LatticeMico32 DDR SDRAM Demonstration

Lattice Semiconductor Corporation
5555 NE Moore Court
Hillsboro, OR 97124
(503) 268-8000

September 2007

Copyright

Trademarks

Disclaimers

to correct any errors contained herein or to advise any user of this document of any correction if such be made. LSC recommends its customers obtain the latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
`.` `.` `.`	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
`|`	A choice between items in syntax descriptions.

Contents

1

LatticeMico32 DDR SDRAM Demonstration

Introduction

This document describes the LatticeMico32 DDR SDRAM IP demonstration for the LatticeMico32 Development Board for LatticeECP FPGA devices. This demonstration is designed to be simple and easy to use, requiring no lengthy setup or complex explanation. It demonstrates the following features of the LatticeMico32 platform design on the LatticeMico32 Development Board for the LatticeECP FPGA:

- Co-locating the FPGA configuration bitstream, as well as a LatticeMico32 software application binary, in a single SPI configuration flash
- Deploying a simple software application in the configuration SPI flash, booting the LatticeMico32 processor from the SPI flash, and copying the application to an external DDR SDRAM memory

The demonstration focuses on the LatticeMico32 Development Board's VGA interface, with a DDR SDRAM module used for executing code and serving as video memory. The LatticeMico32 processor, a WISHBONE master, writes to this video memory, and an incorporated VGA controller, also a WISHBONE master, reads from this video memory. The VGA controller is implemented in VHDL language, and this demonstration shows how to incorporate VHDL custom components into the Verilog-language LatticeMico32 platform framework.

The demonstration provides the necessary ispLEVER design projects and step-by-step instructions on the following:

- Preparing the LatticeMico32 software application for SPI flash deployment
- Generating an SPI flash image containing the FPGA bitstream and the LatticeMico32 software application that can be programmed into the SPI flash using ispVM System

Since the VGA controller is a VHDL design, the demonstration also provides an example of integrating a VHDL WISHBONE-compliant component into the LatticeMico32 platform framework.

For additional details on the steps used in this demonstration for achieving SPI flash deployment, refer to the *LatticeMico32 Software Developer User's Guide*.

Prerequisites

The demonstration assumes that you are familiar with the LatticeMico32 System flow and usage. It also assumes that you are well-versed in debugging, downloading and deploying LatticeMico32 software applications. If you are not familiar with LatticeMico32 System, it is recommended that you complete the *LatticeMico32 Tutorial*.

You should be able to download designs and software applications to the LatticeMico32 Development Board for LatticeECP.

You must have a basic understanding of VGA parameters, such as H-Sync, V-Sync, and the various porches that determine VGA timing.

This demonstration requires a complete installation of Lattice Semiconductor ispLEVER Version 7.0 SP1 software and a complete installation of the LatticeMico32 System Version 7.0 SP1 software.

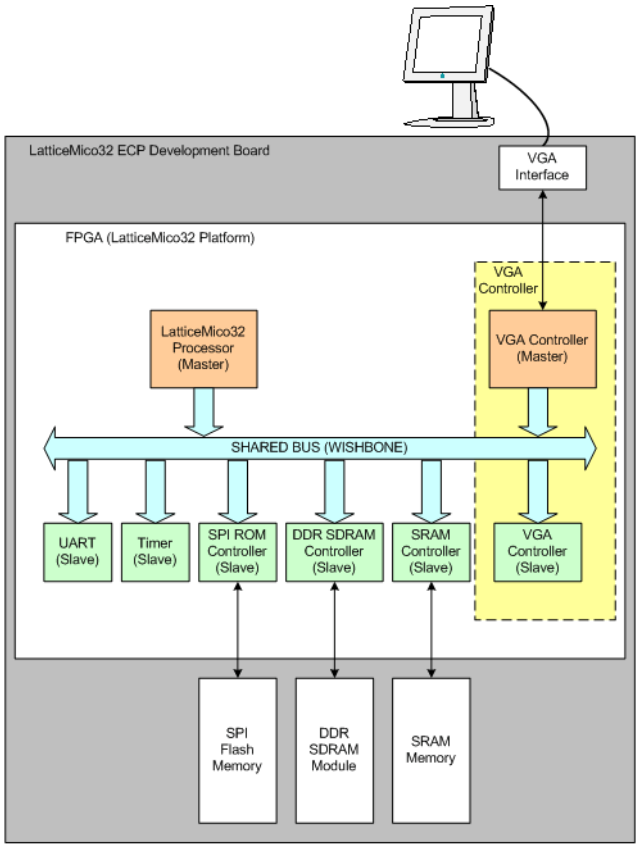
DDR SDRAM Memory Module Requirements

The DDR SDRAM memory module used for this demonstration is a Corsair VS512SDS266 200-pin SODIMM 512-megabyte PC2100 speed memory module. If you are using a different memory module, you must configure the DDR SDRAM IP accordingly.

Design Overview

The demonstration runs on a LatticeMico32 Development Board for LatticeECP. You must have a computer monitor (LCD preferred) connected to the VGA connector of the LatticeMico32 Development Board. Figure 1 shows the components used in the demonstration.

Figure 1: LatticeMico32 DDR SDRAM Demonstration Overview



The platform design objectives are as follows:

- Co-locate the FPGA bitstream and the software application in the SPI configuration flash.
- Prepare the processor so that it executes the boot code stored in the SPI flash.
- Have the boot code copy the main executable into an external DDR SDRAM module.

- Have the processor execute the code from the external DDR SDRAM, once the copy process is done.

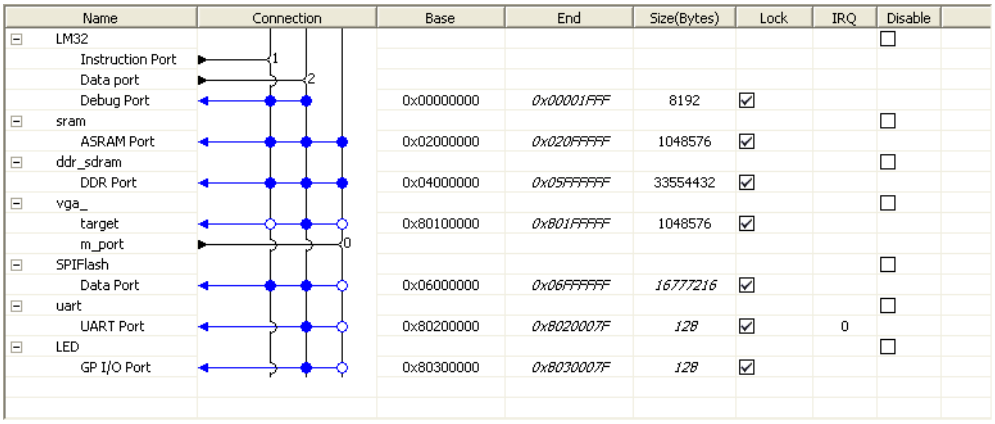
The demonstration partitions the DDR SDRAM so that part of it is used for video while another part executes the code.

To provide a meaningful software application, the software design goal is to write a software application executed by the LatticeMico32 processor that updates the contents of the video memory while the VGA master interface reads the video memory contents, as dictated by the VGA timing parameters.

Platform Design

Figure 2 shows the LatticeMico32 platform design.

Figure 2: LatticeMico32 Platform



The UART in the platform allows you to use printf statements in the software design, and the LED allows you to visually inspect the software execution.

The vga_ component is an instance of a master passthrough custom component, which is explained in “Hardware” on page 39. The main purpose of this component is to provide a passthrough for a WISHBONE master interface to connect to a WISHBONE master component that is not part of the platform generated in the MSB perspective. It also has a slave interface that allows passing arbitrated access to a WISHBONE slave interface of a WISHBONE slave component that is not part of the platform generated in the MSB perspective.

Reset Vector Address (EBA Value) Selection

The SPI flash ROM component, SPIFlash, in the platform is connected to the external configuration SPI flash. It is essential to select a proper reset vector address (also known as the EBA value) for the LatticeMico32 processor for SPI flash deployment.

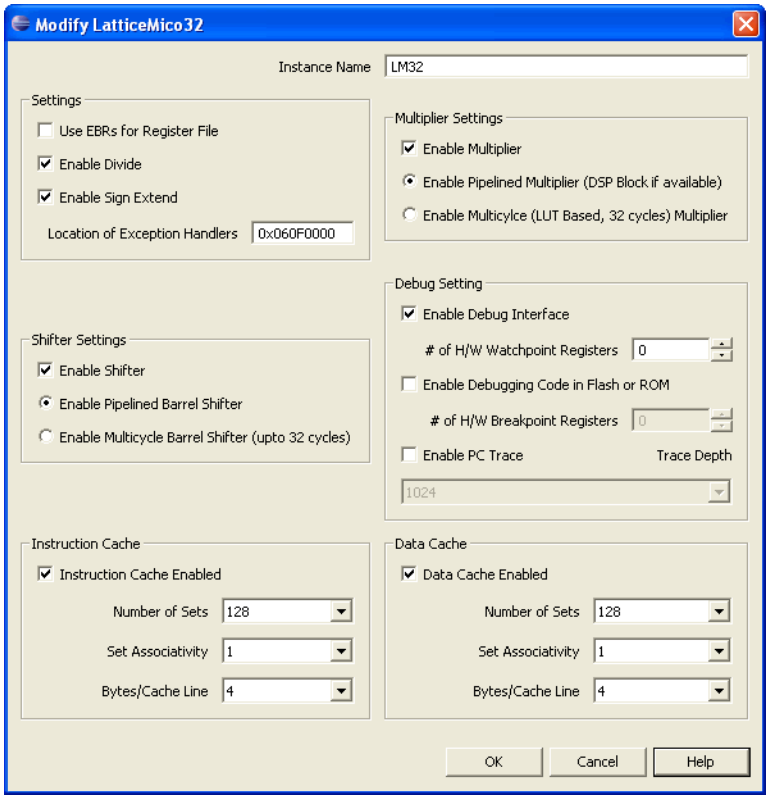
The LatticeMico32 Development Board for the LatticeECP device has an 8-megabit SPI flash. In the Universal File Writer tool, which is part of the ispVM System tool suite, you can visualize this SPI flash as a PROM with 16 64-kilobyte sectors. For the LFECP33E device, which is the LatticeECP FPGA on

the LatticeMico32 Development Board, the uncompressed FPGA bitstream consumes 15 64-kilobyte sectors, although it does not fully consume the 15th sector. Therefore, one 64-kilobyte sector is available for storing the bootable software application. It translates to an offset address of 0xF0000 in the SPI flash device.

Since the MSB has assigned the SPI flash ROM component a base address value of 0x06000000, the processor reset vector address (also known as the EBA value) should be set to 0x060F0000, that is, offset 0xF0000 in the SPI flash ROM component. In addition, the first instruction of the bootable software must be located at offset 0xF0000 in the SPI flash when the demonstration constructs a flash image that consists of the FPGA bitstream and the bootable software.

Figure 3 shows the processor configuration for this platform, with the EBA value set to 0x060F0000.

Figure 3: Modify LatticeMico32 Dialog Box



As shown for this demonstration platform, it is always a good idea to lock all assigned addresses to prevent MSB from reassigning different addresses when regenerating the platform.

Limitations

The VGA output is limited to 6-bit color (2 bits per color component) and does not use a video DAC.

Installing the Demonstration

This demonstration requires a complete installation of the LatticeMico32 System software Version 7.0 SP1, Lattice Semiconductor ispLEVER Version 7.0 SP1 design tools, and ispVM System programming tools.

Lattice Semiconductor releases the demonstration package in a zipped file. This demonstration package contains all the necessary files, including the final bitstream, the application executable, and an Intel Hex file that contains the merged bitstream and the bootable application executable. The projects used for creating these outputs are also provided as part of the zipped package.

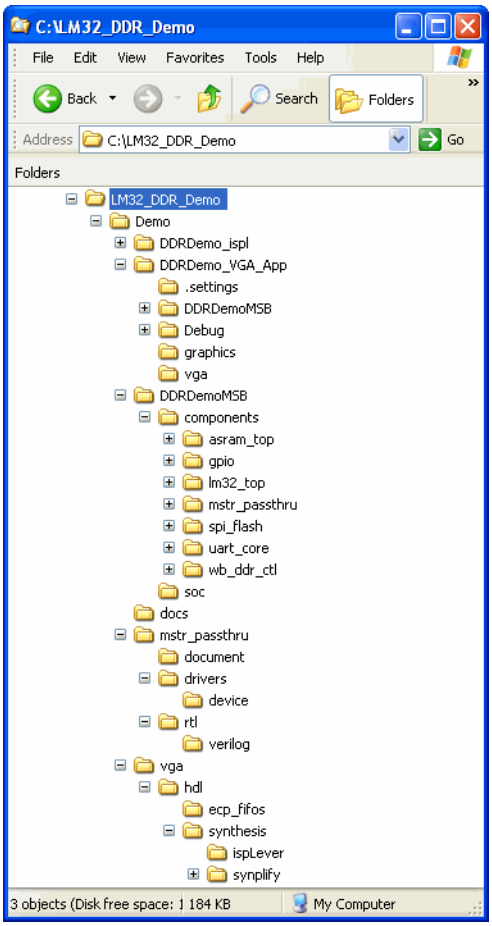
This document provides step-by-step instructions for generating the pre-packaged outputs, since that is one of the goals of this demonstration.

Installation

The LatticeMico32 DDR SDRAM demonstration is distributed in a zipped file. Unzip this file with c:\ as the root directory.

Figure 4 displays the directory hierarchy.

Figure 4: Directory Structure



The contents of the directory structure are as follows:

- Demo/DDRDemo_ispl – Contains the pre-built bitstream, as well as the ispLEVER project used for building the bitstream.
- Demo/DDRDemo_VGA_App – Contains the LatticeMico32 application that writes to the video memory (external SRAM).
- Demo/DDRDemoMSB – Contains the demonstration LatticeMico32 platform.
- Demo/docs – Contains this document.
- Demo/mstr_passthru – Contains the master passthrough custom component that provides connectivity to the VGA controller. Since the VGA controller is a VHDL implementation, it cannot be directly instantiated in the platform.

- Demo/VGA – Contains the VHDL language source files for the VGA controller.
- Demo/VGA/hdl/ecp_fifos – Contains FIFOs used for the VGA controller.
- Demo/VGA/hdl/synthesis/synplify – Contains the Synplify project for generating VGA EDIF file.
- Demo/VGA/hdl/synthesis/ispLever – Contains the ispLEVER project that generates the VGA controller .ngo file (video_chain_top.ngo) from the EDIF file generated by the Synplify project.

Environment Setup

To run the demonstration on the LatticeMico32 Development Board for functionality verification, you must have the following two pieces:

- A pre-built FPGA bitstream containing the LatticeMico32 platform, which requires no modifications
- A C demonstration software project that you must import into C/C++ SPE

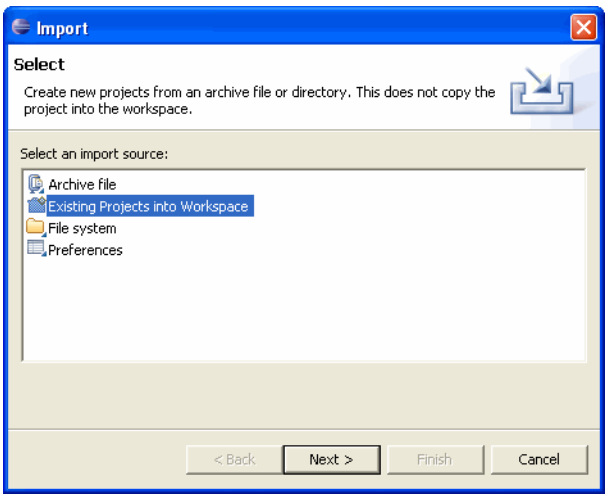
You must import the included C application project into the LatticeMico32 C/C++ SPE environment. Follow the steps outlined in this section to import the demonstration software project, if you are not familiar with importing projects. It is assumed that you have unzipped the demonstration package with c:\ as the root directory.

Note

Importing an existing C/C++ SPE project does not copy the contents into the workspace. Any modification that you make to the application project occurs in your demonstration installation.

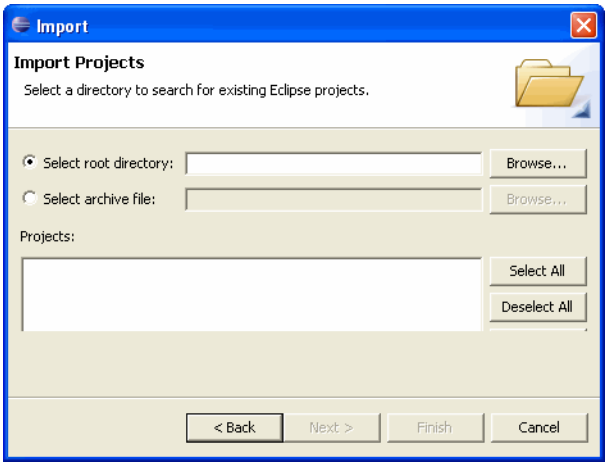
1. Start the LatticeMico32 system and switch to the C/C++ SPE perspective.
2. Choose **File > Import** to activate the Select dialog box, shown in Figure 5.

Figure 5: Select Dialog Box



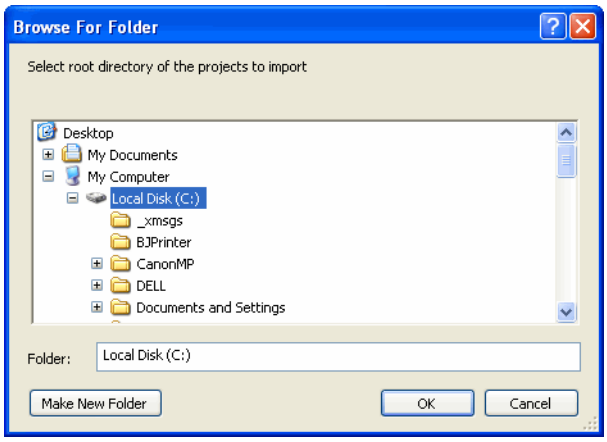
3. Select **Existing Projects into Workspace**.
4. Click **Next** to activate the Import Projects dialog box, shown in Figure 6.

Figure 6: Import Projects Dialog Box



5. Select **Select root directory**, if it is not already selected.
6. Click the **Browse** button next to Select root directory to activate the Browse For Folder dialog box, shown in Figure 7.

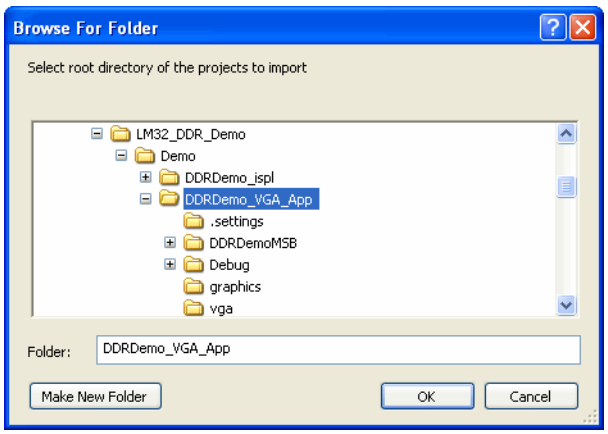
Figure 7: Browse For Folder Dialog Box



7. In the Browse For Folder dialog box, browse to the c:\LM32_DDR_Demo\Demo\DDRDemo_VGA_App directory, as shown in Figure 8.

The figure assumes that the demonstration was installed with c:\ as the root directory.

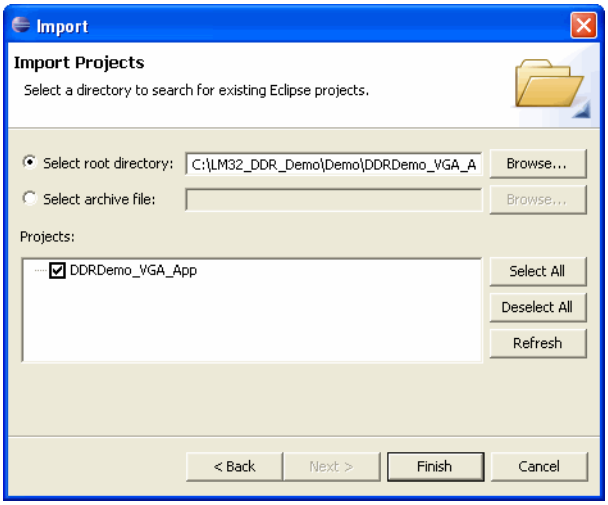
Figure 8: Selecting DDRDemo_VGA_App Software Project Root Directory



8. Click **OK**.

   The Import Project dialog box now shows DDRDemo_VGA_App as an available project in the Projects pane, as shown in Figure 9. If you do not see any listing in the Projects pane, you have not selected the directory correctly in the previous step.

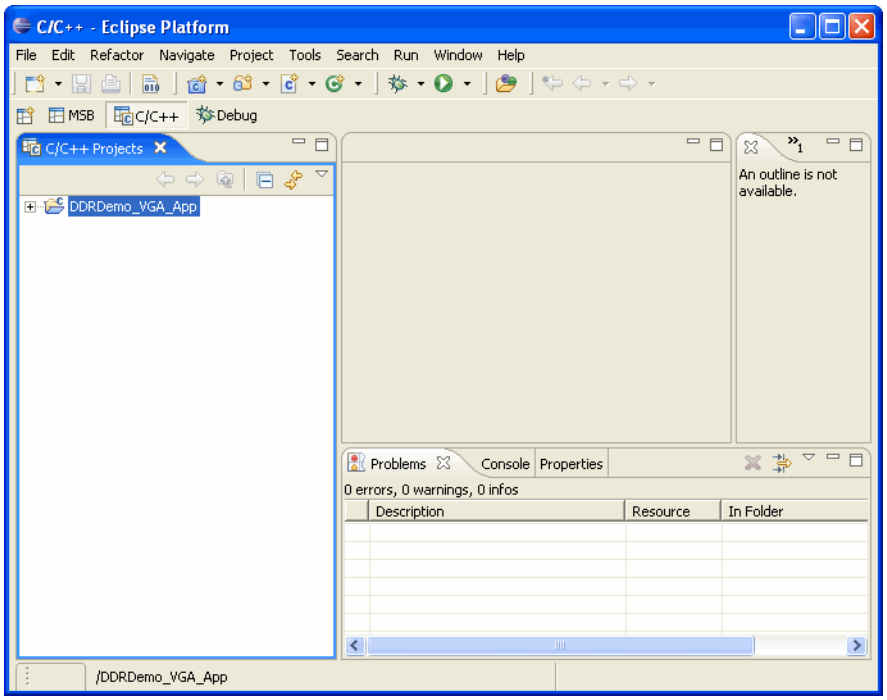
Figure 9: DDR VGA Project Available for Import



9. Click **Finish**.

The C/C++ Projects view in the C/C++ SPE perspective now lists the DDRDemo_VGA_App project, as shown in Figure 10.

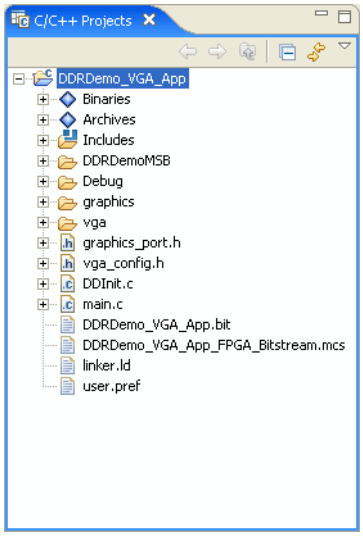
Figure 10: DDRDemo_VGA_App Listed in C/C++ Projects View



10. Expand the project listed in C/C++ SPE to show its contents.

    Figure 11 shows an overview of the project contents.

Figure 11: Project Contents Overview



The main files in this project are the following:

- DDRDemo_VGA_App.bit, which is the binary executable of the project, with an attached boot copier. You will re-generate this file, using step-by-step instructions as part of the main demonstration.
- DDRDemo_VGA_App_FPGA_Bitstream.mcs, which is an Intel-Hex-format file containing the SPI flash image. This image contains the FPGA bitstream merged with the bootable binary executable. This file can be programmed to the SPI flash for a stand-alone demonstration, using ispVM System.
- linker.ld, which is the custom linker file used by the project. It uses the managed-build linker file as its basis and is modified to use only the first 25 percent of the available DDR SDRAM memory.
- vga_config.h, which is the C header file containing the VGA controller parameters.

11. Rebuild the imported project, DDRDemo_VGA_App, to verify a successful build.

    The project is configured to use a custom linker script, and the standard input and output is configured to use the UART. “Design Details” on page 37 describes the project contents.

You have now successfully imported the software application provided.

The Demonstration

The demonstration comprises four separate procedures:

- Configuring VGA timing parameters and verifying the design functionality on the LatticeMico32 ECP Demonstration Board
- Generating a bootable binary image of the executable application
- Generating an Intel Hex file that contains the FPGA bitstream and the bootable application binary image
- Programming the SPI flash device

Configuring VGA Timing Parameters and Verifying the Design Functionality

Before continuing with this step, make sure that you have the LatticeMico32 ECP Demonstration Board powered up and that you have a computer monitor, preferably an LCD monitor, connected to the VGA connector of the LatticeMico32 ECP Demonstration Board.

Configuring the VGA Timing Parameters

The FPGA design is designed with a pixel clock of 50 megahertz and a resolution of 800 pixels x 600 lines.

However, the following frame and line-timing parameters are configurable. The following are the defaults:

- Vertical (frame) timing:
  - Vertical sync width: 7 lines
  - Front porch width: 1 line
  - Back porch width: 23 lines
- Horizontal (line) timing:
  - Horizontal sync width: 128 pixels
  - Front porch width: 40 pixels
  - Back porch width: 88 pixels

These timing parameters are defined in the vga_config.h header file, whose contents are shown in Figure 12.

Figure 12: Contents of the vga_config.h File

```
#ifndef VGA_CONFIG_H_
#define VGA_CONFIG_H_
#include "system_conf.h"

/*
 * DEBUG CONFIGURATION
 */

/* printf vga controller configuration after configuring it */
#define _DEBUG_PRINT_VGA_CONFIG_    (1)



/*
 * VGA VIDEO MEMORY BASE ADDRESS CONFIGURAITON
 */
#if 1
    /* USE SRAM AS VIDEO MEMORY */
#define VGA_VIDMEM_BADDR    (SRAM_BASE_ADDRESS)
#else
    /* USE DDR SDRAM AS VIDEO MEMORY */
#define VGA_VIDMEM_BADDR    (DDR_SDRAM_BASE_ADDRESS + (DDR_SDRAM_SIZE/2))
#endif


/*
 ********************************************************
 *                                                      *
 *          VGA CONTROLLER CONFIGURATION                *
 *                                                      *
 ********************************************************
 */

#define C_NUM_PIXELS_PER_LINE   (800)
#define C_NUM_DISPLAY_LINES     (600)


/* VGA FRAME PARAMETERS */
#define VGA_V_SYNC_WIDTH        (7)
#define VGA_V_FPORCH_WIDTH      (1)
#define VGA_V_BPORCH_WIDTH      (23)


/* VGA LINE PARAMETERS */
#define VGA_H_SYNC_WIDTH        (128)
#define VGA_H_FPORCH_WIDTH      (40)
#define VGA_H_BPORCH_WIDTH      (88)


#endif /*VGA_CONFIG_H_*/
```

The header file contains the following configuration parameters:

- _DEBUG_PRINT_VGA_CONFIG_ – The possible values of this parameter are 0 or 1. Setting it to 1 enables the printf statements in the code so that the application debugs the VGA configuration parameters over the RS-232 UART to the standard out. Setting this parameter to 0 disables the debug printf statements in the code.
- VGA_VIDMEM_BADDR – This parameter determines the start of the video memory used by the LatticeMico32 processor application, as well as the VGA controller core. This demonstration uses the DDR SDRAM as video memory, as well as program memory. If you want to use the SRAM

for video memory, set the value of this parameter to a valid SRAM address.

- C_NUM_PIXELS_PER_LINE – This parameter is used by the software and must not be modified.
- C_NUM_DISPLAY_LINES – This parameter is used by the software and must not be modified.
- VGA_V_SYNC_WIDTH – This parameter sets the vertical sync width (number of horizontal lines that form a vertical sync). You can modify this parameter.
- VGA_V_FPORCH_WIDTH – This parameter sets the front porch width for the vertical sync, in units of horizontal lines. You can modify this parameter.
- VGA_V_BPORCH_WIDTH – This parameter sets the back porch width for the vertical sync, in units of horizontal lines. You can modify this parameter.
- VGA_H_SYNC_WIDTH – This parameter sets the horizontal sync width in units of pixels. You can modify this parameter.
- VGA_H_FPORCH_WIDTH – This parameter sets the front porch width for the horizontal sync, in units of pixels. You can modify this parameter.
- VGA_H_BPORCH_WIDTH – This parameter sets the back porch width for the horizontal sync, in units of pixels. You can modify this parameter.

Once you modify this file, be sure to rebuild the project rather than build it. C/C++ SPE does not correctly identify dependent files when modifying header files. Once the rebuilding is complete, the generated executable contains code to configure the VGA controller with the appropriate timing information.

Verifying the Design Functionality

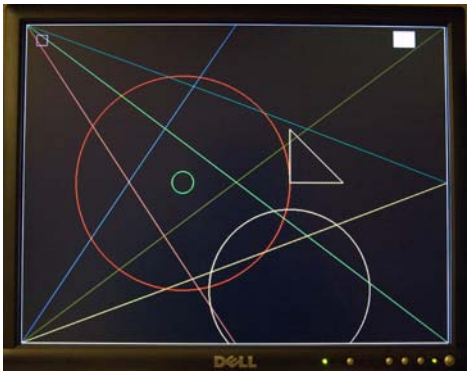
This section guides you through the process of running the demonstration to verify its functionality. It shows you how to debug the application, the RTL design, or both before deploying it to the SPI flash for stand-alone operation.

The pre-built FPGA bitstream, ddrdemo_ispl.bit, is located in the \LM32_DDR_Demo\Demo\DDRDemo_ispl directory. Download this bitstream to the FPGA by using ispVM System. When the FPGA bitstream has been successfully downloaded, the eight discrete LEDs appear solid red. The computer monitor is not active at this time.

Once the FPGA is configured, download the rebuilt software application, DDRDemo_VGA_App, and run it. When the software application is running, the states of the eight discrete LEDs change so that four LEDs remain solid and the other four LEDs turn off. The computer monitor at this time starts displaying the contents of the video memory that is actively being written to by the LatticeMico32 processor and is actively being read by the VGA controller core.

Figure 13 shows the display generated by the LatticeMico32 processor.

Figure 13: VGA Display Generated by LatticeMico32



The processor blacks out the display, writing black values to the video memory, and then regenerates the display shown in Figure 13 by writing to the video memory.

Measuring Performance

The VGA controller performs burst reads from the video memory. These word burst reads are not interruptible and give an idea of the throughput for burst-read operations. The VGA controller maintains a counter that counts the maximum cycles taken for burst line reads and is printed to the standard output by the application software.

Generating a Bootable Binary Image of the Executable Application

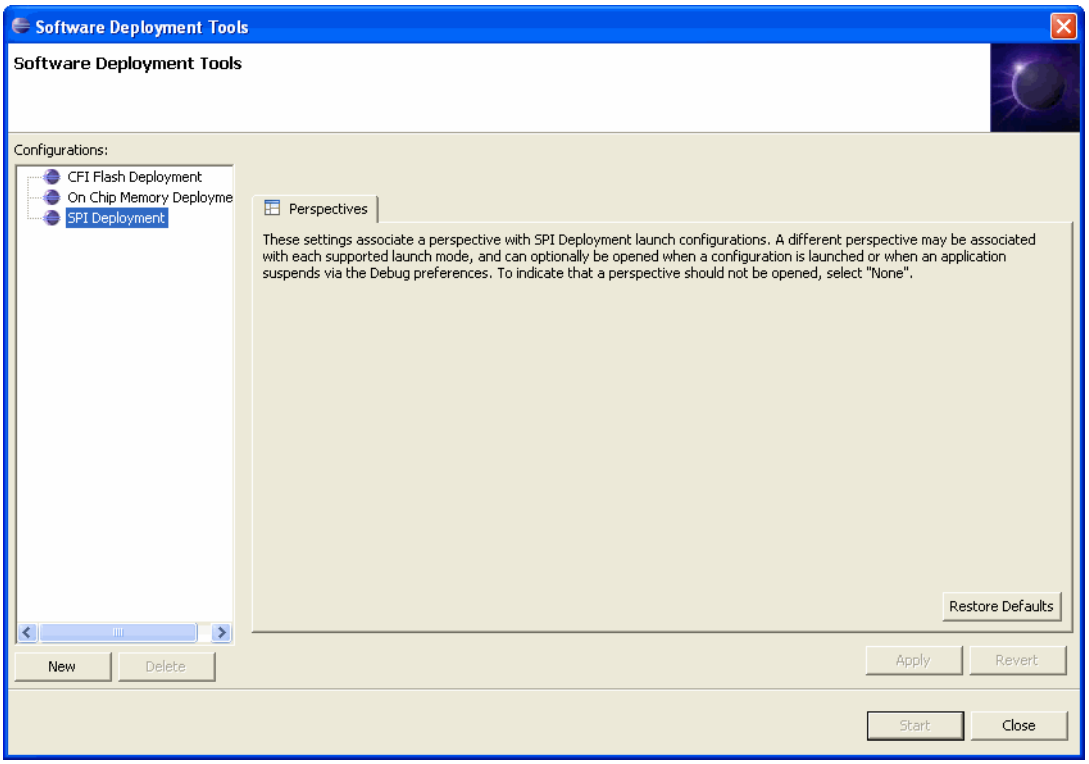
This section of the demonstration shows you how to generate a bootable binary image of the executable application that can be used for booting and loading from the SPI flash.

Before performing the steps in this section, rename the DDRDemo_VGA_App.bit file, located in the DDRDemo_VGA_App project folder, to DDRDemo_VGA_App_orig.bit for backup.

1. From the C/C++ SPE perspective, select the DDRDemo_VGA_App project by selecting it.

2. Choose **Tools > Software Deployment** to activate the Software Deployment Tools dialog box, shown in Figure 14.

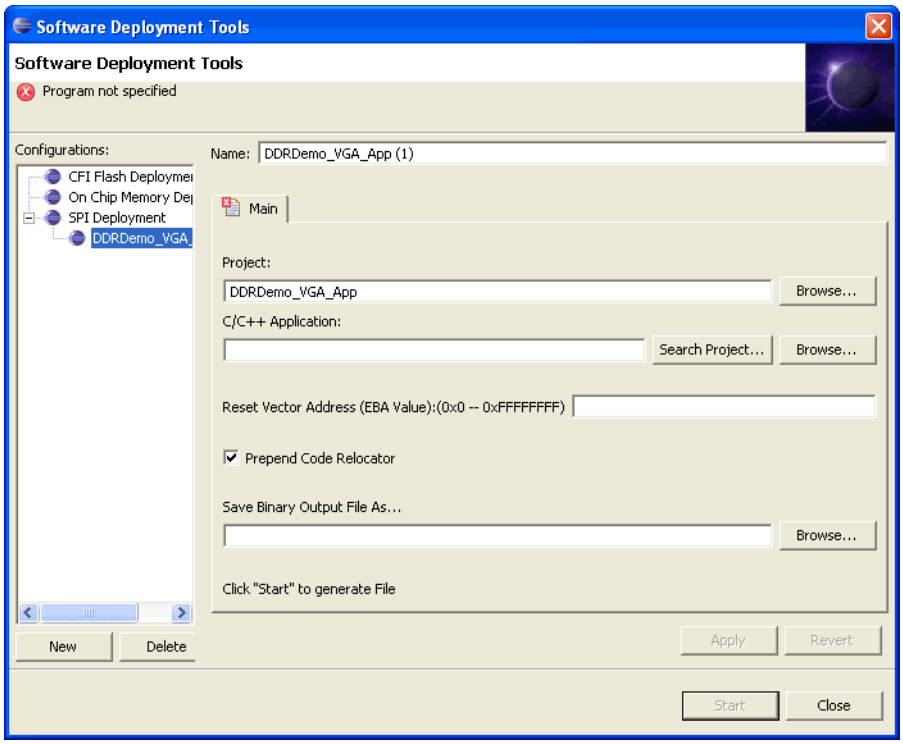
Figure 14: Software Deployment Tools Dialog Box



3. Select **SPI Deployment** from the list of available configurations, as shown in Figure 14.

4. Click the **New** button at the bottom of the list of available configurations to activate a new SPI deployment configuration, as shown in Figure 15.

Figure 15: New SPI Deployment Configuration



5. Configure the dialog box as follows:

   a. In the Project box, select **Browse** to select the DDRDemo_VGA_App project, if it is not already selected.

   b. In the C/C++ Application box, select **Search Project** to select DDRDemo_VGA_App.elf, which is the executable that you want to convert to a binary bootable image.

   c. In the Reset Vector Address box, enter **0x060F0000**, as explained in “Reset Vector Address (EBA Value) Selection” on page 4.

      The deployment tool scripts compile the boot-copier (also known as the boot-loader) for execution from this address, since this is the processor’s reset address.

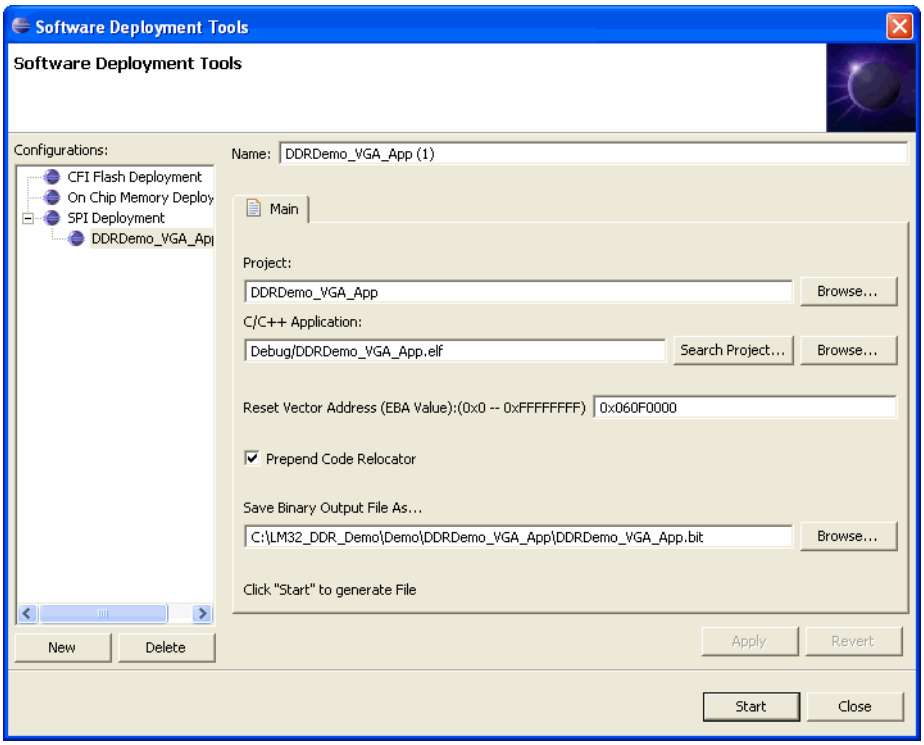
   d. Select the **Prepend Code Relocator** option, if it is not already selected.

      This selection tells the SPI deployment scripts to prefix the boot-copier so that the first code executed by the processor on a reset exception (that is, power-up) is the boot-copier. The boot-copier then copies the executable into the destination DDR SDRAM memory, since the application is compiled for execution from the DDR SDRAM memory.

e. In the Save Binary Output File As box, click **Browse** to activate the File Save As dialog box, and after selecting the DDRDemo_VGA_App directory, enter **DDRDemo_VGA_App.bit** as the file name.

6. After configuring the SPI Deployment Tools dialog box, shown in Figure 16, click **Apply**.

Figure 16: SPI Deployment Configuration



7. Click **Start** at the bottom of the Software Deployment Tools dialog box to generate the DDRDemo_VGA_App.bit binary file, which is a bootable binary file that has the application appended to the code relocator.

For more information on how the deployment tools work, refer to the *LatticeMico32 Software Developer User's Guide*.

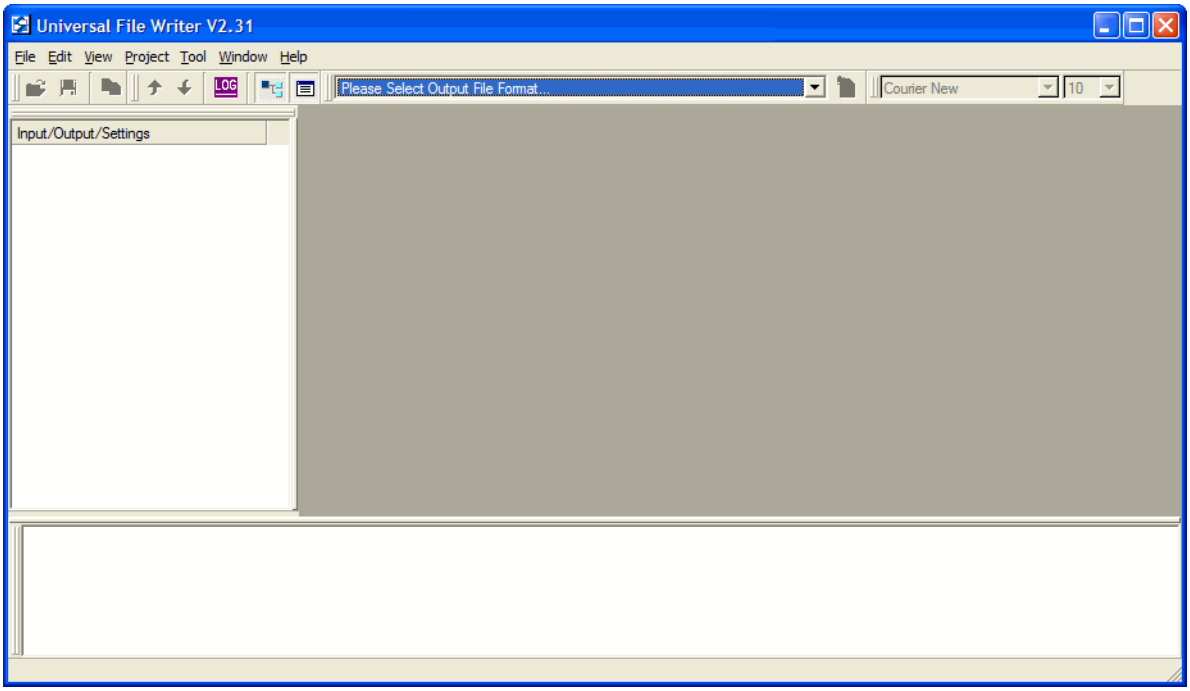
Generating an Intel Hex File

This section of the demonstration provides you with step-by-step instructions for generating an Intel-Hex-format file containing the FPGA bitstream and the binary bootable image generated in the previous step. IspVM System can then use the Intel-Hex-format file for programming the SPI flash on the LatticeMico32 Development Board for the LatticeECP FPGA.

1. Before performing the steps in this section, rename the DDRDemo_VGA_App_FPGA_Bitstream.mcs file in the DDRDemo_VGA_App project folder to a different name for backup.

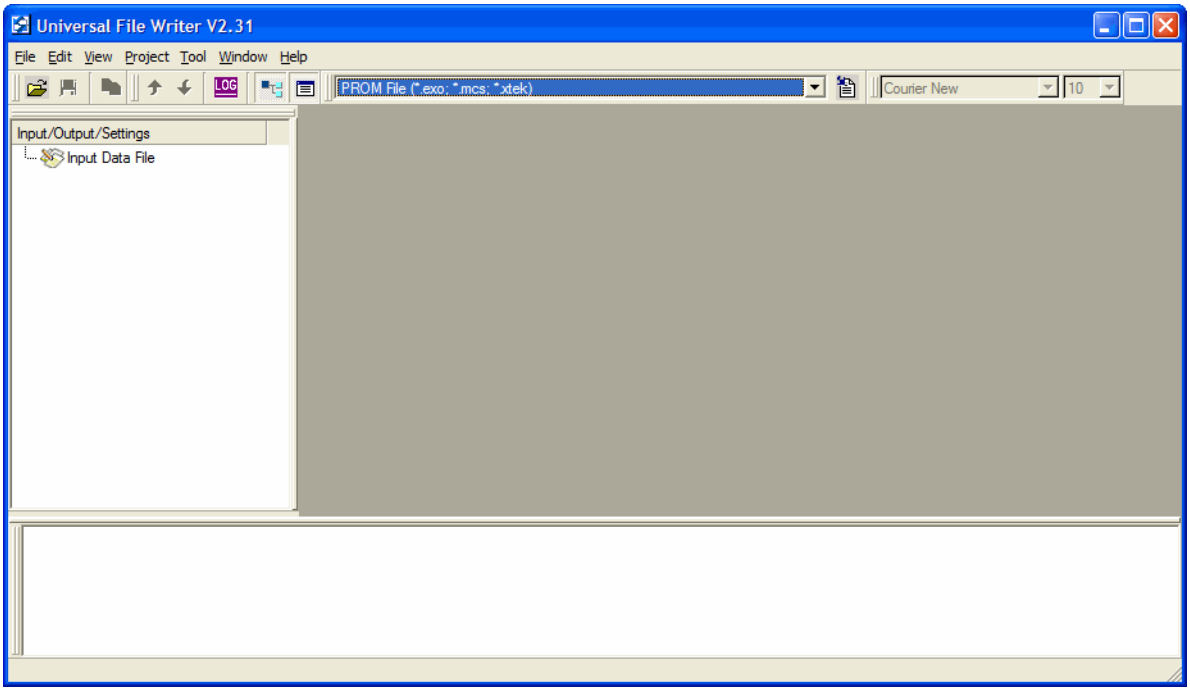
2. Start ispVM System.
3. Choose **ispTools > Universal File Writer** to start the Universal File Writer, as shown in Figure 17.

Figure 17: Universal File Writer



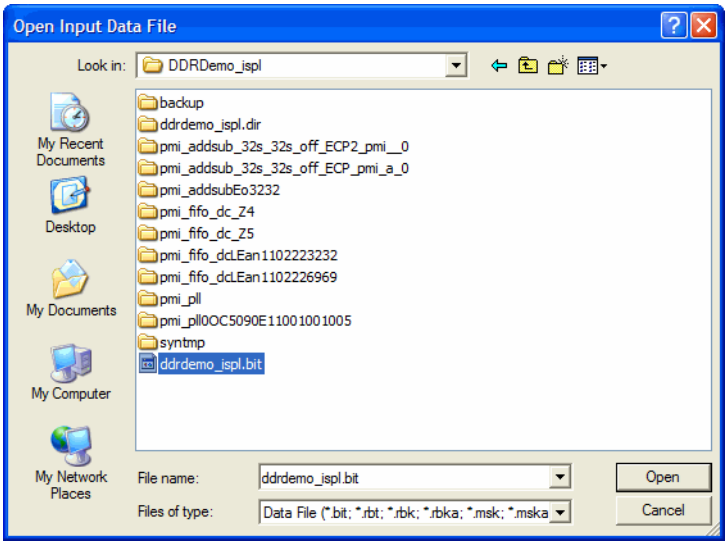
4. In the Please Select Output File Format box, select **PROM File** from the dropdown menu, as shown in Figure 18.

Figure 18: Selecting the PROM file



5. Double-click **Input Data File** in the Input/Output Settings pane to activate the Open Input Data File dialog box, shown in Figure 19.

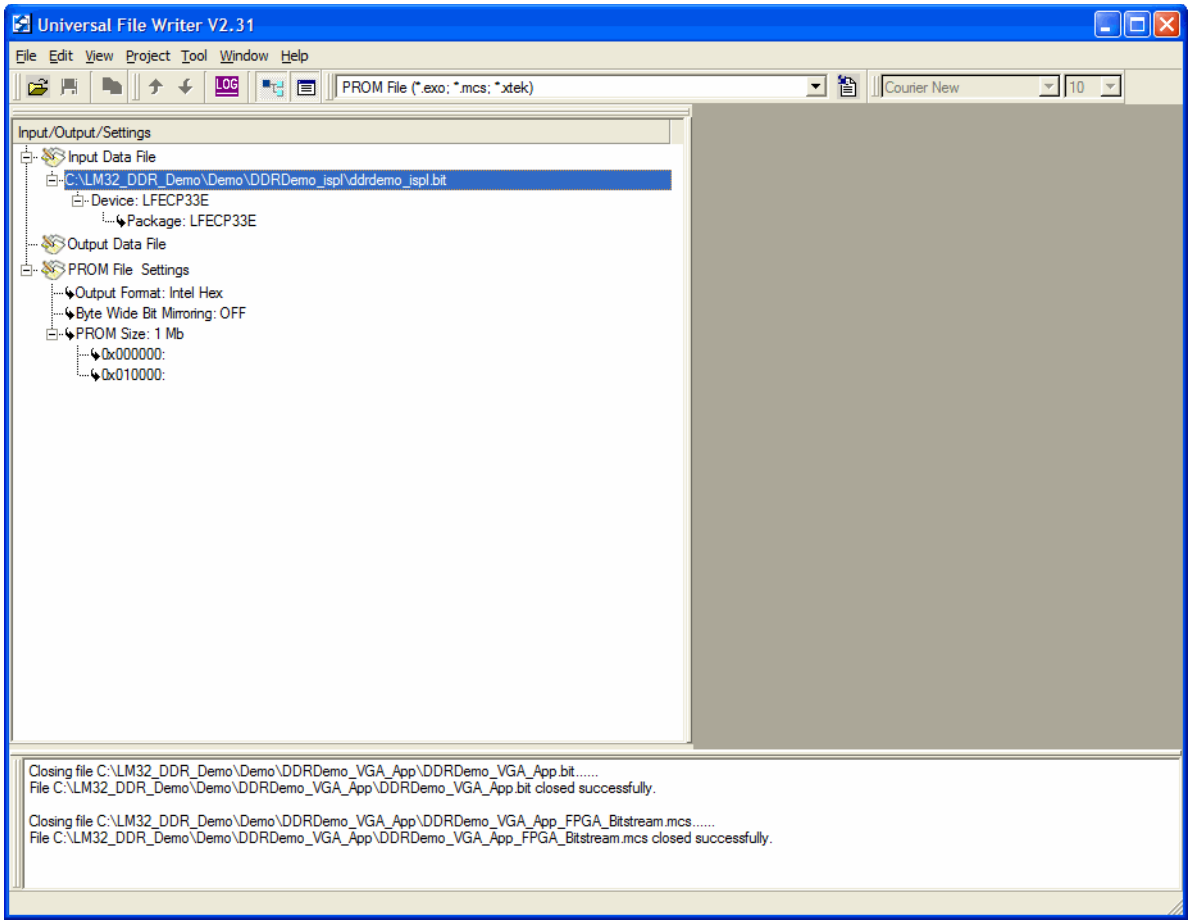
Figure 19: Open Input Data File Dialog Box



6. Select the **ddrdemo_ispl.bit** FPGA bitstream and select **Open**.

The dialog box closes, and this file appears as an Input Data File item, as shown in Figure 20.

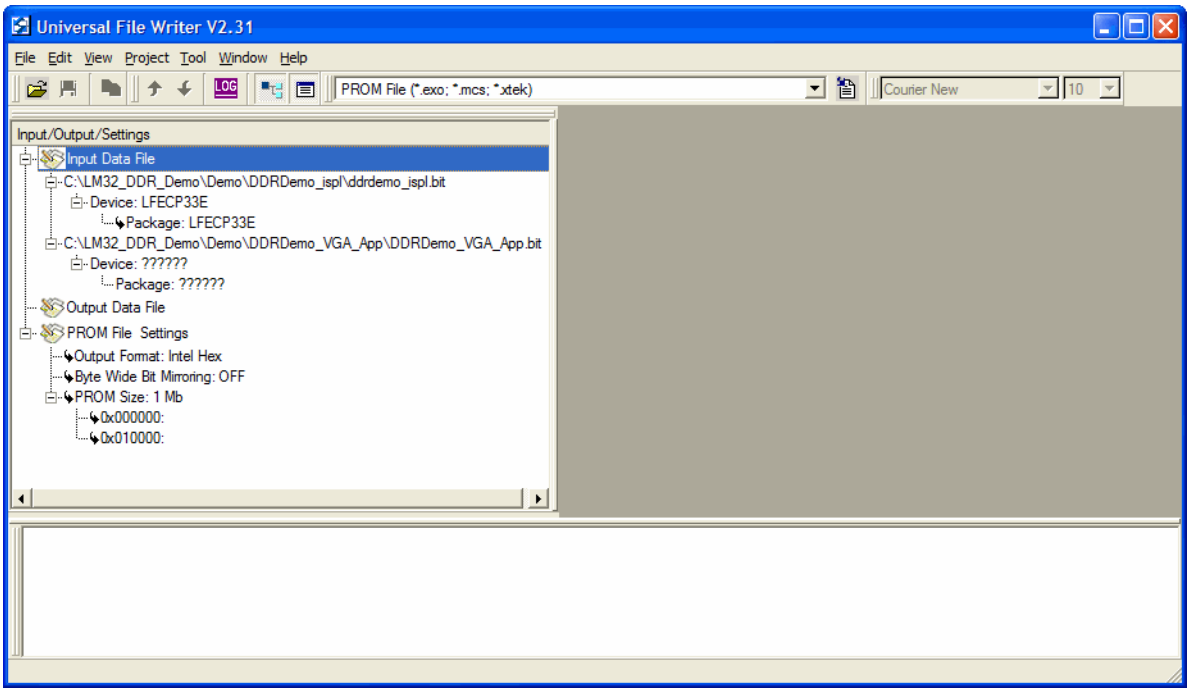
Figure 20: FPGA Bitstream File Selected



7. Double-click **Input Data File** in the Input/Output Settings pane to open the Input File Selection dialog box.
8. Select the **DDRDemo_VGA_App.bit** file and click **Open**.

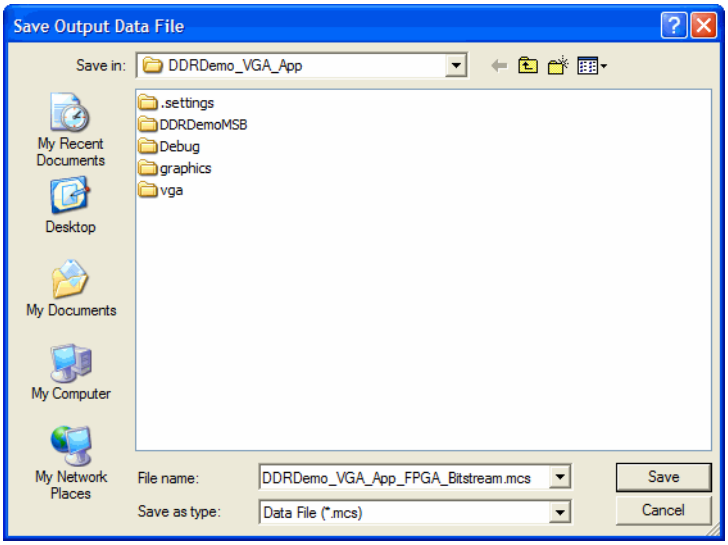
The dialog box closes, and this file appears as an item under Input Data Files, as shown in Figure 21.

Figure 21: DDRDemo_VGA_App.bit Selected



9. Double-click **Output Data File** in the Input/Output Settings pane to open the Save Output Data File dialog box, shown in Figure 22.

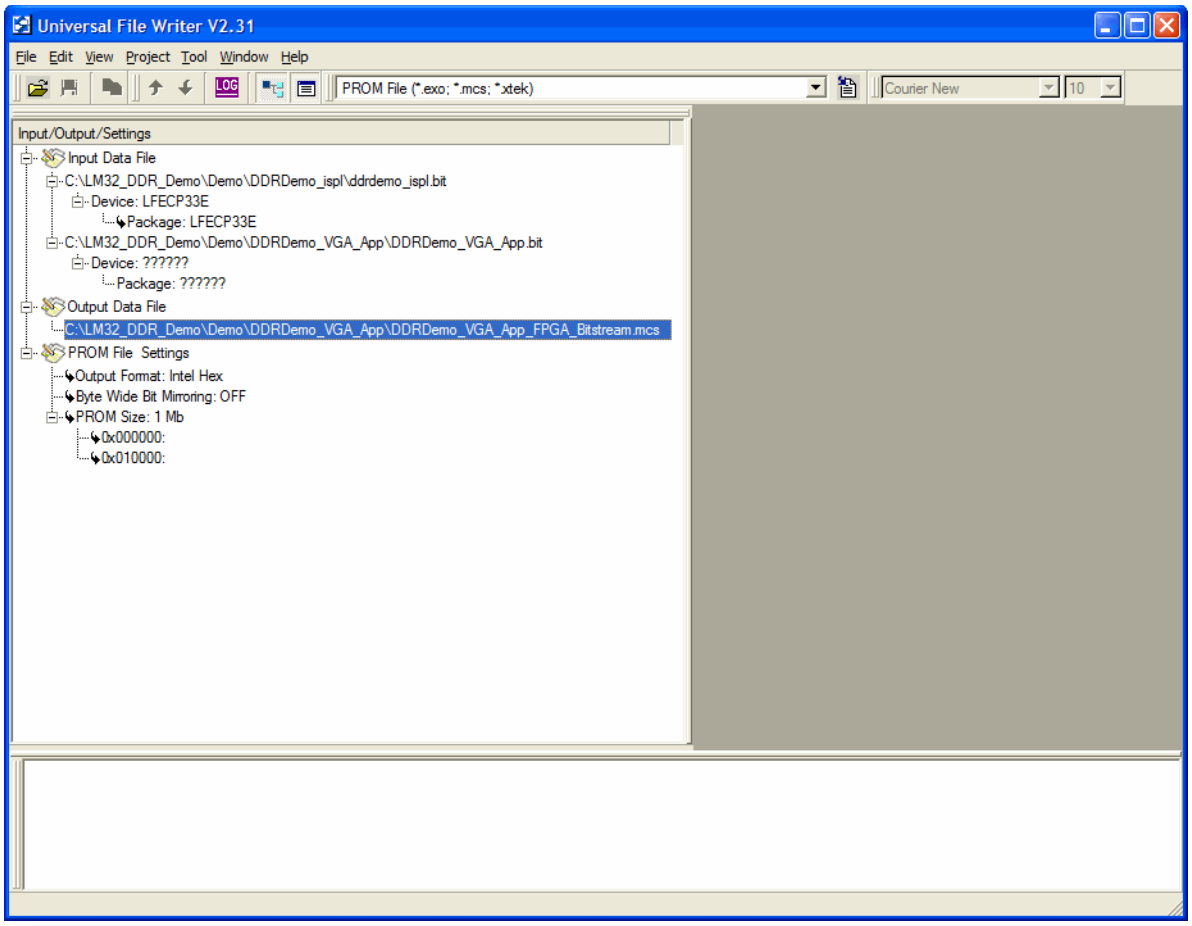
Figure 22: Save Output Data File Dialog Box



10. Browse to the DDRDemo_VGA_App folder, and in the File Name box, enter **DDRDemo_VGA_App_Bistream.mcs**, as shown in Figure 22.
11. Click **Save**.

    The dialog box closes, and the entered file name appears as an Output Data File item, as shown in Figure 23.

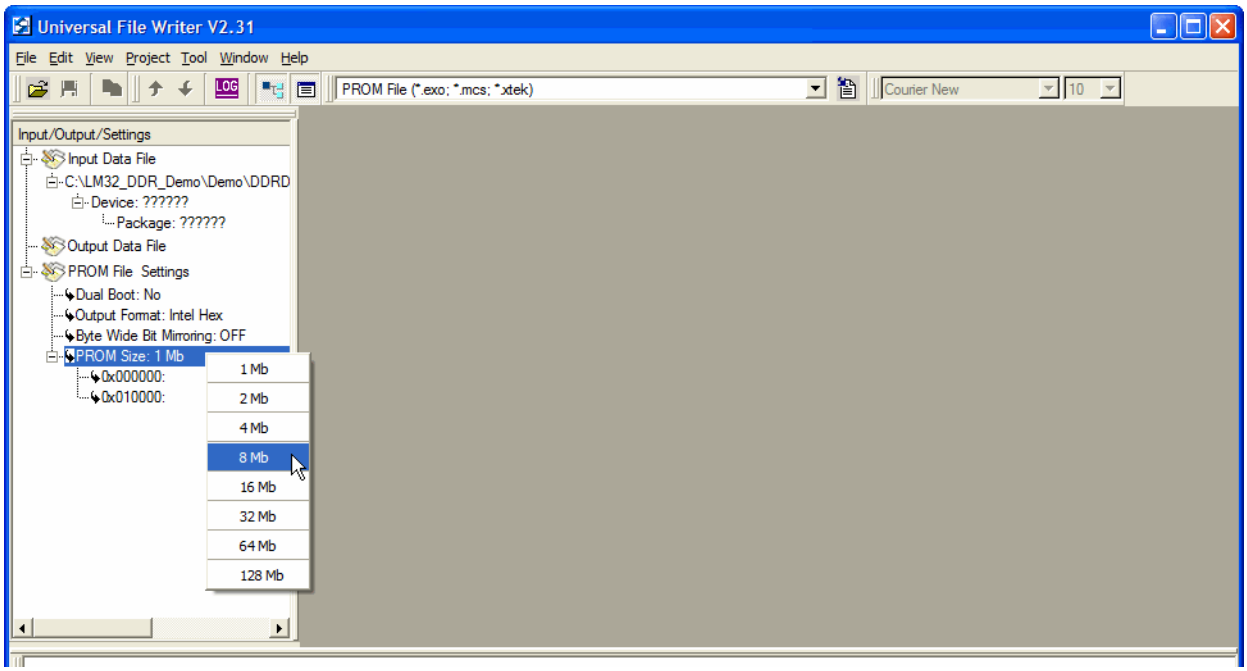
Figure 23: Output Data File Selected



12. Under PROM File Settings, select **PROM Size: 1Mb**.

13. Right-click on **Prom Size: 1Mb** to display the size selection menu, as shown in Figure 24.

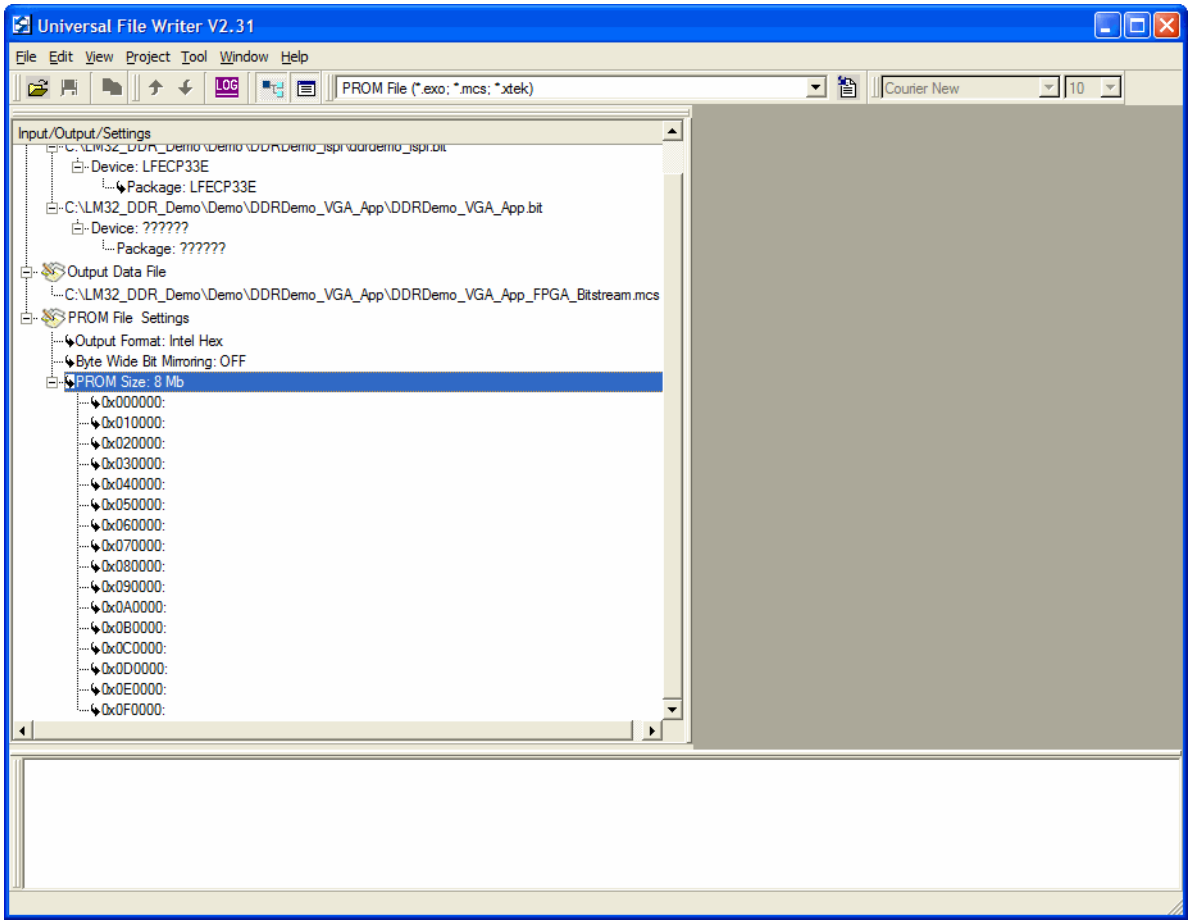
Figure 24: Size Selection Menu



14. Select **8Mb**, since the SPI flash size is 8 megabits.

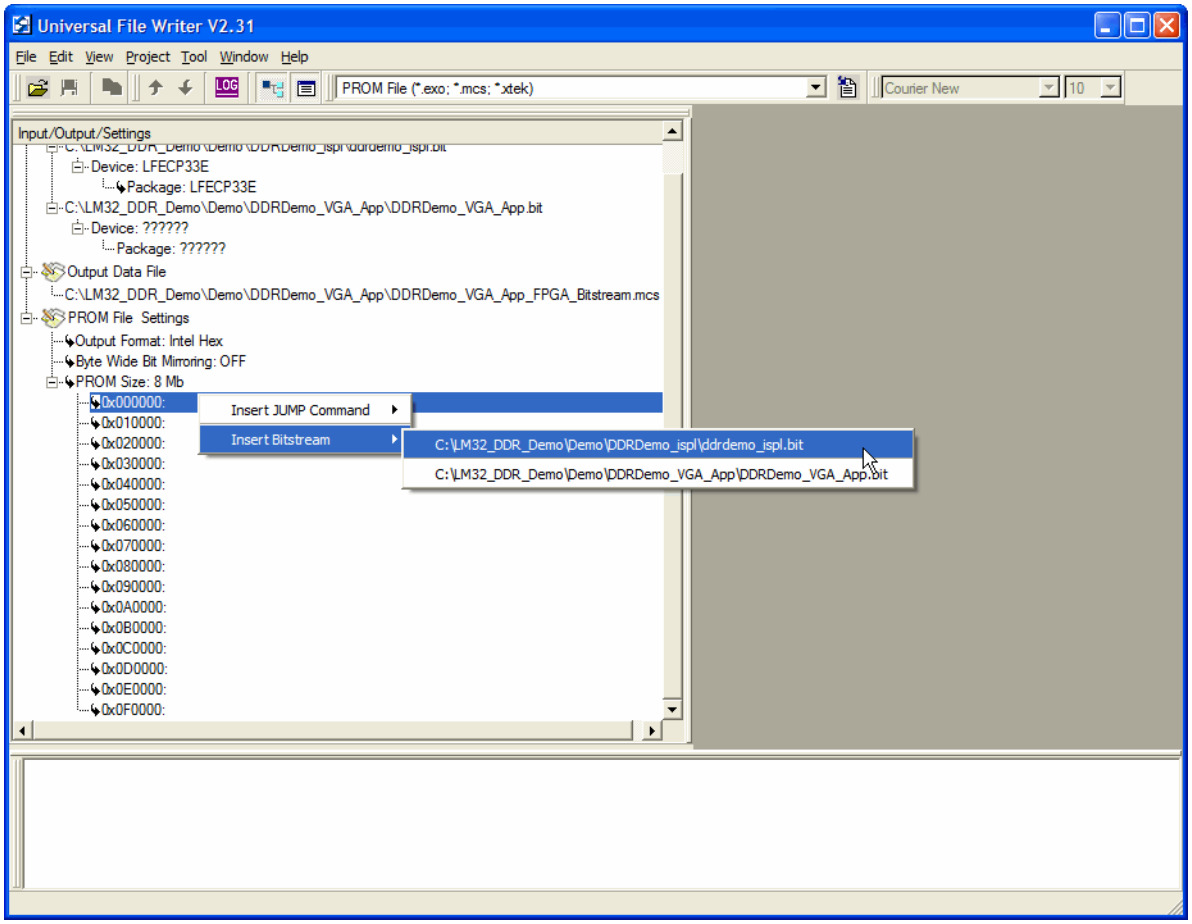
The PROM Size display changes to reflect this new selection, as shown in Figure 25.

Figure 25: PROM Size Selected



15. Under Prom Size: 8Mb, select **0x000000**, and right-click to display the input file selection menu, as shown in Figure 26.

Figure 26: Selecting Input File at Offset 0x0000

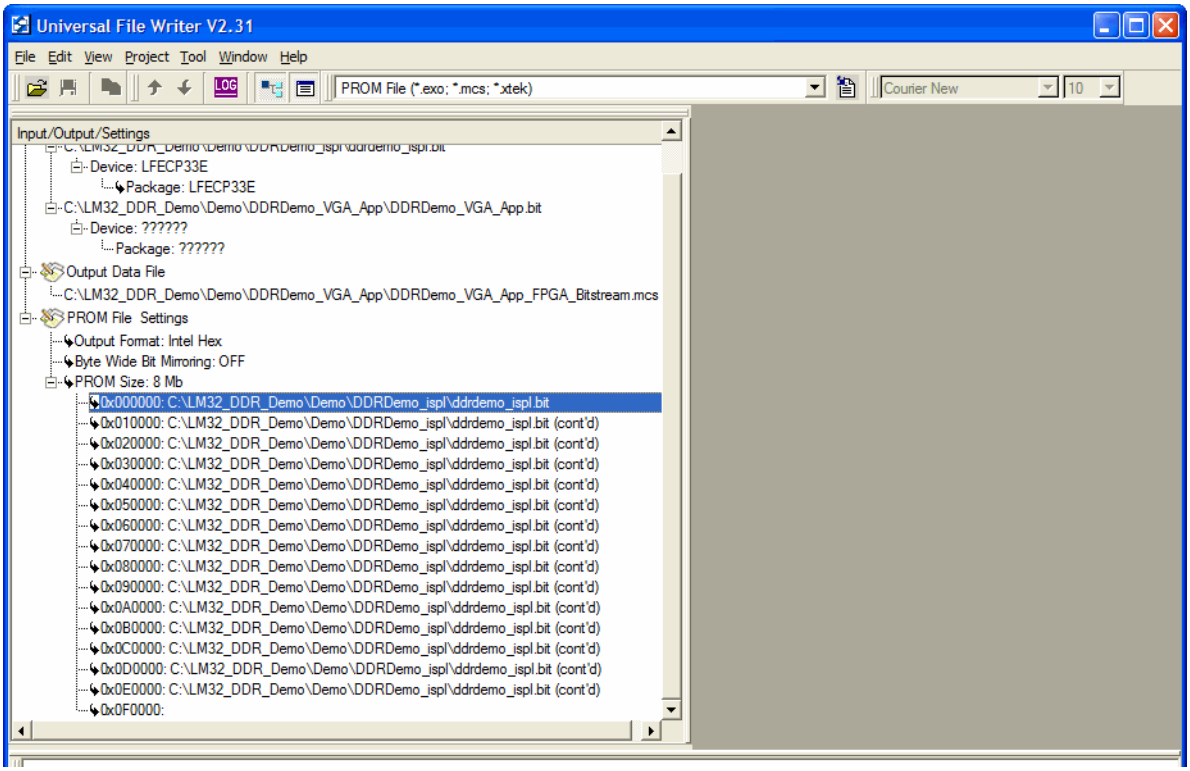


16. Select **ddrdemo_ispl.bit**.

    Offset 0x000000 is the first location in the SPI flash, so the FPGA configuration bitstream must reside here. The FPGA configuration

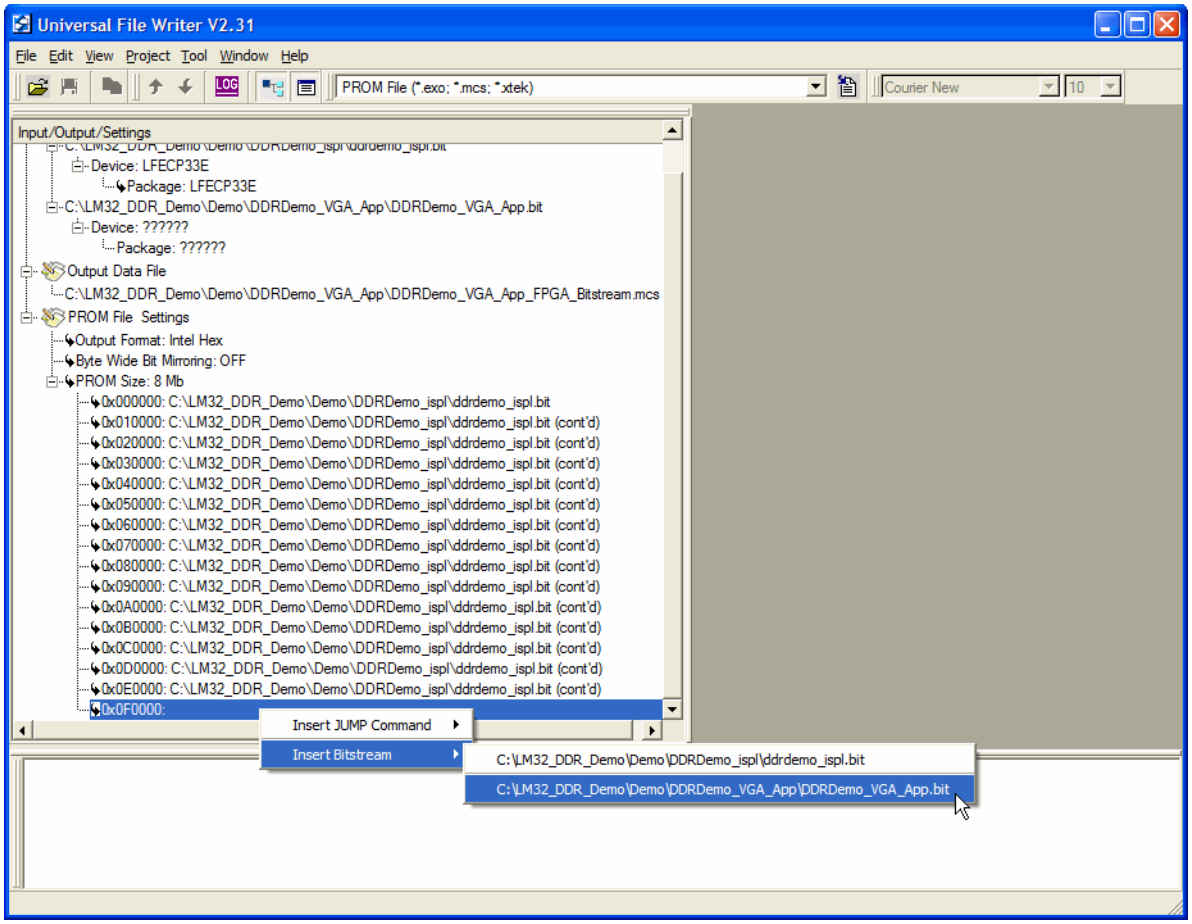
bitstream occupies all but one of the available sectors in the PROM file, as shown in Figure 27.

Figure 27: FPGA Bitstream Placed



17. Under Prom Size: 8Mb, select **0x0F0000**, and right-click to display the input file selection menu, as shown in Figure 28.

Figure 28: Selecting Input Data File at Offset 0x0F0000

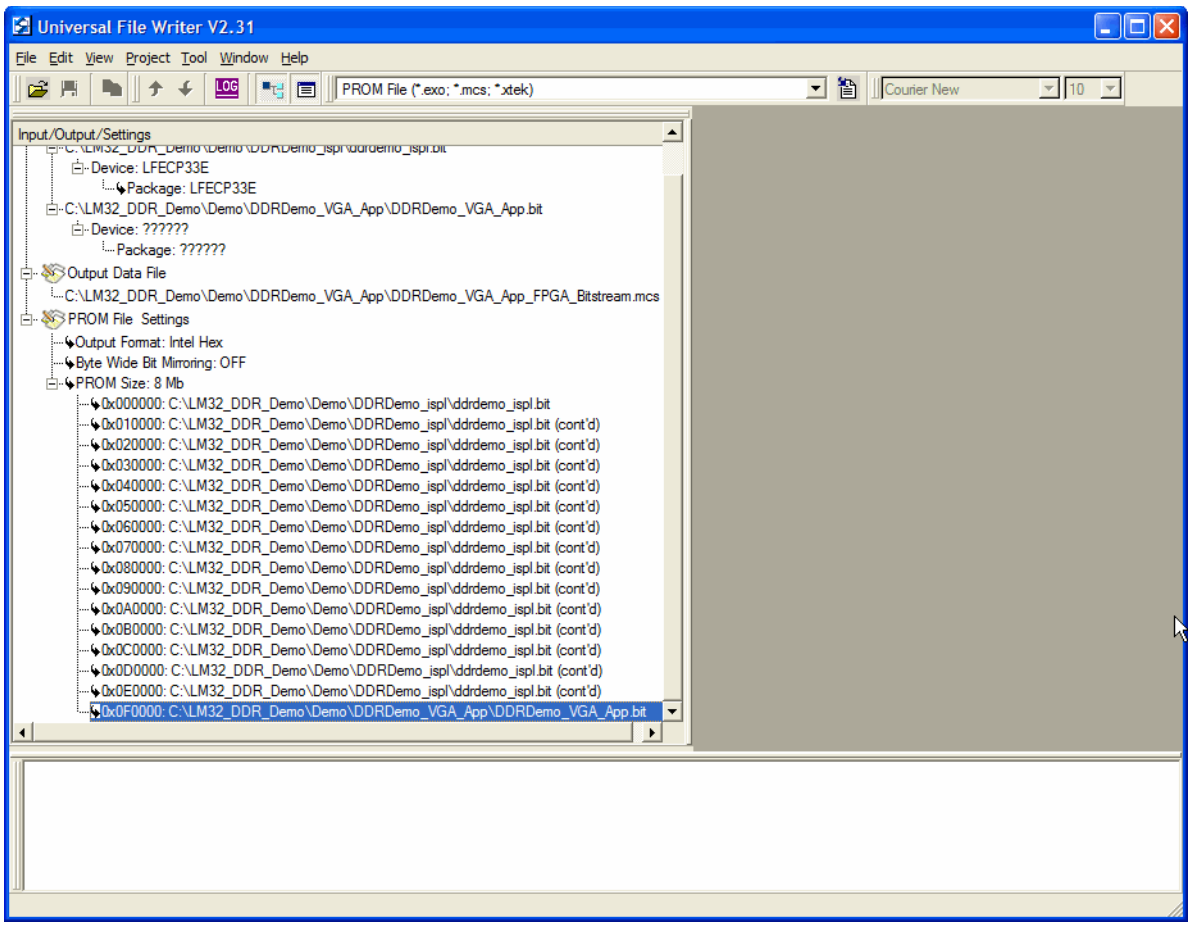


18. Select **DDRDemo_VGA_App.bit**.

    Offset 0x0F0000 is the offset of the first sector available after the FPGA bitstream data in the PROM file. It is also the offset in the SPI flash ROM component that is used as the LatticeMico32 EBA value. The processor starts executing instructions from 0x0F0000, so the executable binary

must be located at this offset in the PROM file. Figure 29 shows the complete PROM file image layout.

Figure 29: PROM File Image Layout



19. Select **Project > Generate** to generate the DDRDemo_VGA_App_FPGA_Bitstream.mcs output file.

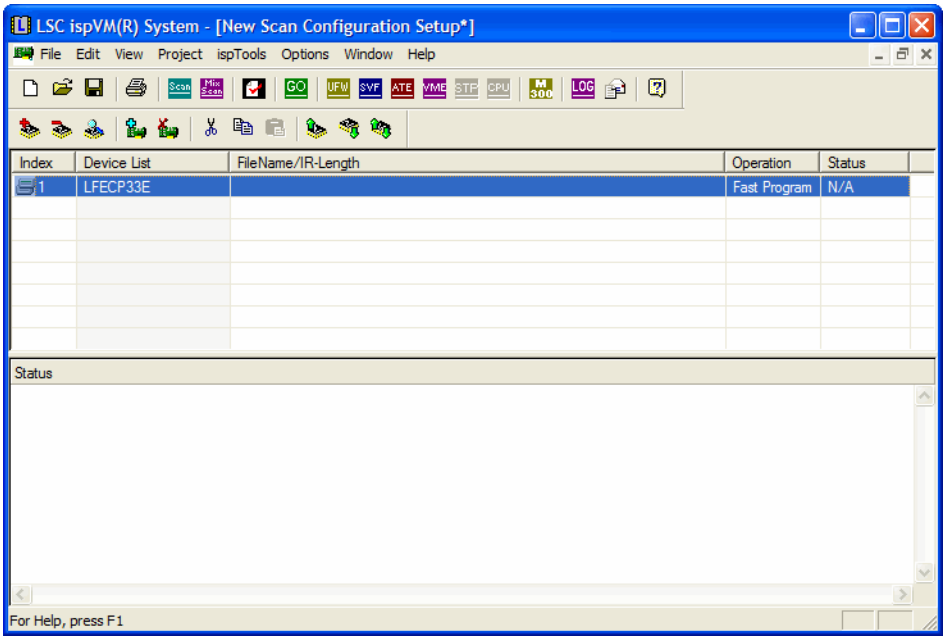
Programming the SPI Flash Device

This section demonstrates how to program the SPI flash device with the Intel Hex file generated in the previous section.

1. Start ispVM System.

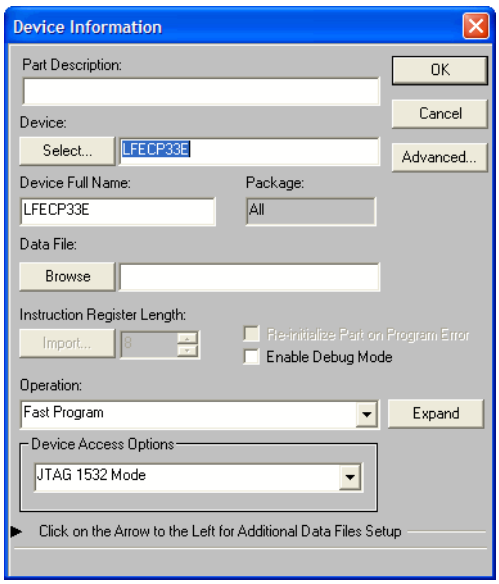
2. Select **Scan** to identify the LatticeECP LFECP33E FPGA device, as shown in Figure 30.

Figure 30: LFECP33E Scanned



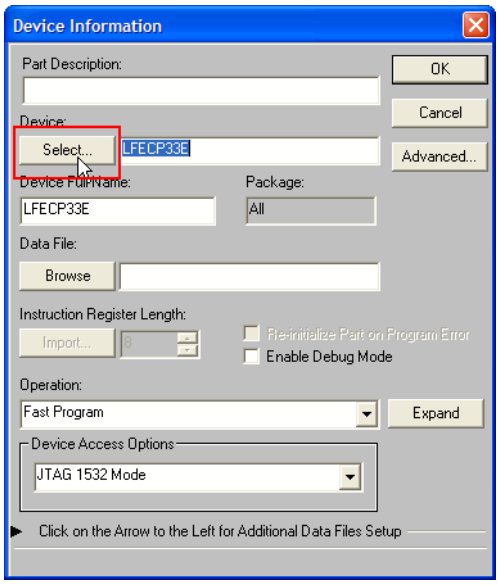
3. Select the **LFECP33E** device, the only device listed.
4. Double-click the **FileName/IR-Length** cell for the LFECP33E device to display the Device Information dialog box, as shown in Figure 31.

Figure 31: Device Information Dialog Box



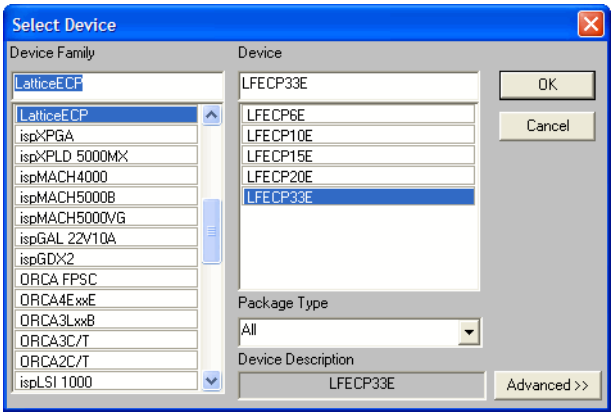
5. Select **Select** under Device, as shown in Figure 32.

Figure 32: Selecting Select



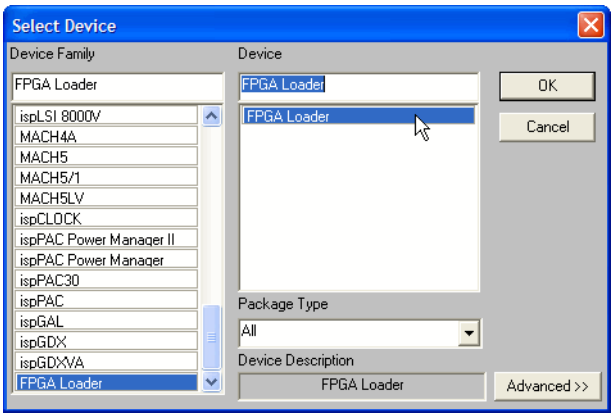
6. From the Select Device dialog box, shown in Figure 33, scroll to the bottom to select the FPGA Loader device.

Figure 33: Select Device Dialog Box



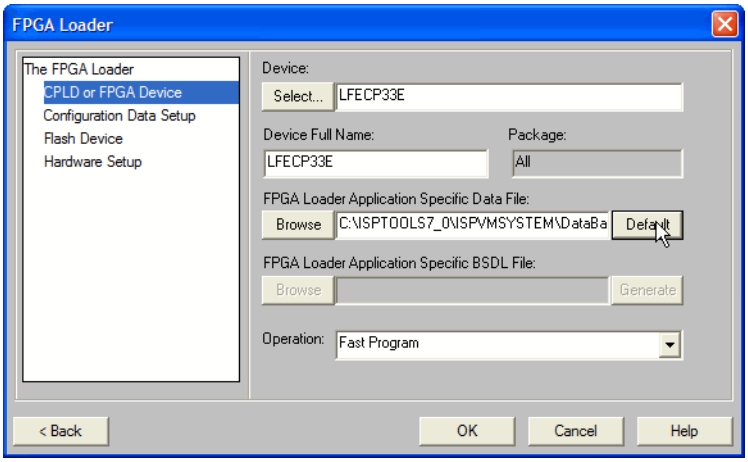
7. Click **OK** after selecting the FPGA Loader device, as shown in Figure 34.

Figure 34: FPGA Loader Selected



8. From the FPGA Loader dialog box, select **CPLD or FPGA Device**, as shown in Figure 35.

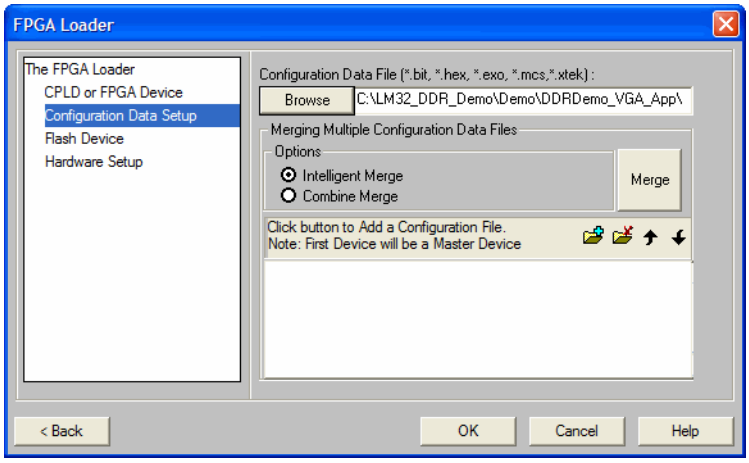
Figure 35: Settings in FPGA Loader Dialog Box



9. In the FPGA Loader dialog box, enter the following:
   a. In the Device box, select **LFECP33E**, if it is not already selected.
   b. In the FPGA Loader Application-Specific Data File box, click **Default** to select the default.
   c. In the Operation box, select **Fast Program**, if it is not already selected.

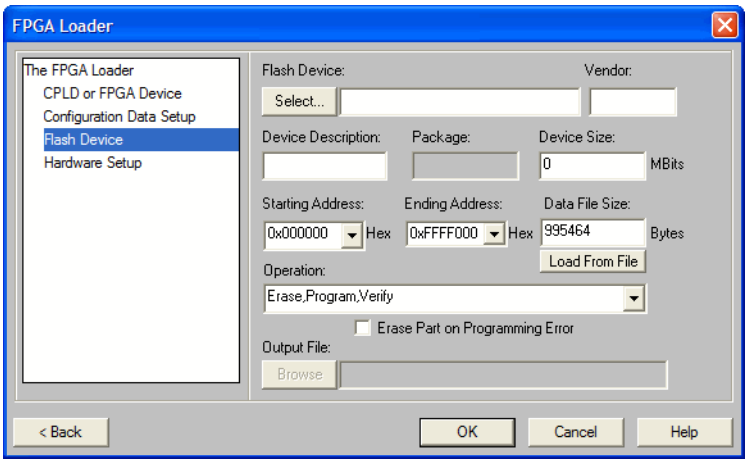
10. Select **Configuration Data Setup**, as shown in Figure 36.

Figure 36: Configuration Data Setup Selected



11. In this tab, select **Browse** under Configuration Data File to open a file selection dialog box, and select the **DDRDemo_VGA_App_FPGA_Bitstream.mcs** file.
12. Select **Flash Device**, as shown in Figure 37.

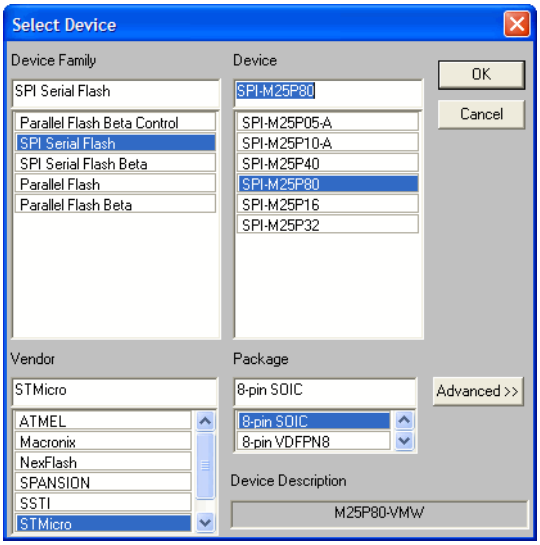
Figure 37: Flash Device Tab



13. Select **Select** for Flash Device to display the Flash Device selection shown in Figure 38.
14. In the Flash Device tab, enter the following, as shown in Figure 38.
    a. In the Device Family section, select **SPI Serial Flash**.
    b. In the Vendor section, select **STMicro**.
    c. In the Device column, select **SPI-M25P80**.

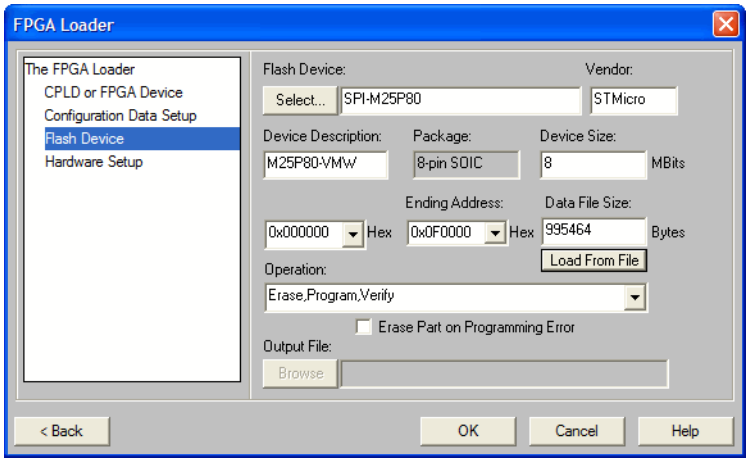
d. In the Package section, select **8-pin SOIC**.

Figure 38: Flash Device Selection



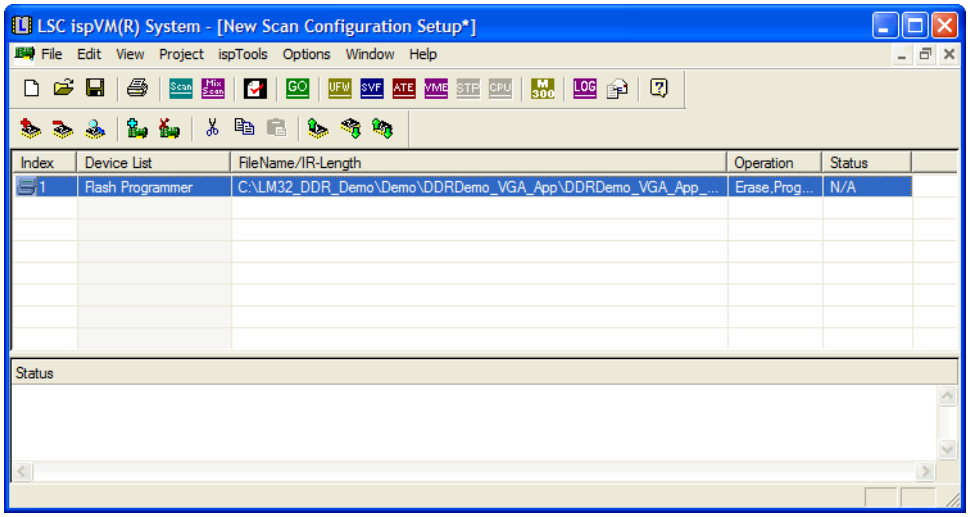
15. Select **OK** to confirm the flash device selection.
16. Select **OK** in the FPGA Loader dialog box after confirming the flash device settings, as shown in Figure 39.

Figure 39: Flash Device Tab After Flash Device Selection



IspVM System now displays Flash Programmer as the device in the Device List column, as shown in Figure 40.

Figure 40: Flash Programmer Selected in ispVM System



17. Select **GO** from ispVM System to program the SPI flash.

After successfully programming the SPI flash, recycle the power to the LatticeMico32 Development Board. Then press the Reset button on the board. The FPGA is configured from the SPI flash, and the processor executes the code copier from the SPI flash, which, in turn, copies the application binary to the external DDR SDRAM memory. Once the boot-copier has copied the application binary, it executes the copied code from the external DDR SDRAM memory.

Rebuilding the FPGA Bitstream

Completely rebuilding the FPGA bitstream is a four-step process:

1. Open the Synplify project located in LM32_DDR_Demo\Demo\vga\hdl\synthesis\synplify directory and generate the EDIF file.
2. Open and rebuild the video_chain_top ispLEVER project, located in the LM32_DDR_Demo\Demo\vga\hd\synthesis\ispLever directory, to generate the video_chain_top.ngo netlist. This is the VGA controller.
3. Copy the .ngo file into the LM32_DDR_Demo\Demo\DDRDemo_ispl\ directory.
4. Rebuild the DDRDemo_ispl project in the LM32_DDR_Demo\Demo\DDRDemo_ispl directory to generate the FPGA bitstream.

If the changes are localized to the platform in MSB, you do not need to regenerate the video_chain_top .ngo netlist.

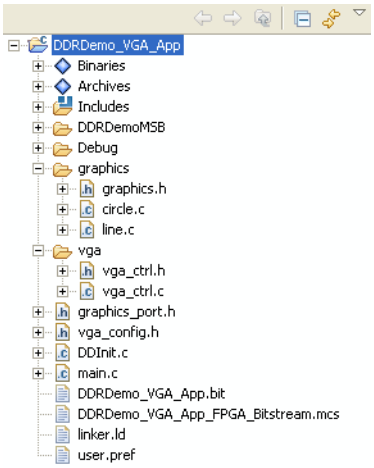
Design Details

This section provides technical information on the system design of the demonstration. The application software is described first, because it is a very simple application and easy to understand. It is followed by a detailed description of the platform components.

Application Software

Figure 41 shows the outline of the project contents as viewed in C/C++ SPE. This application is a managed-make project but overrides the default LatticeDDInit implementation and uses a custom linker script.

Figure 41: Project Contents Overview



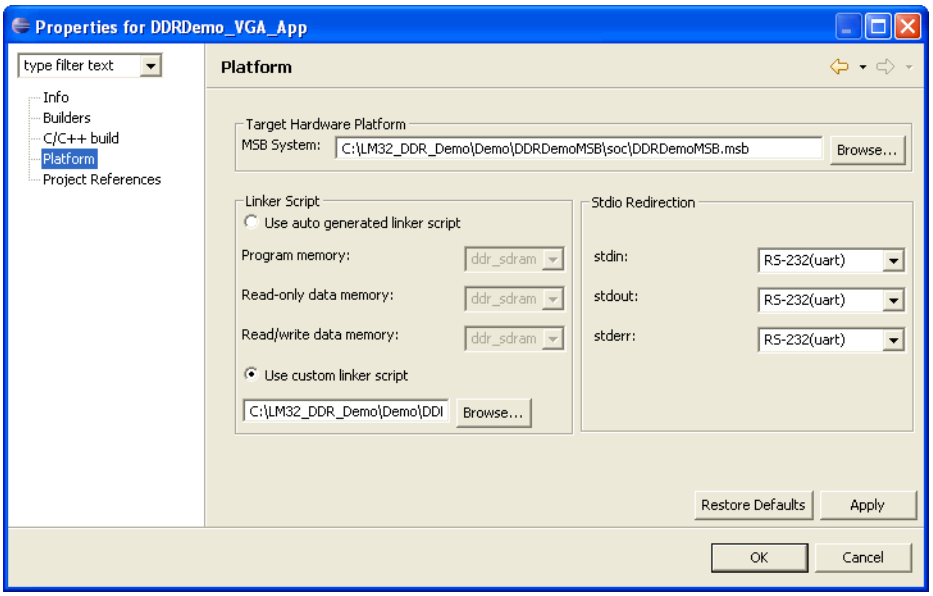
The contents of this application are as follows:

- DDRDemo_VGA_App/graphics – This folder contains basic plotting functions for drawing lines and circles. These functions are based on the Bresenham algorithm, which avoids floating-point operations, making them efficient for processors that lack a floating-point unit.
- DDRDemo_VGA_App/vga – This folder contains functions for configuring the VGA controller and for reading back the configuration information.
- DDRDemo_VGA_App/DDInit.c – This C source file contains the LatticeDDInit implementation that overrides the default LatticeDDInit initialization routine to reduce code size.
- DDRDemo_VGA_App/main.c – This C source file implements “int main(void),” the main application entry point for the software application.
- DDRDemo_VGA_App/graphics_port.h – This C header file contains porting information specific to this demonstration. You do not have to modify this file as part of the demonstration.

- DDRDemo_VGA_App/vga_config.h – This C header file contains VGA configuration information, as described in “Configuring VGA Timing Parameters and Verifying the Design Functionality” on page 12.
- DDRDemo_VGA_App/linker.ld – This file is the custom linker file that is used by the application. It is derived from the linker.ld file generated by the managed build. The default linker.ld file makes the entire DDR SDRAM available for application. This linker.ld file redefines the size of the DDR SDRAM for building the application, so that it uses only 25 percent of the DDR SDRAM size for this application and makes the rest of the DDR SDRAM available for other purposes.

Figure 42 shows the platform properties for this managed-build software application.

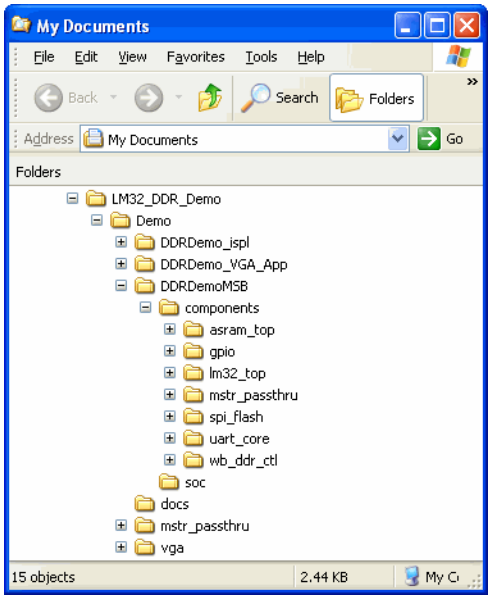
Figure 42: Platform Properties of the Application



Hardware

The directory structure of the demonstration is shown in Figure 43.

Figure 43: Demonstration Directory Structure



Top-Level Module

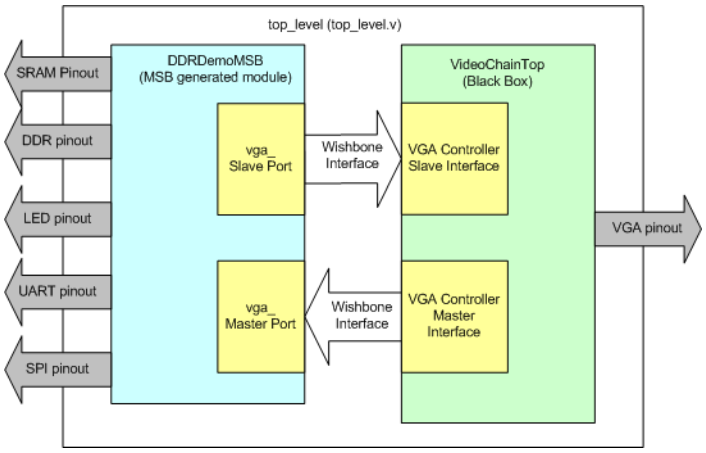
Figure 44 shows the contents of the SOC directory generated by MSB.

Figure 44: SOC Directory Contents

In Figure 44, DDRDemoMSB.v is the platform Verilog file generated by MSB. This demonstration does not use DDRDemoMSB.v as the top-level design. Instead, it uses a hand-written top_level.v Verilog file as the top-level design. This top-level design implements a module named top_level, which is the top-level design. The sole purpose of this module is to wire the Verilog language platform module (DDRDemoMSB) with the VHDL language entity video_chain_top. Video_chain_top is the VGA controller component and is

declared as a black box in the top_level module. The contents of the top_level module are diagrammatically represented in Figure 45.

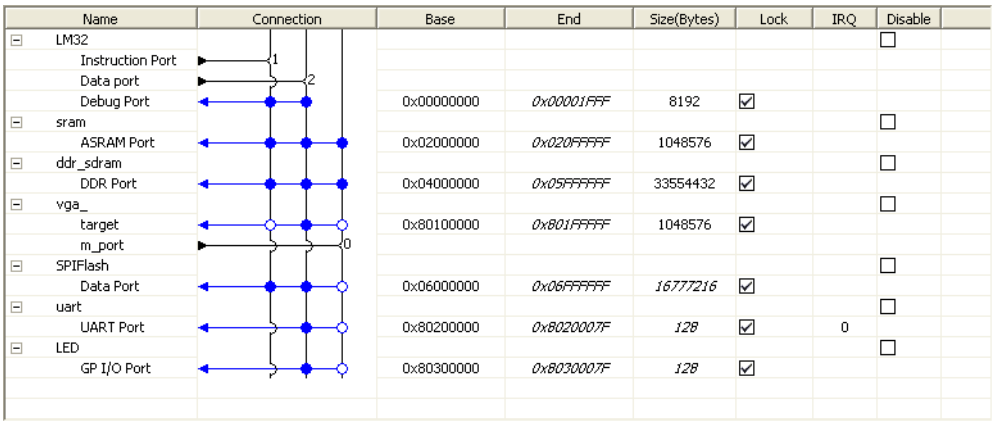
Figure 45: Top-Level Module



DDRDemoMSB Platform Components

Figure 46 shows the connectivity between the various platform components.

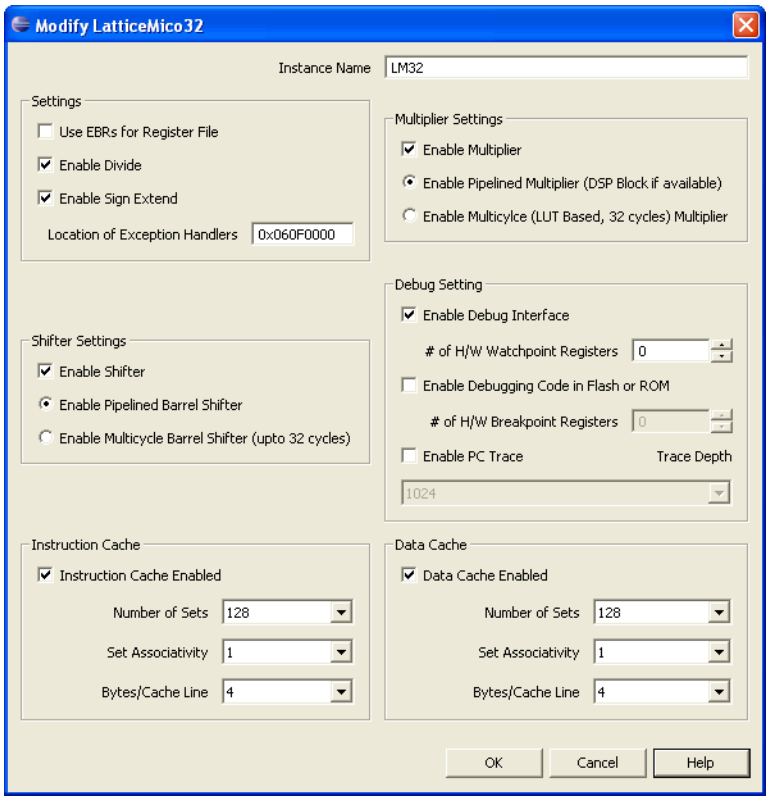
Figure 46: LatticeMico32 Platform



This platform includes two masters: the LatticeMico32 processor (LM32 instance name) and the VGA controller (vga_ instance name). All components other than the vga_ component are part of LatticeMico32 System Version 7.0 SP1. You can obtain information on component-specific configurations by opening their configuration dialog boxes. Only the relevant components are described in this section.

LatticeMico32 Processor (Instance Name: LM32) This LatticeMico32 processor instance is the only processor in the platform. Figure 47 shows the configuration settings for this processor.

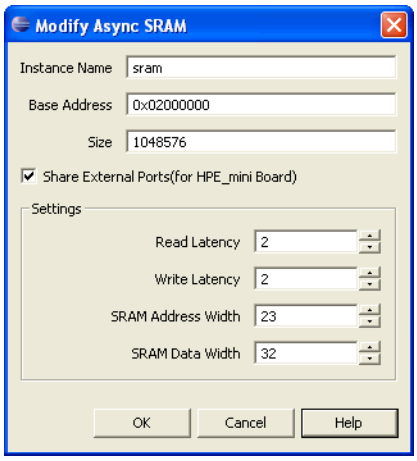
Figure 47: Modify LatticeMico312 Dialog Box



Asynchronous SRAM (Instance name: sram) This asynchronous SRAM serves as the video memory that is written to by the processor and read from

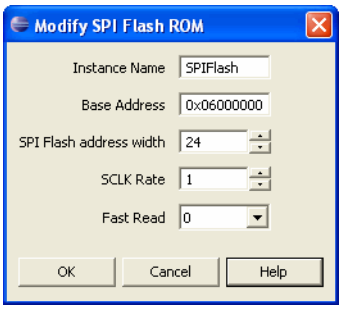
by the VGA controller. Figure 48 shows the configuration dialog box for the component instance.

Figure 48: Modify Async SRAM Dialog Box



SPI Flash ROM (Instance Name: SPIFlash) This SPI flash ROM instance is connected to the external configuration SPI flash on the LatticeMico32 Development Board. It allows the processor to view the SPI flash as a flat read-only memory. It translates all read requests into the appropriate serial communication with the external SPI flash and presents 32-bit data to the processor. Figure 49 shows the configuration dialog box for this component.

Figure 49: Modify SPI Flash ROM Dialog Box



Master Passthru (Instance Name: vga_) This is a custom component that has a WISHBONE slave interface, as well as a WISHBONE master interface. This component passes the slave interface and master interface signals as component input and output signals. It allows exposing arbitrated slave interface signals beyond the top-level module generated by MSB. It also allows the master signals to enter the top-level module generated by MSB.

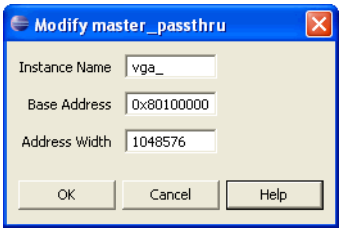
Figure 50 lists the external signals exposed by this passthrough component.

Figure 50: External Signals Exposed by Master Passthrough Component

```
// external signals (master passthru)
input  [31:0] adr,
input  [31:0] master_data,
input  we,
input  [3:0] sel,
input  stb,
input  cyc,
input  lock,
input  [2:0] cti,
input  [1:0] bte,
output [31:0] slave_data,
output ack,
output err,
output rty,
output clk,
output rst,
// external signals (slave passthru)
output [31:0] slv_adr,
output [31:0] slv_master_data,
input [31:0] slv_slave_data,
output slv_strb,
output slv_cyc,
input slv_ack,
input slv_err,
input slv_rty,
output [3:0] slv_sel,
output slv_we,
output [1:0]slv_bte,
output [2:0]slv_cti,
output slv_lock,
```

Figure 51 shows the instance-specific configuration dialog box for the master passthrough.

Figure 51: Modify master_passthru Dialog Box



In Figure 51, the Address Width corresponds to the addressable memory window size of the arbitrated slave interface.

LatticeMico32 DDR SDRAM Controller The LatticeMico32 DDR SDRAM controller used in this design is based on the pipelined DDR SDRAM controller IP version 6.3.

Note

Perform the following two steps if you choose to regenerate the DDR SDRAM IP to use this demonstration with a different DDR memory module than the one mentioned in this document:

- After generating the platform, edit the wb_ddr_ctl.v file and make sure that SYS_FREQ is set to 50, not 25, for the wb_ddr_ctl module. You must perform this check if you regenerate the platform or regenerate the DDR SDRAM IP.
- Select **Command-Burst Enable** as part of the DDR IP parameter selection.

Figure 52 shows the Modify DDR Controller dialog box in MSB that enables you to configure the DDR SDRAM.

Figure 52: Modify DDR Controller Dialog Box

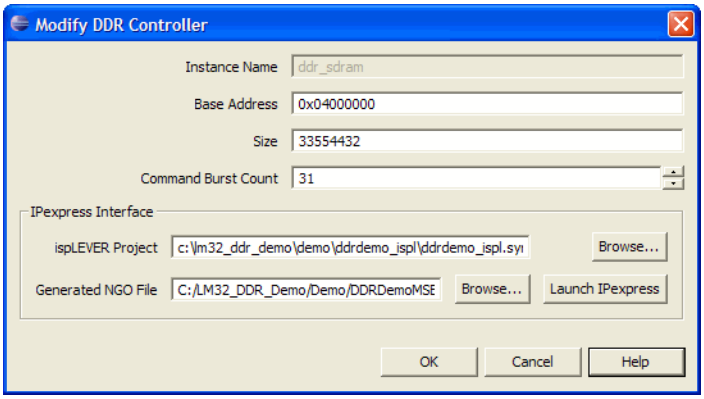


Figure 53 through Figure 57 show the DDR IP configuration selections for the DDR IP used in this demonstration.

Figure 53: DDR IP: Mode Selection

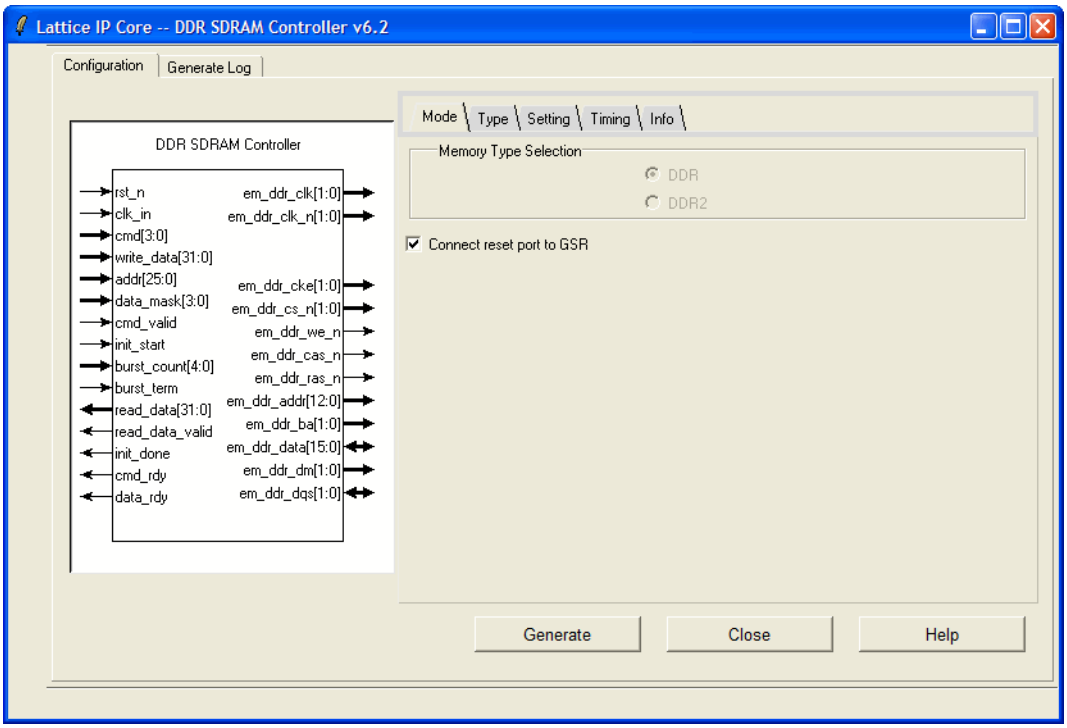


Figure 54: DDR IP: Type Selection

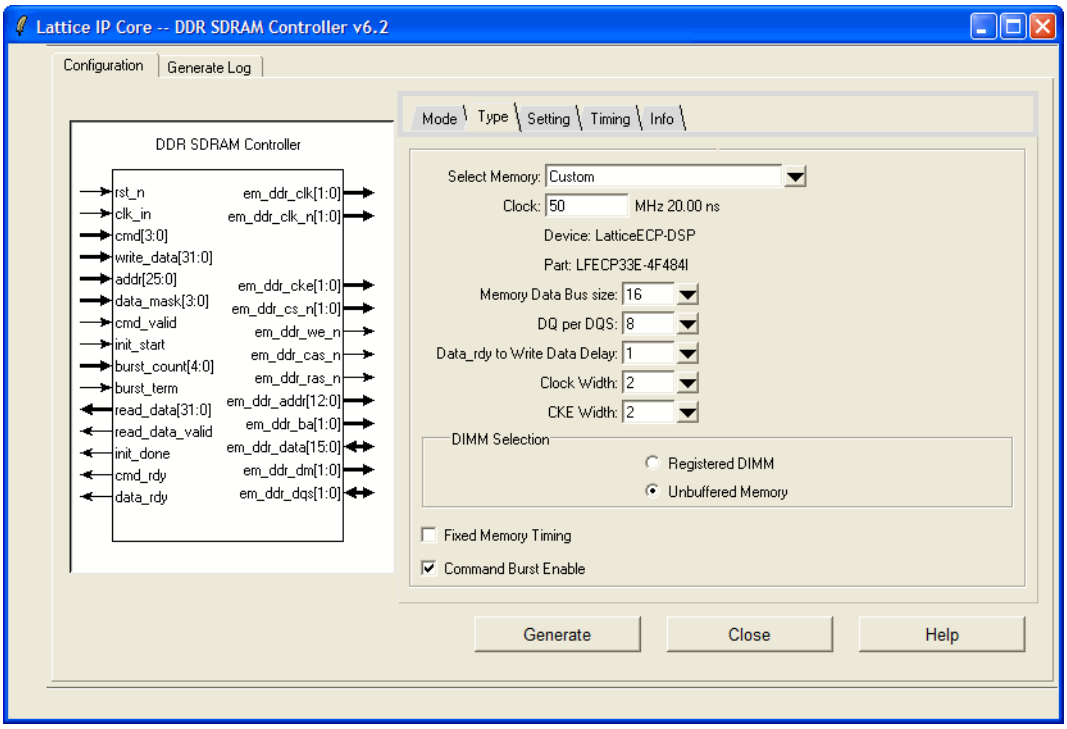


Figure 55: DDR IP: Setting Selection

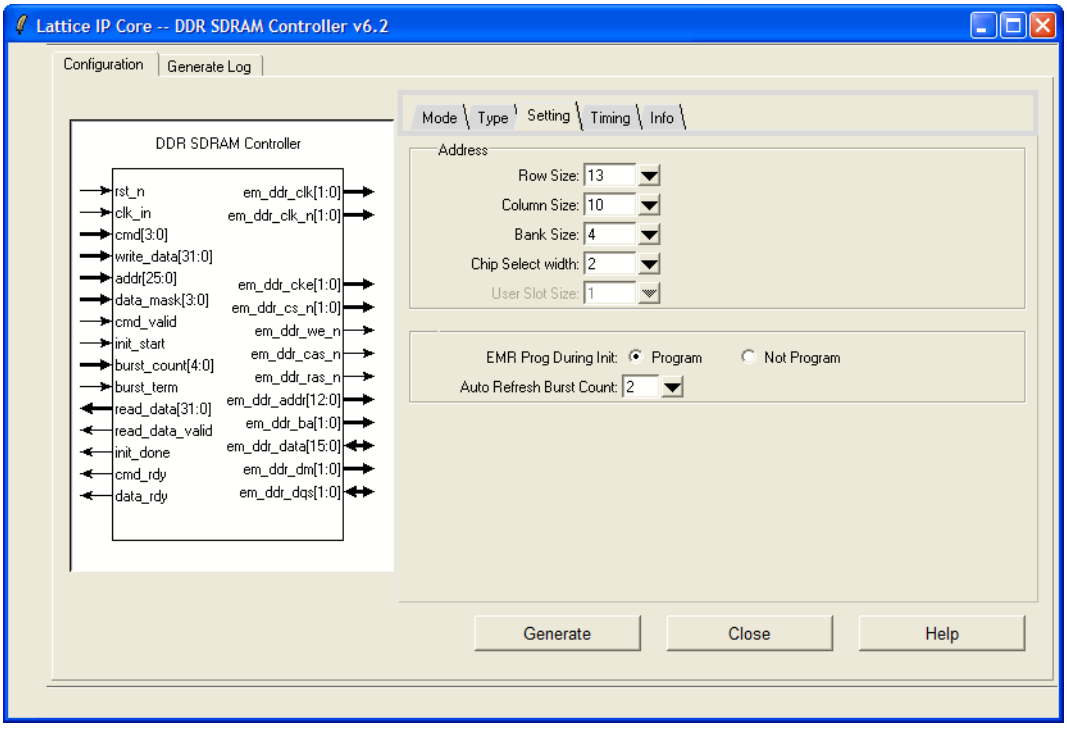


Figure 56: DDR IP: Timing Selection

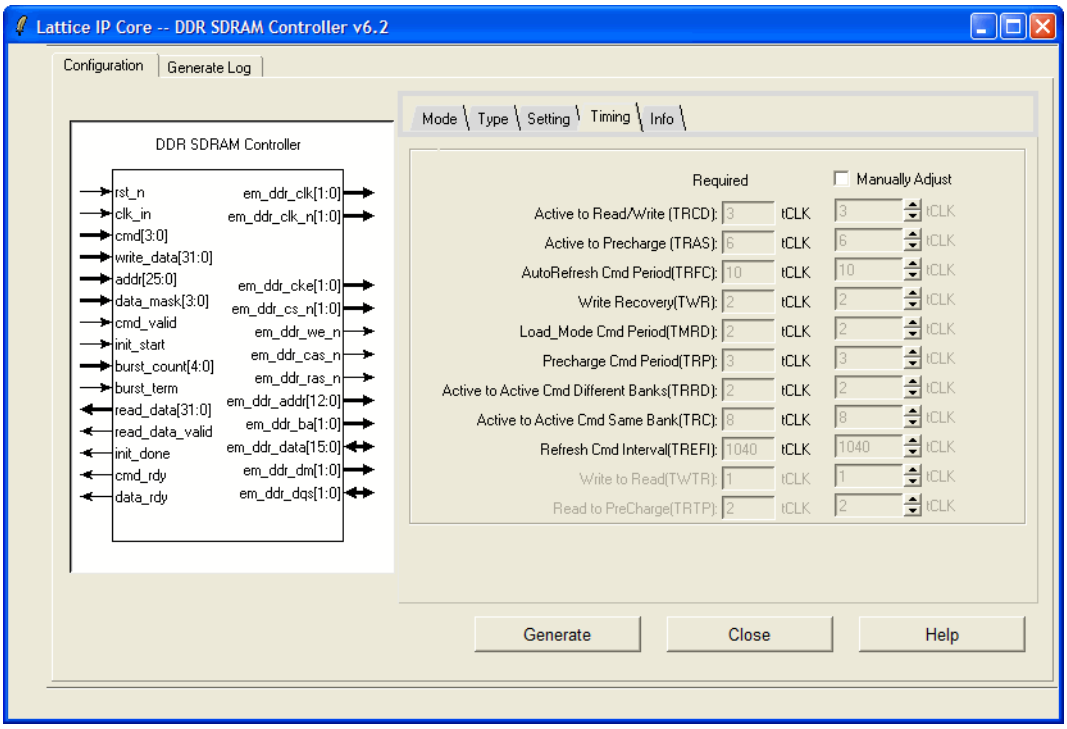
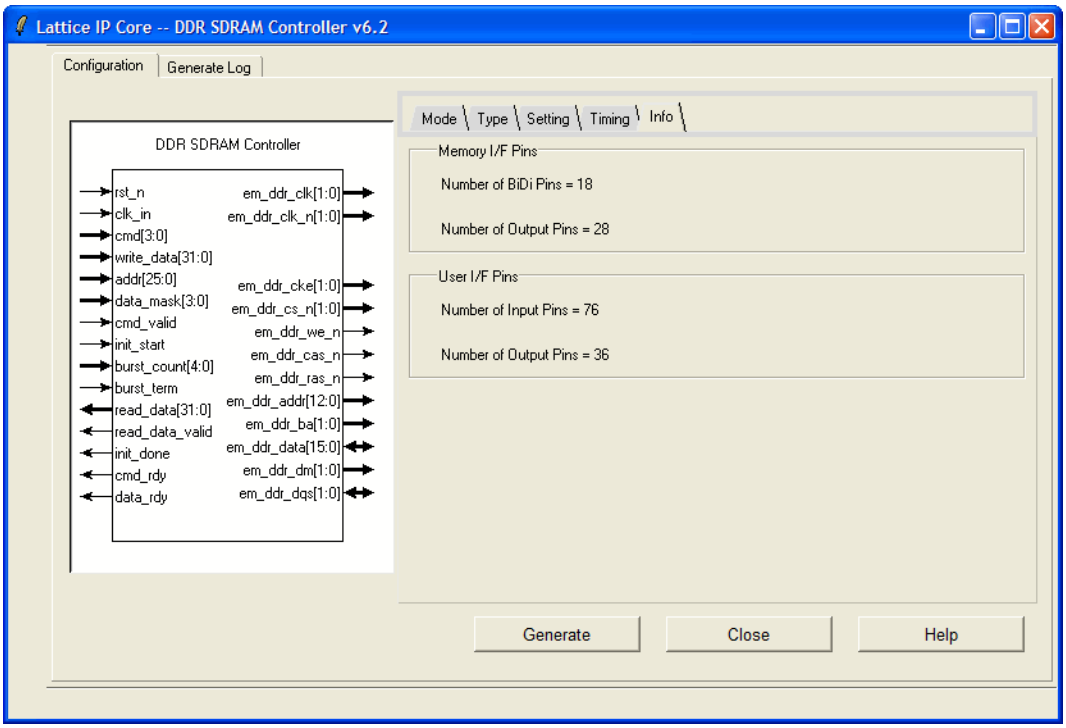


Figure 57: DDR IP: Information



VGA Controller

The VGA controller, implemented as the video_chain_top VHDL entity in the video_pipeline_top.vhd file, consists of several submodules. This section describes the RTL interface and the software interface for the VGA controller.

RTL Interface Table 1 lists the interface signals.

Table 1: VGA Controller (video_chain_top) Interface Signal

Signal	Category	Type	Description
Clock and Reset Signals			
memory_clk	Clock	In	Wishbone clock signal (same as processor platform)
vga_clk	Clock	In	Video clock signal (50MHz)
Reset	Reset	In	Asynchronous reset
wb_vga_ctrl_clk	Clock	In	Wishbone clock signal (same as processor platform)
wb_vga_ctrl_reset	Reset	In	Asynchronous reset
VGA Controller WISHBONE Slave Interface (does not support burst)			
wb_vga_ctrl_adr	WISHBONE	In	VGA Controller Slave: Address Input [31:0]
wb_vga_ctrl_master_data	WISHBONE	In	VGA Controller Slave: Data from master [31:0]
wb_vga_ctrl_stb	WISHBONE	In	VGA Controller Slave: Strobe
wb_vga_ctrl_cyc	WISHBONE	In	VGA Controller Slave: Cycle Valid
wb_vga_ctrl_we	WISHBONE	In	VGA Controller Slave: Write Enable
wb_vga_ctrl_sel	WISHBONE	In	VGA Controller Slave: Byte-Lane Select [3:0]
wb_vga_ctrl_slave_data	WISHBONE	Out	VGA Controller Slave: Data from slave [31:0]
wb_vga_ctrl_ack	WISHBONE	Out	VGA Controller Slave: Ack
wb_vga_ctrl_rty	WISHBONE	Out	VGA Controller Slave: Retry (never asserted)
wb_vga_ctrl_err	WISHBONE	Out	VGA Controller Slave: Error (never asserted)
VGA Controller WISHBONE Master Interface (performs burst reads)			
wb_cyc_o	WISHBONE	Out	VGA Controller Master: Cycle Valid
wb_stb_o	WISHBONE	Out	VGA Controller Master: Strobe
wb_adr_o	WISHBONE	Out	VGA Controller Master: Address Output [31:0]
wb_dat_o	WISHBONE	Out	VGA Controller Master: Data from master [31:0]
wb_sel_o	WISHBONE	Out	VGA Controller Master: Byte-Lane Select [3:0]
wb_we_o	WISHBONE	Out	VGA Controller Master: Write Enable
wb_cti_o	WISHBONE	Out	VGA Controller Master: Cycle Type Identifier [2:0]
wb_bte_o	WISHBONE	Out	VGA Controller Master: Burst Type [1:0]
wb_lock_o	WISHBONE	Out	VGA Controller Master: Lock

Table 1: VGA Controller (video_chain_top) Interface Signal

wb_ack_i	WISHBONE	In	VGA Controller Master: Ack from Slave
wb_err_i	WISHBONE	In	VGA Controller Master: Error from Slave (ignored)
wb_rty_i	WISHBONE	In	VGA Controller Master: Retry from Slave (ignored)
wb_dat_i	WISHBONE	In	VGA Controller Master: Data from slave
VGA Signals (Typically FPGA output)			
red	VGA	Out	VGA Red color [1:0]
blue	VGA	Out	VGA Blue color [1:0]
green	VGA	Out	VGA Green color [1:0]
h_sync_n	VGA	Out	VGA H-Sync (inverted logic) signal
v_sync_n	VGA	Out	VGA V-Sync (inverted logic) signal
Debug Signals (pulled to headers on board for logic analyzer connection)			
snoop_bus	Debug	Out	32-bit debug bus (pulled to headers on board)
snoop_clk_one	Debug	Out	Clock signal (pulled to header on board)

Additionally, the video_chain_top entity has a set of generics that are not described, since you will not need to modify the defaults.

Software Interface The VGA controller slave interface is accessible by the LatticeMico32 processor. It provides access to the VGA controller configuration register interface.

Table 2 lists the available software configurable registers. The offsets in Table 2 are expressed in bytes. These registers support only word writes. Writes that are not words result in unpredictable write values. Reads can be in bytes, half-words, or words.

Table 2: VGA Controller Configuration Registers

Offset	Name	Description
0x00	Control Register	Controls VGA controller operation state
0x04	Status Register	Provides VGA controller operation status
0x08	Video Memory Base Address	Programmable video memory base address
0x0C	Frame Timing Register	Programmable frame timing configuration register
0x10	Line Timing Register	Programmable line timing configuration register
0x14	Max Cycles For Line Burst	Performance counter for measuring maximum cycles taken for line burst

Table 3: Control Register

Bits	Name	R/W	Description
0	Resume	W	Resume bit: Write a 1 to start the VGA controller's video memory reads. Write a 0 to pause the VGA controller's video memory reads. Power-up default is 0. There may be a line-time delay before the VGA controller stops reading from video memory. The Status register bit reflects the status.
31:1	Not Used	N/A	N/A

Table 4: Status Register

Bits	Name	R/W	Description
0	Resume	R	Set to 1 when the VGA controller is enabled to read from the video memory. Set to 0 when the VGA controller is disabled from reading the video memory. Power-up default is 0, reflecting the power-up default of the Resume bit in control register.
31:1	Not Used	N/A	N/A

Table 5: Programmable Video Memory Base Address Register

Bits	Name	R/W	Description
31:0	Video Memory Base Address	R/W	Must be written with 32-bit valid video memory base address. Must be written with a valid memory address before resuming the VGA controller video memory reads. The lower two bits are ignored to generate the word-aligned base address. The power-up default is 0x00000000.

Table 6: Programmable Frame Timing Configuration Register

Bits	Name	R/W	Description
7:0	V-Sync width	R/W	Must be written with a valid value before resuming memory reads. Value represents the number of lines that constitute V-Sync. Power-up default is 0x00.
15:8	V-Sync Front porch width	R/W	Must be written with a valid value before resuming memory reads. Value represents the number of lines that constitute the V-Sync front porch. Power-up default is 0x00.
23:16	V-Sync Back Porch Width	R/W	Must be written with a valid value before resuming memory reads. Value represents the number of lines that constitute the V-Sync back porch. Power-up default is 0x00.
31:24	V-Sync Porch Width	R/W	This value must be the sum of the V-Sync front porch and the V-Sync back porch widths. Power-up default is 0x00.

Table 7: Programmable Line Timing Configuration Register

Bits	Name	R/W	Description
7:0	H-Sync width	R/W	Must be written with a valid value before resuming memory reads. Value represents the number of pixels that constitute the H-Sync. Power-up default is 0x00.
15:8	H-Sync Front porch width	R/W	Must be written with a valid value before resuming memory reads. Value represents the number of pixels that constitute the H-Sync front porch. Power-up default is 0x00.
23:16	H-Sync Back Porch Width	R/W	Must be written with a valid value before resuming memory reads. Value represents number of pixels that constitute the H-Sync back porch. Power-up default is 0x00.
31:24	H-Sync Porch Width	R/W	Must be the sum of the H-Sync front porch and the H-Sync back porch widths. Power-up default is 0x00.

Table 8: Performance Counter Register: Max Cycles for Line Burst

Bits	Name	R/W	Description
9:0	Max Cycles	R/W	Count representing maximum cycles taken for a line-read burst. To reset this count, write 0. Power-up default is 0.
30:10	Not Used	R/W	Not used.
31		R/W	Set to 1 if the maximum cycles value is 0x3FF or exceeds that value. To reset, write a 0.

Design Overview Figure 58 illustrates the VGA controller.

Figure 58: VGA Controller

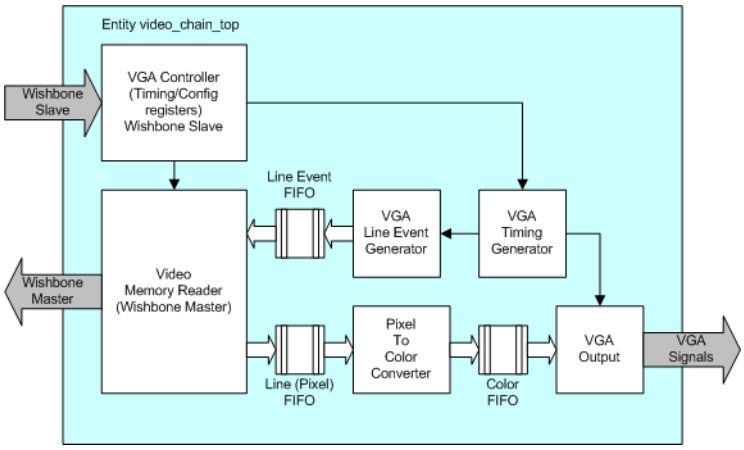


Table 9 lists the VHDL entities that implement the various blocks shown in Figure 53.

Table 9: Design Contents

Design	VHDL Entity	VHDL File
VGA Controller Slave	wishbone_vga_control	wishbone_vga_control.vhd
Video Memory Reader	vid_mem_reader	vid_mem_reader.vhd
VGA Line Event Generator	vga_evt_gen	vga_evt_gen.vhd
VGA Timing Generator	vga	vga_generics.vhd
VGA Output	vga_output	vga_output.vhd
Pixel To Color Converter	pixel_to_color	pixel_to_color.vhd
Line FIFO	pixel_fifo	pixel_fifo.vhd
Color FIFO	color_fifo	color_fifo.vhd
Line Event FIFO	vga_evt_fifo	vga_evt_fifo.vhd

The VGA Line Event Generator, the VGA Timing Generator, and the VGA Output designs have been grouped into a vga_unit structural entity, which is implemented in vga_unit.vhd.

If you have established a set of H/V timing parameters that serve your design's purpose, you can reduce the RTL design size by applying the desired parameters in the video_chain_top generics and then disabling the software-controlled interface by setting the VGA_USE_GENERICS video_chain_top generic value to 1. This step uses an architecture of the VGA timing generation unit that is not controlled by run time to reduce the overall RTL size by 200-300 LUTs.

Reference Information

The following documents provide more information on topics discussed in this guide:

- *LatticeMico32 Software Developer User Guide*, accessible through the LatticeMico32 System Help
- *LatticeMico32 Tutorial*, accessible through the LatticeMico32 System Help
- *Lattice ispLever Core Double Data Rate (DDR1/DDR2) SDRAM Controller – Pipelined*, accessible through IPexpress

Technical Support

If you need technical assistance, you can contact Lattice Semiconductor by any of the following means:

Hotline: 1-800-LATTICE (North America)

+1-503-268-8001 (Outside North America)

Email: techsupport@latticesemi.com

Internet: www.latticesemi.com