# SPI Flash Programming Including Bitstream Revision Selection

Author: Ralf Krueger

Summary

Figure 1 outlines the architecture of a system supporting remote FPGA bitstream updates, bitstream updates through JTAG, and configuring an FPGA from SPI flash with different bitstream revisions. Bitstreams or design specific data are stored in defined locations in the SPI flash through a register interface. A factory installed golden bitstream provides a secure fallback in case of corrupted bitstreams. Depending on the size of the SPI device, a number of bitstream revisions can be stored in the flash device to program the FPGA based on either local or remote events. This application note provides detailed descriptions of the register interface to the SPI device. It also demonstrates how to download bitstreams and other design data to the flash from the Vivado Design Suite through JTAG. The example designs utilize the KCU105 board and a Xilinx download cable.
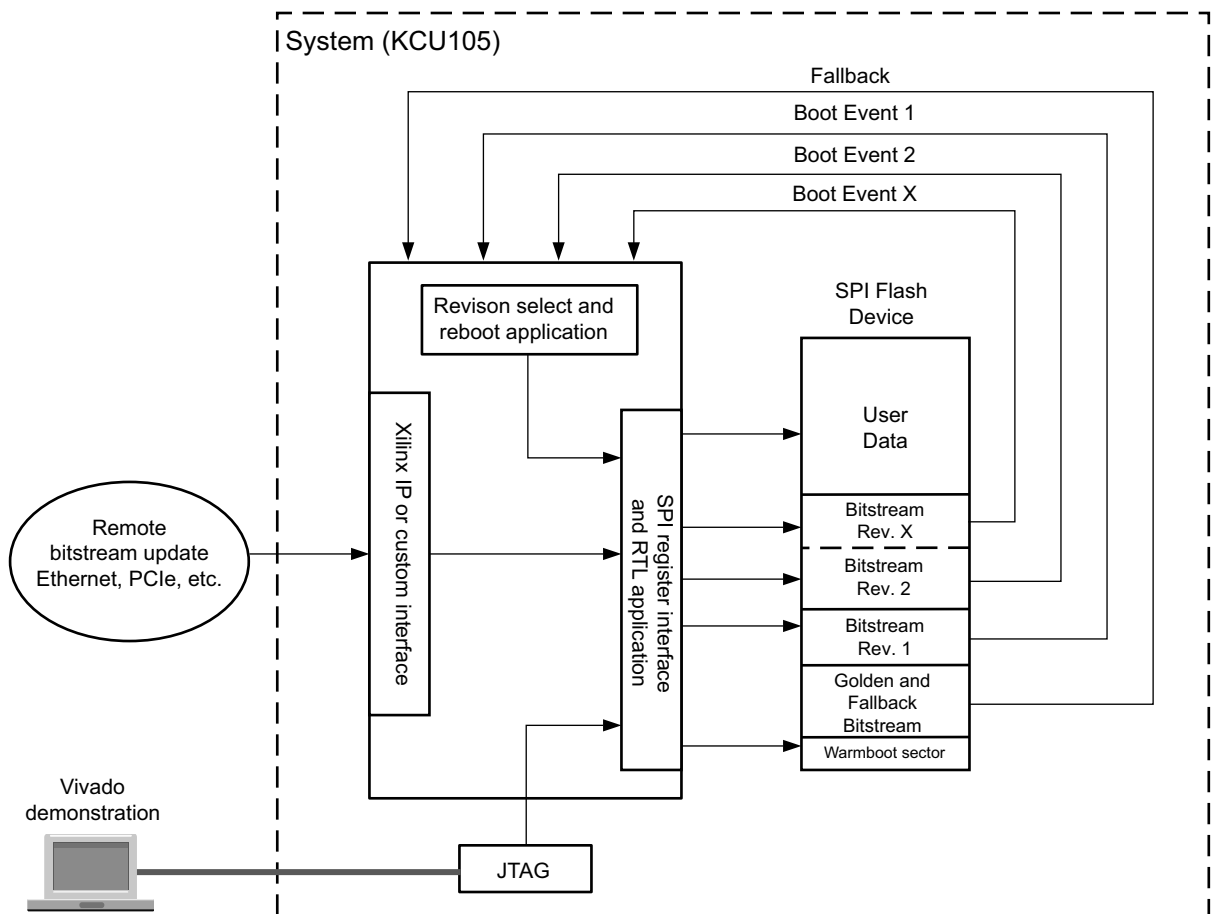
Figure 1: **System Level Block Diagram**

www.xilinx.com

Download the reference design files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design.

Features

- Support for all UltraScale and UltraScale+ FPGAs

- HDL-based flash programmer reference design with register interface

- HDL for event-based bitstream revision selection and automatic reboot examples

- Error recovery/fallback to a known good or *golden* bitstream

Introduction

One of the flash programmer reference designs implements a write interface to a SPI device using the STARTUPE3 primitive and a top-level interface design based on registers and a control signal. A SPI data load address (a subsector start address), a subsector erase count (essentially a data length count), and programming data from your application is presented to the reference design. A FIFO in the reference design buffers the data to be written to the flash device. Handshake signals control the overall erase and data programming procedure. Both bitstreams lengths and design specific data are based on SPI page size transfers and can require no-ops to complete the last page to be transferred.

A second reference design utilizes the flash programmer application to demonstrate the ability to reprogram the FPGA with different bitstream versions stored in the flash and automatically start up the newly loaded bitstream.

A Vivado tools-based example downloads bitstreams in HEX format or arbitrary ASCII data to the SPI through the Xilinx JTAG download cable. A Tcl script utilizing the hw_jtag commands provides a user-defined command-line interface to accomplish the various tasks.

Application Reference Design

At the core of this application note is the SPI flash programmer. It erases the flash, writes 32 bits of data to an intermediate FIFO, and transfers 4 bits of data from the FIFO into the flash. The reference design takes two clock inputs. One clock drives the SPI, the read side of the FIFO and the state machines. A second clock drives the write side of the FIFO. Both clocks can be asynchronous or synchronous.

For data, the top-level application interface is through a set of three basic registers. Data valid signals indicate when those registers contain valid data. A data register is used to transfer the actual SPI programming data to the SPI flash programmer (Figure 2). The beginning of a programming cycle loads the SPI load address (start address), a SPI data page count, and a SPI subsector count. When those three registers are valid, the top-level application signals that the erase process can be started. The programmer asserts an *erase in progress* signal to erase the appropriate number of subsectors in the SPI. The end of the erase cycle is signaled to the top

application by deasserting the *erase in progress* signal. The top-level application can then proceed to download the data, along with a FIFO write enable signal. The aggregate amount of data downloaded must always be a complete number of pages (modulus 256) to be programmed into the flash. If needed, most FIFO related flags are brought out to monitor the top-level application. Finally, a signal is available to indicate the completion of the data-write cycle.

Figure 2: **SPI Flash Programmer Block Diagram**

Table 1 describes the SPI flash programmer ports. Optional ports can be used to monitor any potential issues with the FIFO, notify when the write process is complete, or provide a reset if there are code modifications that cause errors in the erase or transfer processes.

Table 1: **SPI Flash Programmer Interface Ports to the Top Level**

Port	Direction	Width	Description
Clk	Input	1	Clock driving the SPI flash, state machines, and FIFO RDCLK.
fifoclk	Input	1	FIFO write clock.
data_to_fifo	Input	32	Bitstream or user data loaded in to the flash.

Table 1: **SPI Flash Programmer Interface Ports to the Top Level** *(Cont'd)*

Port	Direction	Width	Description
startaddr	Input	32	The bitstream/data load address (start address) for loading data. Must be SPI page aligned.
startaddrvalid	Input	1	The load address in the start address register is loaded and is valid.
pagecount	Input	17	Total data page count. Must be 256 byte aligned.
pagecountvalid	Input	1	The number for data pages is loaded and is valid.
sectorcount	Input	13	Number of subsectors to be erased prior to loading data.
sectorcountvalid	Input	1	The total subsector count for erasing and programming is loaded and is valid.
fifowren	Input	1	FIFO write enable. Assert this signal when data is transferred to the FIFO through the data_to_fifo registers.
fifofull	Output	1	Optional: FIFO full flag.
fifoempty	Output	1	Optional: FIFO empty flag.
fifowrerr	Output	1	Optional: FIFO write error flag.
fiforderr	Output	1	Optional: FIFO read error flag.
writedone	Output	1	Optional: Signals the completion of the data write process.
reset	Input	1	Optional: Design reset. Not required since the state machine always returns to idle.
erase	Input	1	Assert when ready to start the erase process.
erasing	Output	1	Signals the completion of the erase process.

Example Designs

The two example designs in this application note are used to demonstrate all the capabilities of the SPI flash programmer.

• A Vivado tools-based Tcl script downloading bitstreams utilizes the download cables and JTAG. The data is sent through a BSCAN primitive to the flash programmer module that writes the bitstream or design data to the SPI device.

• Revision selected applications are stored in the SPI device at user-specified locations and loaded by pressing switches on the KCU105.

Flash Programmer Example

The block level diagram in Figure 3 shows the top-level design and SPI flash organization.
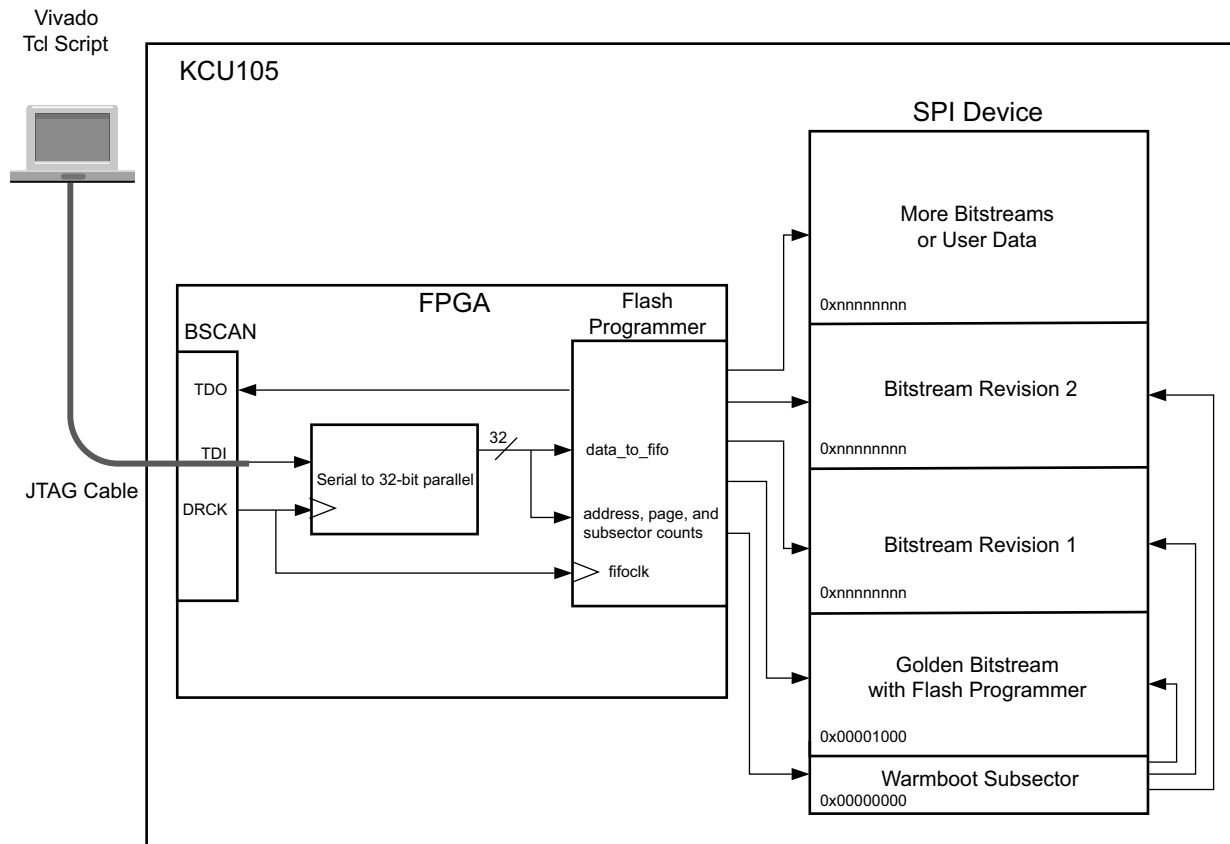
X17265-062416

Figure 3:   **Downloading Bitstreams and Data to the SPI Flash Device**

The principal idea in this example is to have a warmboot SPI subsector at address `0x00000000` that contains a very small bitstream. This bitstream is used to execute a warm boot IPROG instruction to an address somewhere in the SPI address space where a valid bitstream resides. This warmboot sector is updated (as needed) by either the Tcl script or the revision control bitstreams. If one of the bitstreams becomes corrupted, the system automatically falls back to the golden bitstream. The warmboot bitstream typically resides in the subsector above the golden bitstream at address `0x00000000`. The following is an example of the warmboot bitstream loaded by the Tcl script.

```
AA995566  -- Sync Word
20000000  -- NOP
20000000  -- NOP
3003E001  -- BPI/SPI configuration option register
0000066C  -- SPI Read Opcode (Quad Fast Read 32 bit, X4, 32 bit Addr)
30008001  -- Write 1 word to CMD register (flip the 3 to a 0)
00000012  -- BPI/SPI restart bitstream read
20000000  -- NOP
20000000  -- NOP
30020001  -- Write to WBSTAR
XXXXXXXX  -- Warmboot start address
30008001  -- Write 1 word to CMD register
0000000F  -- IPROG
20000000  -- NOP
20000000  -- NOP
```

A Vivado tools-based Tcl script provides a command line interface for downloading bitstreams or user data. The script sets the download cable into JTAG mode and then uses the *HW_JTAG Tcl commands for communication with the BSCAN primitive. The DRCK clock output of the BSCAN primitive clocks and drives a shift register that deserializes the data and loads the appropriate registers. When done, the erase signal is asserted to start the erase process. The flash programmer module asserts the erasing flag and the Tcl script waits for the erasing flag to deassert. Next, the data that was previously loaded from a file into a buffer is downloaded until the buffer is empty. It is important that the buffer size is always a multiple of a SPI page size. The example script automatically takes care of aligning (filling) the data in the buffer in case the file has an incomplete last page (the most likely scenario).

The cable download process and effective cable clock frequency can vary significantly from download to download. The DRCK clock frequency out of the BSCAN primitive cannot be predicted other than that it is significantly slower than the SPI clock. For this reason and in this flash programmer example, it is not necessary to synchronize signals between the two clock domains.

This flash programmer example and the application note description are designed for use with the Xilinx KCU105 evaluation board. The 256 Mb SPI flash used on the board is mounted on the top side of the board and connects to the FPGA through the STARTUPE3 primitive and the associated pins in Bank0. The SPI flash on the bottom side of the board is not used in this example. Normally, the SPI flash device on the KCU105 board can accommodate two uncompressed bitstreams. For this example however, only compressed bitstreams are used because the revision select example requires storage capacity for three bitstreams. Xilinx does not recommend relying on the compressed bitstream sizes for selecting SPI flash, because a relatively small change in the design can result in a significant increase in bitstream size. However, since the example designs are fairly small and utilize less than 1% of the resources, any changes made to the designs should not result in bitstream sizes exceeding the capacity of the 256 Mb SPI flash. For your specific application, Xilinx recommends selecting a SPI flash device that is only capable of handling full-size bitstreams.

IMPORTANT: *This example was developed based on the Micron N25Q256A. Later devices can require changing the 3-byte erase and page program commands to 4-byte commands. For example, CmdSSE24 changes to CmdSSE32 and CmdPP24Quad changes to CMDP32Quad.*

Downloading the Reference Design and Directory Structure

Download the Reference Design and install (uncompress) to a directory of your choosing. The root name of the example design is xapp1191. There are two design directories: `spiflashprogrammer` and `rev_sel`. The `spiflashprogrammer` directory contains the flash programmer design and a top-level design. The top-level design interfaces to the Vivado tools on one side and the flash programmer design on the other side. The `rev_sel` directory contains two designs that demonstrate the revision select portion of this application note and utilize the `spiflashprogrammer` module. It can be viewed as an example application of the `spiflashprogrammer` module. The xapp1191 directory also contains the Tcl script (`write_to_spi.tcl`) used for downloading bitstreams from the Vivado tools to the application running in the FPGA on the KCU105 as well as a number of HEX files (precompiled bitstreams) used in this example.

Hex bitstream files are generated with the `write_cfgmem` command in the Vivado tools. A `write_bitstream` and `write_cfgmem` Tcl script is provided in the xapp1191 directory (`gen_bitstream.tcl`). For the example to work after recompiling the designs, copy the HEX files from the `<application_name>` directories to the xapp1191 root directory or modify the write_cfgmem command to write directly to the root directory.

Note: The -loadbit address option in the write_cfgmem command does not matter.

```
Set all the appropriate properties and variables prior to executing the following:
write_bitstream $impl_path/${top_name}_compressed.bit -force -verbose
write_cfgmem -format hex -interface SPIx4 -size 32 -loadbit "up 0x0
$impl_path/${top_name}_compressed.bit" -file ${top_name}_compressed -force
```

All bitstreams are in a compressed format, which is reflected in the filenames. There is also an uncompressed version of the `flashprogrammer` example to show the difference in erase and load time.

www.xilinx.com

Step by Step Instructions

Loading the Golden Bitstream and Updating the Bitstream and Fallback

The `write_spi_tcl` script contains a number of procedures. The main procedure you directly utilize is called wr_data. To run the wr_data procedure, several command line arguments are used including load address (the SPI start address), warmboot address (likely always 0), filename, and bitstream type (update, warm, other). All arguments can be explicitly spelled out or by using the first character of the command. The -h argument displays all options. The script does some error checking, opens the hardware cable in JTAG mode, loads the appropriate file, performs miscellaneous operations on the data, and then proceeds to erase and program the SPI flash device. While executing the wr_data command, informational and status messages are written to the Td console window.

CAUTION! *Canceling a download process while running can leave the application's state machine in an undefined state. Disconnect the hardware server, reopen the target, and reprogram the device with the* `spiflashprogrammmer_top.bit` *file prior to proceeding.*

---

1. Connect the cable to the KCU105 board and ensure it is powered up.

2. Go to the `spiflashprogrammer` directory and open the `spiflashprogrammer` (`spiflashprogrammer.xpr`) design in the Vivado tools.

Since most of the following steps are executed through the Td console window in the Vivado tools, float and enlarge this window to increase visibility into the trail of messages. Move the window to a free area on the screen, have the window always in view and not covered by the main Vivado tools cockpit.

3. In the Td console window source, the `write_to_spi.tcl` source
   `<path_to_xapp1191_dir>/write_to_spi.tcl`

4. Open the Hardware Manager, connect to the target, and program the device with the design bitstream (`spiflashprogrammmer_top.bit`). The LEDs on the board walk from LED0 to LED7. This is the *signature* of this particular design.

5. This step loads the golden bitstream, which is the top design. All bitstreams should be in the root directory of the xapp1191 directory. In the console command line window, type the following.

   `wr_data -f spiflashprogrammer_top_compressed.hex -l 0x1000`

TIP: *Just entering* `wr_` *uniquely identifies the Tcl command loaded in* *for auto completion. There is also a help command* **wr_data -h** *or* **wr_data -help**.

A standard Tcl command line pop-up window appears with the message **Running scan_dr_hw_jtag** indicating that the JTAG operation and SPI programming is in progress. In the message part of the console window (Figure 4), the INFO messages are displayed first from the hardware manager and then some information about the file including the SPI loading address, erase, and programming in-progress messages. The entire process takes about one to two minutes depending on the host machine and its load. After programming is complete, the next free subsector address in the SPI is reported, as well as an elapsed time. Take note of the next free subsector address (`0x26f000` in the Figure 4 example) because it is the address for the next bitstream to be loaded.

```
Td Console                                                                    ? _ □ ⤢ ×
 wr_data -f spiflashprogrammer_top_compressed.hex -l 0x1000
 INFO: [Labtools 27-2285] Connecting to hw_server url TCP:localhost:3121
 INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Xilinx/00000acb4bc301
 INFO: [Labtoolstcl 44-467] Setting hw_target localhost:3121/xilinx_tcf/Xilinx/00000acb4bc301 into jtag_mode
 INFO: 5088360 ascii Bytes in file,  2544180 spi bytes, 636045 FIFO words
 INFO: Completing last page by adding  408 (0x198h) characters, # of subsectors 623  (0x26fh), # of pages 9939 (0x26d3h)
 INFO: Load Address: 0x1000h, Warmboot bitstream at: NA
 INFO: Erase in progress:
 INFO: Erasing done in 0 Minutes and 39 Seconds
 INFO: Programming SPI in progress

 DONE

 INFO: Next subsector starts at after the current bitstream:  0x26f000h
 INFO: Total Wall Time elaspsed: 1 Minutes and 44 Seconds


Type a Tcl command here
```

X17259-062416

Figure 4:   **Td Console in Step 5**

The golden bitstream is now loaded and stored at address `1000h`.

6. In this step, an updated bitstream is loaded along with a warmboot address to subsector 0. One of the `revsel` bitstreams is used in this example to demonstrate the update bitstream load and the fallback feature. A corrupted version of that very same bitstream is used to demonstrate fallback.

💡 **TIP:** *The Td console window (Figure 5) should be available to view.*

a. In the console command window (Figure 5) type the following (the sequence of the argument does not matter).

```
wr_data -l 0x26f000 -f revsel_leds_midout_compressed.hex
```
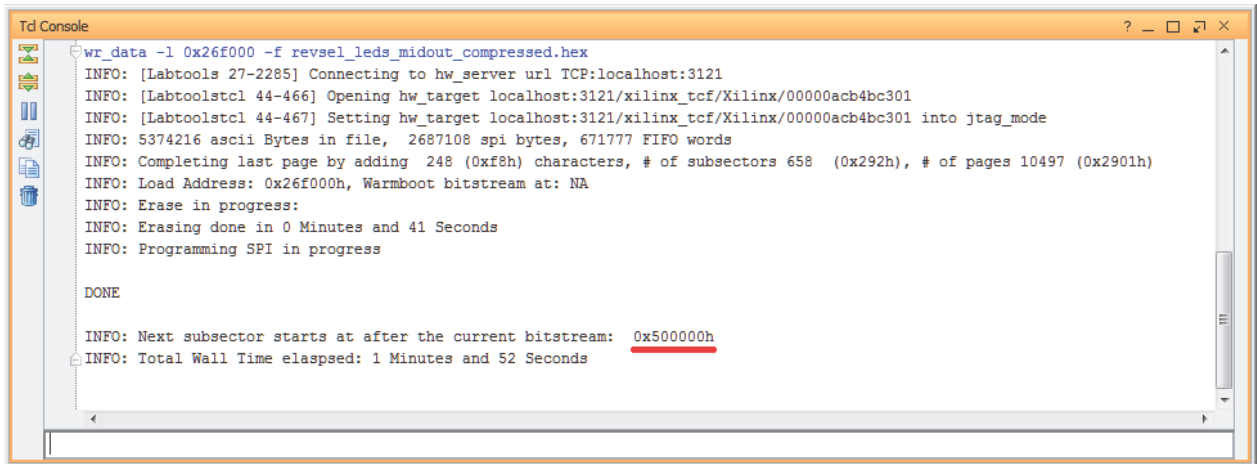
X17260-062416

Figure 5: **Td Console in Step 6**

b. Take note of the next free subsector address at `0x500000`.

c. Load the warmboot address to subsector `0`.

```
wr_data -w 0x26f000 -t w
```

d. While before a `-o` was sufficient (the default), at this point in the process, the file type must be explicitly specified. Subsector 0 now contains the short warmboot bitstream.

e. Push the PROG button on the KCU105. The user LEDs now walk from the middle LEDs out to the left and right. This is the signature of this design/bitstream.

7. This step demonstrates the fallback feature. Locate the five user switches (SW6—SW10) on the KCU105. Press the switch SW9 to return the FPGA to the golden bitstream as indicated by the 0 to 7 walking LEDs. This is further described in the Bitstream Revision Select Example.

    Note: If you accidentally press SW6, the FPGA will attempt to load another bitstream from a higher address in the SPI. Since that bitstream is not yet loaded, the FPGA programming is lost. Go back to step 4 and start over. Pressing SW8 will reload the current bitstream and the other switches have no effect.

    a. A bitstream that has bit number 999 in the HEX file intentionally flipped from a 0 to a 1 will be loaded. This is the `revsel_leds_midout_compressed_fbbad.hex` bitstream in the xapp1191 directory and is otherwise completely identical to the original bitstream `revsel_leds_midout_compressed.hex` that was loaded previously.

```
wr_data -l 0x26f000 -f revsel_leds_midout_compressed_fbbad.hex
```

    b. The warmboot subsector also needs to be reloaded because pressing SW9 caused the subsector `0` warmboot address to be overwritten with address `1000h` (the address of the golden bitstream). This is further described in the Bitstream Revision Select Example.

    c. Typing in the following loads the warmboot bitstream pointing to address `26f000h` into subsector `0`.

```
wr_data -w 0x26f000 -t w
```

    d. Press the PROG button and the attempt to load the bad bitstream fails, and the FPGA programming falls back to the golden bitstream.

e. To reload the good bitstream type in the following.

```
wr_data -l 0x26f000 -f revsel_leds_midout_compressed.hex
```

f. Press the PROG button again and the update bitstream configures the FPGA again without falling back.

Bitstream Revision Select Example

This dynamic bitstream revision select example is based on the ability to reprogram the warmboot subsector `0` with an IPROG address, which points to the boot bitstream already stored in the SPI device. This example utilizes a slightly modified version of the spiflashprogrammer to erase subsector `0` and then write a warmboot bitstream to subsector `0`. Multiple warmboot bitstreams are stored in a single block RAM. Based on an event in the system, in this example it is a switch, the appropriate preloaded warmboot bitstream in the block RAM is selected and written to subsector `0`. After the subsector is programmed, the currently running bitstream revision issues an IPROG command through the ICAP primitive. The FPGA boots from address `0x00000000`, that contains the warmboot bitstream pointing to the desired bitstream location in the SPI device. This example uses the block RAM INIT attribute to properly initialize the block RAM at the desired locations. These bitstreams contain the start address of the selected bitstream to be booted. The warmboot bitstream in the block RAM is constructed identical to the bitstream outlined earlier. The load address in the warmboot bitstream is predetermined by the load addresses used in step 7 (e.g., `0x1000` and `0x26f000`).

The following is an example of warmboot bootstream stored in block RAM.

```
INIT_00 => X"ffffffff_ffffffff_ffffffff_ffffffff_ffffffff_ffffffff_ffffffff_ffffffff",
INIT_01 => X"20000000_00000012_30008001_0000066C_3003E001_20000000_20000000_AA995566",
INIT_02 => X"ffffffff_20000000_20000000_0000000f_30008001_00001000_30020001_20000000",
INIT_03 => X"ffffffff_ffffffff_ffffffff_ffffffff_ffffffff_ffffffff_ffffffff_ffffffff",
```

The following is an ICAP reboot command sequence.

```
AA995566    -- Sync Word
20000000    -- Type 1 NO OP
30020001    -- Type 1 Write 1 Word to WBSTAR
00000000    -- Warm Boot Start Address
20000000    -- Type 1 NO OP
30008001    -- Type 1 Write 1 Words to CMD
0000000F    -- IPROG Command
```

Continuing from the Loading the Golden Bitstream and Updating the Bitstream and Fallback section, it is assumed that the golden bitstream and revision 1 bitstream are loaded.

1. Commence loading the revision 2 bitstream.

   a. Press SW9 to return to the golden bitstream.

   b. The LEDs indicate that the golden bitstream is active.

2. Input the correct address (a `5` followed by five `0`s) and second revision bitstream.

    ```
    wr_data -l 0x500000 -f revsel_leds_7to0_compressed.hex
    ```

    Followed by this command:

    ```
    wr_data -w 0x500000 -t w
    ```

3. The bitstream for revision 2 is loaded into the SPI and is ready to run.

    a. Push the PROG button on the KCU105.

    b. The LEDs now walk from the left to right, which is the signature of this design/bitstream.

4. Press SW8 and the revision 1 bitstream is booted (see LEDs).

    a. Press SW6 and the FPGA returns to the revision 2 bitstream.

    b. By selecting a switch (SW8 or SW6) for revision 1 or 2, reloads the requested bitstream.

5. To demonstrate that the fallback to the golden bitstream still works, a *bad* revision 2 bitstream is loaded with the bit 999 flipped from 0 to 1.

6. By pressing SW9, the FPGA returns to the golden bitstream. Next, load the following address.

    ```
    wr_data -l 0x500000 -f revsel_leds_7to0_compressed_fbbad.hex
    ```

    a. As shown previously, the warmboot sector must be reloaded. This is because pressing SW9 in step 6 cause the subsector `0` warmboot address to be overwritten with address `1000h`, which booted the golden bitstream.

    b. In this example, the revision 1 bitstream is the default revision bitstream.

    c. The following command loads the warmboot bitstream and points address `26f000h` to subsector 0.

    ```
    wr_data -w 0x26f000 -t w
    ```

    d. By pressing the PROG button, the revision 1 bitstream loads.

    e. By pressing the SW6, the revision 2 bitstream attempts to load. However, since this bitstream now has a bad (flipped) bit, the load fails and the FPGA program goes back to the golden bitstream.

7. To reload the good revision 2 bitstream.

    a. Input this command line.

    ```
    wr_data -l 0x500000 -f revsel_leds_7to0_compressed.hex
    ```

    b. Follow with this command line.

    ```
    wr_data -w 0x26f000 -t w
    ```

    c. Press the PROG button, followed by pressing SW6, and the good revision 2 bitstream boots.

    d. The revision can be booted again by pressing SW8 and SW6. Alternately the following command loads the revision 2 bitstream directly after pressing PROG.

    ```
    wr_data -w 0x500000 -t w
    ```

Programming a Full-size Bitstream

The xapp1191 directory contains a full-size bitstream with the **spiflashprogrammer_top** design. The KCU105 SPI device accommodates two full-sized KCU040 bitstreams. The time to load this type of bitstream is significantly longer (in both erase and programming time) than the compressed versions. The actual elapsed time depends on the host machine and the current load of the host machine. The following example loads a full-size bitstream to address `0x1000` of the SPI device. A typical execution time is in the 10 minute range. The full bitstream requires almost ½ of the 256 Mb SPI device (about 120 Mb of the 256 Mb).

Note: Performing this full-size bitstream program overwrites all previously loaded bitstreams and data from previous examples.

1. Press SW9 on the KCU105 to return to the golden bitstream (Figure 6).

   ```
   wr_data -f spiflashprogrammer_top.hex -l 0x1000
   ```
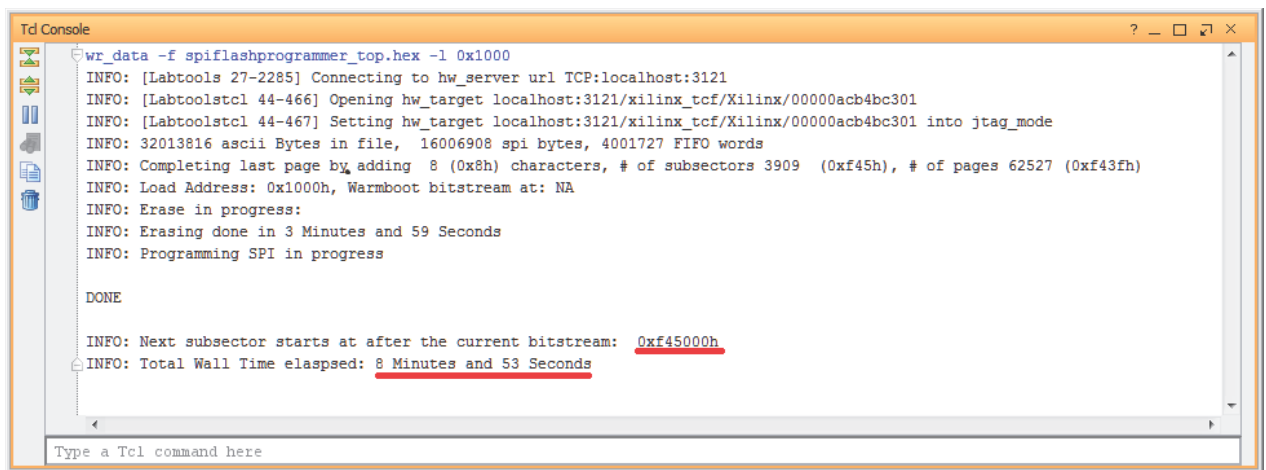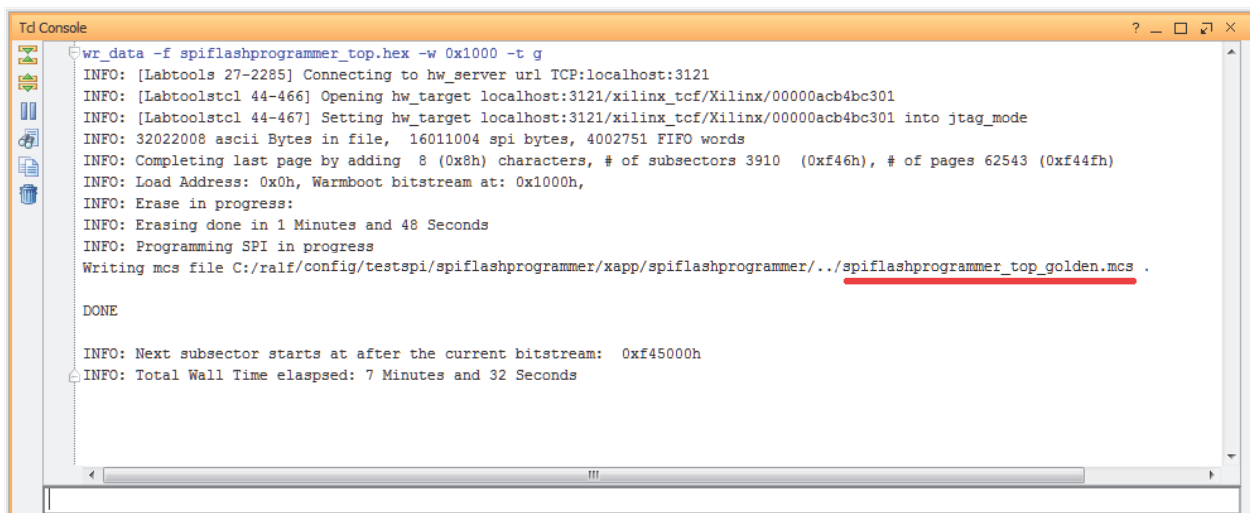

X17261-062416

Figure 6:   **Td Console in Step 1**

2. The Tcl script offers the additional capability to write a single MCS file containing both the warmboot address and golden bitstream. This MCS file can be used to preload the SPI device using the hardware manager. The file name of the generated MCS file has `_golden` appended to it. The following command (Figure 7) loads the warmboot subsector, the golden bitstream, and writes a single MCS file. When programmed into the SPI flash (using the hardware manager), the FPGA is booted and the golden bitstream is loaded because the warmboot subsector is programmed with the golden bitstream start address. The warmboot subsector can also point to a specific bitstream revision. However, that revision must be present/loaded in the SPI flash device for the FPGA to boot from the SPI device.

   ```
   wr_data -f spiflashprogrammer_top.hex -w 0x1000 -t g
   ```

Figure 7: **Td Console in Step 2**

3. To return the download cable to non-JTAG mode, open the target device in the hardware manager again.

Conclusion

The ability to program an SPI flash device remotely or through JTAG, update bitstreams, and revision select different bitstreams is demonstrated in this application note. The examples in the reference design files demonstrate common uses for using SPI flash to program FPGA bitstreams.

Reference Design

Download the reference design files for this application note from the Xilinx website.

Table 2 shows the reference design matrix.

Table 2: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Ralf Krueger
Target devices	UltraScale and UltraScale+ FPGAs
Source code provided	Yes
Source code format	VHDL
Design uses code and IP from existing Xilinx application note and reference designs or third party	No
Simulation	
Functional simulation performed	No
Timing simulation performed	N/A
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version used	Vivado simulator version 2015.4 or later
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado synthesis version 2015.4 or later
Implementation software tools/versions used	Vivado version 2015.4 or later
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KCU105 evaluation board

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/19/2016	1.0	Initial Xilinx release.
02/15/2018	1.1	Updated the Flash Programmer Example section and added an important note on page 6.

XILINX
ALL PROGRAMMABLE™

Please Read: Important Legal Notices