XAPP1034 (v1.2) April 13, 2009

Reference System: Accessing Spartan-3AN In-System Flash using XPS SPI

Author: Sundararajan Ananthakrishnan, Brian Hill, Joshua Lu

Abstract

The application note demonstrates how to access the In-System Flash in the Spartan®-3AN FPGA ***after*** the FPGA is configured. The software application uses the XPS InSystem Flash core to access the In-System Flash in the MicroBlaze™ processor-based reference system.

This application note describes the:

- XPS InSystem Flash parameter settings and Port Connections to access the In-System Flash (ISF) memory of Spartan-3AN FPGA.
- Software Application that uses the Xilinx In-System and Serial Flash software library (xilisf) to communicate with the Spartan-3AN ISF Memory.

This reference system is targeted for the Xilinx Spartan-3AN Starter Kit Revision D board.

Included Systems

The reference system for the Xilinx Spartan-3AN Starter Kit Revision D board is included with this application note. The reference system is available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=104114.

Introduction

Spartan-3AN FPGAs have In-system Flash (ISF) memory. The ISF memory array appears to the software application of a Spartan-3AN FPGA as a SPI-based serial Flash memory.

- The ISF memory is primarily designed to configure the FPGA automatically when power is applied. This is new feature in the Spartan-3AN FPGAs.
- The ISF memory is also available to the FPGA after configuration for various purposes, such as:
  - Simple non-volatile data storage.
  - Storage for identifier codes, serial numbers, IP addresses, etc.
  - Storage of MicroBlaze processor code that can be copied into CPU addressable memory, such as DDR2 SDRAM.

This application note explains about how to access the SPI based ISF memory after configuration using the ISF software library. Two applications are provided - one to program with flash with an application, and a bootloader to copy this application out of the flash into DDR2 memory and execute it.

Hardware and Software Requirements

The hardware and software requirements are:

- Xilinx Spartan-3AN Starter Kit Revision D board
- Xilinx Platform USB Cable or Parallel IV Cable
- Serial null-modem cable
- Serial Communications Utility Program, such as HyperTerminal
- Xilinx Platform Studio 10.1.03 (10.1 with service Pack 3)
- Xilinx Integrated Software Environment (ISE)® 10.1.03 (10.1 with service Pack 3)
- Xilinx Software Development Kit 10.1.03

Reference System Specifics

The reference system has the MicroBlaze processor with the caches enabled to use the instruction cache (I-cache) and the data cache (D-cache) from the external DDR2 memory. In addition, the XPS InSystem Flash core with interrupts, the XPS BRAM controller, the XPS UART Lite core with interrupts, and the XPS Interrupt Controller (XPS INTC), cores are used in the reference system.

The reference system is shown the block diagram shown in Figure 1 and the address map of the reference system is shown in Table 1.

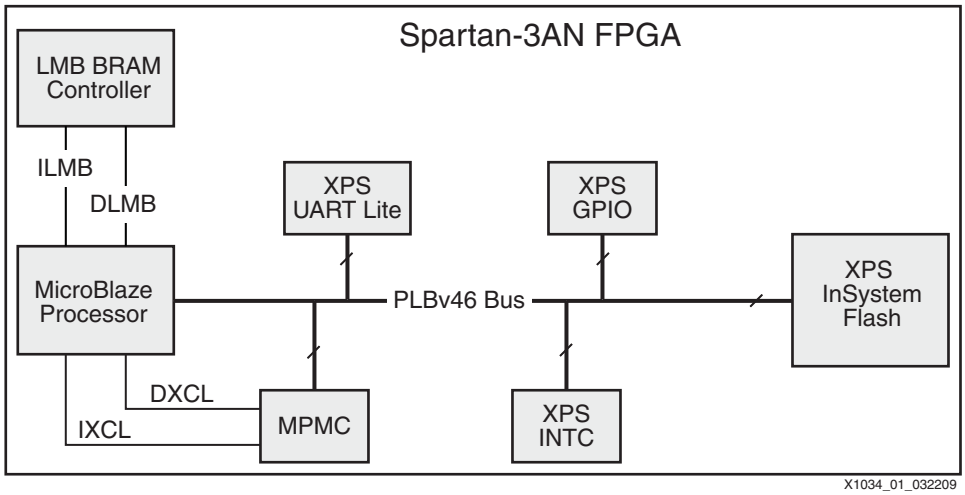


Figure 1: **Reference System Block Diagram**

Address Map

Table 1: **Reference System Address Map**

Instance	Peripheral	Base Address	High Address
dlmb_cntlr / ilmb_cntlr	lmb_bram_if_ctlr	0x00000000	0x00003FFF
LEDs_8Bit	xps_gpio	0x81400000	0x8140FFFF
xps_intc_0	xps intc	0x81800000	0x8180FFFF
xps_insystem_flash_0	xps insystem flash	0x86c00000	0x86c0FFFF
RS232_DTE	xps uartlite	0x84000000	0x8400FFFF
debug_module	mdm	0x84400000	0x8440FFFF
DDR2_SDRAM	mpmc	0x8c000000	0x8FFFFFFF

System Configuration

This Xilinx Spartan-3AN Starter Kit based reference system uses the MicroBlaze processor through the caches enabled. The XPS InSystem Flash core connects the ISF to the MicroBlaze processor through the PLBv46 bus in the reference system. The DDR2 SDRAM is used as the external memory and is configured to allow cacheline transactions from the MicroBlaze processor in the reference system. The Multi-Port Memory Controller (MPMC) is used to connect to the DDR2 SDRAM memory.

The XPS INTC core handles the interrupt signal from the XPS InSystem Flash controller.

The XPS InSystem Flash core is a 32-bit slave peripheral that connects to the PLBv46 and provides access to the ISF using SPI protocol.

Setting the XPS InSystem Flash core parameters

The XPS InSystem Flash core uses the XPS SPI IP core as the base core. The ISF primitive is instantiated in the core along with the XPS SPI IP core instantiation. This core is available as the *Memory and Memory Controller* part as shown in Figure 2.

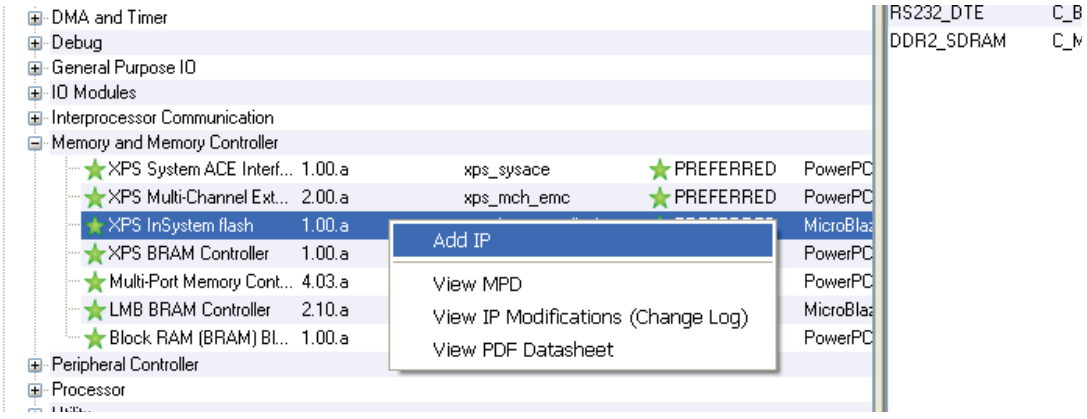


Figure 2: **Add XPS InSystem Flash IP**

Set the parameter, **Include both Receiver and Transmitter FIFOs** to **TRUE** to include the Transmit and Receive FIFOs in the XPS InSystem Flash. Set the the parameter, **Ratio of PLB Clock Frequency To SCK Frequency**, to the default value of **32**. In this system, the PLB bus clock frequency is 62.5 MHz, therefore, the SPI peripheral clock will operate at a frequency of 1.95 MHz. Because the core is the only slave, set the parameter, **Total Number of Slave Select Bits in SS Vector**, to **1**.

The XPS InSystem Flash parameters are shown in Figure 3.

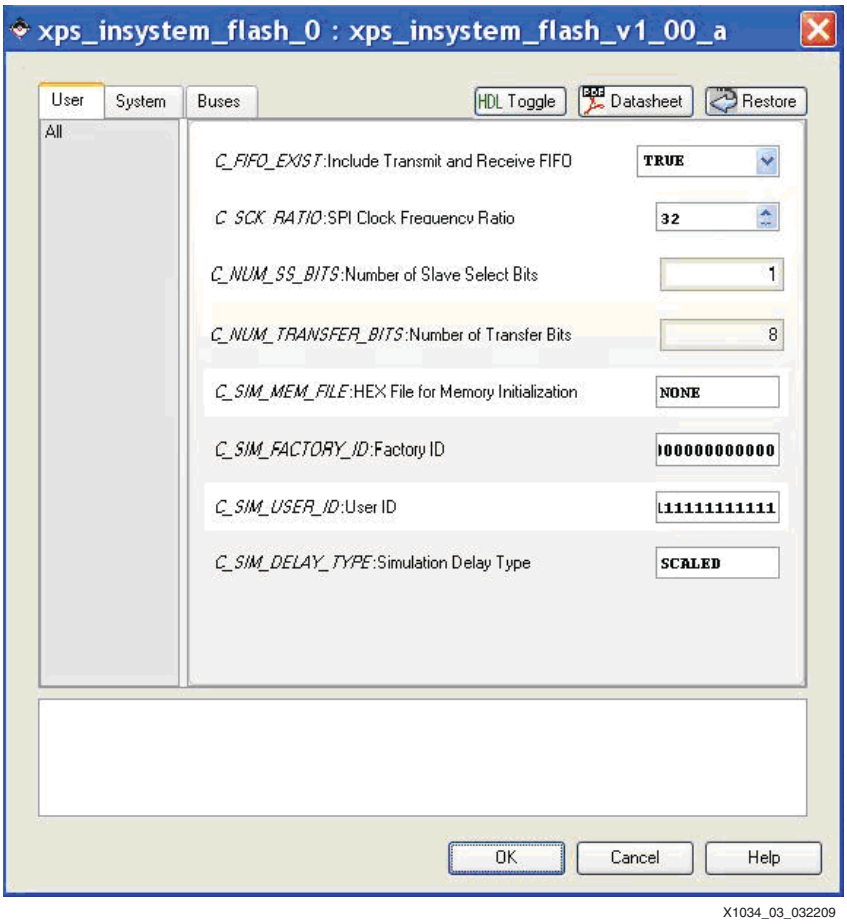


Figure 3: **XPS InSystem Flash Parameter Settings**

XILISF Library

This section describes the Xilinx In-System and Serial Flash Library that accesses the In-System Flash memory and the example application that is used for reading and writing to the ISF memory.

Xilinx In-System and Serial Flash (xilisf) Library

The Xilinx In-System and Serial Flash (xilisf) Library supports the Xilinx In System Flash hardware. The library enables higher layer software (for example, an application) to communicate with the ISF. The library allows the user to write, read, and erase, the ISF. The user can also protect the data stored in the ISF from unwarranted modification by enabling the Sector protection feature.

All the APIs in the library are asynchronous. The transfer is initiated and the control is given back to the user application. The user application must keep track of whether the initiated operation is completed successfully.

The library uses the Xilinx XSpi driver in interrupt-driven mode or polled mode for communicating with the ISF. In interrupt mode, the user application must acknowledge any associated interrupts from the Interrupt Controller.

All the Xilinx ISF library APIs are listed in the Table 2.

Table 2: **Functions of Library Xilisf**

API	Description
int XIsf_Initialize(XSpi *Spi, u32 SlaveSel)	This is the API for initializing the ISF library. This function should be called before any other API in the ISF library can be called
int XIsf_GetStatus(u8 *ReadPtr)	This API returns the contents of the Status Register in the ISF
int XIsf_GetDeviceInfo(u8 *ReadPtr)	This API returns the various JEDEC compatible device information about the ISF which are read from the ISF.
int XIsf_Read(XIsf *InstancePtr, Xlsf_ReadOperation Operation, void *OpParamPtr)	This API reads the data from the ISF.
int XIsf_Write(XIsf *InstancePtr, Xlsf_WriteOperation Operation, void *OpParamPtr)	This API writes the given data to the ISF.
int XIsf_Erase(XIsf *InstancePtr, Xlsf_EraseOperation Operation, u32 Address)	This API is used to erase the contents in the ISF.
int XIsf_SectorProtect(XIsf *InstancePtr, Xlsf_SpOperation Operation, u8 *BufferPtr)	This API is used to perform Sector Protect related operations.
int XIsf_WriteEnable(XIsf *InstancePtr, u8 WriteEnable)	This API Enables/Disables writes to the Intel and STM Serial Flash.
int XIsf_Ioctl (XIsf *InstancePtr, Xlsf_IoctlOperation Operation)	This API configures and controls the Intel and STM Serial Flash.

Note: The address of xilisf stands for the 24-bit physical address of Atmel Flash. UG333 disusses the 24-bit physical address.

To use the xilisf library, the user must choose to include it within their project. In the SDK Software Platform Settings window, under the OS & Library Settings section, select the xilisf library by checking the **Use** column as shown in Figure 4 to add the xilisf library to the system. In addition, the user must configure the xilisf library as shown in Figure 5.

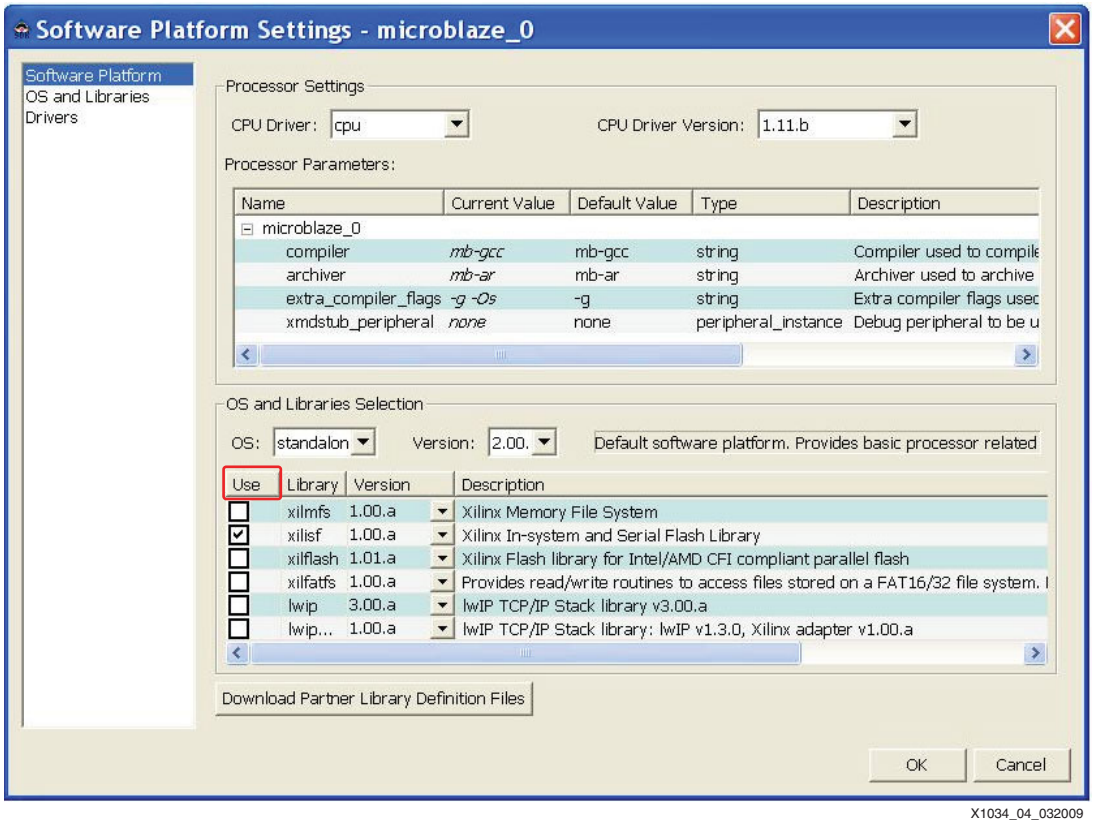


Figure 4: **Software Platform Settings**

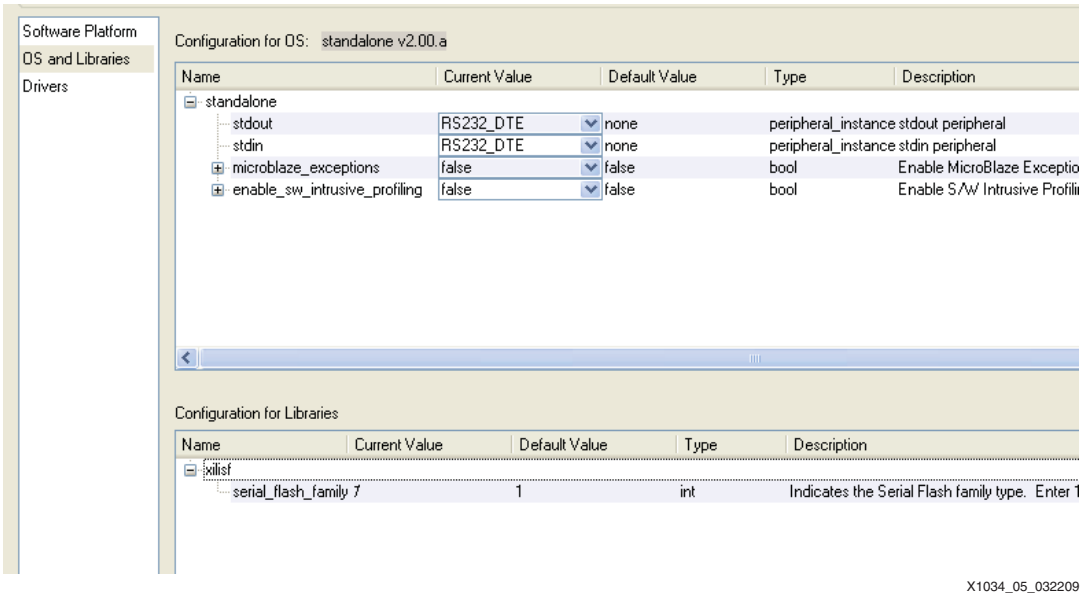


Figure 5: **Configuration of Xillisf Library**

ISF Organization

This application note will discuss the XC3S700AN device **only** as it is configured with the Xilinx Spartan-3AN Starter Kit board. Power-of-2 Addressing mode is not used nor discussed. See the *UG333 Spartan-3AN In-System Flash User Guide* for information on other Spartan 3AN devices.

The In-System Flash provides 8,650,752 bits (1,081,344 bytes) of storage, which is divided into sectors, blocks, and pages as shown in Figure Figure 6. There are a total of 16 sectors, with each sector containing 32 blocks and each block containing 8 pages. One page is 264 bytes.

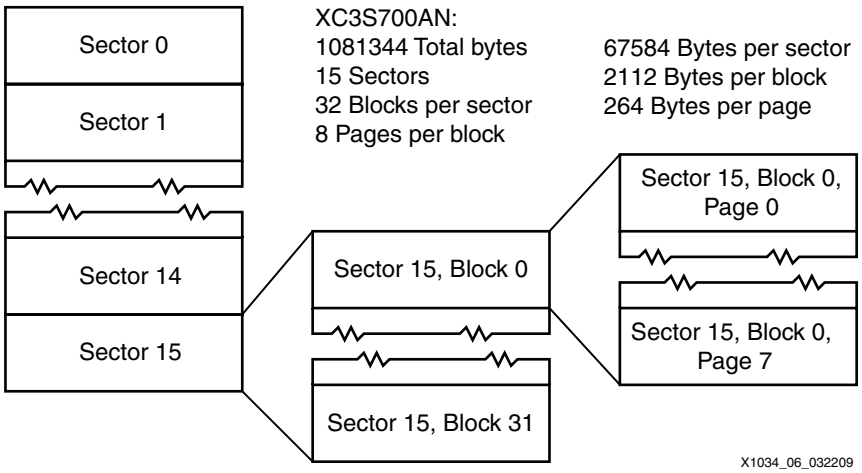


Figure 6: XC3S700AN In-System Flash Organization

The ISF contains enough space for two bitstreams. The system provided with this application note only places one bitstream in the ISF to maximize the space available for user applications.

The uncompressed configuration bitstream for a XC3S700AN device is 2,732,640 bits (341,580 bytes). This consumes 1,294 pages (341,616 bytes) of the flash (pages 0 - 1,293). The flash can not be erased in any quantity smaller than one whole page; it is necessary to round up to the next page to determine the first page usable for other purposes. Page 1294 is the first available for general because the bitstream uses part of page 1293.

Byte offset 341,616 (which can also be referred to as Sector 5, Block 1, Page 6, Byte 0. The user should convert this offset to a 24-bit physical address manually) is where the Program_ISF application, which is provided with this reference system, will program the S-Records of a software application to be loaded by the included bootloader.

Executing the Reference System

To execute the reference system, generate the bitstream and compile the software applications. The bitstream and the compiled software applications for this system are available in the `ready_for_download` directory under the `project root` directory. Configure a HyperTerminal or similar program to use the COM port. Connect the RS232 connector (J27) of the Spartan-3AN Starter Kit board to the COM port via a serial, NULL modem cable.

Set the HyperTerminal to Baud Rate of **9600**, Data Bits to **8**, Parity to **None,** and Flow Control to **None** as shown in Figure 7.

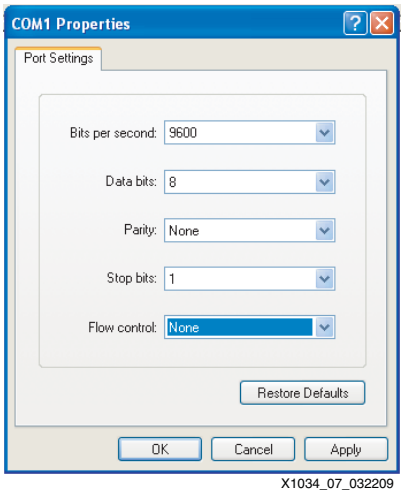


Figure 7: **HyperTerminal Settings**

Executing the Reference System using the Pre-Built Bitstream and the Compiled Software Applications

To execute the system using files inside the `ready_for_download/` directory in the `project root` directory, follow these steps:

1. Change directories to the `ready_for_download` directory.
2. Use iMPACT to download the bitstream by using the following:

   ```
   impact -batch xapp1034.cmd
   ```

3. Invoke XMD and connect to the processor by the following command:

   ```
   xmd -opt xapp1034.opt
   ```

4. Download the desired executable by using the following command:

   ```
   dow <executable name>.elf
   ```

Executing the Reference System from XPS

To execute the system using XPS, follow these steps:

1. Open `system.xmp` inside XPS.
2. Use Hardware **Generate→Bitstream** to generate a bitstream for the system.

Note: If an error is encountered while generating the bitsream, see Figure 8 to modify the ucf file.

3. Use **Software→Launch Platform Studio SDK** to launch SDK.
   a. After SDK has initialized, the Application Wizard is displayed. Click **Cancel**.
   b. Deselect **Project→Build Automatically.**
   c. Choose **File→Import.**
   d. Choose the **Existing Projects into Workspace** wizard.
   e. In the Select root directory field, browse to the Software directory and click **OK**.
   f. The Import Projects window is displayed. Click **Finish** to import all projects.
   g. Build all applications by selecting **Project→Clean**.
   h. Choose **Clean all projects**.

i. Check **Start build immediately**.

j. Click **OK**.

4. Choose the software application to place in BRAM by selecting **Device Configuration** → **Bitstream Settings**. Select **bootloader.elf** from the list box. Click **Save**.
5. Download the bitstream to the board with **Device Configuration**→**FPGA**.
6. The newly compiled binaries are located in the applicable `SDK_projects/<application>/Debug/` subdirectory. Right click on right on the desired project to download the desired application, then select **Run As** →**Run**.

   a. The **Run** window appears. If this is the first time that this application is launched with SDK, click **New**, otherwise select the appropriate project from the list.

   b. Click **Run**.

   c. A warning dialog may appear that there is an existing session. Click **Yes** to proceed with downloading the desired application.

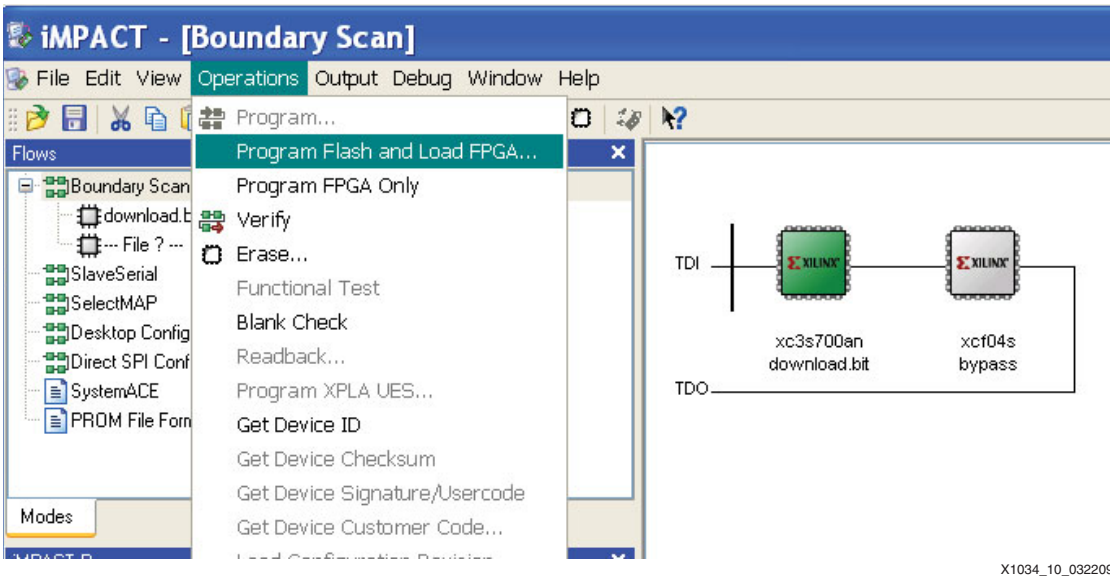
```
604  #NET "SPI_FLASH/SPI_FLASH/XPS_SPI_MODULE_I/I_SPI_MODULE/SS_0<0>" TIG;
605  #NET "SPI_FLASH/SPI_FLASH/XPS_SPI_MODULE_I/I_SPI_MODULE/Serial_Dout" TIG;
606  #NET "SPI_FLASH/SPI_FLASH/XPS_SPI_MODULE_I/I_SPI_MODULE/Serial_Din" TIG;
607
608  NET "xps_insystem_flash_0/xps_insystem_flash_0/XPS_SPI_MODULE_I/I_SPI_MODULE/SS_0<0>" TIG;
609  NET "xps_insystem_flash_0/xps_insystem_flash_0/XPS_SPI_MODULE_I/I_SPI_MODULE/Serial_Dout" TIG;
610  NET "xps_insystem_flash_0/xps_insystem_flash_0/XPS_SPI_MODULE_I/I_SPI_MODULE/Serial_Din" TIG;
```

X1034_08_032209

Figure 8: **Modifying the ucf File**

The Bootloader

A bootloader is provided with this application note. The bootloader copies S-Records (a standard for representing memory data and regions as ASCII text) from the ISF to DRAM. The bootloader code and data are entirely in BRAM. As such, it is placed within the FPGA configuration file `download.bit` (if generated with XPS), or `download_sdk.bit` (if generated with SDK). IMPACT is used to program the `download.bit` file within the ISF. See Figure 9.



X1034_10_032209

Figure 9: **Placing download.bit in the ISF with Impact**

Note: If the bootloader in download.bit is to be automatically used, set up the board to configure the FPGA with the In-System Flash. This is done with jumper block J26 set to Internal Master SPI. See Figure 10.

A previously generated `download.bit` is available in the `ready_for_download` area. If the user has followed the procedures in the "Executing the Reference System" section, a `download_sdk.bit` file will be available in the `SDK_projects/implementation` directory.

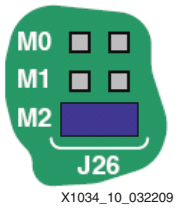


Figure 10: **J26 Internal Master SPI**

Now, when the board is reset, the bootloader will automatically run. At this time, no S-Records have been placed into the ISF for the bootloader to copy, therefore, the bootloader will display an error stating that invalid S-Resods are present:

```
ERROR: SREC line is corrupted
```

Impact is no longer needed, and may be closed at this time.

Program_ISF

This application note discusses placing the TestApp_Peripheral application in the Spartan-3AN In-System Flash. The Program_ISF application is provided to perform this task. Program_ISF accomplishes this by embedding the application to be placed in the ISF, in this case, TestApp_Peripheral, within its own executable image.

The process of embedding an application within Program_ISF begins by converting it to S-Records. This is done with **objcopy** in an EDK shell as shown below:

```
$ mb-objcopy -O srec <the executable>.elf flashimage.srec
```

The text file **`flashimage.srec`** is created. Use the linker to embed this data within the Program_ISF application. The linker can only link files of a format known by the linker. The `flashimage.srec` file is converted to an object file using objcopy:

```
$ mb-objcopy -I binary -O elf32-microblaze -B microblaze \
  --rename-section .data=.rodata,alloc,load,readonly,data,contents \
  flashimage.srec flashimage.o
```

This command will take the input file, **`flashimage.srec`**, and create an object file, **`flashimage.o`**, suitable for linking. The symbol for this S-Record data is determined by the input file name, in this case, `flashimage.srec`. The symbols, _binary_flashimage_srec_start, _binary_flashimage_srec_end, and _binary_flashimage_srec_size, are created in the .rodata section.

This object file is then placed into a static library archive for ease of use.

```
$ mb-ar r libflashimage.a flashimage.o
```

The result is that when Program_ISF is built, it can link with the library flashimage. With the symbols previously discussed, it can directly access the S-Record data.

To simplify this process, the **`make_flash_obj`** script is provided in the `SDK_projects/Program_ISF` directory. Rather than utilize this script by hand, a feature of the Xilinx Software Development Kit (SDK) is used. Prior to building an application, any user-provided script may be run. Configure Program_ISF to utilize this feature as shown in Figure 11.

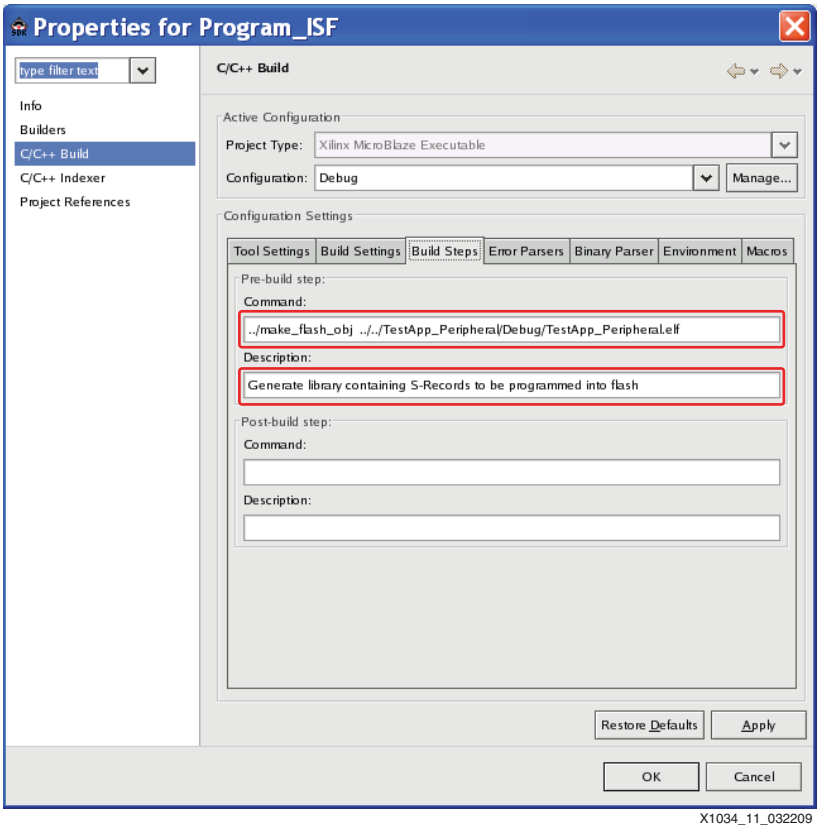


Figure 11: **Using the make_flash_obj Script with SDK**

TestApp_Peripheral is specified as the application to embed. To ensure that this binary is available and up-to-date, configure the Program_ISF application with this as a dependant project, so that it will be built before Program_ISF. See Figure 12.

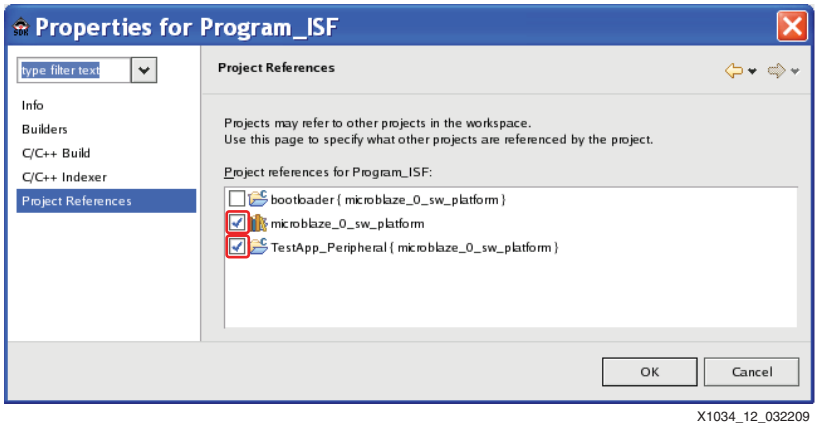


Figure 12: **Program_ISF Dependencies**

Configure the Program_ISF application to link with the library which is generated by make_flash_obj, as well as with the libxilisf library. See Figure 13.

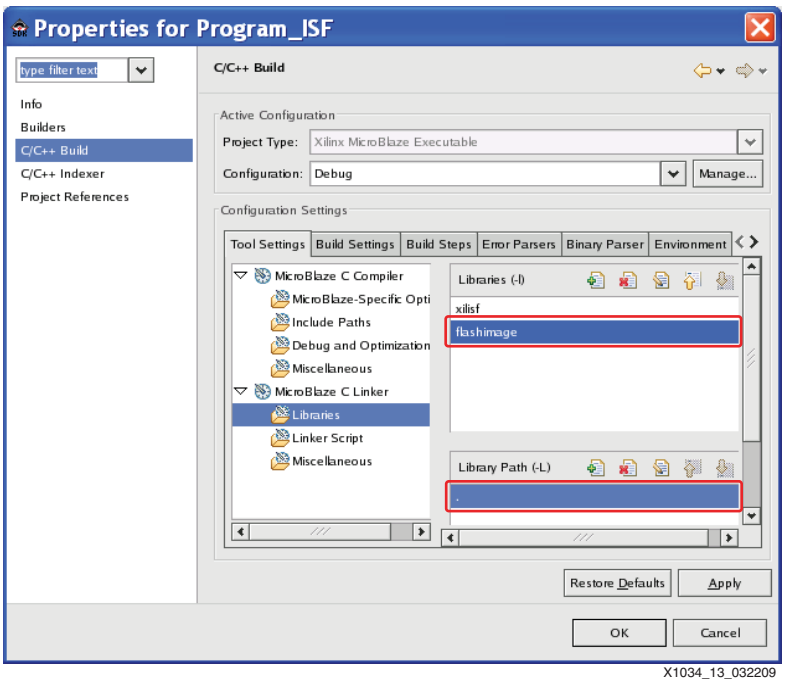


Figure 13: **Program_ISF Linker Settings**

Download the Program_ISF binary built in the "Executing the Reference System" section.

1. Right click on the Program_ISF project, then select **Run As** → **Run**.
2. The Run window appears. If this is the first time that this application is launched with SDK, click **New**, otherwise select the Program_ISF project from the list.
3. Click **Run**.
4. A warning dialog may appear stating that there is an existing session. Click **Yes** to proceed with downloading the desired application.

When run, the expected output is:

```
Programming 22494 bytes of ISF @ offset 341616:
Programming flash starting at sector 5 block 1 page 6
Programming flash starting at sector 5 block 1 page 7
Programming flash starting at sector 5 block 2 page 0
Programming flash starting at sector 5 block 2 page 1
Programming flash starting at sector 5 block 2 page 2
Programming flash starting at sector 5 block 2 page 3
Programming flash starting at sector 5 block 2 page 4
Programming flash starting at sector 5 block 2 page 5
Programming flash starting at sector 5 block 2 page 6
Programming flash starting at sector 5 block 2 page 7
Programming flash starting at sector 5 block 3 page 0
.....
.....
Programming flash starting at sector 5 block 12 page 2
Programming flash starting at sector 5 block 12 page 3
Verifying flash image, Please Wait......
Done.
```

When the board is reset by the user, the bootloader should now copy TestApp_Peripheral out of the ISF and execute it as is shown below:

```
EDK Bootloader:
Copying S-Records from flash offset 0x00053670
Executing program starting at address: 00000000
-- Entering main() --

Running GpioOutputExample() for LEDs_8Bit...
GpioOutputExample PASSED.

Running UartLiteSelfTestExample() for debug_module...
UartLiteSelfTestExample PASSED
-- Exiting main() --
```

Note: If the bootloader cannot fit in the 16K BRAM, the user should set the compiler Optimization option as shown in Figure Figure 14 or the extra compiler flag as shown in Figure Figure 15.

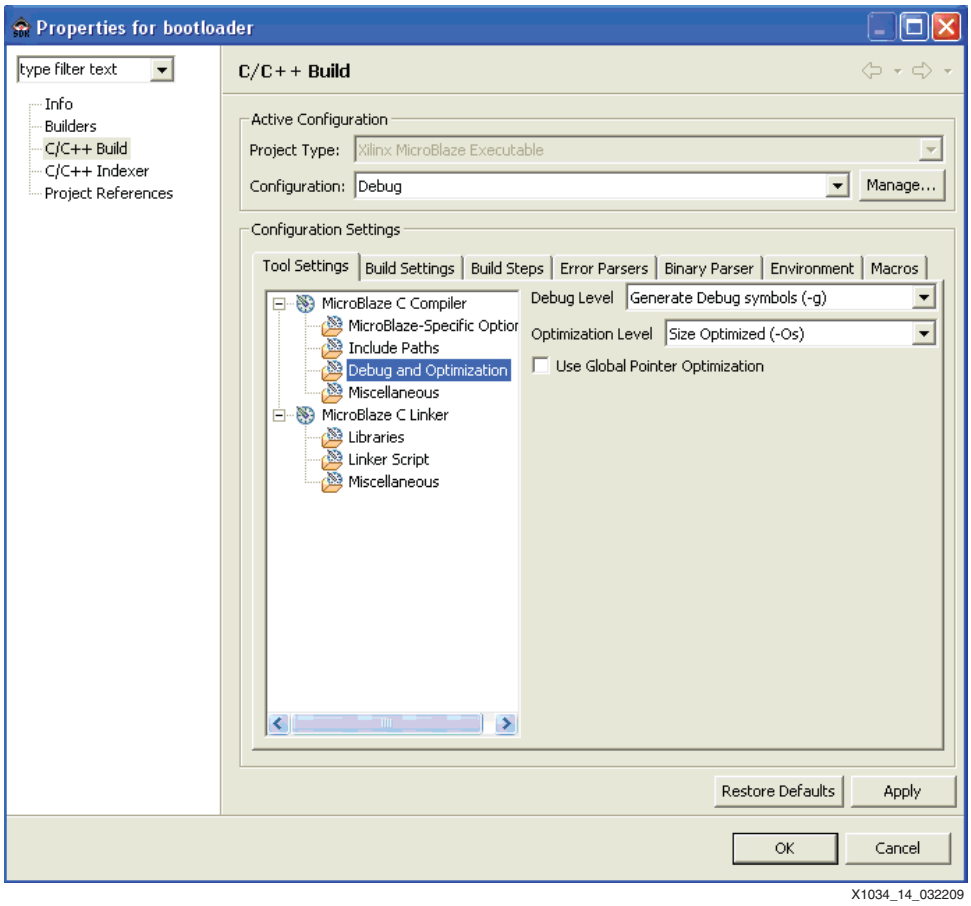


Figure 14: **Debug and Optimization Settings**

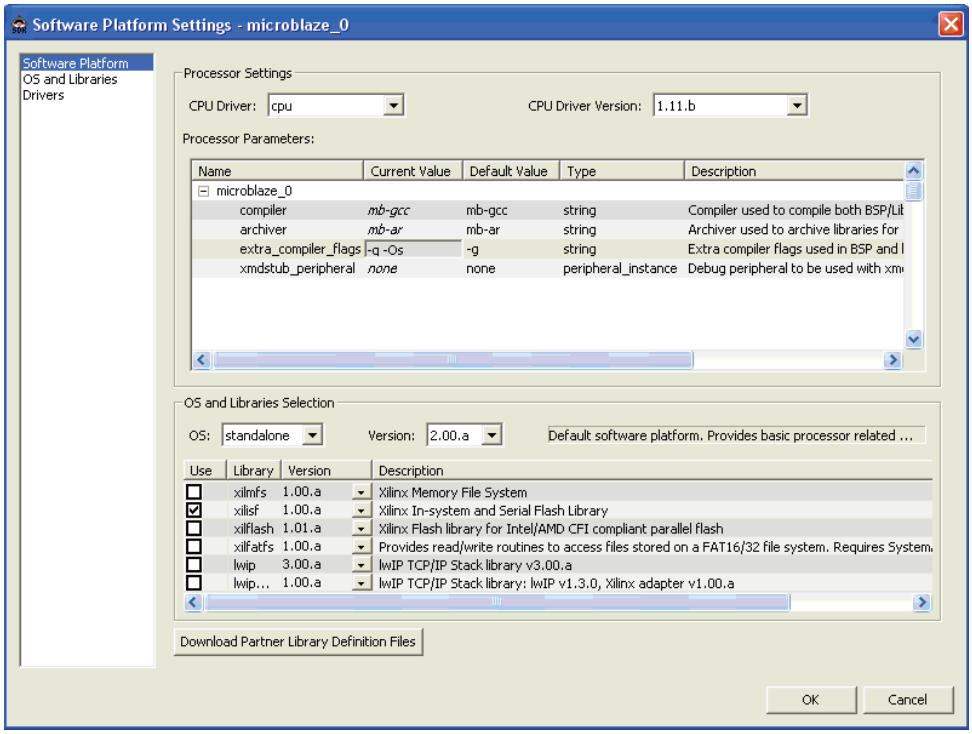


Figure 15: **Extra Compiler Flag Settings**

References

1. DS698 *XPS InSystem Flash v1.00a Product Specification*
2. UG332 *Spartan-3 Generation Configuration User Guide*
3. UG333 *Spartan-3AN FPGA In-System Flash User Guide*
4. UG334 *Spartan-3A/3AN Starter Kit Board User Guide*

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/21/07	1.0	Initial Xilinx release.
3/5/08	1.1	Updated to include bootloader.
4/13/09	1.2	Updated to use XPS Insystem Flash and the officially released xilisf library.

Notice of Disclaimer