. In the Hierarchy window pane of Project Navigator, select **axi_lite_master_vdma_1**. Then, in the Processes window pane, select **Design Utilities** > **View HDL Instantiation Template** (Figure 81).

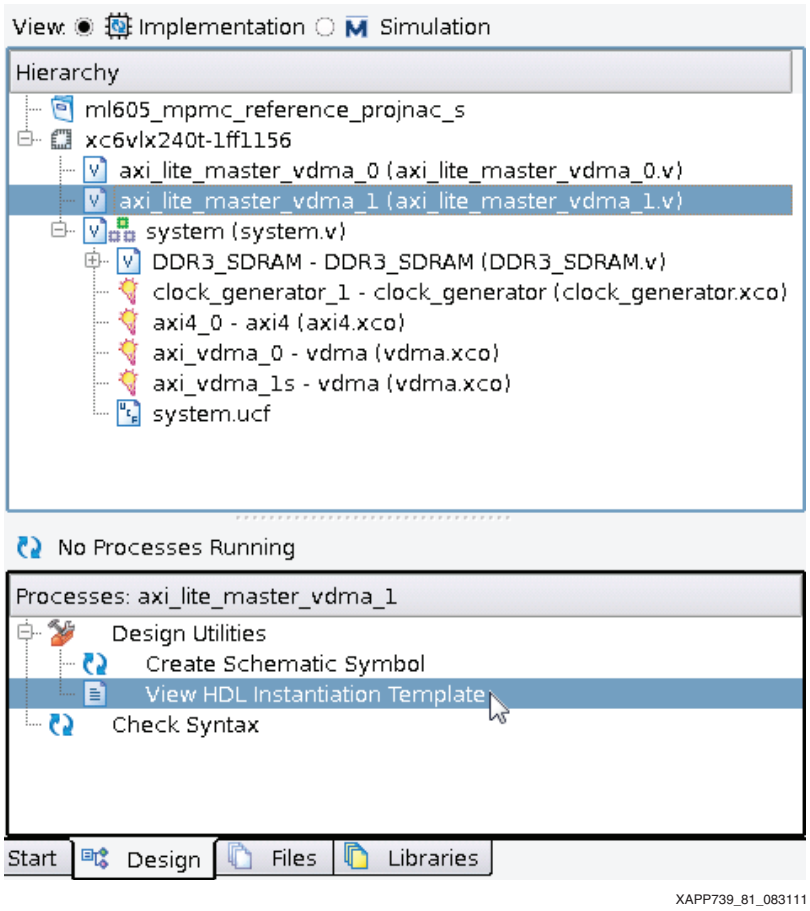



*Figure 81:* **Viewing a CORE Generator Tool IP Instantiation Template From Project Navigator (axi_lite_master_vdma_1)**

78. In the Project Navigator editor window, copy the instantiation template for axi_lite_master_vdma_1 and place it into the `<user_dir>/system.v` file. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead.
79. To edit `system.v` to make the connections to the axi_lite masters, edit the `<user_dir>/system.v` file. Define HDL wires and connect them as shown in Table 5.

*Table 5:* **Connections in `system.v` with axi_lite Masters Added**

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| Clock Generator | CLK_OUT1 (75 MHz) | axi_lite_master_vdma_* | M_AXI_ACLK |
| AXI Interconnect | S00_AXI_ARESET_OUT_N | axi_lite_master_vdma_0 | M_AXI_ARESETN |
| AXI Interconnect | S02_AXI_ARESET_OUT_N | axi_lite_master_vdma_1 | M_AXI_ARESETN |
| axi_lite_master_vdma_0 | M_AXI_* | AXI_VDMA_0 | s_axi_lite_* |
| axi_lite_master_vdma_1 | M_AXI_* | AXI_VDMA_1 | s_axi_lite_* |
| MIG | phy_init_done | axi_lite_master_vdma_* | DDRX_PHY_INIT_DONE |

## Adding the AXI4-Stream Test Pattern Generator

The AXI4-Stream source driving AXI VDMA 0 is a video test pattern generator with a timebase generator (to generate video frame sync signals). The TPG block is delivered as a fixed netlist. The fixed netlist is configured to drive a 1280x720p, 60 Hz video pattern into AXI VDMA 0 using an AXI4-Stream protocol.

In this design, the AXI TPG transfers 24-bit RGB data padded to the 32-bit wide datapath of the AXI VDMA. The AXI VDMA and AXI MPMC datapaths are also capable of transferring full 32-bit RGBA and 32-bit YUVA data.

80. Click **Project** > **Add Source** to add the source code for the AXI4-Stream TPG master to the system (Figure 82).

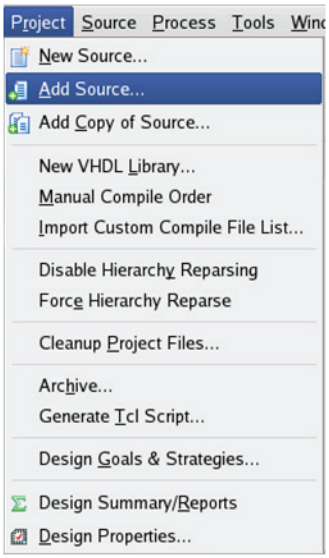



*Figure 82:* **Adding Source Files to Project Navigator**

81. In the Add Source window (Figure 83), browse to the `<user_dir>/ipcore_example/` directory and add the files `axis_master_example_0_wrapper_synth.v`, `axis_master_example.ngc`, and `axis_master_example.v`. Then click **Open**.

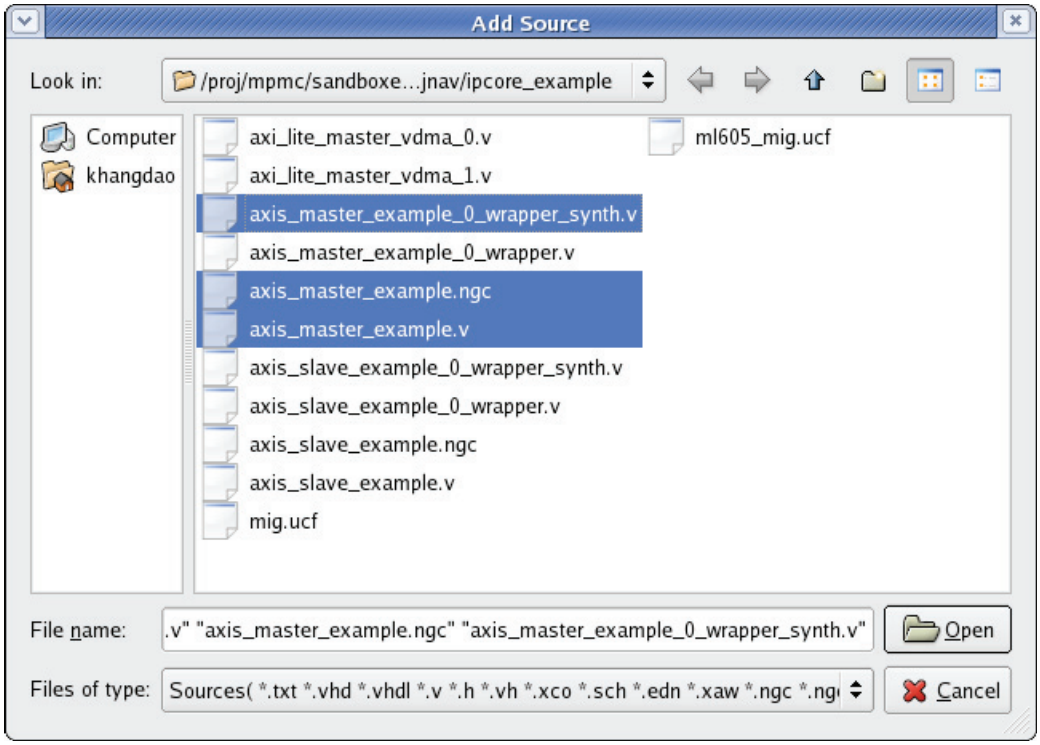



*Figure 83:* **Browser to Add Source Files**

82. Associate these files as shown below and click **OK**. This associates the source code for simulation and synthesis as needed. The files `axis_master_example_0_wrapper_synth.v` and `axis_master_example.ngc` define an HDL black-box definition and netlist for the core whereas `axis_master_example.v` is a netlist-based HDL simulation model of the IP (Figure 84).

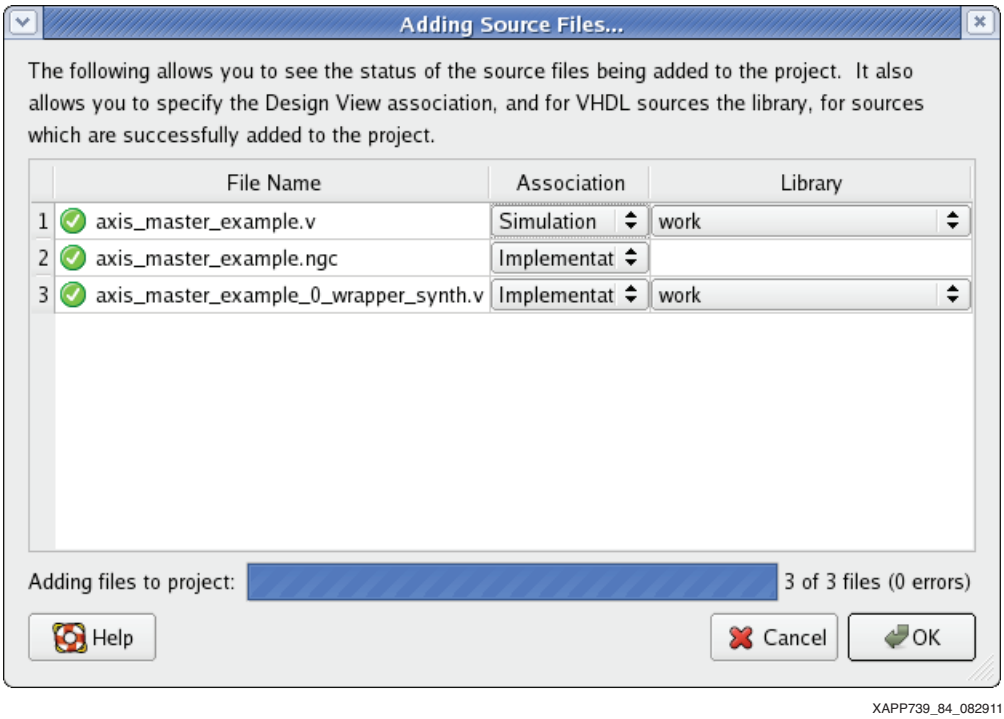



*Figure 84:* **New Source File Association Selection**

83. In the Hierarchy window pane of Project Navigator, select **axis_master_example**. Then, in the Processes window pane, double-click **Design Utilities** > **View HDL Instantiation Template** (Figure 85).

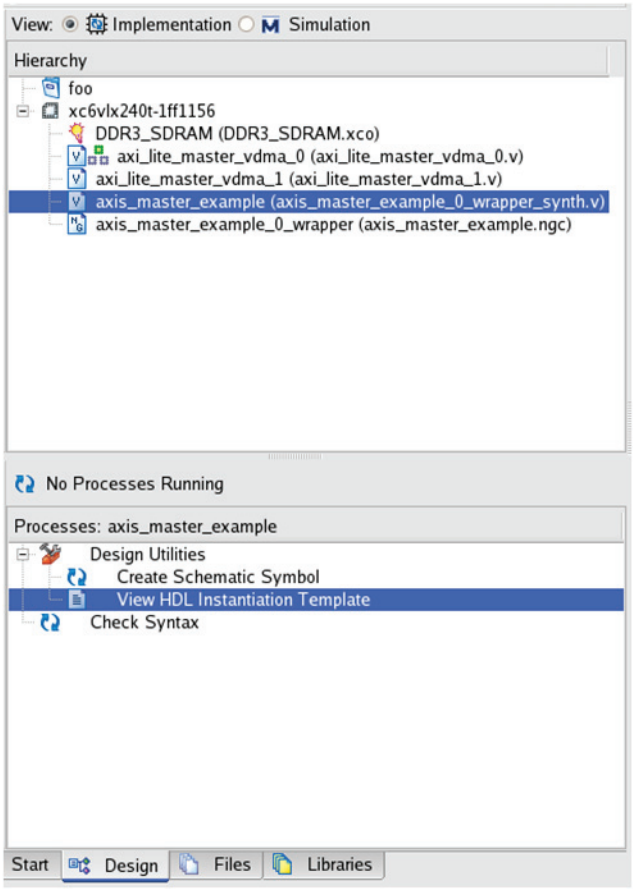



*Figure 85:* **Viewing a CORE Generator IP Instantiation Template From Project Navigator (axis_master_example)**

84. In the Project Navigator editor window, copy the instantiation template for axis_master_example and place it into the `<user_dir>/system.v` file. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead. To edit `system.v` to make the connections to the AXI4-Stream TPG master, edit the `<user_dir>/system.v` file. Define HDL wires and connect them as shown in Table 6.

*Table 6:* **Connections in `system.v` with AXI4-Stream TPG Masters Added**

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| AXI4-Stream TPG (axis_master_example) | m_axis_* (except m_axis_aresetn) | AXI_VDMA_0 | s_axis_s2mm |
| AXI_VDMA_0 | s2mm_prmry_reset_out_n | AXI4-Stream TPG (axis_master_example) | m_axis_aresetn |
| Clock Generator | CLK_OUT1 (75 MHz) | AXI4-Stream TPG (axis_master_example) | aclk |
| AXI4-Stream TPG (axis_master_example) | fsync_o | AXI_VDMA_0 | s2mm_fsync |
| MIG | phy_init_done | AXI4-Stream TPG (axis_master_example) | DDRX_PHY_INIT_DONE |

## Adding the AXI4-Stream DVI Display Controller

AXI VDMA 1 drives an output AXI4-Stream into a DVI Display Controller with a timebase generator (to generate video frame sync signals). The DVI Display Controller block also includes an IIC driver to configure the Chrontel CH7301 video chip on the ML605 board. This IP block is delivered as a fixed netlist. The fixed netlist is configured to display a 1280x720p, 60 Hz video signal on the ML605 board using an AXI4-Stream protocol from the AXI VDMA 1.

85. Click **Project** > **Add Source** to add the source code for the AXI4-Stream DVI master to the system (Figure 86).

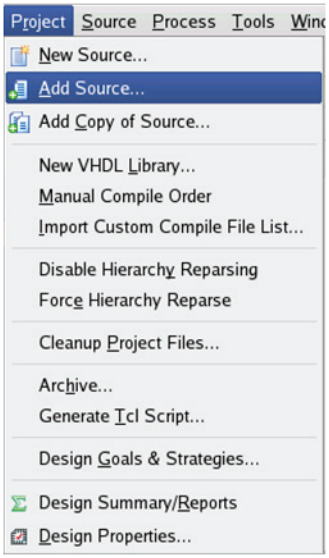



*Figure 86:* **Adding Source Files to Project Navigator**

86. In the Add Source window, browse to the `<user_dir>/ipcore_example/` directory and add the files `axis_slave_example_0_wrapper_synth.v`, `axis_slave_example.ngc`, and `axis_slave_example.v`. Then click **Open** (Figure 87).

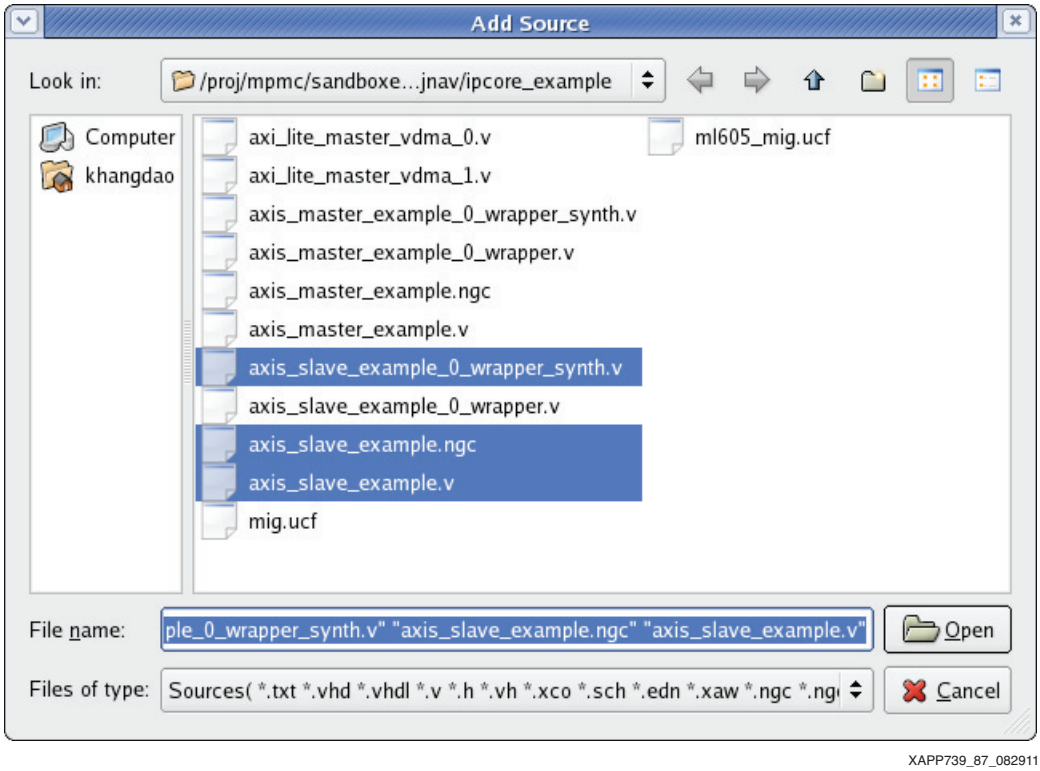



*Figure 87:* **Browser to Add source Files**

87. Associate the files as shown in Figure 88 and click **OK**. This associates the source code for simulation and synthesis as needed. The files `axis_slave_example_0_wrapper_synth.v` and `axis_slave_example.ngc` define an HDL black-box definition and netlist for the core whereas `axis_slave_example.v` is a netlist-based HDL simulation model of the IP.

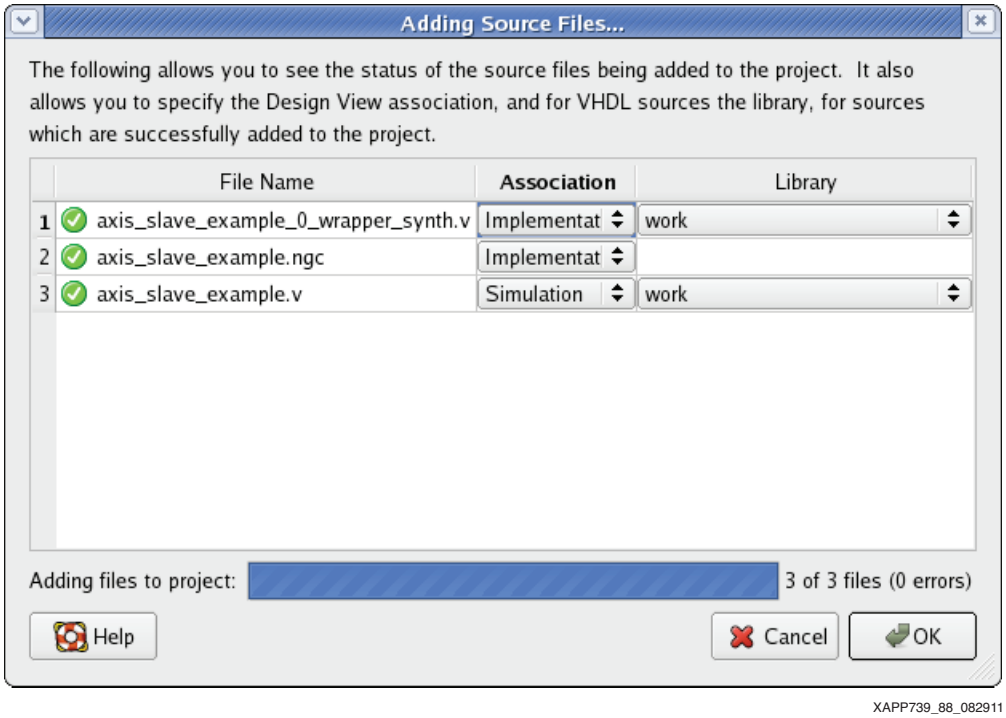



*Figure 88:* **New Source File Association Selection**

88. In the Hierarchy window pane in Project Navigator, select **axis_slave_example**. Then, in the Processes window pane, double-click **Design Utilities** > **View HDL Instantiation Template** (Figure 89).

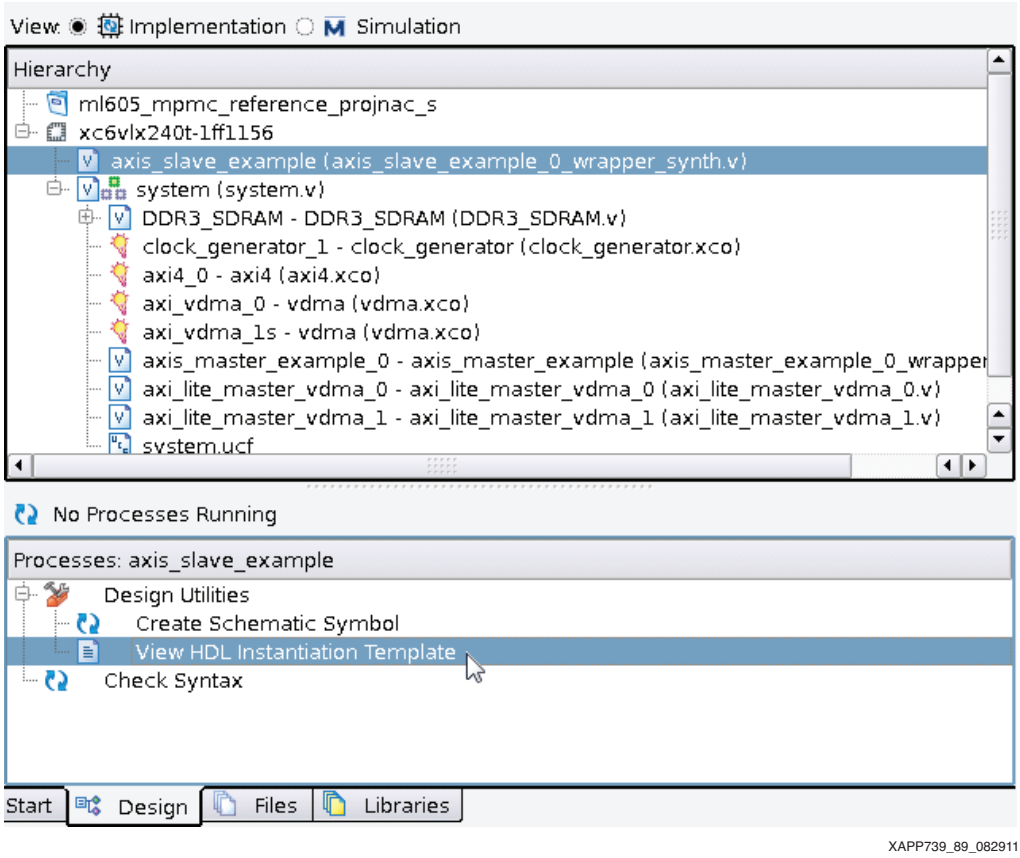



*Figure 89:* **Viewing a CORE Generator IP Instantiation Template From Project Navigator (axis_slave_example)**

89. In the Project Navigator editor window, copy the instantiation template for axis_slave_example and place it into the `<user_dir>/system.v` file. To save time, the completed `<design_dir>/projnav/system.v` file can be used instead. To edit `system.v` to make the connections to the AXI4-Stream DVI Display Controller slave, edit the `<user_dir>/system.v` file. Define HDL wires and connect them as shown in Table 7.

*Table 7:* **Connections in `system.v` with AXI4-Stream DVI Display Controller Slave Added**

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| AXI_VDMA_1 | m_axis_mm2s_* | AXI4-Stream DVI (axis_slave_example) | m_axis_mm2s_* |
| AXI_VDMA_1 | s2mm_prmry_reset_out_n | AXI4-Stream DVI (axis_slave_example) | aresetn |
| Clock Generator | CLK_OUT1 (75 MHz) | AXI4-Stream DVI (axis_slave_example) | aclk |
| AXI4-Stream DVI (axis_slave_example) | fsync_o | AXI_VDMA_* | mm2s_fsync |
| AXI4-Stream DVI (axis_slave_example) | fsync_o | AXI_VDMA_1 | s2mm_fsync |

*Table 7:* **Connections in `system.v` with AXI4-Stream DVI Display Controller Slave Added *(Cont'd)***

| From | | To | |
|---|---|---|---|
| **IP Core** | **Port Name** | **IP Core** | **Port Name** |
| MIG | phy_init_done | AXI4-Stream DVI (axis_slave_example) | DDRX_PHY_INIT_DONE |
| AXI4-Stream DVI (axis_slave_example) | CHRONTEL_INIT_DONE (gates start of TPG and axi_lite masters until IIC operations complete) | AXI4-Stream TPG (axis_master_example) | CHRONTEL_INIT_DONE |
| AXI4-Stream DVI (axis_slave_example) | CHRONTEL_INIT_DONE | axi_lite_master_vdma_* | CHRONTEL_INIT_DONE |
| AXI4-Stream DVI (axis_slave_example) | SDA/SCL* | Top Level I/O Port | Connect to video_out_scl and video_out_sda I/O port via manually instantiated IOBUF primitive. See IOBUF Connections for SDA/SCL in `system.v`, page 72 |
| AXI4-Stream DVI (axis_slave_example) | de, vsync, hsync, dvi_data, dvi_clk_p, dvi_clk_n | Top Level I/O Port | dvi_out_de, dvi_out_vsync, dvi_out_hsync, dvi_out_data, dvi_out_clk_p, dvi_out_clk_n, |
| N/A | N/A | Top Level I/O Port | dvi_out_reset_n (drive this output port to a 1 value). |

## IOBUF Connections for SDA/SCL in `system.v`

The IOBUF connections for SDA/SCL in `system.v` should look like this:

```
IOBUF
    scl_iobuf (
      .I ( SCL_O ),
      .IO ( video_out_scl ),
      .O ( SCL_I ),
      .T ( SCL_T )
    );

  IOBUF
    sda_iobuf (
      .I ( SDA_O ),
      .IO ( video_out_sda ),
      .O ( SDA_I ),
      .T ( SDA_T )
    );
```

## Reset

The MIG DDR3 controller contains its own clock generation and reset logic, and provides a reset output via ui_clk_sync_rst, which can be used to reset the rest of the system. The ui_clk_sync_rst should be connected to the reset input of the clock_generator instance. The LOCKED output should be used to reset the other cores that are clocked off the 75 MHz clock to ensure that a stable clock is available before operation. The AXI Interconnect provides synchronized reset signals to its masters. The AXI_VDMA provides resets to the AXI4-Stream master and slave example cores.

## System Connections

After the necessary system connections are made between the IP instances and the top-level I/O, the final `system.v` file can be rebuilt to a bitstream. Successfully completing the steps to

build the AXI MPMC design from a new project can be checked by building the design and running it in hardware according to the steps in Quick Start, page 2. If the build or hardware fails, the design should be compared to the original pre-generated design in `<design_dir>/projnav`.

The `system.v` files and `.xco` files in `<design_dir>/projnav/ipcore_dir` should be compared with the `.xco` files in `<user_dir>/ipcore_dir` to look for incorrectly configured cores. The `ipcore_dir/DDR3_SDRAM` directories should also be compared to check that the proper MIG file modifications were made.

***Note:*** If the design fails in synthesis due to an `unknown module <axi_vdma>` error, the workaround described in Quick Start, page 2 should be applied.

### Simulation Testbench

This design does not support simulation. Because this is video design with long frame times, simulations of multiple video frames would be prohibitive in a simulator.

## Equivalent XPS Design

For reference, an equivalent Xilinx Platform Studio (XPS) stand-alone design is also provided. This design has the same IP and overall functionality as the Project Navigator design but is delivered as a complete XPS design that can be built to a bitstream within the XPS tool. Figure 90 shows the block diagram of the XPS design. This design differs from the Project Navigator design by its use of proc_sys_reset and clock_generator blocks native to XPS and the extra AXI Interconnect blocks for the AXI4-Lite connections. The AXI Interconnect blocks connecting the AXI4-lite masters to each VDMA is required for XPS modeling of AXI connections, but is essentially a pass-through logically.

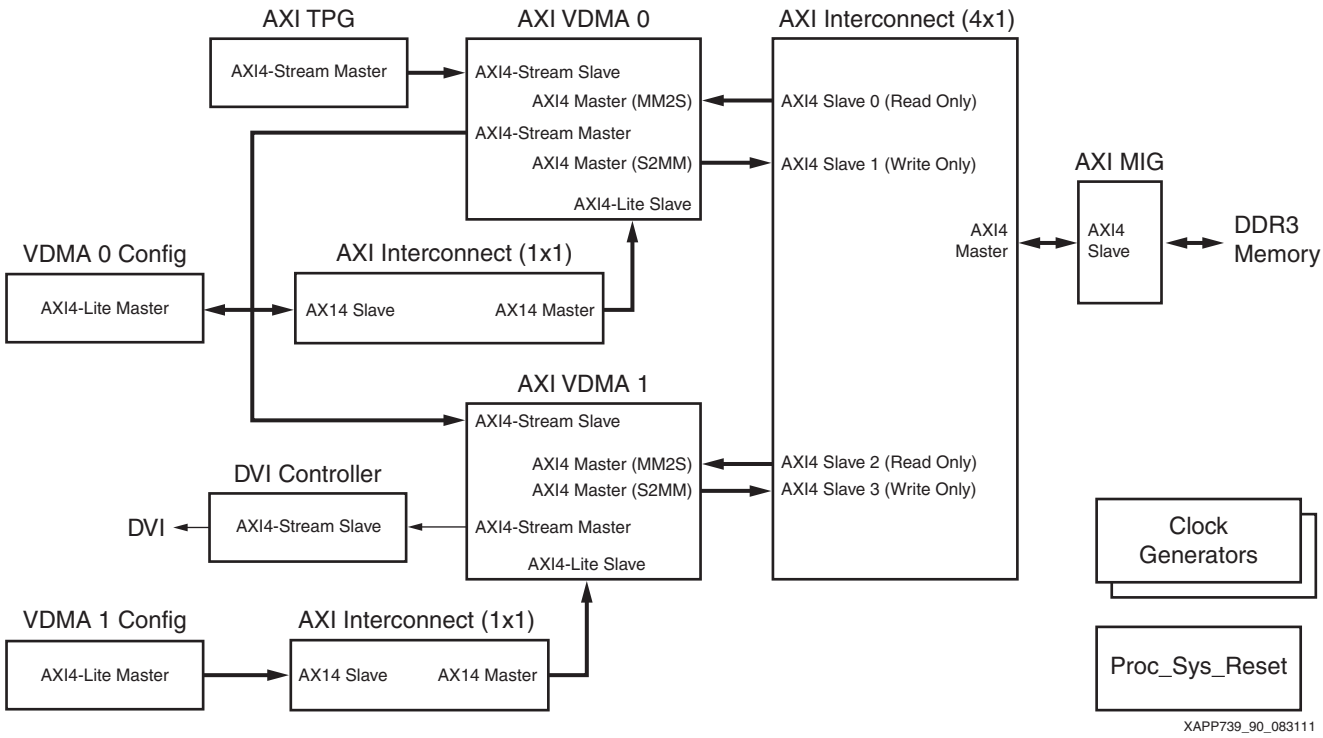



*Figure 90:* **Block Diagram Overview of AXI MPMC System (EDK)**

This design can be rebuilt using normal build flows in the XPS tools. See *EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design* [Ref 6] for more information about the XPS tools. The EDK project file is stored in `<design_dir>/edk/system.xmp`.

## Reference Design

The reference design files for this application note can be downloaded at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=175135

The reference design checklist is shown in Table 8.

*Table 8:* **Reference Design Matrix**

| Parameter | Description |
|---|---|
| **General** | |
| Developer Name | Xilinx |
| Target Devices (Stepping Level, ES, Production, Speed Grades) | Virtex-6 FPGA |
| Source Code Provided? | Yes (except two blocks delivered as a netlist) |
| Source Code Format | Verilog |
| Design Uses Code or IP from Existing Reference Design, Application Note, 3rd party, or CORE Generator™ Software? | Reference designs provided for Project Navigator and EDK |
| **Simulation** | |
| Functional Simulation Performed? | N/A (simulation not supported) |
| Timing Simulation Performed? | N/A (simulation not supported) |
| Testbench Provided for Functional and Timing Simulations? | N/A (simulation not supported) |
| Testbench Format | N/A (simulation not supported) |
| Simulator Software and Version | N/A (simulation not supported) |
| SPICE/IBIS Simulations? | N/A (simulation not supported) |
| **Implementation** | |
| Synthesis Software Tools and Version | XST 13.2 |
| Implementation Software Tools and Version | ISE Design Suite 13.2 or higher |
| Static Timing Analysis Performed? | Yes (passing timing in PAR/TRCE) |
| **Hardware Verification** | |
| Hardware Verified? | Yes |
| Hardware Platform Used for Verification | ML605 board |

The resource utilization and clock frequency of the system are summarized in Table 9. The system is designed to fit the FPGA resources and speed grade of the XC6VLX240T-FF1156-1 FPGA on the ML605 board. It has not been characterized for other FPGA devices or speed grades.

***Note:*** The AXI MPMC Interconnect and MIG tool are configured for high performance. The system is not area optimized, but is optimized for throughput. The AXI MPMC portion of the system has more available throughput than the two AXI VDMA masters can consume. The extra available throughput can be used for expansion of the system to include additional AXI4 masters. See the *AXI Reference Guide* [Ref 2] for more information about optimizing the AXI MPMC for different trade-offs of area, timing, throughput, latency, and ease of use.

*Table 9:* **Device Utilization**

| Parameters | Specification/Details | |
|---|---|---|
| Maximum Frequency (by speed grade) | -1 | 200 MHz |
| Device Utilization | Slices | 8,650 |
| | Slice LUTs | 18,375 |
| | Slice Registers | 25,445 |
| | GCLK Buffers | 4 |
| | Block RAMs | 28 |
| HDL Language Support | | Verilog |
| DDR3 Memory Configuration | | 64-Bit, 400 MHz DDR3-SDRAM |
| Video Clock Frequency | | 75 MHz |
| AXI Interconnect and MIG Main Clocking | | 200 MHz |

Device resource utilization is detailed in Table 10 for each IP core in Figure 1.

***Note:*** The information in Table 10 is taken from the **Design Summary** Tab in Project Navigator under the **Design Overview** > **Module Level Utilization** report selection. This information is only generated when the Map process property option Generate Detailed MAP Report (-detail) is enabled. The utilization information is approximate due to cross-boundary logic optimizations and logic sharing between modules. Slices can be packed with basic elements from multiple IP cores and hierarchies. Therefore, a slice is counted in every hierarchical module that each of its packed basic elements belong to. This results in some double counting of slice counts when adding up the slice counts across modules.

*Table 10:* **Module Level Resource Utilization**

| IP Core | Instance Name | Slices | Slice LUTs | Slice Registers | GCLK Buffers | Block RAMs |
|---|---|---|---|---|---|---|
| AXI MIG | DDR3_SDRAM | 4100 | 6605 | 7787 | 3 | 0 |
| AXI Interconnect | axi4_0 | 3712 | 5980 | 9943 | 0 | 18 |
| VDMA 0 Config | axi_lite_master_vdma_0 | 14 | 26 | 14 | 0 | 0 |
| VDMA 1 Config | axi_lite_master_vdma_1 | 14 | 26 | 14 | 0 | 0 |
| AXI VDMA 0 | axi_vdma_0 | 1356 | 2222 | 2955 | 0 | 4 |
| AXI VMDA 1 | axi_vdma_1 | 1344 | 2190 | 2929 | 0 | 4 |
| AXI TPG | axis_master_example_0 | 429 | 903 | 1223 | 0 | 2 |
| DVI Controller | axis_slave_example_0 | 193 | 410 | 524 | 0 | 0 |
| Clock Generator | clock_generator_1 | 0 | 0 | 0 | 1 | 0 |

## Conclusion

This application note describes the steps to build a high-performance AXI MPMC (MIG and AXI Interconnect) system using the Project Navigator tool. It illustrates the tool steps and design considerations necessary to create a working system in hardware. The AXI MPMC system is exercised by AXI VDMA IP cores moving video frames through DDR3 memory from a test pattern generator IP block to a DVI Display IP block. This system can be used as a template and training information for a new design.

## References

This document uses the following references:

1. UG406, *Virtex-6 FPGA Memory Interface Solutions User Guide*
2. UG761, *AXI Reference Guide*

3. DS768, *LogiCORE IP AXI Interconnect Datasheet*
4. ISE Design Suite (IDS) 13.2 Documentation
http://www.xilinx.com/support/documentation/dt_ise13-2.htm
5. Virtex-6 FPGA ML605 Evaluation Kit
http://www.xilinx.com/products/boards/ml605/reference_designs.htm
6. UG683, *EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design*
7. Advanced Microcontroller Bus Architecture (AMBA) ARM AXI4 specifications
http://www.amba.com

## Revision History

The following table shows the revision history for this document.

| Date | Version | Description of Revisions |
|---|---|---|
| 09/23/11 | 1.0 | Initial Xilinx release. |

## Notice of Disclaimer