Using Xilinx XCF02S/XCF04S JTAG PROMs for Data Storage Applications

Author: Arthur Khu, Farshid Shokouhi, Jameel Hussein, Amrish Patel

XAPP544 (v1.1) January 11, 2008

Summary

This application note describes a method for combining FPGA configuration data and general-purpose user data into a Xilinx XCF02S (2 Mbit) or XCF04S (4 Mbit) configuration PROM. Using a software utility to format an FPGA configuration data file and sample MicroBlaze™ C code, an FPGA configured with a MicroBlaze processor can perform a limited number of user data write operations to the unused area of the configuration PROM and an unlimited number of read operations through the PROM's JTAG interface. Board connectivity testing through JTAG is retained.

Introduction

FPGA configuration data files can be stored in one or more Xilinx XCF02S or XCF04S configuration PROMs. After the FPGA is configured, a MicroBlaze processor instantiated in the FPGA uses the JTAG port to store and retrieve code or general-purpose data from the unused area in the PROM or the last PROM in a cascaded configuration chain. Figure 1 shows the FPGA and PROM configuration with the optional HiZ_CTRL pin.

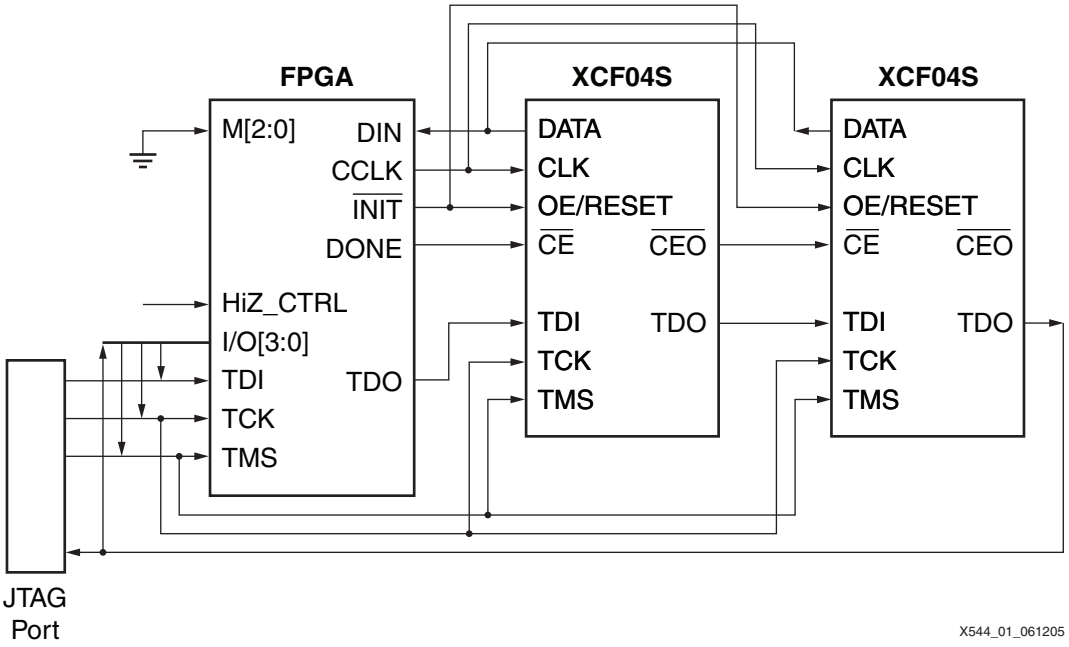

Figure 1: **Xilinx FPGA Configured by Two XCF04S 4-Mbit Platform Flash Devices**

Figure 1 shows three devices in the JTAG chain: one FPGA and two XCF04S devices. The MicroBlaze controller instantiated in the FPGA uses four I/O pins (I/O[3:0]) connected to the JTAG TAP interface to write and read user data from the last PROM in the JTAG chain. An external HiZ_CTRL pin can be added to the design, which when active 3-states I/O[3:0] to allow an external JTAG controller to drive the JTAG chain (for example, for interconnectivity testing through JTAG Boundary-Scan) without contention from the MicroBlaze controller.

The XCF02S and XCF04S PROMs are JTAG-reprogrammable Flash devices with chip erase capabilities. Data is programmed into the device one row at a time (4096 bits). Figure 2 shows the XCF04S, which has 1024 rows (the XCF02S has 512 rows).

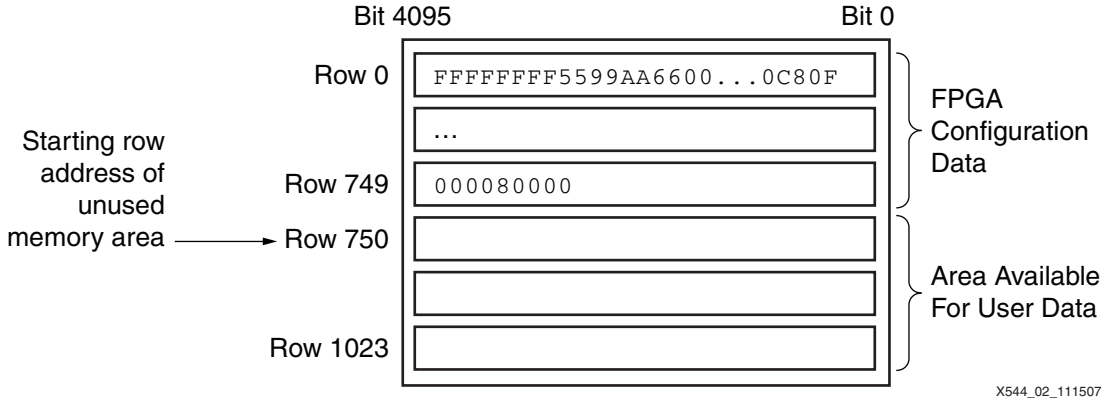

Figure 2: **Platform Flash PROM Row Architecture**

Unused rows set aside for XCF04S general-purpose data storage are partitioned into 32 128-bit pages, with page 0 holding the page status field for each of the sequential 31 data pages (Figure 3). A page is either free, used, or *stale*.

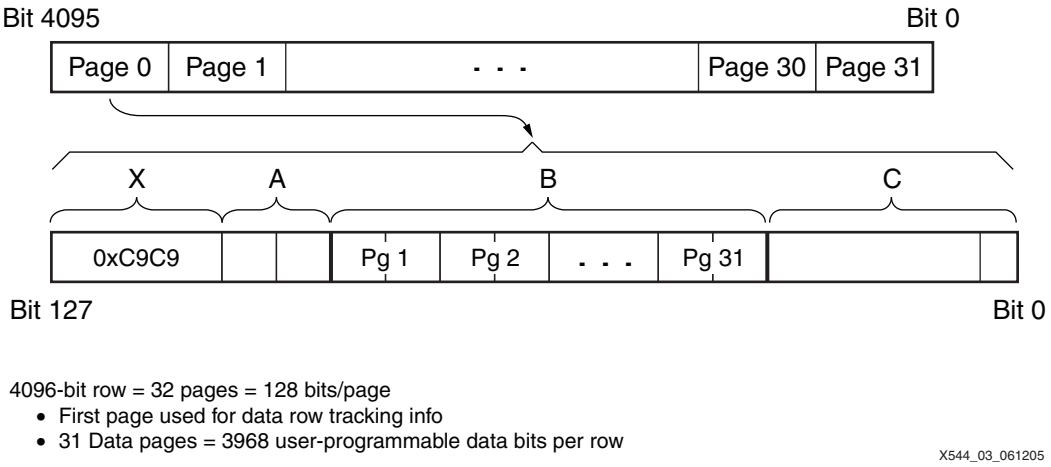

Figure 3: **Page Fields in Rows in Unused Memory Area**

Page 0 is used for data page information:

- X = Fixed 16-bit pattern (for example, `0xC9C9`) to indicate row is a user-data row
- A = Two bits to indicate state of row
  - `11` = row available
  - `01` = at least one page used
  - `00` = all pages in row are stale (do not use)
- B = 62 bits (2 bits/page for 31 pages) to indicate page state
  - `11` = page N available
  - `01` = page N used and contains valid data
  - `00` = page N data is stale (contains invalid or obsolete data)
- C = 48 user-programmable data bits
  - 32 bits of this field in the last PROM row are used to store the first row address where user data can be written

Prior to programming the PROM with the configuration bitstream data, the data is first preprocessed by a software utility called XMCSUTIL. This utility finds the first unused row after the bitstream data and inserts this information into the last row of the bitstream data file. This utility also adds a two-byte header sequence (0xC9C9) at the start of each row after the bitstream (Figure 4) to indicate that these are data rows. The PROM is then programmed with this preprocessed data file using iMPACT. After the FPGA is configured with the bitstream data from the PROM, the MicroBlaze controller reads the last row to determine where it can start writing general-purpose user data.

For example, the following command prepares an XCV50E MCS file for the PROM read/write system:

```
C:\> xmcsutil -i testfile.mcs -o new_file.mcs -17 n
```

where:

- **-17 n** specifies the PROM R/W processing option.
- **n** = 2 or 4 specifies XCF02S or XCF04S, respectively.

For this example, XMCSUTIL outputs the following:

```
xmcsutil(tm) Version 1.10 (c) 2001-2004 Xilinx, Inc.
Xilinx MCS/HEX Byte Data Utility

Mon Jan 10 16:02:27 2005

Calculating [testfile.mcs] bitstream size
==> Counted [630048] bits
==> [870 of 1024] rows available for user data
    - each row has 31 user data blocks w/ 128 bits/block
    - max [26970] user data blocks available
- max MCS byte addr [0x00080000]
Elapsed clock time = 0 seconds
```

The XMCSUTIL software reports:

- The number of bits required by the bitstream (XCV50E requires 630,048 bits or 154 PROM rows).
- The number of PROM rows available for general-purpose user data (1024 – 154 = 870).

With 31 pages (or user data blocks) per row, there are 26,970 (870 × 31) pages available (with 128 bits per page) for write/update operations.

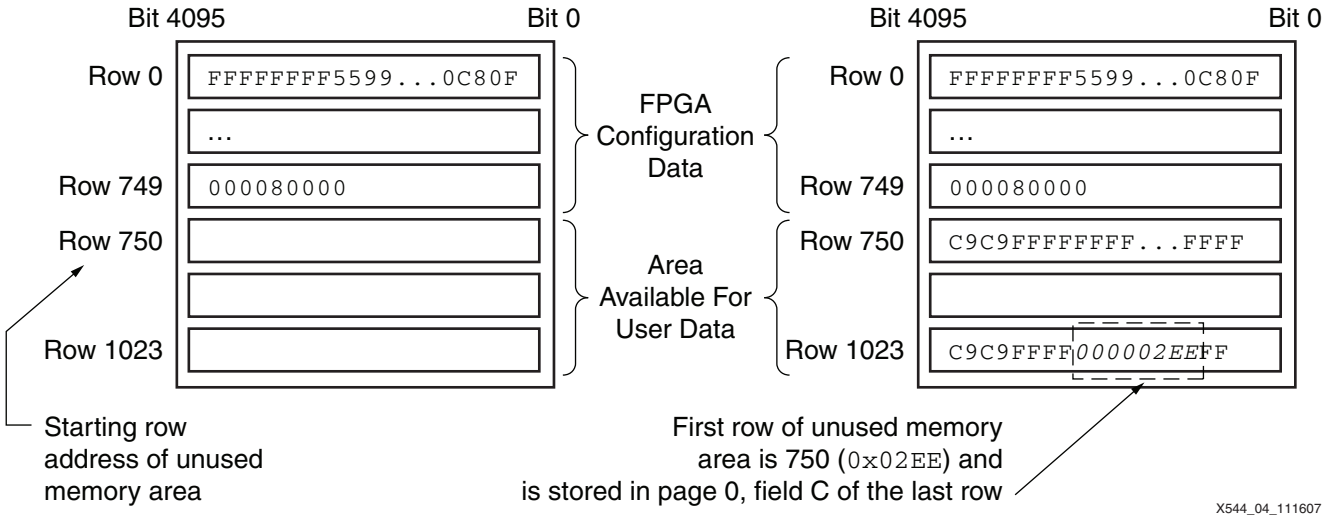

Figure 4: **Configuration PROM Data File Processed through XMCSUTIL**

Programming and Updating User Data

To program user data, the MicroBlaze controller first reads the contents of the first row in the user-data section. It checks if the first 16 bits of the row contain the `0xC9C9` pattern to indicate that this is a user-data row. It then looks for the first free page (if programming up to 128 bits) or pages (if programming more than 128 bits) by searching the page status fields in a row. The status field of the free page is changed to the used state, and the data is loaded into the corresponding page before the row is programmed (Figure 5).

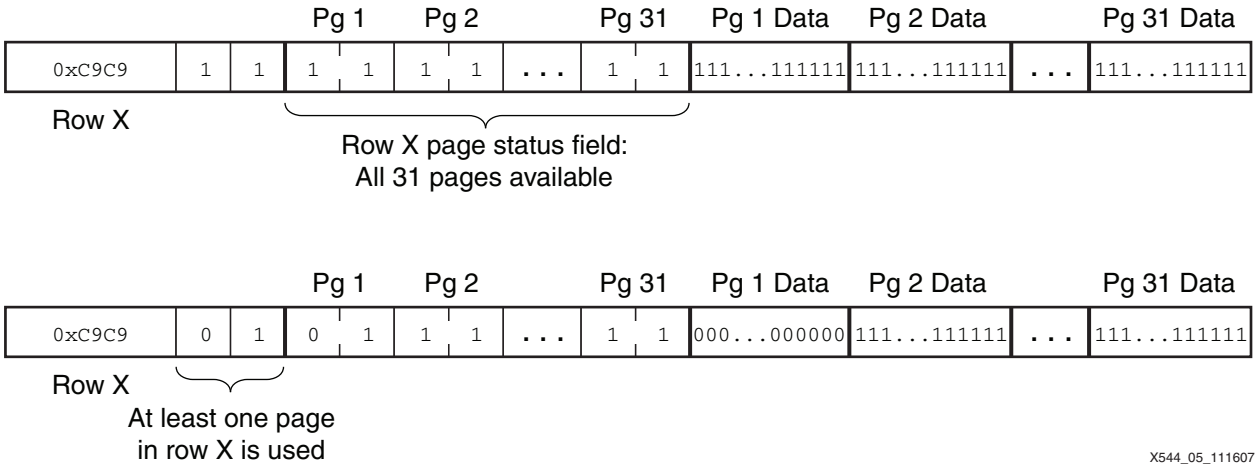

Figure 5: **Programming Page 1 with up to 128 User Data Bits**

In the sample C code `XAPP544.C`, data is written/updated and read 128 bits at a time. The function Xapp544_WritePage() calls Xapp544_ReadRow() to read the 4096-bit row into a 512-element uiRowBuf[] byte array. The FOR loop in Xapp544_WritePage() looks for the first unused row (status field = `11`) or a row which has at least one used page (status field = `01`). The function FindFreePage() then finds the first free page in this row. If the current row does not have a free page and this is not the last row in the device, FindFreePage() programs the last used page in this row as stale, marks the row as stale, and then moves to the next row to continue looking for a free page.

Once a row with a free page is found, UpdateStatusPage() updates the appropriate page status field bits. ProgramDataPage() then loads the 128 bits of user data into the appropriate location in the 4096-bit uiRowBuf[] register before calling Xapp544_ProgramRow() to program the row rowNumber set by FindFreePage().

When updating the 128-bit user data page, the system does not have to erase the entire device before programming the new data page. The system instead searches for the first used page, marks it as stale, marks the next free page as used, and then loads the new or updated data into the next free page (Figure 6). This row of data is then programmed into row X. Because the contents of the previous data page are unchanged, the system can read the contents of the stale pages to construct a change or update history.

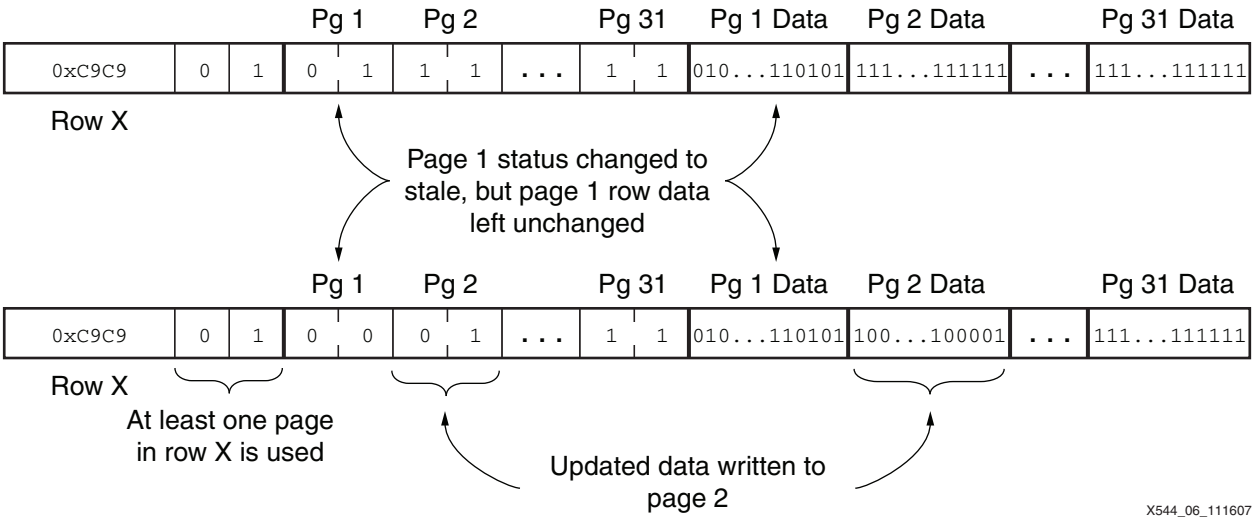

Figure 6: **Updating a 128-bit Page of Data**

In Figure 6, the previously used page 1 status is changed to stale, and the 128-bit data area of page 1 is left unchanged. The page 2 status field is marked as used, and new data is loaded into the page 2 data area. The row X status is left unchanged at `01`.

Data can be updated as long as free pages are available. Once all free pages are used, the developer has the option to block all subsequent write operations, or erase the entire PROM and then reprogram the PROM with the preprocessed bitstream data (Figure 4, page 3).

Reading User Data

The MicroBlaze controller reads the most recently written page of data by skipping all stale pages and looking for the first used page. The controller reads the first 4096-bit row in the user-data area, and then checks the 2-bit row-use field (see Figure 3, page 2 for field encodings). If the row-use field is `00`, then this row is skipped and the next 4096-bit row is read because all pages in this row are marked as stale, and the current page is not in this row. If the row-use field is `01`, then at least one page in this row is used. The controller should then check the page status fields of all 31 pages in this row sequentially to find the used page (Figure 7). See function readCurrentPageData() in the sample C code `XAPP544.C`.

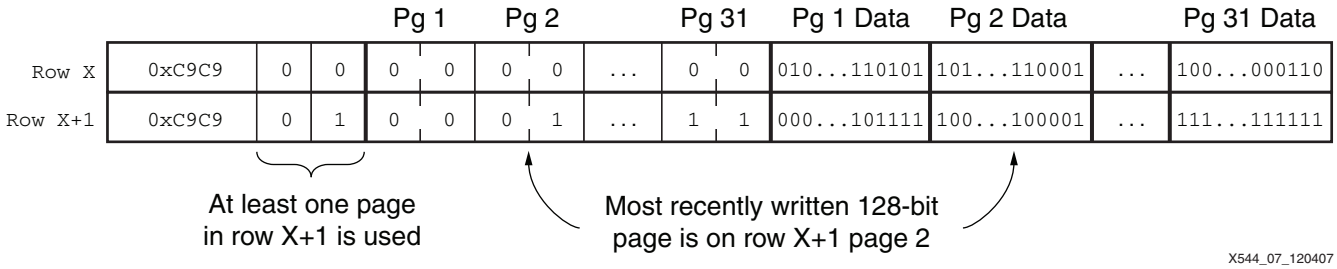

Figure 7: **The Current 128-bit Page of Data is in a Row with Row and Page Status Fields Marked as Used**

The C code can be modified to write more than one 128-bit page at a time. Also, more than one page status field can be updated at a time to indicate the writing and reading of multiple pages. For example, if the MicroBlaze controller has to write 512 bits (four pages) of user data at a time, then the following changes are required:

- FindFreePage(): look for four consecutive pages
- UpdateStatusPage(): update four status pages at a time
- ProgramDataPage(): load 512 bits of user data into the appropriate pages of the 4096-bit row

Reference Design Files

The reference design files are available as a zip archive via the following location:

https://secure.xilinx.com/webreg/clickthrough.do?cid=55769

References

1. DS123, *Platform Flash PROM Data Sheet*
2. UG161, *Platform Flash PROM User Guide*
3. Xilinx Embedded Processor Solutions

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/20/05	1.0	Initial Xilinx release.
01/11/08	1.1	• Updated template. • Updated Figure 7, page 5. • Added "Reference Design Files," page 6 and "References," page 6.

Notice of Disclaimer