XAPP972 (v1.2) September 15, 2009

Updating a Platform Flash PROM Design Revision In-System Using SVF

Contact: Randal Kuramoto

Summary

The Platform Flash XCFP PROM can store multiple design revisions (FPGA bitstreams), of which one design revision can be selected for downloading to an FPGA during a configuration cycle. The ability to store multiple design revisions enables the system to configure an FPGA with one of many functions or with an updated design. This application note demonstrates the process required to update a single design revision in a Platform Flash XCFP PROM using an IEEE Std 1149.1 Boundary-Scan (JTAG) Serial Vector Format (SVF) file. This document includes the requirements for creating and programming an initial image containing two design revisions into the PROM, as well as the requirements for creating and programming only the second revision. The ability to update only the second revision enables the system to add an alternate FPGA design while retaining the initial, base design. The information in this application note complements the details provided in the Platform Flash PROM and Virtex®-5 data sheets and user guides (see [Ref 1], [Ref 2], [Ref 3], [Ref 4], and [Ref 5]). Review of the data sheets and user guides is recommended prior to reading this document.

Introduction

When designing for a flexible FPGA configuration solution, one important factor to consider is how to update the FPGA bitstream in-system. For a PROM-based, FPGA configuration solution, the FPGA bitstream is stored within the PROM's non-volatile memory, and the PROM delivers a bitstream at power-on through the FPGA serial or SelectMAP configuration interface. The Platform Flash XCFP PROM provides the added feature of allowing multiple FPGA bitstreams to be stored and easily accessed from a single PROM. This capability allows the user to store a base bitstream image in one portion of the memory space while actively updating a secondary bitstream image stored in a separate portion of the memory space within the same PROM. With multiple images available to the FPGA, the system can be reconfigured to take on a particular functionality, while safely retaining a base image in the PROM, allowing the system to revert to the base image upon a system power cycle. For example, the base image could contain a design with scaled-down functionality while a customized upgrade image with advanced features could be stored in the second image location and uploaded to the FPGA on-demand.

The Platform Flash XCFP PROM family is available in 8, 16, or 32 Mb densities. The data within each PROM is organized into 8 Mb data blocks, allowing the user to scale each revision in 8 Mb increments. If additional memory is needed to store images that require more than 32 Mb total, then additional PROMs can be cascaded to provide access to additional memory. Each 8 Mb data block can be individually erased, programmed, and verified using the IEEE Std 1149.1 (JTAG) bus (Figure 1).

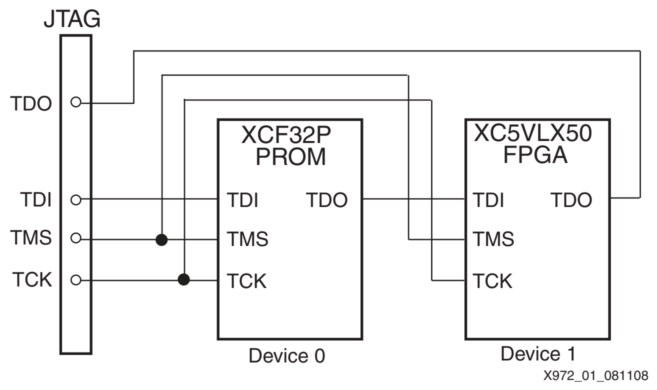

Figure 1: **JTAG Update Chain for XCF32P and XC5VLX50**

For a robust, revision update flow, the PROM is first programmed with an initial image containing two revisions (a primary revision and a secondary revision), with both revisions containing a separate copy of the same, initial FPGA design bitstream. This initial programming sets up the internal control registers, and creates a memory map for both revisions. The secondary revision can be updated at any time, leaving the copy of the initial FPGA design bitstream in the primary revision unaltered. The update sequence must not change critical control registers, or the FPGA's ability to access the primary revision could be corrupted.

This application note specifically demonstrates the Platform Flash XCFP PROM update flow using an example which initially programs an XCF32P PROM with an image containing two revisions: revision 0 spanning blocks 0 to 1, and revision 1 spanning blocks 2 to 3. Each revision contains a single XC5VLX50 bitstream. After programming the initial image, the revision 1 data (stored in blocks 2 to 3) is updated with a new FPGA bitstream design. The revision structure and other PROM settings can be customized to match the settings required for any targeted system. This application note provides a step-by-step guide for generating the required PROM image files (`.mcs`) and PROM JTAG programming files (`.svf`) from the initial and updated FPGA design files (`.bit`). Suggestions and references are provided for executing the `.svf` files. In addition, details for PROM JTAG algorithms are provided for an advanced understanding of the PROM programming sequences.

Platform Flash XCF32P PROM In-System Update

The example in this application note targets an XCF32P containing two revisions. Each revision contains a single uncompressed Virtex-5 FPGA (XC5VLX50) bitstream. The size of each bitstream is 12,556,672 bits, requiring two 8 Mb blocks to fully contain each revision (Figure 2).

The initial image containing the original bitstream replicated for both revisions is created and used to program the device when only one revision is initially available. The copy of the initial bitstream stored in the second revision is used as a placeholder for the future update bitstream. The update image contains the original bitstream stored in the first revision (revision 0) and the update bitstream stored in the second revision (revision 1). The update image is created and used to program revision 1 data into data blocks 2 to 3 in the PROM during the update sequence.

Note: For the case where the user has only one design image initially available, storing a duplicate of that image in the space reserved for the second revision ensures that both sufficient space is actually available to store two full images on the PROM, and that the PROM control registers are initially programmed correctly, allowing the user to access either revision.

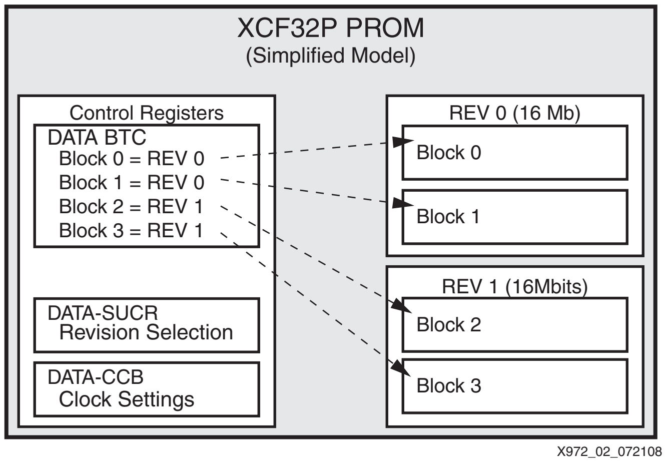

Figure 2: **An XCF32P Containing Two Revisions**

The example in this application note has two phases (in other words, two software flows) that result in files for initial programming and files for in-system update of the PROM, as shown in Figure 3. The first phase generates the initial files to program the PROM with the primary FPGA design as well as reserving a second revision space in the PROM that is ready for future design revision updates. The second phase generates the update files that can program the updated design into the reserved, second revision space of the PROM.

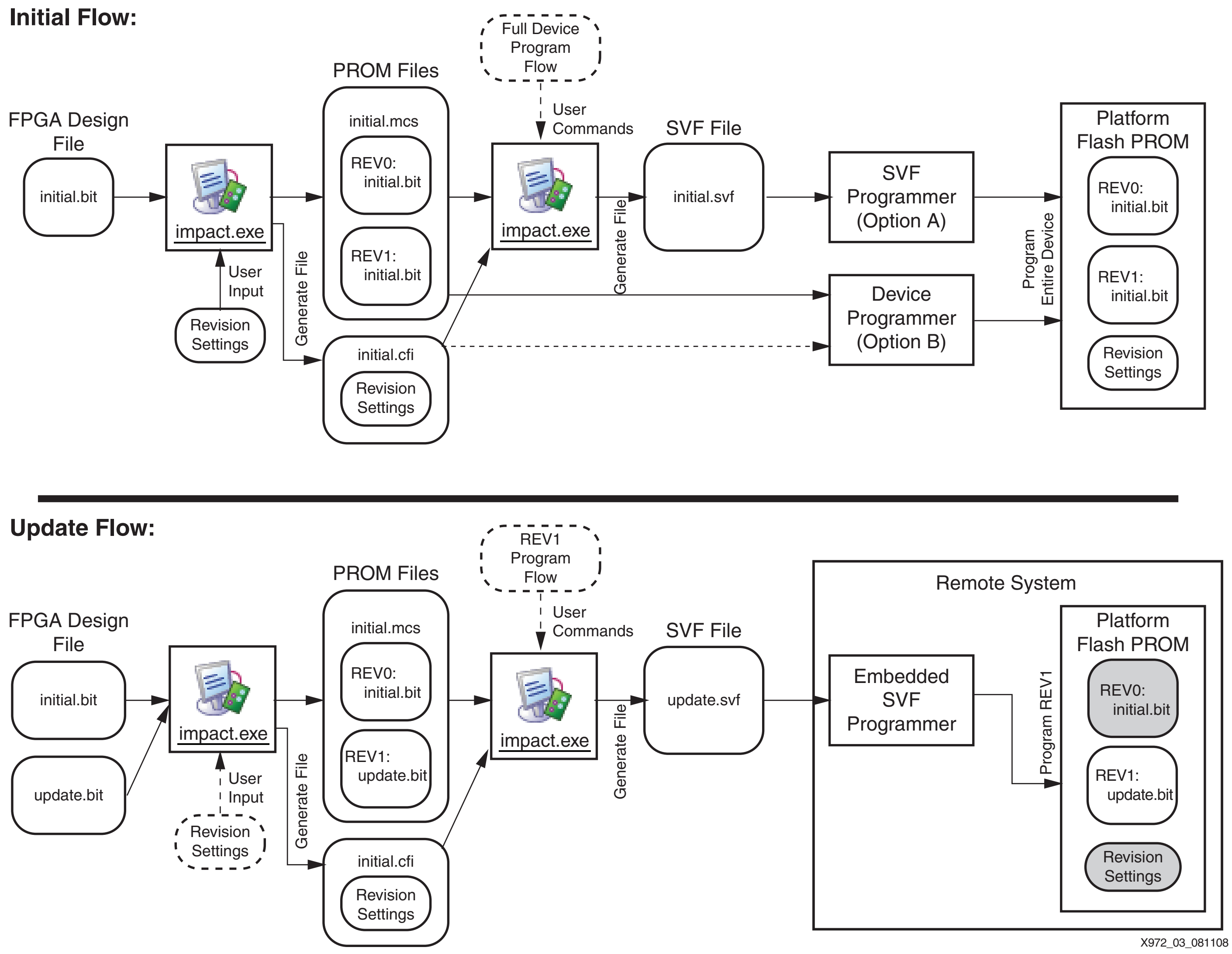

Figure 3: **iMPACT Software Flows for Initial and Update SVF Files**

The initial file generation phase takes the initial FPGA design bitstream as input. The initial file generation flow uses the iMPACT software available in the ISE® Design Suite for the following tasks:

1. Generate the initial PROM image files (`.mcs` and `.cfi` files).
2. Generate the initial PROM JTAG programming file (`.svf`).

The initial PROM image files can be used to program the Platform Flash PROM on a third-party device programmer to set the initial PROM image. Alternatively, the `.svf` file can be used to program the PROM via an SVF-based, in-system programming solution.

The update file generation phase takes the updated FPGA design bitstream and the initial FPGA design bitstream as inputs. The update file generation flow uses the iMPACT tool for the following tasks:

1. Generate the update PROM image files (`.mcs` and `.cfi` files).
2. Generate the update PROM JTAG programming file (`.svf`).

The `.svf` file can be executed via an embedded SVF-based programming solution to in-system update the Platform Flash PROM design revision.

When programming the initial image, in addition to the data blocks, several internal registers must be programmed. The critical registers are DATA-SUCR, DATA-CCB, and DATA-BTC. For details on these registers, see "Appendix A: Boundary Scan for the Platform Flash PROM."

These three registers control the basic functionality of the PROM, such as serial or parallel data output, clocking options, and the default design revision selection. During a single-revision update flow, these three registers must not be changed. After the PROM data blocks and control registers are programmed via the JTAG interface, the revision data can be accessed for FPGA configuration over the serial or parallel data interface by selecting a particular revision.The update image programming sequence must not change the critical control registers, or the update could potentially disable access to the original image.

If the critical control registers (DATA-SUCR, DATA-CCB, and DATA-BTC) are erased but not programmed, then the default erased states will control PROM operation. The default erased state for the DATA-SUCR register causes the PROM to default to selecting revision 3 (`11`). However, this default internal revision selection is overridden by the external revision select pins if the EN_EXT_SEL pin is Low. The default erased state for the DATA-CCB register causes the PROM to default to the slave clock and serial data output settings, corresponding to the FPGA Master Serial configuration mode. The default erased state for the DATA-BTC register causes the PROM to have no data associated with any of the four possible revision locations, so the DATA-BTC must be programmed to access any data in any FPGA configuration mode. Therefore, when the update data is programmed, these critical control registers should all remain in their initial programmed state.

File Generation Requirements

A few basic software options and preferences are outlined here to ensure that the correct files are generated. "Platform Flash XCF32P PROM In-System Update" details the specific system targeted in this application note, while "Initial Image PROM File and SVF Generation" and "Update Image PROM File and SVF File Generation" outline the specific steps required to generate files for the example target system. Refer to the ISE software user manuals for additional details on specific software option settings and preference settings.

Software Support

This application note provides SVF file generation flows for iMPACT software version 10.1. Version 10.1 can execute the generated SVF files to drive a target chain of JTAG devices via a Xilinx download cable. The SVF execution feature in version 10.1 enables cross-validation of the SVF file programming functions with the iMPACT software's verification operations.

FPGA Bitstream Generation

The file generation flows in this application note take FPGA design (bitstream) files as input. A few requirements of the FPGA bitstreams are described in this section.

Bitgen is used to convert the FPGA design into a bitstream (`.bit`), which is used to configure the FPGA. The designer must generate the bitstream with options that are appropriate for the target FPGA. One option that is used for all FPGAs is the startup clock option. The FPGA bitstream to be stored in the PROM must be generated with the BitGen CCLK startup clock option set.

```
bitgen -g startupclk:cclk designName.ncd
```

This option ensures that the FPGA uses CCLK to clock the FPGA configuration startup sequence when loading the bitstream from the PROM.

Alternatively, iMPACT software is capable of auto-correcting the FPGA bitstream startup clock settings during PROM file generation. Within the iMPACT Tool, select **Edit** → **Preferences** → **Configuration Preferences**, and set the option **Startup Clock (FPGA)** to **Automatic Correction**. Alternatively, from within the iMPACT tool's batch mode, add the command:

```
    setPreference -pref StartupClock:AUTO_CORRECTION
```

Note: If the **Startup Clock** was not set to **cclk** during the initial `.bit` file generation, then iMPACT software is capable of automatically correcting the **Startup Clock** setting, but only if the preference is enabled.

PROM File Generation

After the bitstream(s) have been generated, iMPACT software is used to generate the PROM files, which will be stored in the XCFP Platform Flash PROM. For this application, the PROM will be storing two revisions, with one bitstream in each revision. Both bitstreams are stored in a single file in `.mcs` format (Intel MCS-86 hexadecimal object file format). The `.mcs` formatted files contain both address and data. When the `.mcs` file is generated, a corresponding configuration format information (`.cfi`) file is generated, which contains the details on the size and location of each design revision stored in the `.mcs` file. The `.cfi` file also contains information about auxiliary PROM features that are not utilized in this application note, such as decompression (for compressed PROM files) and a free-running clock option (to meet FPGA startup wait clocking requirements for the Digital Clock Manager (DCM) or Digitally Controlled Impedance (DCI)).

SVF File Generation

The FPGA bitstream is used to generate the PROM data file, which in turn is used to generate the `.svf` file. The `.svf` file contains the sequence of JTAG instructions and data shifts required to access the PROM for the selected erase, program, or verify operations.

The following iMPACT tool preferences should be set when generating the `.svf` file for the typical `SVF` player. Within the iMPACT tool, select **Edit** → **Preferences** → **Configuration Preferences** to open the **Configuration Preferences** dialog box. From this dialog box, uncheck the following options: **ConcurrentMode (CPLD PROM), Use HIGHZ instead of BYPASS**, and **Automatic Checksum Insertion (CPLD PROM)**. Next, select **Edit** → **Preferences** → **File Generation Controls**, and uncheck **Use Absolute Time in SVF File**. Alternatively, from within the iMPACT tool's batch mode, add the commands:

```
    setPreference -pref ConcurrentMode:FALSE
    setPreference -pref UseHighz:FALSE
    setPreference -pref AutoSignature:FALSE
    setPreference -pref svfUseTime:FALSE
```

The Platform Flash PROM programming options must be set to match the expected PROM functionality. The PROM options include serial or parallel mode, clocking options, decompression, up to four design revisions, read protection, and write protection. See [Ref 2] for details on the PROM options. For the example files generated by the sequences given in this application note, the PROM is set to output data in parallel mode with an external clock driving the CLK input pin. The PROM is also setup to store two revisions. The revision structure is shown in Figure 2.

Serial Vector File Format

Platform Flash PROM devices are compatible with IEEE Std 1149.1 Boundary-Scan protocols. This application note utilizes the JTAG SVF file to access the PROM for erase, program, and verify operations. SVF was jointly developed by Texas Instruments and Teradyne in 1991 as a standard ASCII format for expressing test patterns for IEEE Std 1149.1 based tests. SVF controls the JTAG test bus using commands that specify the transition of the JTAG TAP controller from one steady state to another. SVF does not describe the explicit state of the JTAG system on every TCK cycle, but rather describes it in terms of transactions conducted between stable states. The target system is left to interpret the SVF instructions, control the operation of the JTAG TAP controller, and clock in the data. Accessing the Platform Flash PROM device for erase, program, or verify operations is simply a matter of executing a particular sequence of SVF instructions as defined by the algorithm for that particular operation.

Caution! The XCFxxP JTAG TAP pause states are not fully compliant with the IEEE Std 1149.1 specification. If a temporary pause of a JTAG shift operation is required, then stop the JTAG TCK clock and maintain the JTAG TAP within the JTAG Shift-IR or Shift-DR TAP state. Do not transition the XCFxxP JTAG TAP through the JTAG Pause-IR or Pause-DR TAP state to temporarily pause a JTAG shift operation.

For more information on the JTAG and SVF specifications, refer to [Ref 6].

Initial Image PROM File and SVF Generation

The initial image programming sequence completely erases and programs the entire device, including the data registers and the critical control registers. After the PROM is completely programmed, either revision can be selected for FPGA configuration.

Initial Image PROM File Generation

iMPACT software can generate a set of PROM files describing the initial design revision architecture and data, taking the initial FPGA bitstream as input and generating the needed PROM files (`.mcs` and `.cfi`).

The `.mcs` and `.cfi` files can be used to program the PROM with the initial image on a third-party device programmer, such as a BPM Microsystems device programmer. Alternatively, the initial PROM files can be used to generate an `.svf` file for in-system programming the initial PROM image on board via SVF-based programming solutions such as third-party Boundary-Scan test tools.

MCS File Generation for the Initial Image Using the iMPACT Software GUI

This section details the process for creating the PROM `.mcs` file for the initial image using the iMPACT software GUI.

1. Launch the iMPACT tool.
2. Select **Edit** → **Launch Wizard** to open the iMPACT **Welcome to iMPACT** dialog box. From this dialog box, select **Prepare a PROM File**, and click **Next>**.
3. From the iMPACT **Prepare PROM Files** dialog box, select **Xilinx PROM**. Then select **MCS**, and set:

Check Sum Fill Value	=	**FF**
PROM File Name	=	**initial**

   Enter or browse to desired project location, and click **Next>**.
4. From iMPACT, select **PROM/Flash Mode**, then select **I am using a Xilinx PROM in a Parallel Mode and the data bus width for my FPGA is:,** and **the same as the data width for my Flash/PROM device**. Click **Next>**.
5. From the iMPACT **Specify Xilinx PROM Device** dialog box, select **Enable Revisioning**, and set:

Number of Revisions	=	**2**
Select a PROM	=	**xcf32p**

   Click **Add** once to add one PROM to the list, and click **Next>**.
6. In the iMPACT **File Generation Summary** dialog box, the summary should read:

```
PROM Type:      Version
File Format:    mcs
Fill Value:     FF
PROM filename: initial
Number of PROMs: 1
Position:       0
Part Name:      xcf32p
```

   After confirming the PROM information, click **Finish**.

7. From the Add Device notification window (**Start adding device file to Revision: 0)**, click **OK** to open the Add Device dialog box. From the dialog box, browse to and select the file `v5_initial.bit`, and click **Open**.
8. From the Add Device notification window (**Would you like to add another design file to Revision:0?**), click **No**.
9. From the next Add Device notification window, browse to and select the file `v5_initial.bit`, and click **Open**.
10. From the **Add Device** notification window (**Would you like to add another design file to Revision:1?**), click **No**.
11. From the **Add Device** notification window (**You have completed the device file entry**), click **OK** to complete.
12. In the iMPACT **Processes** dialog box, double click on **Generate File**.

iMPACT software then generates the `.mcs`/`.cfi` files, `initial.mcs` and `initial.cfi`, and reports "File Generation Succeeded."

MCS File Generation for the Initial Image Using the iMPACT Tool in Batch Mode

This section details the process for creating the PROM `.mcs` file for the initial image using the iMPACT tool in batch mode.

The following batch commands create a `.mcs` file for the initial image:

```
setPreference -pref
StartupClock:AUTO_CORRECTION
setMode -pff
setSubmode -pffversion
addPromDevice -position 1 -name xcf32p
addCollection -name initial
addDesign -version 0 -name 0000
addDeviceChain -index 0
setCurrentDesign -version 0
addDevice -position 1 -file v5_initial.bit
addDesign -version 1 -name 2000
addDeviceChain -index 0
setCurrentDesign -version 1
addDevice -position 1 -file v5_initial.bit
generate -format mcs -fillvalue FF
quit
```

Initial Image SVF Generation

The iMPACT software can be used to generate an `.svf` file, when an SVF-based programming solution (for example, a Boundary-Scan test tool) is used to program the initial PROM image. iMPACT software takes the initial set of PROM (`.mcs` and `.cfi`) files as input and generates an `.svf` file that contains the JTAG sequence for programming the initial image into the PROM.

SVF File Generation for the Initial Image Using the iMPACT Tool GUI

This section details the iMPACT tool GUI flow for creating the `.svf` that erases and programs the initial PROM image.

Setup Boundary-Scan Chain

The Boundary-Scan chain for this example consists of an XCF32P and an XC5VLX50.

To Auto Detect the Chain

1. Connect the programming cable to powered target board, and launch the iMPACT tool.
2. In Boundary-Scan Flow Window, right-click and select **Initialize Chain (CTRL-I)**. iMPACT software then identifies the target chain.

3. In the first Assign New Configuration File dialog box, assign the file `initial.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the second Assign New Configuration File dialog box, select **BYPASS** to bypass the FPGA.

To Manually Set the Chain

1. Launch the iMPACT tool.
2. Anywhere in Boundary-Scan Flow Window, right-click, select **Add Xilinx Device (CTRL-D)**, and add `initial.mcs`. Cursor to the right of the PROM, right-click, select **Add Xilinx Device (CTRL-D),** and add `xc5vlx50.bsd`.

   Note: The file `xc5vlx50.bsd` can be found in the Xilinx ISE software installation directory`%XILINX%\virtex5\data`.

Erase Initial Image SVF Using GUI

1. Select **Output** → **SVF File** → **Create SVF File…**, browse to the correct directory location, and enter **erase_all** as the file name.
2. Right-click on XCF32P, and select **Assign New Configuration File**. Assign the file `initial.mcs` to the XCF32P PROM, and click **OPEN**.
3. In the Boundary-Scan Flow window, right-click on the XCF32P, and select **erase…** to open the **Erase Revision Options** dialog box. From this dialog box, select **All Revisions**, and then click **OK**.

   iMPACT software then creates an `erase_all.svf` file containing the sequence needed to erase the entire PROM and reports "Erase Succeeded."
4. Select **Output** → **SVF File** → **Stop Writing to SVF File**

Program Initial Image SVF Using GUI

1. Select **Output** → **SVF File** → **Create SVF File…**, and browse to the correct directory location. Enter **initial** as the file name, and click **Save**.
2. From the File Generation Mode window (**All device operation will be directed to the file:**), click **OK**.
3. Right-click on **XCF32P**, and select **Assign New Configuration File**. Assign `initial.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the Boundary-Scan Flow window, right-click on the **XCF32P**, and select **Set Programming Properties…**, opening the **Programming Properties – Device 1 Programming Properties** dialog box. The PROM Programming Properties can be modified to match the desired FPGA configuration mode. For this specific example, the FPGA and Platform Flash PROM are setup for FPGA Master SelectMAP x8 configuration mode. In the dialog box, **Parallel Mode** under "PROM Specific Properties" should be checked.
5. In **Programming Properties – Device 1 Programming Properties** dialog box under "Advanced PROM Programming Properties," select **During Configuration: PROM is Slave (clocked externally)**.
6. In the **Programming Properties – Device 1 Programming Properties** dialog box under "Advanced PROM Programming Properties – Design Revision," select **Enable Revision 0 properties** and **Enable Revision 1 properties**. Click **Apply**, and **OK** to complete.

   iMPACT software creates an `initial.svf` file containing the sequence needed to program the entire PROM and report "Programming Succeeded."
7. Select **Output** → **SVF File** → **Stop Writing to SVF File**.

SVF File Generation for the Initial Image Using the iMPACT Tool in Batch Mode

This section details the process for creating the `.svf` that erases and programs the initial PROM image using the iMPACT tool in batch mode.

The following batch commands create an `.svf` file for full erase:

```
setPreference -pref AutoSignature:FALSE
setPreference -pref KeepSVF:TRUE
setPreference -pref ConcurrentMode:FALSE
setPreference -pref UseHighz:FALSE
setPreference -pref svfUseTime:FALSE
setMode -bs
setCable -port svf -file erase_all.svf
addDevice -p 1 -sprom xcf32p -file initial.mcs
addDevice -p 2 -part xc5vlx50
Erase -p 1
Quit
```

The following batch commands create an `.svf` file for full initial program:

```
setPreference -pref AutoSignature:FALSE
setPreference -pref KeepSVF:TRUE
setPreference -pref ConcurrentMode:FALSE
setPreference -pref UseHighz:FALSE
setPreference -pref svfUseTime:FALSE
setMode -bs
setCable -port svf -file initial.svf
addDevice -p 1 -sprom xcf32p -file initial.mcs
addDevice -p 2 -part xc5vlx50
Program -p 1 -parallel -ver 0 -ver 1 -defaultVersion 0
Quit
```

Detailed Descriptions of Initial Image SVF Content

Initial Image Erase

Table 1 outlines the command sequence required to erase the entire PROM. A full device erase includes data and control registers, and requires a maximum of 140 seconds of erase wait time to complete (this maximum does not include the small overhead for JTAG instruction and data shift command sequences).

The full-device-erase SVF sequence generated by iMPACT software contains some operational redundancy. The result does not match the SVF sequence outlined in "UNLOCK/ERASE/BLANK CHECK All SVF Sequence:" exactly. However, the full-device-erase SVF file, generated by iMPACT software, works without any editing.

Table 1: **Full-Device-Erase Sequence**

Step	Operation
1	IDCODE CHECK
2	ENTER ISP MODE
3	UNLOCK ALL(1) [optional]
4	ERASE ALL
5	CHECK STATUS [optional]
6	BLANK CHECK ALL [optional]

Table 1: **Full-Device-Erase Sequence *(Cont'd)***

Step	Operation
7	EXIT ISP MODE

Notes:

1. Unlocks the write protection settings and permits erase/program.

UNLOCK/ERASE/BLANK CHECK All SVF Sequence:

The following example SVF targets a XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain:

```
//================================================================
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0;
HIR 10 TDI (03ff) SMASK (03ff);
HDR 1 TDI (00) SMASK (01);
TDR 0;
//Loading device with 'IDCODE' instruction.
SIR 16 TDI (00fe) SMASK (ffff);
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f5059093) MASK (0fffffff) ;
STATE RESET;
//Loading device with 'ISPEN' instruction.
SIR 16 TDI (00e8) SMASK (ffff) ;
SDR 8 TDI (03) SMASK (ff) ;
// Loading device with a 'XSC_UNLOCK' instruction.
SIR 16 TDI (aa55) SMASK (ffff);
SDR 24 TDI (00003f) SMASK (ffffff) ;
// Loading device with a 'ISC_ERASE' instruction.
// XAPP972: Cannot go to RUN-TEST-IDLE state between SIR and SDR
ENDIR IRPAUSE;
SIR 16 TDI (00ec) SMASK (ffff);
SDR 24 TDI (00003f) SMASK (ffffff);
ENDIR IDLE;
RUNTEST 140000000 TCK;
// Check Device status.
SIR 16 TDI (00e3) SMASK (ffff);
SDR 8 TDI (00) SMASK (ff) TDO (36) MASK (7f) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading device with a 'XSC_BLANK_CHECK' all instruction.
SIR 16 TDI (000d) SMASK (ffff);
RUNTEST 500000 TCK;
SDR 8 TDI (00) SMASK (ff) TDO (00) MASK (3f) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'BYPASS' instruction.
SIR 16 TDI (ffff) SMASK (ffff);
SDR 1 TDI (00) SMASK (01) ;
```

Initial Image Program

Table 2 outlines the command sequence required to program both design revisions and the user controlled registers in the PROM. Programming time varies, depending on the data image and control registers targeted. For a 32 Mb image (131,072 pages x 256 bits/page) with the critical control registers programmed, the maximum programming wait time is approximately 132 seconds (this maximum does not include the small overhead for JTAG instruction and data shift command sequences).

The full-device-program SVF sequence generated by iMPACT software contains some operational redundancy. As a result, it does not match the SVF sequence in "Initial Image Program" exactly. However, the full-device-program SVF file, generated by iMPACT software, works without any editing.

Table 2: **Full-Device-Program Sequence**

Step	Operation
1	IDCODE CHECK
2	ENTER ISP MODE
3	BLANK CHECK ALL [optional]
4	PROGRAM ALL
5	PROGRAM DONE
6	CHECK STATUS [optional]
7	EXIT ISP MODE
8	CONFIG [optional]

PROGRAM All SVF Sequence:

The following example SVF targets a XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain.

```
//================================================================
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0 ;
HIR 10 TDI (03ff) SMASK (03ff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'IDCODE' instruction.
SIR 16 TDI (00fe) SMASK (ffff);
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f5059093) MASK (0fffffff) ;
STATE RESET;
//Loading device with 'ISPEN' instruction.
SIR 16 TDI (00e8) SMASK (ffff) ;
SDR 8 TDI (03) SMASK (ff) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading device with a 'XSC_BLANK_CHECK' all instruction.
SIR 16 TDI (000d) SMASK (ffff);
RUNTEST 500000 TCK;
SDR 8 TDI (00) SMASK (ff) TDO (00) MASK (3f) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
```

```
RUNTEST 50 TCK;
// Loading device with 'XSC_DATA_BTC' instruction.
SIR 16 TDI (00f2) SMASK (ffff);
SDR 32 TDI (fffffba4) SMASK (ffffffff) TDO (00000000) MASK (00000000) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 120 TCK;
// XAPP972: Begin Program Data Blocks Sequence
// XAPP972: First 256-bit page program sequence requires start address,
// XAPP972: but page address automatically increments for pages which
follow.
// Loading device with a 'ISC_DATA_SHIFT' instruction.
SIR 16 TDI (00ed) SMASK (ffff);
SDR 256 TDI (256-bits of data…) SMASK (256-bits of mask data…) ;
// Loading device with a 'ISC_ADDRESS_SHIFT' instruction.
SIR 16 TDI (00eb) SMASK (ffff);
SDR 24 TDI (000000) SMASK (ffffff) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 1000 TCK;
// Loading device with a 'ISC_DATA_SHIFT' instruction.
SIR 16 TDI (00ed) SMASK (ffff);
SDR 256 TDI (256-bits of data…) SMASK (256-bits of mask data…) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 1000 TCK;
// XAPP972: …continue ISC_DATA_SHIFT/ISC_PROGRAM sequence
// XAPP972: until all data pages are programmed …
//
// Loading device with a 'XSC_DATA_SUCR' instruction.
SIR 16 TDI (000e) SMASK (ffff);
SDR 16 TDI (fffc) SMASK (ffff) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 60 TCK;
// Loading device with a 'XSC_DATA_CCB' instruction.
SIR 16 TDI (000c) SMASK (ffff);
SDR 16 TDI (fff9) SMASK (ffff) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 60 TCK;
// Loading device with 'XSC_DATA_DONE' instruction.
SIR 16 TDI (0009) SMASK (ffff);
SDR 8 TDI (c0) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 60 TCK;
// Check Device status.
SIR 16 TDI (00e3) SMASK (ffff);
SDR 8 TDI (00) SMASK (ff) TDO (36) MASK (7f) ;
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'BYPASS' instruction.
SIR 16 TDI (ffff) SMASK (ffff);
SDR 1 TDI (00) SMASK (01) ;
```

Update Image PROM File and SVF File Generation

File generation for updating the PROM is a two-step process that involves the creation of the update PROM image file and an update PROM programming file. The update PROM image file adds the updated FPGA design bitstream to the revision 1 location in the PROM image. The update PROM programming sequence re-programs the revision 1 location with the updated FPGA design bitstream.

The update image programming sequence must not change the critical control registers (DATA-SUCR, DATA-CCB, and DATA-BTC), or the update could potentially disable access to the original image. If the critical control registers are erased but not programmed, then the default erased states will control PROM operation. The iMPACT software can generate SVF that does not change the critical control registers.

Update Image PROM File Generation

The iMPACT software can generate a set of PROM files that includes the updated design revision with the initial, primary revision. iMPACT software takes the updated FPGA design bitstream and initial FPGA design bitstream as inputs and generates the PROM (`.mcs` and `.cfi`) files. The updated PROM image is a representation of the actual PROM contents after the updated image has been programmed into the revision 1 location.

MCS File Generation for the Update Image Using the iMPACT Tool GUI

This section details the process for creating the PROM `.mcs` file for the update image using the iMPACT tool GUI. The update image contains the initial bitstream stored in revision 0, and the update bitstream stored in revision 1.

1. Launch iMPACT software.
2. Select **Edit** → **Launch Wizard** to open the **iMPACT - Welcome to iMPACT** dialog box. From this dialog box, select **Prepare a PROM File**, and click **Next>**.
3. From the **iMPACT - Prepare PROM Files** dialog box, select **Xilinx PROM**. Then select **MCS**, and set:

Check Sum Fill Value	=	**FF**
PROM File Name	=	**update**

   Enter or browse to desired project location, and click **Next>**.
4. From the **iMPACT - Select PROM/Flash Mode**, select **I am using a Xilinx PROM in a Parallel Mode and the data bus width for my FPGA is:**, and **The same as the data width for my Flash/PROM device**. Click **Next>**.
5. From the **iMPACT - Specify Xilinx PROM Device** dialog box, select **Enable Revisioning**, and set:

Number of Revisions	=	**2**
Select a PROM	=	**xcf32p**

   Click **Add** once to add one PROM to the list, and click **Next>**.
6. In the **iMPACT - File Generation Summary** dialog box, the summary should read:

   ```
   PROM Type:     Version
   File Format:   mcs
   Fill Value:    FF
   PROM filename: update
   Number of PROMs: 1
   Position:      0
   Part Name:     xcf32p
   ```

   After confirming the PROM information, click **Finish**.
7. From the Add Device notification window (**Start adding device file to Revision: 0)**, click **OK** to open the Add Device dialog box. From the dialog box, browse to and select the file `v5_update.bit`, and click **Open**.

8. From the Add Device notification window (**Would you like to add another design file to Revision:0?**), click **No**.
9. From the next Add Device notification window, browse to and select the file `v5_update.bit`, and click **Open**.
10. From the Add Device notification window (**Would you like to add another design file to Revision:1?**), click **No**.
11. From the Add Device notification window (**You have completed the device file entry**), click **OK** to complete.
12. In the **iMPACT Processes** dialog box, double click on **Generate File**.

    iMPACT software then generates the `.mcs`/`.cfi` files, `update.mcs` and `update.cfi`, and reports "File Generation Succeeded."

MCS File Generation for the Update Image Using IMPACT in Batch Mode

This section details the process for creating the PROM `.mcs` file for the update image using iMPACT software in batch mode.

The following batch commands create an `.mcs` file for the update image:

```
setPreference -pref
  StartupClock:AUTO_CORRECTION
setMode -pff
setSubmode -pffversion
addPromDevice -position 1 -name xcf32p
addCollection -name update
addDesign -version 0 -name 0000
addDeviceChain -index 0
setCurrentDesign -version 0
addDevice -position 1
  -file v5_update.bit
addDesign -version 1 -name 2000
addDeviceChain -index 0
setCurrentDesign -version 1
addDevice -position 1
  -file v5_update.bit
generate -format mcs -fillvalue FF
Quit
```

Update Image SVF File Generation

iMPACT software takes the updated set of PROM (`.mcs` and `.cfi`) files as input, extracts only the updated FPGA design from the revision 1 location, and generates an `.svf` file that contains the JTAG sequence for reprogramming only the revision 1 location in the PROM. The `.svf` does not modify the PROM settings or initial FPGA design bitstream in the PROM revision 0 location. In this example, application of the update `.svf` to the Platform Flash PROM, which has been previously programmed to the initial PROM image, results in a PROM containing the `V5_initial.bit` file stored in revision 0, and the `V5_update.bit` file stored in revision 1.

The flow outlined in this section creates one file, `rev1_update.svf`, which will update the revision 1 data, as well as a file, `rev1_initial.svf`, which will revert the revision 1 data back to the initial design.

SVF File Generation for the Update Image Using the iMPACT Software GUI

This section details the process for creating the `.svf` that erases and programs the update PROM image using the iMPACT software GUI. The `.svf` file for the update image only changes the data stored in revision 1 and must not change the critical control registers in the Platform Flash PROM.

Setup Boundary-Scan Chain:

The Boundary-Scan chain for this example consists of an XCF32P and a XC5VLX50.

To Auto Detect the Chain

1. Connect the programming cable to powered target board, and launch iMPACT software.
2. In Boundary-Scan Flow Window, right-click and select **Initialize Chain (CTRL-I)**. iMPACT software then identifies the target chain.
3. In the first Assign New Configuration File dialog box, assign the file `initial.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the second Assign New Configuration File dialog box, select **BYPASS** to bypass the FPGA.

To Manually Set the Chain

1. Launch iMPACT software.
2. In Boundary-Scan Flow Window, right-click and select **Add Xilinx Device (CTRL-D)**, and add `initial.mcs`. Cursor to the right of the PROM, and right-click and select **Add Xilinx Device (CTRL-D)** and add `xc5vlx50.bsd`.

Erase Revision 1

1. Select **Output** → **SVF File** → **Create SVF File…**, browse to the correct directory location, and enter erase_rev1 as the file name, and click Save.
2. From the File Generation Mode window (**All device operation will be directed to the file:** `erase_rev1.svf`), click **OK**.
3. Right-click on **XCF32P**, and select **Assign New Configuration File**. Assign the file `update.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the Boundary-Scan Flow window, right-click on the **XCF32P**, and select **Set Erase Properties…** to open the Erase Properties – Device 1 Erase Properties dialog box. From this dialog box, select **Erase Selected Revision**, select **Revision 1**. Click **Apply** and **OK** to complete.
5. In the Boundary-Scan Flow window, right-click on the **XCF32P**, and select **Erase**.

   iMPACT software then creates `erase_rev1.svf`, containing the sequence needed to erase revision 1 date from the PROM and report "Erase Succeeded."
6. Select **Output** → **SVF File** → **Stop Writing to SVF File** to complete the process.

Program Revision 1 Update

1. Select **Output** → **SVF File** → **Create SVF File…**, and browse to the correct directory location. Enter **rev1_update** as the file name, and click **Save**.
2. From the File Generation Mode window (**All device operation will be directed to the file:** `rev1_update.svf`), click **OK**.
3. Right-click on **XCF32P**, and select **Assign New Configuration File**. Assign `update.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the Boundary-Scan Flow window, right-click on the **XCF32P**, and select **Set Programming Properties…**, opening the Programming Properties – Device 1 Programming Properties dialog box. The PROM Programming Properties can be modified to match the desired FPGA configuration mode. For this specific example, the FPGA and Platform Flash PROM are setup for FPGA Master SelectMAP x8 configuration mode. In the dialog box, **Parallel Mode** under "PROM Specific Properties" should be checked.
5. In Programming Properties – Device 1 Programming Properties dialog box under "Advanced PROM Programming Properties," select **During Configuration: PROM is Slave (clocked externally)**.
6. In the Programming Properties – Device 1 Programming Properties dialog box under "Advanced PROM Programming Properties – Design Revision," select **Enable Revision 1 properties** (the **Enable Revision 1 properties** box should be unchecked).

7. In the Programming Properties – Device 1 Programming Properties dialog box, select **Safe design Revision In-System Update**. Click **Apply**, and **OK** to complete.

   iMPACT software creates a rev1`_update.svf` file containing the sequence needed to program the entire PROM and report "Programming Succeeded."
8. Select **Output** → **SVF File** → **Stop Writing to SVF File** to complete the process.

   Note: Always erase before programming.

Program Revision 1 Initial

1. Select **Output** → **SVF File** → **Create SVF File…**, and browse to the correct directory location. Enter **rev1_initial** as the file name, and click **Save**.
2. From the File Generation Mode window (**All device operation will be directed to the file:** `rev1_initial.svf`), click **OK**.
3. Right-click on **XCF32P**, and select **Assign New Configuration File**. Assign `initial.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the Boundary-Scan Flow window, right-click on the **XCF32P**, and select **Set Programming Properties…**, opening the Programming Properties – Device 1 Programming Properties dialog box. The PROM Programming Properties can be modified to match the desired FPGA configuration mode. For this specific example, the FPGA and Platform Flash PROM are setup for FPGA Master SelectMAP x8 configuration mode. In the dialog box, **Parallel Mode** under "PROM Specific Properties" should be checked.
5. In Programming Properties – Device 1 Programming Properties dialog box under "Advanced PROM Programming Properties," select **During Configuration: PROM is Slave (clocked externally)**.
6. In the Programming Properties – Device 1 Programming Properties dialog box under "Advanced PROM Programming Properties – Design Revision," select **Enable Revision 1 properties** (the **Enable Revision 2 properties** box should be unchecked).
7. In the Programming Properties – Device 1 Programming Properties dialog box, select **Safe design Revision In-System Update**. Click **Apply**, and **OK** to complete.

   iMPACT software creates a `rev1_initial.svf` file containing the sequence needed to program the entire PROM and report "Programming Succeeded."
8. Select **Output** → **SVF File** → **Stop Writing to SVF File** to complete the process.

Note: Always erase before programming.

SVF File Generation for the Update Image Using the iMPACT Tool in Batch Mode

This section details the process for creating the `.svf` that erases and programs the update PROM image using iMPACT software in batch mode. The `.svf` file for the update image only changes the data stored in revision 1.

The following batch commands create a `.svf` file for the erase revision 1:

```
setPreference -pref AutoSignature:FALSE
setPreference -pref KeepSVF:TRUE
setPreference -pref ConcurrentMode:FALSE
setPreference -pref UseHighz:FALSE
setPreference -pref svfUseTime:FALSE
setMode -bs
setCable -port svf -file erase_rev1.svf
addDevice -p 1 -sprom xcf32p -file initial.mcs
addDevice -p 2 -part xc5vlx50
Erase -p 1 -ver 1
Quit
```

The following batch commands create an `.svf` file for the program revision 1 update:

```
setPreference -pref AutoSignature:FALSE
setPreference -pref KeepSVF:TRUE
setPreference -pref ConcurrentMode:FALSE
setPreference -pref UseHighz:FALSE
setPreference -pref svfUseTime:FALSE
setMode -bs
setCable -port svf -file rev1_update.svf
addDevice -p 1 -sprom xcf32p -file update.mcs
addDevice -p 2 -part xc5vlx50
Program -p 1 -parallel -ver 1 -defaultVersion 0 -inSystemUpdate
quit
```

The following batch commands create an `.svf` file for the program revision 1 initial:

```
setPreference -pref AutoSignature:FALSE
setPreference -pref KeepSVF:TRUE
setPreference -pref ConcurrentMode:FALSE
setPreference -pref UseHighz:FALSE
setPreference -pref svfUseTime:FALSE
setMode -bs
setCable -port svf -file rev1_initial.svf
addDevice -p 1 -sprom xcf32p -file initial.mcs
addDevice -p 2 -part xc5vlx50
Program -p 1 -parallel -ver 1 -defaultVersion 0 -inSystemUpdate
quit
```

Detailed Descriptions of Update Image SVF Content

Update Image Erase

Table 3 outlines the command sequence required to erase a single design revision. The 8 Mb data blocks are erased one block at a time and require a maximum of 30 seconds to successfully complete erasing each block.

The register values can be changed but it is inadvisable. For the safe-update flow, the objective is to minimize the risk associated with changing registers, affecting the ability to access at least one of the revisions. If a problem/interrupt occurs while re-writing the critical registers, then the PROM could be left in a state where no data is accessible by the FPGA until the critical registers are reprogrammed.

Table 3: **Single-Revision Erase Sequence**

Step	Operation
1	IDCODE CHECK
2	ENTER ISP MODE
3	ERASE TARGETED BLOCKS(1) [optional]
4	CHECK STATUS [optional]
5	BLANK CHECK TARGETED BLOCKS [optional]
6	EXIT ISP MODE

Notes:

1. Write protected blocks cannot be touched without using UNLOCK.

ERASE/BLANK_CHECK Single-Revision SVF Sequence:

The following example SVF targets a XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain.

```
//=================================================================
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0 ;
HIR 10 TDI (03ff) SMASK (03ff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'IDCODE' instruction.
SIR 16 TDI (00fe) SMASK (ffff);
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f5059093) MASK (0fffffff) ;
STATE RESET;
//Loading device with 'ISPEN' instruction.
SIR 16 TDI (00e8) SMASK (ffff) ;
SDR 8 TDI (03) SMASK (ff) ;
// Loading device with a 'ISC_ERASE' instruction.
// XAPP972: Cannot go to RUN-TEST-IDLE state between SIR and SDR
ENDIR IRPAUSE;
SIR 16 TDI (00ec) SMASK (ffff);
SDR 24 TDI (00000c) SMASK (ffffff);
ENDIR IDLE;
RUNTEST 60000000 TCK;
// Check Device status.
SIR 16 TDI (00e3) SMASK (ffff);
SDR 8 TDI (00) SMASK (ff) TDO (36) MASK (7f) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading device with a 'XSC_BLANK_CHECK' block 2-3 instruction.
SIR 16 TDI (000d) SMASK (ffff);
RUNTEST 500000 TCK;
SDR 8 TDI (00) SMASK (ff) TDO (33) MASK (0c) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'BYPASS' instruction.
SIR 16 TDI (ffff) SMASK (ffff);
SDR 1 TDI (00) SMASK (01) ;
```

Update Image Program

Table 4 outlines the command sequence required to program only a single design revision into the PROM (refer to Figure 5 for the TAP controller states). The 8 Mb data blocks are programmed 256 bits at a time and require a maximum of 1 ms to successfully complete programming each 256-bit page.

Table 4: **Single-Revision Program Sequence**

Step	Operation
1	IDCODE CHECK
2	ENTER ISP MODE
3	BLANK CHECK TARGETED BLOCKS [optional]
4	PROGRAM TARGETED BLOCKS
5	PROGRAM DONE TARGETED BLOCKS
6	CHECK STATUS [optional]

Table 4: **Single-Revision Program Sequence *(Cont'd)***

Step	Operation
7	EXIT ISP MODE
8	CONFIG [optional]

PROGRAM Single-Revision SVF Sequence:

The following example SVF targets an XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain:

```
//=================================================================
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0 ;
HIR 10 TDI (03ff) SMASK (03ff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 16 TDI (00fe) SMASK (ffff);
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f5059093) MASK (0fffffff) ;
STATE RESET;
//Loading device with 'ispen' instruction.
SIR 16 TDI (00e8) SMASK (ffff) ;
SDR 8 TDI (03) SMASK (ff) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading device with a 'XSC_BLANK_CHECK' block 2-3 instruction.
SIR 16 TDI (000d) SMASK (ffff);
RUNTEST 500000 TCK;
SDR 8 TDI (00) SMASK (ff) TDO (33) MASK (0c) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// XAPP972: Begin Program Data Blocks Sequence
// XAPP972: First 256-bit page program sequence requires start address,
// XAPP972: but page address automatically increments for pages which
follow.
// Loading device with a 'ISC_DATA_SHIFT' instruction.
SIR 16 TDI (00ed) SMASK (ffff);
SDR 256 TDI (256-bits of data…) SMASK (256-bits of mask data…) ;
// Loading device with a 'ISC_ADDRESS_SHIFT' instruction.
SIR 16 TDI (00eb) SMASK (ffff);
SDR 24 TDI (200000) SMASK (ffffff) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 1000 TCK;
// Loading device with a 'ISC_DATA_SHIFT' instruction.
SIR 16 TDI (00ed) SMASK (ffff) ;
SDR 256 TDI (256-bits of data…) SMASK (256-bits of mask data…) ;
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 1000 TCK;
// XAPP972: …continue ISC_DATA_SHIFT/ISC_PROGRAM sequence
// XAPP972: until all revision 1 data pages are programmed …
//
```

```
// Loading device with 'XSC_DATA_DONE' instruction.
SIR 16 TDI (0009) SMASK (ffff) ;
SDR 8 TDI (f3) SMASK (ff);
// Loading device with a 'ISC_PROGRAM' instruction.
SIR 16 TDI (00ea) SMASK (ffff);
RUNTEST 60 TCK;
// Check Device status.
SIR 16 TDI (00e3) SMASK (ffff);
SDR 8 TDI (00) SMASK (ff) TDO (36) MASK (7f) ;
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff) ;
RUNTEST 50 TCK;
// Loading devices with 'BYPASS' instruction.
SIR 16 TDI (ffff) SMASK (ffff);
SDR 1 TDI (00) SMASK (01) ;
```

Executing SVF Files

The file generation flows in this application note result in SVF files. The SVF files provide a standard format in which to describe the JTAG sequences that are required to program the Platform Flash PROM with specific data. The SVF can be interpreted directly for conversion to board-level signals that drive the JTAG interface of the Platform Flash PROM. For the initial PROM image, third-party JTAG test tools can execute SVF as part of the board-level test program. For in-system update, reference SVF-based programming solutions are available in the form of C code or HDL code for embedding within a system. These reference solutions convert the SVF to more compact binary formats (for example, XSVF or ACE file formats) and execute the binaries via embedded C code or HDL to drive the board-level JTAG interface of the Platform Flash PROM. For details on these SVF-based programming solutions, see [Ref 7] or [Ref 8].

The iMPACT software has integrated support for the SVF translation to the file formats (XSVF and ACE) used in the reference solutions. The SVF generation steps provided in this application note can be followed with the iMPACT software batch mode file translation commands shown below, to convert the SVF into the file format needed for the referenced SVF-based programming solutions.

Translating SVF to ACE

The following batch file translates an SVF file targeting a PROM into the ACE file format using IMPACT in batch mode:

```
>impact -batch
>svf2ace -d -i update_rev1_xcf32p.svf -o update_rev1_xcf32p.ace
>quit
```

Translating SVF to XSVF

The following batch fine translates an SVF file targeting a PROM into the XSVF file format using IMPACT in batch mode:

```
>impact -batch
>svf2xsvf -d -r 0 -extensions -i update_rev1_xcf32p.svf -o
update_rev1_xcf32p.xsvf
>quit
```

Verifying and Debugging Platform Flash PROM Programming Issues

This application note is primarily focused on defining the SVF sequences required to update a single design revision on the Platform Flash PROM. However, when designing the update system, there are a number of additional elements to consider beyond the basic erase and program SVF sequences to ensure that the PROM is correctly updated and able to deliver the updated bitstream to the FPGA. Those elements are discussed in the following sections:

- "Basic System Checks"
  - "Verification of Device Data"
  - "Verification of Device Control Registers"
  - "Device Status Check"
  - "Exit In-System Configuration Mode"
- "JTAG Design Considerations"

Basic System Checks

After executing an SVF file to perform a particular operation, several methods can be used by the system to ensure that the PROM ends up in a valid state and contains the expected data. These methods are particularly important in systems that allow interruptions to the delivery of the SVF file by the SVF player, and in systems that are not designed to automatically detect where or if such an interruption has occurred. Although each SVF sequence does include a status check and a disable ISC mode sequence, an interrupted SVF sequence can result in the attempted operation failing partially or completely and/or the device being left in ISC mode. If the system does not automatically detect SVF failures, then there are four additional sequences that can be implemented in the system to confirm the PROM contents, and ensure the PROM is left in a valid state.

Verification of Device Data

Verification of the device data requires nearly doubling the amount of SVF data that must be delivered to the SVF player. This method does, however, provide a additional assurance that the Platform Flash PROM contains the expected data.

SVF Generation for Verifying the Update Image Using the iMPACT Tool GUI

Verify Revision 1.

1. Select **Output** → **SVF File** → **Create SVF File…**, browse to the correct directory location, enter **verify_rev1** as the file name, and click **Save**.
2. From the File Generation Mode window (**All device operation will be directed to the file:** `verify_rev1.svf`), click **OK**.
3. Right-click on **XCF32P**, and select **Assign New Configuration File**. Assign the file `update.mcs` to the XCF32P PROM, and click **OPEN**.
4. In the Boundary-Scan Flow window, right-click on the **XCF32P**, and select **Verify** to open the Verify Revision Options – Device 1 dialog box. From this dialog box, select **Selected Revision(s)**, and **Revision 1**. Click **OK** to complete.

   iMPACT software then creates `verify_rev1.svf`, containing the sequence needed to verify revision 1 date from the PROM and report "Verify Succeeded."
5. Select **Output** → **SVF File** → **Stop Writing to SVF File** to complete the process.

Table 5 outlines the command sequence required to verify a single design revision (revision 1 spanning blocks 2 to 3).

SVF Generation for Verifying the Update Image Using the iMPACT Tool in Batch Mode

The following batch commands create an `.svf` file to verify the data for revision 1:

```
setPreference -pref AutoSignature:FALSE
setPreference -pref KeepSVF:TRUE
setPreference -pref ConcurrentMode:FALSE
setPreference -pref UseHighz:FALSE
setPreference -pref svfUseTime:FALSE
setMode -bs
setCable -port svf -file verify_rev1.svf
addDevice -p 1 -sprom xcf32p -file update.mcs
addDevice -p 2 -part xc5vlx50
Verify -p 1 -ver 1
Quit
```

Table 5: **Single Revision Data Verify Sequence**

Step	Operation
1	IDCODE Check(1)
2	Enter ISC Mode
3	Check Blank All(2)
4	Check Read Protect For Selected Revision Data Blocks(3)
5	Verify Selected Revision Data Blocks
6	Check Status [Optional]
7	Exit ISC Mode

Notes:

1. If IDCODE does not match, then skip remaining verify sequence.
2. If all blank, then skip remaining verify sequence.
3. If read protection is set, then skip remaining verify sequence.

SVF Sequence to Verify the Data for a Single Revision

The following example SVF verifies a single design revision (revision 1 spanning blocks 2 to 3) targeting an XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain:

```
//================================================================
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0 ;
HIR 10 TDI (03ff) SMASK (03ff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'IDCODE' instruction.
SIR 16 TDI (00fe) SMASK (ffff) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f5059093) MASK (0fffffff) ;
STATE RESET;
//Loading device with 'ISPEN' instruction.
SIR 16 TDI (00e8) SMASK (ffff) ;
SDR 8 TDI (03) SMASK (ff) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
// Loading device with a 'XSC_BLANK_CHECK' instruction.
SIR 16 TDI (000d) SMASK (ffff);
RUNTEST 500000 TCK;
SDR 8 TDI (00) SMASK (ff) TDO (3f) MASK (3f) ;
// Loading device with a 'XSC_CLEAR_STATUS' instruction.
```

```
SIR 16 TDI (00f4) SMASK (ffff);
RUNTEST 50 TCK;
//Check for Read Protection.
SIR 16 TDI (0004) SMASK (ffff);
SDR 16 TDI (0000) SMASK (ffff) TDO (ffff) MASK (ffff) ;
// XAPP972: Begin Verify Data Blocks Sequence
// XAPP972: Verify 32 256-bit pages at a time (8192 bits), then
// XAPP972: increment address by 0x000020 for the next verify sequence
// XAPP972: Block 0 = start address 0x000000
// XAPP972: Block 1 = start address 0x100000
// XAPP972: Block 2 = start address 0x200000
// XAPP972: Block 3 = start address 0x300000
// Loading device with a 'ISC_ADDRESS_SHIFT' instruction.
SIR 16 TDI (00eb) SMASK (ffff);
SDR 24 TDI (200000) SMASK (ffffff) ;
// Loading device with a 'XSC_READ' instruction.
SIR 16 TDI (00ef) SMASK (ffff);
RUNTEST 15 TCK;
SDR 8192 TDI (8192-bits all zero…) SMASK (8192-bits of all ones mask data…)
TDO (8192-bits expected data…) MASK (8192-bits of all ones mask data…);
// Loading device with a 'ISC_ADDRESS_SHIFT' instruction.
SIR 16 TDI (00eb) SMASK (ffff);
SDR 24 TDI (200400) SMASK (ffffff) ;
// Loading device with a 'XSC_READ' instruction.
SIR 16 TDI (00ef) SMASK (ffff);
RUNTEST 15 TCK;
SDR 8192 TDI (8192-bits all zero…) SMASK (8192-bits of all ones mask data…)
TDO (8192-bits expected data…) MASK (8192-bits of all ones mask data…);
// XAPP972: …continue ISC_ADDRESS_SHIFT/XSC_READ sequence
// XAPP972: until all revision 1 data bits are verified …
//
// Check Device status.
SIR 16 TDI (00e3) SMASK (ffff);
SDR 8 TDI (00) SMASK (ff) TDO (36) MASK (7f) ;
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'BYPASS' instruction.
SIR 16 TDI (ffff) SMASK (ffff);
SDR 1 TDI (00) SMASK (01) ;
```

Verification of Device Control Registers

Verification of the device control registers is recommended because even though the update operation should be designed not to touch the control registers, there might not be any system checks in place to monitor the SVF file generation and content to avoid playing SVF files, which can modify the control registers. If other system checks exist, then the control register verification is not required.

Table 6 outlines the command sequence required to verify the user controlled registers in the PROM.

Table 6: **Internal Register Verify Sequence**

Step	Operation
1	IDCODE Check(1)
2	Enter ISC Mode
3	Check Blank All(2)
4	Check Read Protect For Non-data Registers(3)

Table 6: **Internal Register Verify Sequence *(Cont'd)***

Step	Operation
5	Verify Non-data Registers
6	Check Status [Optional]
7	Exit ISC Mode

Notes:

1. If IDCODE does not match, then skip remaining verify sequence.
2. If all blank, then skip remaining verify sequence.
3. If read protection is set, then skip remaining verify sequence.

SVF Sequence to Verify the Control Registers

The following example SVF verifies the control registers: DATA-RDPT, DATA-WRPT, DATA-CCB, DATA-DONE, DATA-SUCR, and DATA-BTC; targeting a XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain:

```
//================================================================
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0;
HIR 10 TDI (03ff) SMASK (03ff);
HDR 1 TDI (00) SMASK (01);
TDR 0;
//Loading device with 'IDCODE' instruction.
SIR 16 TDI (00fe) SMASK (FAA);
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f5059093) MASK (0fffffff) ;
STATE RESET;
//Enter ISC mode and check read/write protection (DATA-RDPT/DATA-WRPT)
SIR 16 TDI (00e8) SMASK (ffff) TDO (0155) MASK (01ff); //enter ISC mode
SDR 8 TDI (00) ;                                 //optional don't-care bits
SIR 16 TDI (0004) SMASK (ffff);                         //RDPT
SDR 16 TDI (003f) SMASK (ffff) TDO (0000) MASK (ffff); //'1'=no read
protection
SIR 16 TDI (00F7) SMASK (ffff);                         //WRPT
SDR 16 TDI (000f) SMASK (ffff) TDO (0000) MASK (ffff); //'1'=no write
protection
RUNTEST 1 TCK;
// Verify DATA-CCB/DATA-DONE/DATA-SUCR/DATA-BTC REGISTERS:
SIR 16 TDI (000c) SMASK (ffff);                         //DATA-CCB
SDR 16 TDI (0000) SMASK (ffff) TDO (0000) MASK (ffff); //
SIR 16 TDI (0009) SMASK (ffff);                         //DONE
SDR 8 TDI (00) SMASK (ff) TDO (00) MASK (ff);           //
SIR 16 TDI (000e) SMASK (ffff);                         //DATA-SUCR
SDR 16 TDI (0000) SMASK (ffff) TDO (0000) MASK (ffff); //
SIR 16 TDI (00f2) SMASK (ffff);                         //DATA-BTC
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (00000000) MASK (ffffffff); //
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff);
RUNTEST 50 TCK;
// Loading devices with 'BYPASS' instruction.
SIR 16 TDI (ffff) SMASK (ffff);
SDR 1 TDI (00) SMASK (01) ;
//================================================================
```

Device Status Check

Some systems can implement a quick status check using the XSC_OP_STATUS command sequence to verify the state of the device and determine if the last operation failed. The status check accesses the ISC-DEFAULT register, which contains bits that indicate the programming progress.

Table 7 outlines the command sequence required to check the status register. The status register is cleared following an CLR_STATUS instruction, ISC_DISABLE instruction, TEST-LOGIC-RESET state, or V_{CC} power-down.

Table 7: **Device Status Register Check Sequence**

Step	Operation
1	Check Status Register(1)

Notes:
1. While still in ISC mode, add status register check after the sequence needing a status check.

SVF Sequence to Check the Device Status

The following example SVF sequence checks the device status targeting an XCF32P PROM with header and trailer set for an XCF32P-to-XC5VLX50 JTAG chain. The status register check is appended to an existing sequence, so header and trailer settings are not required:

```
//=================================================================
TIR 0;
HIR 10 TDI (03ff) SMASK (03ff);
HDR 1 TDI (00) SMASK (01);
TDR 0;
// Check Device status.
SIR 16 TDI (00e3) ;
SDR 8 TDI (00) SMASK (ff) TDO (36) MASK (7f) ;
```

Exit In-System Configuration Mode

To perform any read or write operations on internal control or data registers for the Platform Flash PROM, the PROM must first enter ISC mode. Entering ISC requires starting any read/write sequence by loading the ISC_ENABLE instruction into the JTAG Instruction register. While the PROM remains in ISC mode, the CEO output is driven High. All other outputs are held in a high-impedance state or at clamp levels during in-system programming. To enable normal serial/parallel data access after completing any ISC sequence, the PROM must exit ISC mode. Exiting ISC mode requires either loading the ISC_DISABLE instruction or power cycling the PROM. If the PROM is inadvertently left in ISC mode, the PROM's non-JTAG pins remain in a disabled state.

The SVF files generated by iMPACT software automatically include the ISC_DISABLE instruction sequence at the end of the file. However, it is recommended to design a redundant reset function into the system to ensure that if the SVF player encounters any file instruction or execution errors, or if the player operation is interrupted, the PROM still exits from ISP mode after the error or interrupt. To reset the PROM, the system can cycle power to the PROM, or the system can apply the following JTAG sequence to the PROM:

1. Ensure a known JTAG TAP start state: Drive TMS High and apply a minimum of five TCK clock cycles to place the JTAG TAP in the TEST-LOGIC-RESET state.
2. Load the PROM with the JTAG ISC_DISABLE instruction.
3. Go to the JTAG TAP RUN-TEST/IDLE state and wait for a minimum of 50 µs within the RUN-TEST/IDLE state.
4. Reset the JTAG TAP: Drive TMS High and apply a minimum of five TCK clock cycles to place the JTAG TAP in the TEST-LOGIC-RESET state.

The above sequence can be applied via an SVF file. The sequence for the SVF file can be extracted from an iMPACT SVF file that programs the PROM. Extract the STATE RESET line, the TIR, HIR, HDR, TDR lines, and the ISC_DISABLE section. After extracting appropriate SVF lines, a sample, resulting SVF is shown here:

```
STATE RESET;
TIR 0;
HIR 10 TDI (03ff) SMASK (03ff);
HDR 1 TDI (00) SMASK (01);
TDR 0;
// Loading devices with 'ISC_DISABLE' instruction.
SIR 16 TDI (00f0) SMASK (ffff);
RUNTEST 50 TCK;
STATE RESET;
```

JTAG Design Considerations

Confirm Target Chain

A simple SVF file which only checks the device IDCODEs can be generated by iMPACT software and used to verify the JTAG chain device contents and connectivity.

PAUSE-IR and PAUSE-DR TAP States

The TAP should not be returned to SHIFT-IR after transitioning to the PAUSE-IR state. Also, the TAP should not be returned to SHIFT-DR after transitioning to the PAUSE-DR state. If the TAP is returned to the SHIFT-IR or SHIFT-DR state after a temporary pause in the PAUSE-IR or PAUSE-DR state, the shift register incorrectly recaptures its initial values, resetting the contents from whatever partially shifted state existed previously. If the application executing the JTAG instruction sequence needs to pause, then the TAP inputs should be held at their current values. A free-running TCK should not be used if pauses are required.

RUNTEST Scaling

All PROM algorithm RUNTEST statements are time-based. Unlike some cycle-based FPGA SVF sequences, RUNTEST statements should be scaled to match the frequency of the JTAG clock. The TCK cycle count value written by iMPACT software assumes a 1 MHz TCK frequency.

Table 8 lists example wait times from SVF sequences in this application note.

Table 8: **Example RUNTEST Wait Times**

TCK Cycle Count at 1 MHz	Wait Time
50	50 µs
60	60 µs
500,000	0.5s
60,000,000	60s
140,000,000	140s

Dynamically Defining the JTAG Target Chain Containing the PROM for SVF

Normally, when generating the JTAG SVF files for a particular board, the preferred method is to generate an `.svf` file specifically created for the intended target board. In this way, the PROM data, which typically contains an FPGA design targeted to a specific board, can only be delivered to the intended target board (or a board with an identical JTAG target chain). This method also eliminates the potential for error when actively defining the JTAG target chain at file execution time. However, it is possible to create a general-purpose `.svf` file to configure a

Platform Flash PROM in any target chain (where the chain can be dynamically modified at run-time). The `.svf` file must be generated targeting a JTAG chain containing only a Platform Flash PROM. The `.svf` header and trailer values can then be modified prior to executing the `.svf` file to specify any devices that precede the PROM and any devices that follow the PROM in the physical target JTAG chain. If the `.svf` file is generated targeting a JTAG chain containing multiple devices, then some of the instruction shifts and data shifts in the `.svf` file are pre-padded with the header and trailer values. The shift values in the multiple-device, pre-padded `.svf` file cannot be easily modified to target a different chain. Therefore, creating a general-purpose `.svf` file, and then modifying the header and trailer values in the file, is the preferred solution when the JTAG target chain must be dynamically defined.

To create an general purpose `.svf` file for dynamic targeting, use iMPACT software to:

1. Define a Boundary-Scan chain containing only a single Platform Flash PROM device.
2. Assign the `.mcs` file to the PROM.
3. Select the PROM as the operation target.
4. Generate an `.svf` file containing the desired erase, program, and/or verify operation commands.
5. As required, define the `.svf` header (HDR and HIR) and trailer (TDR and TIR) values based on the targeted device chain to ensure that the instruction and data register values are appropriately padded.

Advanced Revision Update

This application note describes the basic flows for initial programming and for updating a revision of the Platform Flash XCFP PROM. Additional design or programming steps can be taken to achieve advanced or enhanced functionality.

Configuration Management for Recovery from Failed Updates

The design revision update flow can be combined with an advanced configuration manager (controller) to obtain an FPGA design update solution that can recover from update failure scenarios. The revision update solution in this application note includes a primary design. With an appropriate configuration control mechanism, the system can attempt to configure the FPGA from the updated design at power-on and can fallback to the primary design when the updated design within the PROM is corrupted or incomplete. Increasingly sophisticated configuration controllers can compensate for increasingly larger sets of failure scenarios.

There are many different approaches to monitoring the system to ensure successful update and configuration of the FPGA. A general solution for controlling, monitoring, and managing multiple design revisions is shown in Figure 4. For details, see [Ref 9]. Some systems can take advantage of the configuration and fallback features built into the Virtex-5 FPGA. Refer to [Ref 1], [Ref 3], and [Ref 5] for details on the Virtex-5 FPGA fallback feature.

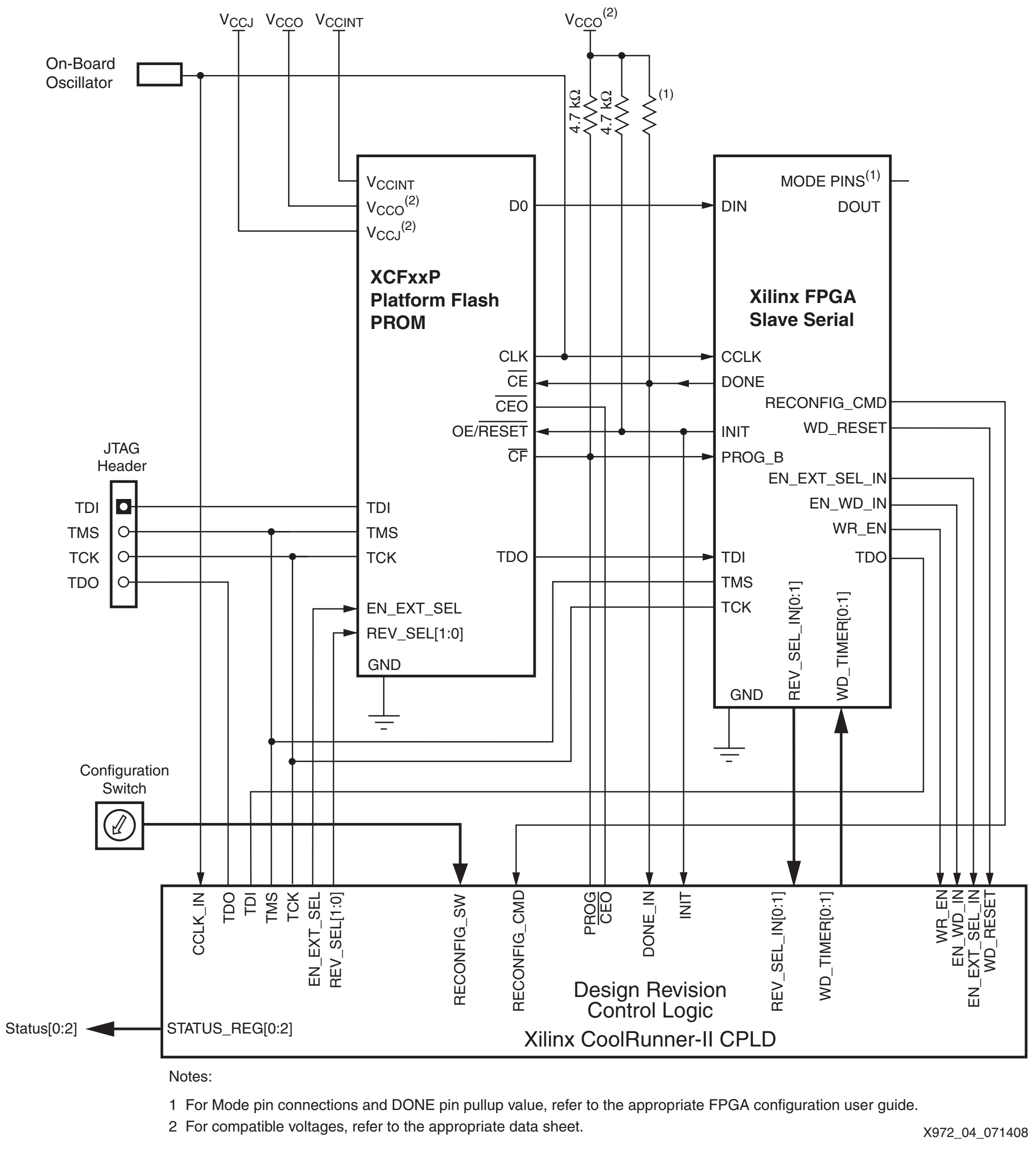

Figure 4: **Sample PROM Revision Controller and Configuration Manager from [Ref 9]**

Revisioned PROM Identification

The Platform Flash XCFP PROM supports user-programmable registers that can be used for identification of the PROM contents. The XCFP PROM has an IEEE Std 1149.1, 32-bit USERCODE register.

The 32-bit USERCODE register can be used as an overall identifier for the PROM and can be programmed during the initial PROM programming flow.

> ***Caution!*** Do not erase (re-program) the USERCODE in a revision update flow. The USERCODE is in the same erasable block as the critical PROM configuration registers (DATA-BTC and DATA-CCB). Thus, an erase operation on the USERCODE also erases the DATA-BTC and DATA-CCB registers. Such an erase operation can render the PROM in a state in which the revision information is unknown.

The iMPACT software and most third-party device programmers support programming of the USERCODE for the initial programming step.

The USERCODE value can be read via JTAG using the IEEE Std 1149.1 USERCODE instruction.

Conclusion

The guidelines presented in this application note provide a robust method for updating an FPGA design in-system using the JTAG programming feature and multiple design revision feature of the Platform Flash PROM. The example demonstrates SVF file generation flows that are supported by the ISE iMPACT software. References to SVF-based, in-system programming solutions are provided.

References

1. DS100, *Virtex-5 Family Overview LX, LXT, and SXT Platforms*
2. DS123, *Platform Flash In-System Programmable Configuration PROMs*
3. DS202, *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*
4. UG161, *Platform Flash PROM User Guide*
5. UG191, *Virtex-5 FPGA Configuration User Guide*
6. Texas Instruments IEEE JTAG 1149.1 Primer: http://focus.ti.com/lit/an/ssya002c/ssya002c.pdf
7. XAPP058, *Xilinx In-System Programming Using an Embedded Microcontroller*
8. XAPP424, *Embedded JTAG ACE Player*
9. XAPP693, *A CPLD-Based Configuration and Revision Manager for Xilinx Platform Flash PROMs and FPGAs*

Appendix A: Boundary Scan for the Platform Flash PROM

Compatibility with IEEE Std 1149.1 Boundary-Scan protocols requires that the PROM includes a test access port (TAP), TAP controller, and JTAG registers. The JTAG registers support all required Boundary-Scan instructions, as well as many of the optional instructions specified by IEEE Std 1149.1. In addition to these standard and optional instructions, the PROM's JTAG interface also supports an instruction set used to access the PROM during in-system programming. When accessing the PROM's internal data and control registers for erase, program, or verify operations, the PROM must be in in-system configuration (ISC) mode. Outlined in the following sections are the fundamentals of the JTAG architecture for the Platform Flash PROM devices.

Test Access Port

The Platform Flash PROM TAP contains four mandatory dedicated pins as specified by the IEEE Std 1149.1 protocol (Table 9). There are three input pins and one output pin to control the Boundary-Scan TAP controller. In addition to the required pins, devices from other manufactures can also contain optional control pins such as TRST (test reset) and enable pins. Be aware of these optional signals when interfacing Xilinx devices with devices from different vendors, because these signals might need to be driven on the other devices connected in the JTAG chain to enable normal TAP operations.

Table 9: **XCFxxP JTAG TAP Pin Names and Descriptions (VO48/VOG48 and FS48/FSG48)**

Pin Name	Boundary-Scan Function	Pin Description	48-pin TSOP (VO48/VOG48)	48-pin TFBGA (FS48/FSG48)
TMS	Mode Select	**JTAG Mode Select Input**. The state of TMS on the rising edge of TCK determines the state transitions at the Test Access Port (TAP) controller. TMS has an internal 50 KΩ resistive pull-up to V_{CCJ} to provide a logic `1` to the device if the pin is not driven.	21	E2
TCK	Clock	**JTAG Clock Input.** This pin is the JTAG test clock. It sequences the TAP controller and all the JTAG test and programming electronics.	20	H3
TDI	Data In	**JTAG Serial Data Input**. This pin is the serial input to all JTAG instruction and data registers.TDI has an internal 50 KΩ resistive pull-up to V_{CCJ} to provide a logic `1` to the device if the pin is not driven.(1)	19	G1
TDO	Data Out	**JTAG Serial Data Output.** This pin is the serial output for all JTAG instruction and data registers. TDO has an internal 50 KΩ resistive pull-up to V_{CCJ} to provide a logic `1` to the device if the pin is not driven.(2)	22	E6

Notes:
1. During an instruction or data shift, TDI is clocked in on the rising edge of TCK.
2. TDO is only active during an instruction or data shift and changes state on the falling edge of TCK. This pin is High-Z at all other times.

TAP Controller

Figure 5 diagrams the 16-state finite state machine of the Platform Flash PROM TAP controller. The four TAP pins control how the data is scanned into the various registers. The state of the TMS pin at the rising edge of the TCK determines the sequence of state transitions. There are two main sequences: one for shifting data into the data register, and the other for shifting an instruction into the instruction register.

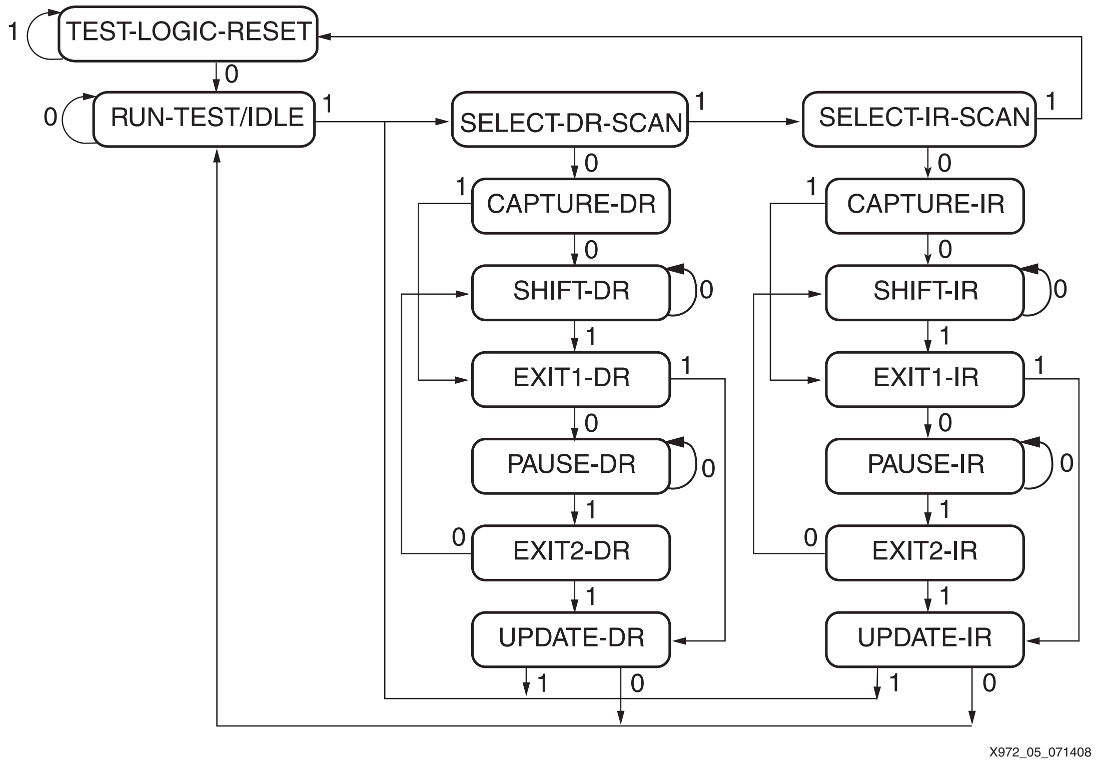

Figure 5: **State Diagram for the TAP Controller**

Boundary-Scan Instruction Register, Instruction Set, and Data Registers

Mandatory and optional IEEE Std 1149.1 and IEEE Std 1532 instructions are supported in Platform Flash PROM devices, along with many device specific instructions, including instructions allowing read/write access to the internal control registers. To determine the operation to be invoked, a 16-bit instruction is loaded into the instruction register while the TAP controller is in the instruction shift state (SHIFT-IR). The new instruction is registered on the falling edge of TCK, after the TAP controller enters the instruction update state (UPDATE-IR).

Table 10 lists the instructions associated with the Platform Flash PROM erase, program, and verify operations, as well as the data registers associated with those instructions.

Table 10: **Platform Flash PROM JTAG Instructions Required for Read/Write Access**

Instruction (SIR)	Instruction Code	Data Register (SDR)	Data Register Width	Instruction Type
BYPASS	`1111111111111111 (0xFFFF)`	BYPASS	1	1149.1 Mandatory
IDCODE	`0000000011111110 (0x00FE)`	IDCODE	32	1149.1 Optional
ISC_ADDRESS_SHIFT(1)	`0000000011101011 (0x00EB)`	ADDRESS	24	1532 Optional
ISC_DATA_SHIFT(1)	`0000000011101101 (0x00ED)`	DATA-0	256	1532 Optional
ISC_DISABLE(1)	`0000000011110000 (0x00F0)`	BYPASS	1	1532 Mandatory
ISC_ENABLE(1)	`0000000011101000 (0x00E8)`	ISC-ENABLE	8	1532 Mandatory
ISC_ERASE(1)	`0000000011101100 (0x00EC)`	SELECT-BLOCK	24	1532 Optional
ISC_PROGRAM(1)	`0000000011101010 (0x00EA)`	ISC-DEFAULT	8	1532 Mandatory
XSC_BLANK_CHECK(1)	`0000000000001101 (0x000D)`	BLANK	8	Device Specific
XSC_CLR_STATUS	`0000000011110100 (0x00F4)`	ISC-DEFAULT	8	Device Specific
XSC_CONFIG(1)	`0000000011101110 (0x00EE)`	BYPASS	1	Device Specific
XSC_DATA_BTC	`0000000011110010 (0x00F2)`	DATA-BTC	32	Device Specific
XSC_DATA_CC	`0000000000000111 (0x0007)`	DATA-CC	256	Device Specific
XSC_DATA_CCB	`0000000000001100 (0x000C)`	DATA-CCB	16	Device Specific
XSC_DATA_DONE	`0000000000001001 (0x0009)`	DONE	8	Device Specific
XSC_DATA_RDPT	`0000000000000100 (0x0004)`	DATA-RDPT	16	Device Specific
XSC_DATA_SUCR	`0000000000001110 (0x000E)`	DATA-SUCR	16	Device Specific
XSC_DATA_UC	`0000000000000110 (0x0006)`	USERCODE	32	Device Specific
XSC_DATA_WRPT	`0000000011110111 (0x00F7)`	DATA-WRPT	16	Device Specific
XSC_OP_STATUS	`0000000011100011 (0x00E3)`	ISC-DEFAULT	8	Device Specific
XSC_READ(1)	`0000000011101111 (0x00EF)`	DATA-1	8M block	Device Specific
XSC_UNLOCK	`1010101001010101 (0xAA55)`	SELECT-BLOCK	24	Device Specific

Notes:

1. Abort array operation.

Instruction Register

The instruction register (IR) is connected between TDI and TDO during an instruction scan sequence. In preparation for an instruction scan sequence, the instruction register is parallel loaded with a fixed instruction-capture pattern. This pattern is shifted out onto TDO (LSB first), while an instruction is shifted into the instruction register from TDI.

The instruction-capture pattern shifted out of the XCFxxP device includes IR[15:0]. IR[15:9] are reserved bits and are set to a logic 0. The ISC Error field, IR[8:7], contains 10 when an ISC operation is successful; otherwise, the field is set to 01 when an in-system configuration (ISC) operation fails. The Erase/Program (ER/PROG) Error field, IR[6:5], contains 10 when an erase or program operation is a success; otherwise, the field contains 01 when an erase or program operation fails. The Erase/Program (ER/PROG) Status field, IR[4], contains a logic 0 when the device is busy performing an erase or programming operation; otherwise, the field contains a logic 1. The ISC Status field, IR[3], contains logic 1 if the device is currently in ISC mode; otherwise, the field contains logic 0. The DONE field, IR[2], contains logic 1 if the sampled design revision is successfully programmed; otherwise, the field contains a logic 0 indicating incomplete programming. The remaining bits, IR[1:0], are set to 01 as defined by IEEE Std 1149.1.

Table 11: **XCFxxP Instruction Capture Values Loaded into IR as Part of an Instruction Scan Sequence**

TDI →	IR[15:9]	IR[8:7]	IR[6:5]	IR[4]	IR[3]	IR[2]	IR[1:0]	→ TDO
	Reserved	ISC Error	ER/PROG Error	ER/PROG Status	ISC Status	DONE	01	

Instruction Set

BYPASS

Short Description:

Selects the BYPASS Register between TDI and TDO, effectively causing the device to be "bypassed" in a JTAG chain during data shifts.

Requirements:

This instruction does not interfere with normal device operation.

Data Register Enabled:

BYPASS (1 bit)

IDCODE

Short Description:

Selects the IDCODE Register between TDI and TDO.

Requirements:

The IDCODE is shifted out on the TDO pin in the SHIFT-DR state. This instruction does not interfere with normal device operation.

Data Register:

IDCODE (32 Bits)

Note: Whenever the TAP controller enters the TEST-LOGIC-RESET state, the IDCODE instruction is automatically loaded into the JTAG Instruction Register and immediately becomes the active instruction.

ISC_DISABLE

Short Description:

Disables in-system programming operations and initiates an internal reset. Selects the BYPASS Register between TDI and TDO.

Requirements:

An internal reset pulse is generated after entering the RUN-TEST/IDLE state while the ISC_DISABLE instruction is active. This operation requires the device to remain in the RUN-TEST/IDLE state for at least 50 μs for the ISC_DISABLE operation to complete, during which:

- Any current ISC or XSC operation is aborted
- The previous Boundary-Scan cell configuration (HIGHZ, CLAMP, etc.) is reset to the normal operation
- Internal registers are re-initialized
- The address counter is reset to point to the start address for the selected revision

If the device was previously in ISP mode, after an ISC_DISABLE instruction is loaded, the device pins are released to normal operation on the rising edge of TCK upon exiting from the UPDATE-IR state.

Data Register Enabled:

BYPASS (1 bit)

ISC_ENABLE (ISPEN)

Short Description:

Enables in-system-programming operations, by placing the device in ISC mode. The ISC_ENABLE instruction must be the first instruction executed during any in-system-programming sequence. No other 'ISC' or 'XSC' type instructions is allowed to execute without first issuing the ISC_ENABLE instruction. Selects the ISC-ENABLE register between TDI and TDO.

Requirements:

All non-TAP pins on the device are disabled (3-state) on the rising edge of TCK upon exit from the UPDATE-IR state. Also, a bit in the ISC-DEFAULT register is set to indicate that the device is in ISC mode. The device remains in ISC mode until either the ISC_DISABLE instruction is executed, the TAP controller is returned to TEST-LOGIC-RESET, or the PROM is power cycled. ISC_ENABLE Instruction aborts any current ISC or XSC operation.

Data Register Enabled:

ISC-ENABLE (8 Bits)

ISC_PROGRAM

Short Description:

Initiates the programming pulse for user registers or data registers. Selects the ISC-DEFAULT register between TDI and TDO. This instruction is disabled by write protection on selected block (see “XSC_DATA_WRPT,” page 40).

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. The ISC_PROGRAM operation is disabled for any data blocks or registers which have write protection set. A *program pulse* is generated after entering the RUN-TEST/IDLE state and pulsing TCK one additional time with TMS held Low (holding the TAP controller in the RUN-TEST/IDLE state) while the ISC_PROGRAM instruction is active. The TAP controller must remain in the RUN-TEST/IDLE state for the recommended program time, and TCK can continue to toggle. The program pulse is a self-timed internal operation that programs the pre-loaded data to the pre-selected address.

This self-timed programming operation takes up to a maximum of 1 ms for each 256-bit data page, depending on device operating conditions such as voltage, temperature, etc. Entering the Test-Logic-Reset state or executing any abort instruction (see Table 10, page 33) interrupts the programming operation. The device requires 50 µs to reset internal nodes after premature termination of an array operation. During this time, executing an array-access instruction does not yield valid results. In addition, polling the device via the XSC_OP_STATUS instruction or through the Capture-IR bits returns a status of busy during these 50 µs.

The page address to be programmed is determined by loading the internal ADDRESS Register using an ISC_ADDRESS_SHIFT instruction sequence. The page address automatically increments to the next 256-bit page address after each programming sequence until the last address in the 8 Mb block is reached. To continue programming the next block, the next 8 Mb block's starting address must be loaded into the internal ADDRESS register. The page data to be programmed is determined by loading the DATA-0 register using an ISC_DATA_SHIFT instruction sequence.

Programming operation starts in RUN-TEST/IDLE, and the programming instruction aborts any current ISC or XSC operation.

Data Register Enabled:

ISC-DEFAULT (8 Bits)

Additional Data Register Contents Used by Instruction:

ADDRESS (24 Bits)

DATA-0 (256 Bits)

ISC_ADDRESS_SHIFT

Short Description:

Selects the ADDRESS Register between TDI and TDO.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. The ISC_ADDRESS_SHIFT instruction aborts any current ISC or XSC operation. The ADDRESS register does not need to be reloaded for each program sequence, as returning to the SELECT-DR-SCAN state while ISC_DATA_SHIFT is loaded automatically increments the address to the next 256-bit word's memory location.

Data Register Enabled:

ADDRESS (24 Bits)

ISC_DATA_SHIFT

Short Description:

Selects the DATA-0 Register between TDI and TDO.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. The ISC_DATA_SHIFT instruction aborts any current ISC or XSC operation. Returning to the SELECT-DR-SCAN state while ISC_DATA_SHIFT is loaded automatically increments the address to the next 256-bit word's memory location.

Data Register Enabled:

DATA-0 (256 Bits)

ISC_ERASE

Short Description:

Erase user registers and data registers as indicated by SELECT-BLOCK register settings. The SELECT-BLOCK register is immediately enabled between TDI and TDO. After the ISC_ERASE instruction is loaded, the TAP controller must move from the UPDATE-IR state to the SHIFT-DR state to select which registers will be erased, without passing through the RUN-TEST/IDLE state. The erase instruction is disabled by write protection on selected block (see "XSC_DATA_WRPT," page 40).

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are when the instruction is loaded. Erase operation starts in RUN-TEST/IDLE and aborts any current ISC or XSC operation. If the targeted blocks are write protected, then an XSC_UNLOCK instruction sequence must be used to unlock the targeted blocks to override any write protection (DATA-WRPT) before an erase operation can succeed.

- For a 32 Mb XCF32P device, six blocks can be individually erased: Block[0], Block[1], Block[2], Block[3], DATA-SUCR, and GUCR.
- For a 16 Mb XCF16P device, four blocks can be individually erased: Block[0], Block[1], DATA-SUCR, and GUCR.
- For an 8 Mb XCF08P device, three blocks can be individually erased: Block[0], SUCR, and GUCR.

Note: Erasing an 8 Mb memory array block automatically erases its corresponding BUCR.

Data Register Enabled:

SELECT-BLOCK (24 Bits)

XSC_BLANK_CHECK

Short Description:

Blank checks the whole device. Selects the BLANK register between TDI and TDO.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. Blank check operation starts in RUN-TEST/IDLE and aborts any current ISC or XSC operation.

Data Register Enabled:

BLANK (8 Bits)

XSC_CONFIG

Short Description:

Drives the $\overline{\text{CF}}$ output pin low for 300 ns. Selects the BYPASS register between TDI and TDO.

Requirements:

The $\overline{\text{CF}}$ pulse falling edge starts in RUN-TEST/IDLE. The XSC_CONFIG instruction must remain in the instruction register while the pulse is driven. OE/$\overline{\text{RESET}}$ is also held Low while the $\overline{\text{CF}}$ pulse is driven. The XSC_CONFIG instruction aborts any current ISC or XSC operation. The device does not need to be in ISC mode to execute the XSC_CONFIG instruction.

Data Register Enabled:

BYPASS (1 bit)

XSC_READ

Short Description:

Read data starting at indicated address. This instruction is disabled by read protection (see "XSC_DATA_RDPT," page 39).

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. The page address from which data is read is determined by loading the internal ADDRESS register using an ISC_ADDRESS_SHIFT instruction sequence. Data can be read out until the end of the block is reached (up to 8 Mb per read sequence). To continue reading from the next block, the next 8 Mb block's starting address must be loaded into the internal ADDRESS register. If the data block is read protected, or if attempting to read past the end of the 8 Mb block, then the SHIFT-DR state shifts out all `1`s. The XSC_READ instruction aborts any current ISC or XSC operation.

Data Register Enabled:

DATA-1 (8 Mb data block)

Additional Data Register Contents Used by Instruction:

ADDRESS (24 Bits)

XSC_CLR_STATUS

Short Description:

Clears the internal erase or program operation error status in the ISC-DEFAULT register.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. Operation starts in RUN-TEST/IDLE.

This instruction must not be executed while an internal erase, program, or read operation is in progress. The results of the instruction in this case are not predictable because they depend on the state of the internal erase, program, or read operation at the time the instruction is received.

If the XSC_CLR_STATUS instruction is executed while no internal erase, program, or read operation are in progress, then the internal erase, program, or read operation error status bits are cleared and shifted out. If no error is flagged in the status register, then there is no change in the content of the ISC-DEFAULT register.

In addition to XSC_CLR_STATUS instruction, the internal erase, program, or read operation error status bits can also be cleared during V_{CC} power-down, returning to the TEST-LOGIC-RESET state, or loading the ISC_DISABLE instruction.

It is not mandatory to clear an error flagged by an internal erase, program, or read operation — the device continues to function normally. However, the error flag remains set in the ISC-DEFAULT register until cleared by a subsequent XSC_CLR_STATUS, ISC_DISABLE instruction, TEST-LOGIC-RESET, or V_{CC} power-down.

Data Register Enabled:

ISC-DEFAULT (8 Bits)

XSC_DATA_BTC

Short Description:

Load Data into the Block Table register. The Block Table contains the settings controlling the size and other options for the four permitted design revisions.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

Data Register Enabled:

DATA-BTC (32 Bits)

XSC_DATA_CC

Short Description:

Load Data into the 'Customer Code' Register.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. The customer code register corresponds to a particular block, and the customer code register to be programmed is selected by loading the corresponding block start address into the internal ADDRESS Register using an ISC_ADDRESS_SHIFT instruction sequence, before loading the 256 bits of data to be programmed using an XSC_DATA_CC instruction sequence.

Data Register Enabled:

DATA-CC (256 Bits)

Additional Data Register Contents Used by Instruction:

ADDRESS (24 Bits)

XSC_DATA_CCB

Short Description:

Loads data into the Customer-Controlled Bits register.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

Data Register Enabled:

DATA-CCB (16 Bits)

XSC_DATA_DONE

Short Description:

Loads data into the Done register.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

Data Register Enabled:

DONE (8 Bits)

XSC_DATA_RDPT

Short Description:

Loads data into READ-PROTECTION register. Read protection is used to disable JTAG readback. An erase sequence is required to override read protection.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

Data Register Enabled:

DATA-RDPT (16 Bits)

XSC_DATA_SUCR

Short Description:

Loads data into the DATA-SUCR register.

Requirements:

The device does not need to be in ISC mode to execute the XSC_DATA_SUCR instruction.

Data Register Enabled:

DATA-SUCR (16 Bits)

XSC_DATA_UC

Short Description:

Load Data into the USERCODE register.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

Data Register Enabled:

USERCODE (32 Bits)

XSC_DATA_WRPT

Short Description:

Loads data into the WRITE-PROTECTION register. Write protection is used to avoid inadvertent program or erase operations. An unlock sequence is required to override write protection.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

Data Register Enabled:

DATA-WRPT (16 Bits)

XSC_OP_STATUS

Short Description:

Read the ISC-DEFAULT register. XCS_OP_STATUS can be used to check the status of the current (or previous) erase, program, or read operation.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded. V_{CC} power-down, returning to the TEST-LOGIC-RESET state, or loading the ISC_DISABLE instruction clears the internal erase, program, or read operation error status.

Data Register Enabled:

ISC-DEFAULT (8 Bits)

XSC_UNLOCK

Short Description:

Unlocks user registers and data registers as indicated by SELECT-BLOCK Register settings to remove write protection set by the corresponding DATA-WRPT register bits.

Requirements:

If the device is not in ISC mode, then error bits in the ISC-DEFAULT register are set when the instruction is loaded.

- For a 32 Mb XCF32P device, six blocks can be individually unlocked: Block[0], Block[1], Block[2], Block[3], DATA-SUCR, and GUCR.
- For a 16 Mb XCF16P device, four blocks can be individually unlocked: Block[0], Block[1], DATA-SUCR, and GUCR.
- For an 8 Mb XCF08P device, three blocks can be individually unlocked: Block[0], DATA-SUCR, and GUCR.

Note: Unlocking an 8 Mb memory array block will automatically unlock its corresponding BUCR.

Data Register Enabled:

SELECT-BLOCK (24 Bits)

Data Registers

Platform Flash PROM devices have several data registers including the registers required by IEEE Std 1149.1 and IEEE Std 1532. Which Data Register (DR) is connected between TDI and TDO during an data scan sequence is determined by the JTAG instruction currently loaded (see Table 10, page 33). In preparation for a new data scan sequence, the data register is parallel loaded with a fixed data capture pattern while in the CAPTURE-DR TAP state. This data capture pattern is shifted out onto TDO (LSB first), while new data is shifted into the data register from TDI while in the SHIFT-DR state. The new data loads on the first falling edge of TCK in the UPDATE-DR state.

When programming any register, insert ones for any unused bits. For example, if intending to program bits [2:0] of an 8-bit register to `000`, the value shifted into the register should be [8:0] = `11111000`.

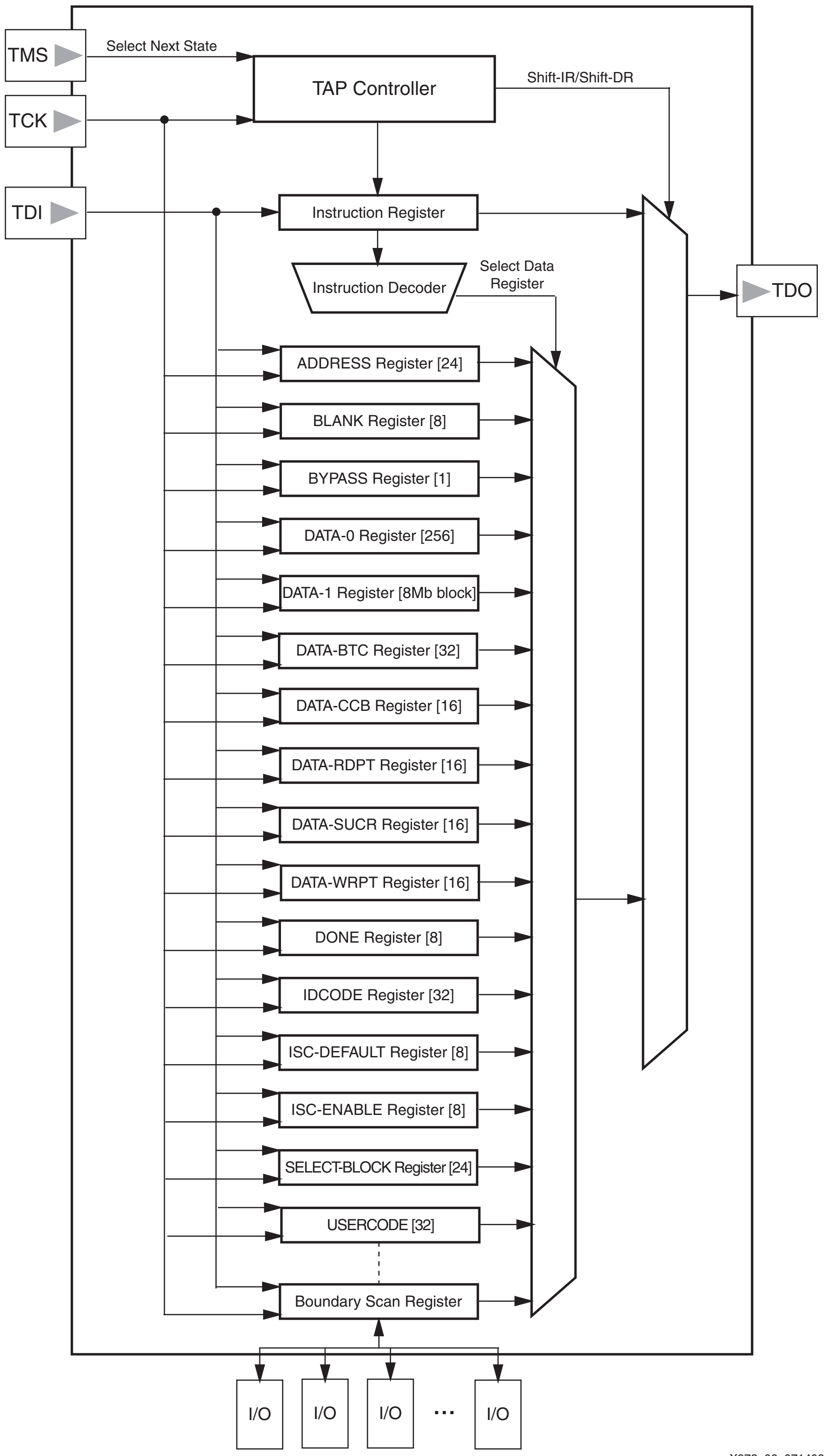

Figure 6: **XCFP Platform Flash PROM JTAG Data Registers**

ADDRESS Register (24 Bits)

The XCFxxP series contains an internal 24-bit address register (ADDRESS [23:0]). Each address selects a 256-bit page from the memory array. For program and read operations, the internal address register automatically increments to the next page address within the block after the operation completes. Thus, only the starting address is required prior to an array access operation. To manually increment to the next 256-bit page address, add `0x000020` to the current address. The ADDRESS register value is updated using the ISC_ADDRESS_SHIFT instruction and is always reloaded with address `0x000000` on the rising edge of TCK in the CAPTURE-DR state. The ADDRESS register value is used for ISC_PROGRAM, XSC_READ, and XSC_DATA_CC instructions.

Table 12: **Addressing Scheme**

Address [23:0]	Address Description
0000 0000 0000 0000 0000 0000	Data Block[0] Start Address
0000 1111 1111 1111 1110 0000	Data Block[0] End Address
0001 0000 0000 0000 0000 0000	Data Block[1] Start Address
0001 1111 1111 1111 1110 0000	Data Block[1] End Address
0010 0000 0000 0000 0000 0000	Data Block[2] Start Address
0010 1111 1111 1111 1110 0000	Data Block[2] End Address
0011 0000 0000 0000 0000 0000	Data Block[3] Start Address
0011 1111 1111 1111 1110 0000	Data Block[3] End Address
1000 1000 0000 0000 0000 0000	Block[0] User Controlled Registers
1001 1000 0000 0000 0000 0000	Block[1] User Controlled Registers
1010 1000 0000 0000 0000 0000	Block[2] User Controlled Registers
1011 1000 0000 0000 0000 0000	Block[3] User Controlled Registers
1100 0000 0000 0000 0000 0000	Global User Controlled Registers
1101 0001 0000 0000 0000 0000	Special User Controlled Registers

Block User Controlled Registers (BUCR):

The BUCR includes DATA-0 (DATA-CC) register, DONE, DATA-RDPT, and DATA-WRPT bits. The starting address of a block's BUCR is also the address of the block's customer code (XSC_DATA_CC).

Global User Controlled Registers (GUCR):

The GUCR includes USERCODE, DATA-BTC, and DATA-CCB registers, and DONE and DATA-WRPT bits.

Special User Controlled Registers (SUCR):

The SUCR includes DATA-SUCR register and DONE and DATA-WRPT bits.

Unlock and Erase operations are performed using the SELECT-BLOCK Register (see "ISC_ERASE," page 37 and "XSC_UNLOCK," page 41 for details). Unlock and Erase operations do not use the ADDRESS register.

BLANK Register (8 Bits)

The BLANK register is accessed using the XSC_BLANK_CHECK instruction. Internal logic automatically blank checks the device and stores the results of the blank check in the BLANK register. Read protected blocks show as `1` (not blank) in the BLANK register. The current

BLANK Register value is always reloaded on the rising edge of TCK in the CAPTURE-DR state.

Table 13: **BLANK Register**

Bit	Name	Description
7…6	–	Not Used (set to `1`)
5	SUCR	`1`: not blank `0`: blank
4	GUCR	`1`: not blank `0`: blank
3	Block[3]	`1`: not blank `0`: blank
2	Block[2]	`1`: not blank `0`: blank
1	Block[1]	`1`: not blank `0`: blank
0	Block[0]	`1`: not blank `0`: blank

Boundary Scan Register (32 Bits)

See [Ref 2] (used for device JTAG tests).

BYPASS Register (1 Bit)

The BYPASS register is a single-bit register that directly registers and passes data serially from the TDI to TDO pin while the BYPASS instruction is active. The BYPASS Register is always loaded with a logic `0` on the rising edge of TCK in the CAPTURE-DR state.

DATA-0/DATA-CC Customer Code Register (256 Bits)

The DATA-0 register is enabled between TDI and TDO during the ISC_DATA_SHIFT or XSC_DATA_CC instructions. The DATA-0 register is primarily used during programming to write a 256-bit word of data to a location specified by the ADDRESS register. Since the 256-bit shift register is shared between DATA-0 and DATA-CC operations, loading data into the register using TDI during an ISC_DATA_SHIFT results in the device outputting the most recently accessed customer code bits on TDO. The default customer code loaded is the Block[0] DATA-CC. After loading the data, the DATA-0 register data is used with the ISC_PROGRAM instruction.

The stored customer code DATA-CC values cannot be read protected but can be write protected (see [Ref 2] for additional customer code information).

DATA-1 Register (8 Mb Block)

The DATA-1 register is used during the XSC_READ instruction to read the contents of the 8 Mb memory block selected by the ADDRESS register. When the DATA-1 register is selected, the contents of the selected memory block are shifted out on the TDO pin, starting at the selected address and ending at the last address within the selected block.

DATA-BTC Register (32 Bits)

The DATA-BTC register is used to program the settings for up to four separate design revisions into the block table. The design revision settings include:

- One bit to enable the revision
- One bit to enable decompression for the revision
- The revision start block address

- The revision stop block address
- One bit to enable a free-running clock for the revision

If the design revision is disabled, then selecting that revision results in the data pins 3-stating.

If decompression is disabled, then the PROM simply outputs the data as it is stored in memory.

If the revision start address and stop address are set to the same block, then the revision only exists for that block. The revision stop address indicates the last block containing data for the revision. For example, a revision starting at block[2] and ending somewhere in block[3] has a Start Block Address = `10` and Stop Block Address = `11`.

If CLKOUT is enabled, then enabling the free-running clock for a revision allows CLKOUT to continue to run after all of the bits from the blocks in the selected revision are shifted out. Also, if the free-running clock is enabled, then CLKOUT also continues to run after the CE pin transitions High. CLKOUT is enabled by the DATA-CCB[5:3] bits.

Table 14: **BTC Register**

Bit	Name	Description
31	Reserved	`1`
30	Reserved	`1`
29	Reserved	`1`
28	Reserved	`1`
27	Free-running clock 3	`0`: enable free-running clock `1`: disable free-running clock
26	Free-running clock 2	`0`: enable free-running clock `1`: disable free-running clock
25	Free-running clock 1	`0`: enable free-running clock `1`: disable free-running clock
24	Free-running clock 0	`0`: enable free-running clock `1`: disable free-running clock
23	Decompression	`0`: enable decompression `1`: disable decompression
22	Enable Revision	`0`: enable revision `1`: disable revision
21:20	Stop Block Address	`00`, `01`, `10`, or `11`(1)
19:18	Start Block Address	`00`, `01`, `10`, or `11`(1)
17	Decompression	`0`: enable decompression `1`: disable decompression
16	Enable Revision	`0`: enable revision `1`: disable revision
15:14	Stop Block Address	`00`, `01`, `10`, or `11`(1)
13:12	Start Block Address	`00`, `01`, `10`, or `11`(1)
11	Decompression	`0`: enable decompression `1`: disable decompression
10	Enable Revision	`0`: enable revision `1`: disable revision
9:8	Stop Block Address	`00`, `01`, `10`, or `11`(1)

Table 14: **BTC Register *(Cont'd)***

Bit	Name	Description
7:6	Start Block Address	`00`, `01`, `10`, or `11`(1)
5	Decompression	`0`: enable decompression `1`: disable decompression
4	Enable Revision	`0`: enable revision `1`: disable revision
3:2	Stop Block Address	`00`, `01`, `10`, or `11`(1)
1:0	Start Block Address	`00`, `01`, `10`, or `11`(1)

Notes:

1. For a 32 Mb XCF32P device, four 8 Mb data blocks are selectable as the start/stop addresses for each revision: Block[0]=`00`, Block[1]=`01`, Block[2]=`10`, and Block[3]=`11`.
   For a 16 Mb XCF16P device, two 8 Mb data blocks are selectable as the start/stop addresses for each revision: Block[0]=`00` and Block[1]=`01`
   For an 8 Mb XCF08P device, only one 8 Mb data blocks is selectable as the start/stop addresses for each revision: Block[0]=`00`

DATA-WRPT Register (16 Bits)

The DATA-WRPT register protects the selected registers or data blocks from inadvertent write operations (erase or program). An unlock sequence is required to override write protection.

Table 15: **DATA-WRPT Register**

Bit	Name	Description
15...6	–	Not Used (set to `1`)
5	SUCR	`1`: not protected `0`: protected
4	GUCR	`1`: not protected `0`: protected
3	Block[3]	`1`: not protected `0`: protected
2	Block[2]	`1`: not protected `0`: protected
1	Block[1]	`1`: not protected `0`: protected
0	Block[0]	`1`: not protected `0`: protected

DATA-RDPT Register (16 Bits)

The DATA-RDPT register protects the selected data blocks from JTAG read commands. The data block must be erased to reset the read protection bits. The SUCR and GUCR registers cannot be read protected.

Table 16: **DATA-RDPT Register**

Bit	Name	Description
15..4	-	Not Used (set to `1`)
3	Block[3]	`1`: not protected `0`: protected
2	Block[2]	`1`: not protected `0`: protected
1	Block[1]	`1`: not protected `0`: protected
0	Block[0]	`1`: not protected `0`: protected

DATA-CCB Register (16 Bits)

The DATA-CCB register controls the clock and data width options for the PROM as indicated in Table 17.

Table 17: **DATA-CCB Register**

Bit	Name	Description
15..6	–	Not Used (set to `1`)
5:4	Clock Oscillator Frequency	`11` = default 40 MHz `01` = 20 MHz
3	Master/Slave Clocking Mode	`1` = PROM is Slave (FPGA in Master Mode) `0` = PROM is Master (FPGA in Slave Mode)
2:1	I/O Data Width	`11` = Serial (1 bit) Data Output `00` = SelectMAP (8 bit) Data Output
0	Clock Source	`1` = External (input from CLK pin) `0` = Internal Clock Oscillator

DATA-SUCR Register (16 Bits)

The DATA-SUCR register contains the programmable internal revision selection bits. These internal bit setting are overridden by the external REV_SEL[1:0] pin settings when the EN_EXT_SEL pin is held Low (see [Ref 2] for additional information on revision control).

Table 18: **DATA-SUCR Register**

Bit	Name	Description
15..3	–	Not Used (set to `1`)
2	Enable External Select	`1`: use internal bits `0`: use external pins
1:0	Revision Selection	`00`, `01`, `10`, or `11`

DONE Register (8 Bits)

After programming the internal control registers and data registers, the associated DONE bits in the DONE register must be programmed to enable normal device operation and allow access to the selected design revision. Each bit in the DONE register determines whether the corresponding block behaves as programmed or erased (not programmed).

Table 19: **DONE Register**

Bit	Name	Description
7...6	–	Not Used (set to `1`)
5	SUCR	`1`: not programmed `0`: programmed
4	GUCR	`1`: not programmed `0`: programmed
3	Block[3]	`1`: not programmed `0`: programmed
2	Block[2]	`1`: not programmed `0`: programmed
1	Block[1]	`1`: not programmed `0`: programmed
0	Block[0]	`1`: not programmed `0`: programmed

IDCODE Register (32 Bits)

The IDCODE register allows easy identification of the part being tested or programmed via Boundary-Scan. While the IDCODE instruction is active, the IDCODE Register is always (re)loaded with the device IDCODE on the rising edge of TCK in the CAPTURE-DR state (See [Ref 2] for details).

ISC-ENABLE Register (8 Bits)

The ISC-ENABLE register is the active data register during the ISC_ENABLE instruction, but the register data is not used.

ISC-DEFAULT Register (8 Bits)

The ISC-DEFAULT register is selected by ISC_PROGRAM, XSC_OP_STATUS, and XSC_CLR_STATUS instructions. The contents can be checked to monitor the status of the current erase, program, or read operation. The register is always refreshed on the rising edge of TCK in the CAPTURE-DR state and is shifted out on the TDO pin while in the SHIFT-DR state.

Table 20: **ISC-DEFAULT Register**

Bit	Name	Description
7	–	Not Used (set to 1)
6	RTI Entry Error	`1`: RTI Error `0`: no RTI error
5	ISC Mode	`1`: ISC mode `0`: not in ISC mode
4:3	Internal Erase/Program Error	`11`: Address Error(1) `10`: Success (default value) `01`: Fail `00`: Locked

Table 20: **ISC-DEFAULT Register *(Cont'd)***

Bit	Name	Description
1:0	ISC Error	`11`: N/A `10`: Success `01`: Error `00`: N/A
2	Internal Erase/Program Status	`1`: ready `0`: busy

Notes:
1. Not applicable for erase operations.

ISC Error [1:0]

The ISC Error bits indicate the ISC status of the previously executed instruction (ISC_ERROR is updated any time a new instruction is loaded).

For XCFxxP, the ISC Error bits indicate `01` (error) if:

- An ISC instruction is loaded and the part is not in ISC mode, or
- Skipping RunTest/Idle after an Update-IR for instructions that require time in RunTest/Idle (sets bit[6] to `1` as well, and does not clear it until a CLR_STATUS or ISC_DISABLE instruction).

Internal Erase/Program Error [4:3]

The Internal Erase/Program Error bits (bits[4:3]) indicates an error if:

- an attempt to read from a read-protected block or writing to a write-protected block is performed (bits[4:3] are set to `00`)
- an attempt to read/write from an address beyond the array's set density is performed (bits[4:3] are set to `11`)
- programming the main memory or the customer code at a non-existing/not-allowed address (bits[4:3] are set to `11`)
- the device reports an erase or program error (bits[4:3] are set to `01`)

It is not mandatory to clear an internal erase/program error flagged by an internal ERASE, PROGRAM or READ operation (the device continues to function normally). However, the error flag remains set in the ISC-Default and the instruction register bits[6:5] until cleared by a subsequent XSC-CLR-STATUS or ISC_DISABLE instruction, Test-Logic-Reset, or V_{CC} power-down.

ISC Mode [5]

The ISC Mode bit indicates if the device is currently in ISC Mode. Several operations require that the device remain in ISC mode until that operation is complete.

RTI Entry Error [6]

RTI_ENTRY_ERROR is set when RTI is skipped after ISC_PROGRAM, ISC_READ, XSC_READ, or ISC_BLANK_CHECK. However, RTI_ENTRY_ERROR is set when the device enters RTI after ISC_ERASE.

Internal Erase/Program Status [2]

The Internal Erase/Program Status (busy indicator) bit goes Low 1 µs after an internal erase or program operation is initiated. The polling loop that checks the busy status to determine the end of the operation must not begin before the 1 µs delay, avoiding a false end-of-operation status. If polling functions are available over JTAG, then the busy indicator (bit[6]) should be polled to determine when any operation completes. Otherwise, the deterministic absolute maximum erase and programming times indicated in the SVF file should be used.

SELECT-BLOCK Register (24 Bits)

The SELECT-BLOCK register is used to select which blocks are the target of an ISC_ERASE or XSC_UNLOCK instruction. To unlock or erase a particular register or block, a `1` is loaded into the corresponding bit location in the SELECT-BLOCK register.

Table 21: **SELECT-BLOCK Register**

Bit	Name	Description
23..6	–	Not Used (set to `1`)
5	SUCR	Includes DATA-SUCR register, DONE bit, and DATA-WRPT bit
4	GUCR	Includes USERCODE, DATA-BTC, and DATA-CCB registers, and DONE bit and DATA-WRPT bits
3	Block[3]	Data block 3
2	Block[2]	Data block 2
1	Block[1]	Data block 1
0	Block[0]	Data block 0

Register Description:

USERCODE Register [32 Bits]

The USERCODE register allows easy identification of the design currently programmed into the part. While the USERCODE instruction is active, the USERCODE Register is always reloaded with the programmed USERCODE on the rising edge of TCK in the CAPTURE-DR state before moving into the SHIFT-DR state (See [Ref 2] for details).

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
08/05/08	1.0	Initial Xilinx release.
08/13/08	1.0.1	Minor style edits.
02/13/09	1.1	Updated "SVF File Generation," "MCS File Generation for the Initial Image Using the iMPACT Tool in Batch Mode," "Program Initial Image SVF Using GUI," "Initial Image Erase," "PROGRAM All SVF Sequence:," "MCS File Generation for the Update Image Using the iMPACT Tool GUI," "Update Image SVF File Generation," "Basic System Checks," "SVF Generation for Verifying the Update Image Using the iMPACT Tool GUI" sections, and Table 2.
09/15/09	1.2	Updated "Exit In-System Configuration Mode", "DONE Register (8 Bits)" and descriptions in Table 19.

Notice of Disclaimer