XILINX®

XAPP975 (v1.0.3) May 12, 2008

Low-Profile In-System Programming Using XCF32P Platform Flash PROMs

Authors: Jameel Hussein, and Rish Patel

Summary

Not all applications using in-system programming (ISP) require a full-featured solution with the capability to execute a wide range of JTAG functions in addition to programming. This application note describes a *low-profile* ISP solution, consisting of HDL IP and Xilinx® software tools, designed to handle only the JTAG functions needed for programming. The XCF32P Platform Flash PROM (PFP) programming module requires less logic than other full-featured solutions in addition to a smaller programming file.

Note: In its current form, this solution does not support the advanced revisioning feature of Platform Flash PROMs.

Introduction

In-system programming support provides designers with the flexibility to update and revise existing designs in the field. Several solutions currently exist that allow the user to package the new bitstream programming files with provided software utilities, and update the remote system via the JTAG interface. Examples of these solutions can be found in both XAPP058, *Xilinx In-System Programming Using an Embedded Microcontroller*, and XAPP424, *Embedded JTAG ACE Player*.

These solutions require either a program running in a microprocessor or special IP residing in either a CPLD or FPGA to read and process specialized files. A specialized file for programming the PROM contains the programming data wrapped in a JTAG instruction sequence. The data and instructions are sequenced as required by the PROM's programming algorithm in order to successfully program the PROM over the JTAG interface. Both solutions act as a translator, reading the source file and sending the appropriate JTAG signals to the PROM.

While these solutions allow the user to embed any functionality supported via the JTAG interface for any device in the target chain, this capability adds overhead to the programming file. In some cases, the overhead for a standard erase, program, verify flow could add between 10% on the low end, to nearly 1000% in extreme cases. Not all applications can afford this overhead, nor require the ability to perform JTAG functions beyond those needed for reconfiguration. The Platform Flash PROM programming module provides a low profile solution, requiring no command overhead and less logic to implement. The actual size of the design is greatly reduced because it only performs the necessary tasks to update the PROM. The speed of the actual programming process is greatly increased due to:

- No extra instruction needed to control the programming once started.
- Data being sent to the PFP programming module is piped straight to the PROM.
- Polling also being used to eliminate long wait times during Erase and Program.

Table 1: **Low-Profile In-System Programming Highlights**

Parameter	Value
Number of Slices	250
Erase Time	~ 30 sec
Programming Time	~ 9 sec per 8 Mbit
Verification Time	~ 4 sec

Programming Basics

The standard sequence for programming Xilinx Platform Flash PROMs is erase, program, and verify. Associated with each operation is a sequence of instructions necessary to command the PROM to perform each of these tasks. In the case of iMPACT, the programming utility sends the appropriate signals to the PROM via its JTAG port to complete each operation. For example, to erase the PROM, iMPACT must first send the command to place the device in ISP mode, then send the actual erase command specifying which blocks to erase before triggering the actual erasure of the PROM. Once erasure has started, iMPACT checks for errors and monitors the device for completion of the erase cycle.

The PFP programming module described in this application note has all of the specific steps for each operation (erase, program, and verify) hard-coded in the IP module, eliminating the need to embed these instructions in the programming file.

The PFP Programming Module

The PFP programming module consists of a state machine, hard-coding each of the steps necessary to control the PROM during programming. As the state machine issues all of the necessary JTAG commands, no additional overhead is required (no commands need to be embedded in the programming file).

This module can reside internal to the FPGA to be configured, consuming approximately 250 slices and only requiring four user I/Os to connect to the JTAG port of the PROM (Figure 1). The solution does have a few restrictions:

- The FPGA and the PROM cannot reside in the same JTAG chain.
- No other devices can be connected to the JTAG chain driven by the PFP programming module.
- Updates are made only to the PROM data contents, and by default, starts at address '0' and sequentially loads each data segment to the next 256-bit address.
- Update are not made to the PROM internal setup registers (as briefly noted in "Hardware Setup"). Therefore, the update image must only target data addresses existing in the original PROM image.

Note: The PFP programming module does not interfere with the actual PROM-FPGA configuration interface.

The full programming solution requires the user application, external logic or microprocessor program to send the `.hex` programming file to the PFP programming module (see "Programming Data Preparation and Transmittal" for details on generating the `.hex` file). The only requirements are that the data must be sent at the correct time and rate (details of the correct handshaking protocol are described in "Module Operation"). Construction of the byte-wide interface between the user application and the module are left to the user.

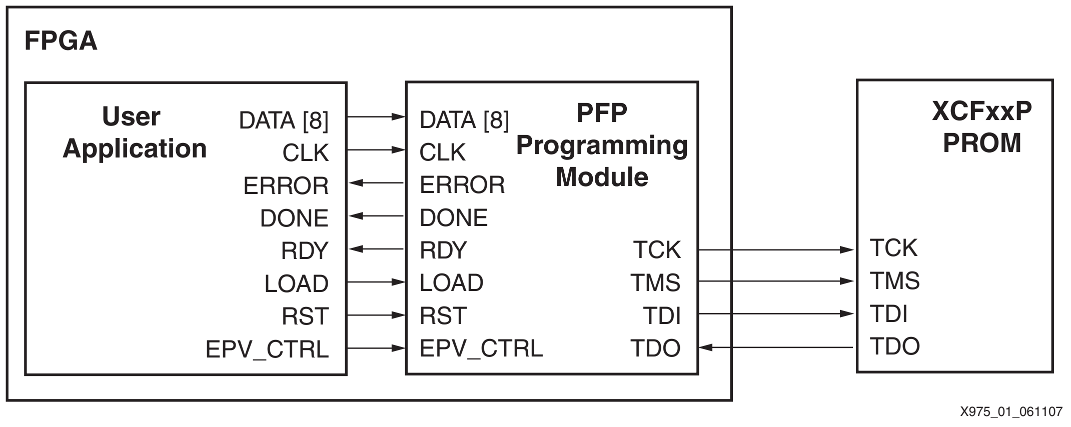


Figure 1: **Module Interface**

Module Signals

Table 2 list the various module signals, their direction and function.

Table 2: **Module Signals**

Signal Name	Signal Direction	Signal Description
RST	Input	RST is held High for one clock cycle to reset the control logic state machine and abort the current operation. After the reset sequence is complete, READY is asserted.
DONE	Output	DONE is asserted High when all operations (erase, program, and verify) are completed successfully.
ERROR	Output	ERROR is asserted High if errors occur during any of the operations (erase, program, or verify).
DATA[7:0]	Input	DATA [7:0] is the data input to the module.
CLK	Input	CLK is the clock signal to the module; maximum frequency is 10 MHz.
RDY/LOAD	Output/Input	RDY, and LOAD function as handshaking between the user application and the module to ensure that data is only sent when both parties are ready (see "Module Operation" for more details).
EPV_CTRL	Input	EPV_CTRL begins the sequence of operations to erase, program and verify the PROM.
TCK,TMS,TDI, and TDO	Output/Input	TCK, TMS, TDI and TDO are the JTAG interface signals to the PROM, using user I/Os of the FPGA.

Notes:

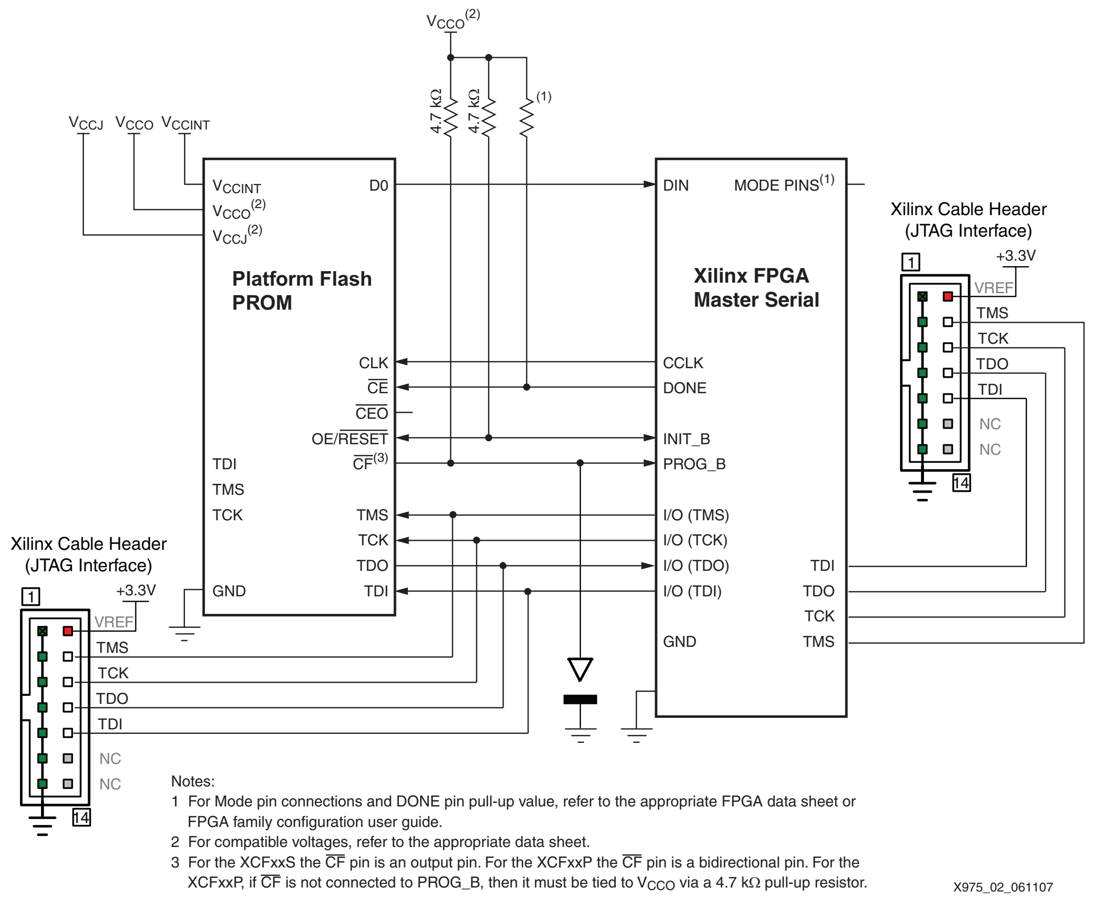
1. By default, if an error condition occurs during an operation, at the end of that operation, the controller also sends the PROM "Disable ISP" instruction in addition to the ERROR flag. If the PROM is left in ISP mode, the configuration interface cannot be accessed if the FPGA is reset for any reason.

Hardware Layout

Figure 2 illustrates the connections between a Platform Flash PROM and a Xilinx FPGA in Master Serial mode as required for configuration. Also shown are the four signals required for programming with the PFP programming module (the JTAG signals from the PROM connected to the four user I/Os on the FPGA).

Note: The PROM JTAG connections to the JTAG connections of the module form an independent JTAG chain, separate from the FPGA's JTAG chain.

Refer to DS123, *Platform Flash In-System Programmable Configuration PROMS*, for a full specification of the hardware setup for configuration in Master Serial mode.



Notes:
1 For Mode pin connections and DONE pin pull-up value, refer to the appropriate FPGA data sheet or FPGA family configuration user guide.
2 For compatible voltages, refer to the appropriate data sheet.
3 For the XCFxxS the $\overline{CF}$ pin is an output pin. For the XCFxxP the $\overline{CF}$ pin is a bidirectional pin. For the XCFxxP, if $\overline{CF}$ is not connected to PROG_B, then it must be tied to V_{CCO} via a 4.7 kΩ pull-up resistor.

Figure 2: **Hardware Layout**

Hardware Setup

During prototyping, the Platform Flash PROM is programmed by connecting the programming cable to the PROM JTAG header and using iMPACT to perform the erase, program and verify operations. After the PROM is successfully programmed, the FPGA is configured from the Platform Flash PROM. The successful configuration of the FPGA should validate the FPGA configuration hardware is setup correctly, and the PROM setup area is programmed correctly.

The PROM setup area is specially programmed to accommodate different configuration options. After the PROM setup area is programmed, the PFP programming module does not modify the setup area. The PFP programming module only changes the configuration data and leaves the setup area as originally programmed.

Note: To avoid contention on the PROM JTAG lines during programming with iMPACT, the user must ensure that the programming module is not running on the FPGA. One method is to set the FPGA mode pins to configure from JTAG and pulse the PROG_B pin. This action clears the memory of the FPGA and 3-state the I/Os connected to the PROM JTAG header.

Programming Data Preparation and Transmittal

Once a `.bit` file is created for the design, it must be properly reformatted for use with the PFP programming module. The included utility, XBINUTIL, is used to translate the binary `.bit` file into an ASCII `.hex` file (padding the last page of the translated file data to a 256-bit page boundary), while calculating and prepending a 8-bit checksum (CRC) to the file. The PFP programming module uses this CRC value to verify the programmed PROM contents during the verification operation.

For example, to create the required `.hex` file with prepended CRC:

```
XBINUTIL inputfile.bit outputfile.hex
```

Note: In order for XBINUTIL to run successfully, user must install the `CRC.pm` library file from the Perl website in the directory `\perl\site\lib\digest`.

The module expects the user application to send the programming data as it appears in the `.hex` file, reading from left to right, top to bottom. The data is stored in little endian order, for example, if the first byte of programming data is:

```
11010001
```

Then D[0] = 1, D[1] = 1, D[2] = 0, etc.

Module Operation

The control logic state machine (Figure 4) performs three operations: erase, program and verify. While the erase operation can be performed independently by the module, the program and verify operations require a more complex interaction on the part of the user application.

Erase

Erase is the first operation performed by the PFP programmer module once EPV_CTRL is asserted (High). The module erases the entire PROM during this operation and reads the status of the PROM to check for completion and any errors. For a XCF32P PROM, this operation should take approximately 30 seconds to complete. After the module completes erasing the PROM, it asserts RDY on the next rising clock edge and begins the programming sequence.

If ERROR is asserted by the module, the PROM was not erased properly. The user application must reset the module (via RST) and assert EPV_CTRL to attempt the operation again. Refer to "Debugging" for more details on the ERROR signal.

Program

After erasing the PROM, the PFP programming module asserts RDY to signal the user application that the module is ready to receive programming data. Data is read in by the module from the user application, one byte (eight bits) at a time in every load sequence (Figure 3). The data is shifted out serially to the PROM, one bit at a time. Thus, the module requires at least eight clock cycles to shift out the new data byte before signaling that it is ready for more data by reasserting RDY. The user application must monitor RDY before attempting to send new data to the module.

During the each load sequence:

- The PFP programming module asserts RDY to start the sequence.
- The user application loads eights bits of data on the DATA bus and asserts LOAD at the same rising clock edge. The data is latched by the module on the falling edge of the same clock cycle.

  Note: The module ignores any data on the bus until LOAD is asserted.
- On the next rising edge of the clock, RDY is deasserted by the module. Once RDY is taken Low, the user application must deassert LOAD before any new data can be latched.

  Note: The user application must wait for RDY to be asserted before updating the data on the DATA bus and asserting LOAD.

The module expects the CRC at the first load sequences, followed by actual programming data. After the module has received 256 bits of data, the PROM has to program the data into the actual Flash memory, requiring approximately 15 µs. The user application must accommodate this delay when transferring data to the module.

During PROM programming, the module monitors the status of the PROM and asserts ERROR if any problem occurred (refer to "Debugging" for more details on the ERROR signal). Once the user application completes sending all programming data to the module, the user application must deassert EPV_CTRL. The module then completes the transfer of the last 256 bits of data to the PROM, and if no errors occur, begins the verify operation.

Verify

Once the user application deasserted EPV_CTRL, and the PFP programming module completes sending the last set of data to the PROM, the verification sequence begins. The module reads the entire contents of the PROM and calculates a 8-bit CRC for the data. This value is compared to the CRC prepended to `.hex` file read by the module. If the CRCs do not match, then ERROR is asserted (DONE is held Low). If verification completes successfully, then DONE is asserted, all I/Os connected to the JTAG port on the PROM are set to high impedance, and the module waits for RST assertion.

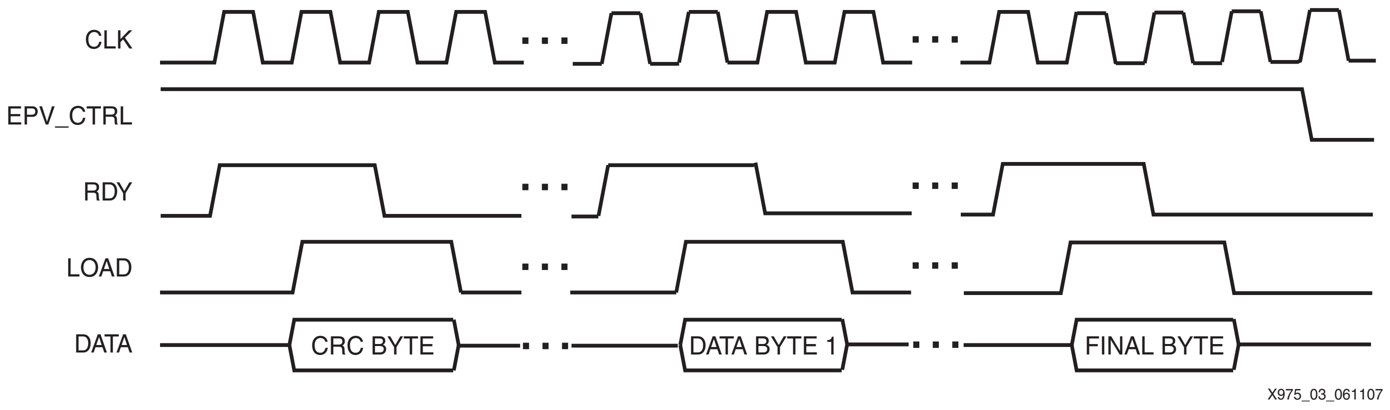


Figure 3: **Program Operation Timing**

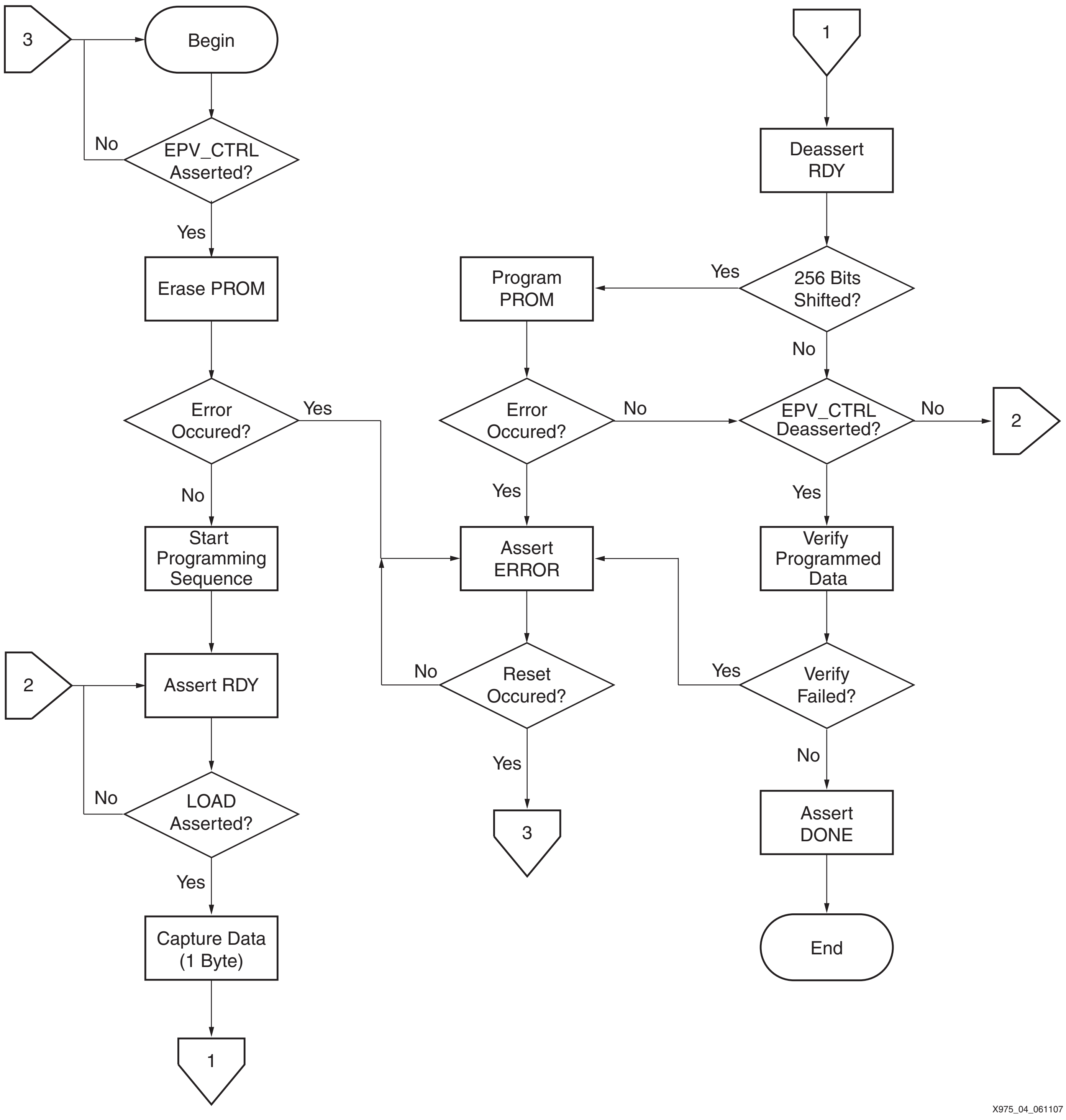


Figure 4: **Design Flow**

Debugging

ERROR can be asserted for various reasons during configuration.

During Erase

After the erase command is sent to the PROM, the PFP programming module polls the status register in the PROM to check if the erase operation is complete or if an error occurred. The status bits are read from the PROM on TDO. The module asserts ERROR if an error occurred.

During lab checkout, the erase operation can be verified by connecting the programming cable to the PROM JTAG header and using iMPACT to perform the blank-check operation.

During Program

After the program command is sent to the PROM, the module polls the status register in the PROM to check if the program operation is complete or if an error occurred. The status bits are read from the PROM on TDO, and the module asserts ERROR if an error occurred.

During lab checkout, the program operation can be verified by connecting the programming cable to the PROM JTAG header and using the iMPACT read-back operation.

During Verify

The CRC calculated during the verification operation by the module is compared against the CRC sent by the user application during programming operation. The module asserts ERROR if the CRC comparison fails. If iMPACT verifies the PROM as being correctly programmed, verify that the CRC is sent to the module in correct order.

Design Files

A `.zip` archive containing the PFP programming module design files and XBINUTIL are available for download at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=55648

Conclusion

The PFP programming module provides a low-profile, in-field programming solution when using Platform Flash PROMs. The module can reside either in the FPGA or in logic elsewhere in the system. By limiting the module's functionality to programming only, both the required logic and programming file size are reduced versus earlier solutions.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/09/07	1.0	Initial Xilinx release.
11/26/07	1.0.1	• Updated URLs. • Updated document template.
03/19/08	1.0.2	Updated associated source files.
05/12/08	1.0.3	• Correct typo regarding CRC bit size on page 6. • Updated trademark notations.

Notice of Disclaimer