



XAPP468 (v1.1) July 7, 2009

Fail-Safe MultiBoot Reference Design

Author: Jim Wesselkamper

Summary

This application note describes a reference design that adds fail-safe mechanisms to the MultiBoot capabilities of the Extended Spartan®-3A family of FPGAs (Spartan-3A, Spartan-3AN, and Spartan-3A DSP FPGAs). The reference design configures specific FPGA logic via an initial bitstream that determines which application (by alternate bitstreams) to load. The decision as to which bitstream to load, if an alternate is loaded at all, is based on the bitstream revision, the number of prior configuration attempts, and the integrity of the alternate bitstreams. The algorithms that test bitstream integrity and select the bitstream image to load are implemented using a PicoBlaze™ controller. Additional independent modules manage communication with the Internal Configuration Access Port (ICAP) and the Serial Peripheral Interface (SPI) flash device.

Introduction

Traditionally, embedded applications using an FPGA with a changeable configuration also include a microprocessor to manage bitstream loading. For example, the microprocessor determines which of several bitstreams stored in a central repository is the latest version, stores it in local memory, and loads the FPGA with this newer version. It can also monitor INIT_B and DONE and handle file corruption or failed configuration attempts by reverting back to an older or known good bitstream.

System-on-chip (SOC) designs using a microprocessor with dependencies on the FPGA present a paradox. If the processor is embedded in the FPGA, how can it recover from a failed FPGA configuration attempt? If the processor uses the FPGA to access memory, how can it load the FPGA before the FPGA is configured?

The Extended Spartan-3A family solves these problems without the need for external components. The MultiBoot capabilities of these devices enable the FPGA to load a configuration from one of several bitstreams stored in local memory. The bitstream located at 0x0 always loads first. This bitstream can configure logic with the intelligence to load an alternate bitstream version and to detect if this alternate bitstream is corrupted. If a short configuration time is required, the initial bitstream can be a streamlined bootstrap designed to quickly load a second bitstream. The MultiBoot functionality provides the best trade-off between the low cost of a small flash memory with a single upgradeable bitstream and a large, robust flash containing multiple bitstreams.

Figure 1 shows how this reference design organizes the bitstream images in flash. A bootstrap image is always loaded first. The bootstrap image then examines the application images and decides whether or not to use one of those images by issuing a MultiBoot command to the ICAP.

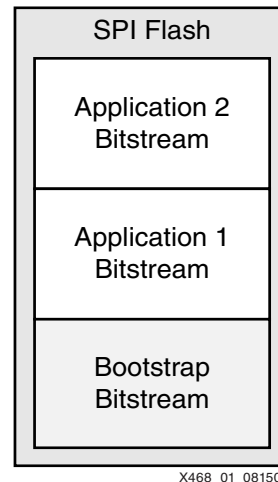


Figure 1: SPI Flash Memory Implementation

The content of the SPI flash can be organized in many other ways. The number of application bitstreams can vary. Also, the amount of logic in the bootstrap image can vary from an image that contains the bootstrap only to an image that contains the bootstrap and a complete application as well.

Implementation

The reference design implements the bootstrap loader (Figure 1) using SPI flash memory for bitstream storage. The large, external SPI flash memory provided on the Spartan-3AN FPGA Starter Kit board accommodates three application bitstreams plus the bootstrap bitstream. The In-System Flash (ISF) memory of the Spartan-3AN device accommodates one application bitstream plus the bootstrap bitstream. The MultiBoot functionality is not restricted to SPI memories and can be used with other types of memory as well. Parts of the reference design source code can be used to implement similar solutions on other Extended Spartan-3A family designs using other packages and configuration modes.

The bootstrap loader determines which application bitstream is used to configure the FPGA. The choice is based on the application bitstream revision, the history of prior configuration attempts, and valid bitstream integrity. The bootstrap loader marks each configuration attempt in a history record just before it instructs the ICAP to begin configuration with the application bitstream. After the configuration is complete, the embedded application is then responsible for writing and updating the application bitstreams in flash memory and marking a successful configuration attempt as successful in the history record. Optionally, the configuration can obtain a new application bitstream revision from a repository and write it into SPI memory.

The bootstrap design and the application designs use the same PicoBlaze processor-based submodule to interface to the SPI flash memory. The PicoBlaze processor firmware supports the functions needed by both sets of designs. These include:

- Evaluating and selecting the optimal application bitstream to load
- Accessing the flash memory to perform read, write, and erase operations
- Maintaining algorithms that keep track of each application bitstream's integrity
- Maintaining algorithms that update the history record to keep track of configuration attempts

The firmware routines differ slightly between a bootstrap design (where a decision to configure the FPGA a second time can take place) and an application design (where there is no decision, but the current configuration in the history record must be marked as successful). The PicoBlaze controller determines whether it is executing a bootstrap configuration or an

application configuration by examining the history record. [Figure 2](#) shows the PicoBlaze controller block diagram.

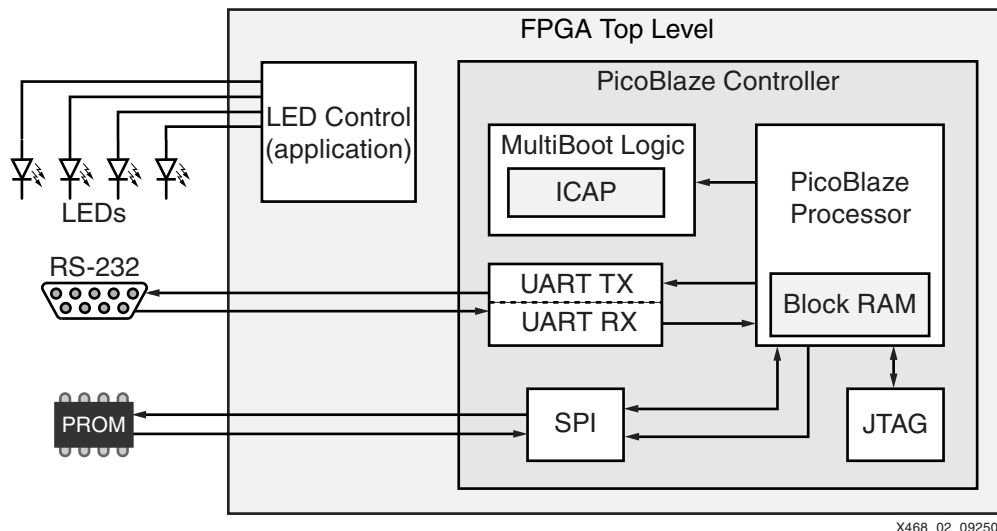


Figure 2: **PicoBlaze Controller Block Diagram**

The PicoBlaze controller contains the PicoBlaze processor and several peripherals including:

- A MultiBoot state machine including the ICAP controller
- Configuration and status registers
- An SPI interface (connected to either the external or ISF memory)
- A UART interface for external control (optional)
- A JTAG interface for external control (optional)
- A register-based interface for external control (optional)

Registers are used to access and control these peripherals. The PicoBlaze controller also accesses the peripheral registers and its own scratchpad memory on behalf of an external entity (in this case, a script used by the reference design). This allows the external entity to read and write the flash memory, query the status of the PicoBlaze controller, and initiate reconfigurations.

The bootstrap and application bitstreams are stored in flash memory. The reference design uses either the external 16 Mb SPI flash memory at location IC21 on the Spartan-3AN FPGA Starter Kit board or the internal ISF memory of the Spartan-3AN FPGA. The larger external SPI memory is divided into four equal parts, and the smaller internal ISF memory is divided into two equal parts. The bootstrap loader starts at address $0x000000$. Note that the Atmel PROM at location IC9 can be used instead of the ISF.

[Figure 3](#) shows the memory map used by the external M25P16 PROM at IC21. [Figure 4](#) shows the memory map used by either the internal ISF memory of the Spartan-3AN FPGA or the external Atmel AT45DB161D PROM at IC9. These are only example memory maps. It is important for the user to be aware of the structure of the flash memory that is used in their application. For example, the use of 264-byte pages with the default addressing scheme of the ISF (or Atmel PROM) result in addresses that do not exist. For this reason, the first application bitstream begins at $0x100000$ (a page boundary for the Atmel PROM) rather than $0x080000$ (256 bytes into a 264-byte page).

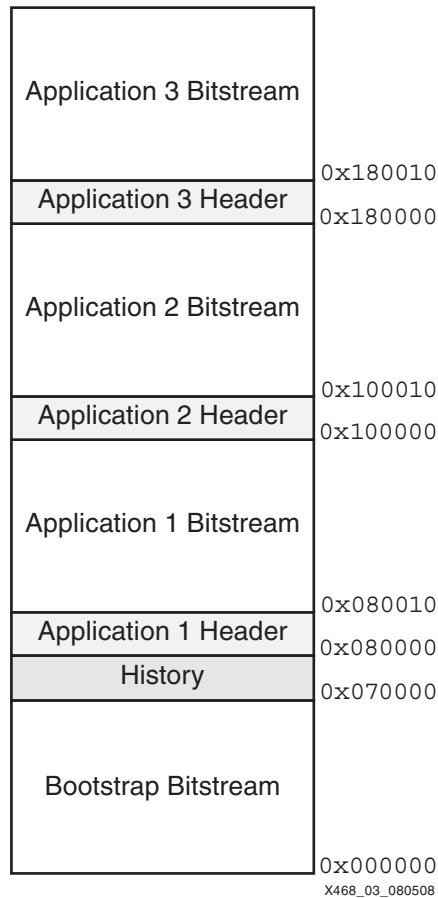


Figure 3: External Flash Memory Map (ST Microelectronics M25P16 PROM at IC21)

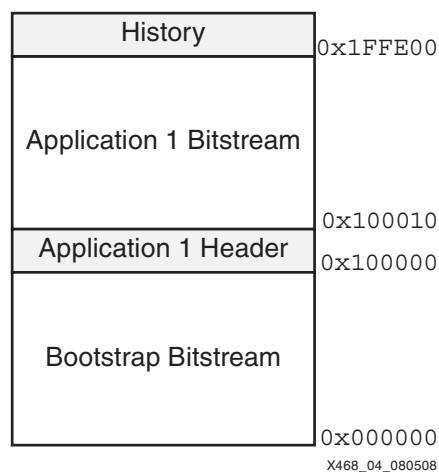


Figure 4: ISF Memory Map (or External Atmel AT45DB161D PROM at IC9)

The nature of the SPI flash memory forces several aspects of the implementation. Because the flash memory is only sector erasable, the bitstream headers are placed in the same sector as the bitstream loads. Therefore, each individual bitstream header must reside in a sector separate from the other bitstream headers. Also, after a sector is erased, all the bytes contain 0xFF. A write command changes individual bits from a logic 1 to a logic 0, but a bit cannot be

written back to logic 1 after it is a logic 0. The bitstream header stores four pieces of information:

- A 16-bit valid/invalid/empty code
- A 16-bit revision code
- A 16-bit cyclic redundancy check value (CRC-16)
- A 24-bit length field

The first header word indicates whether the associated bitstream is valid, invalid, or empty. A value of `0xFFFF` indicates that the flash memory contents for that bitstream are empty. A value of `0x00FF` indicates that the bitstream is valid. A value of `0x0000` indicates that the bitstream was written but is no longer valid.

The second word contains the bitstream revision. A larger number indicates a more recent bitstream revision. In this reference design, the bootstrap checks the revision to ensure that it loads the most recent valid bitstream. Other implementations might use a date or time stamp for this purpose.

The third word contains a CRC-16 value. To reduce configuration time, the reference design does not process this value. Typically, a bootstrap or application configuration would process this value to test bitstream integrity. An external entity might use this word to verify flash memory content.

The fourth word contains a 24-bit value indicating the number of bytes in the bitstream. The PicoBlaze processor uses this value to determine which portion of memory it should perform CRC calculations on.

A bitstream is considered valid if all of the following conditions are met:

- The 16-bit valid/invalid/empty code is `0x00FF` (valid).
- The bitstream images begins with some number of bytes containing `0xFF` followed by `0xAA` and `0x99`. The ICAP controller uses this pattern to check for the reset sequence and the synchronization sequence.
- The bitstream image has the sequence `0x31,0x61,0x<data>`, where the MSB of `<data>` is set within the first 64 bytes of the image. This is to check whether the BitGen option `-g reset_on_err:Yes` is set when the bitstream is generated. This option is enabled in the ISE® software's Project Navigator by selecting **Retry Configuration if CRC Error Occurs** under the General Options for the Generate Programming File process.

The bootstrap and application bitstreams use a page of flash memory as a history record to keep track of unsuccessful configuration attempts by the application bitstreams (Figure 3 and Figure 4). The history record occupies its own sector so it can be erased separately from the bitstreams and headers. If the reference design uses the external SPI memory, the history page is located at `0x070000`. If the reference design is located in the ISF memory, the history page is located at the last page of the flash memory. Aligning to an ISF page boundary provides the additional advantage of allowing independent protection or lockdown of the bootstrap image.

The history record is a list of bytes. The bootstrap uses the last byte in the list to remember which application bitstream it is attempting to load and how many attempts are made. The fact that a logic 0 cannot be written to a logic 1 in the SPI flash memory requires special encoding of the number of attempts. For each load attempt, the bootstrap logic writes the bitstream number to the most significant nibble of the history byte. The lower nibble is written with a `0xE` (`0b1110`) if this is the first attempt with the bitstream. The second and third attempts are `0xC` (`0b1100`) and `0x8` (`0b1000`), respectively. If the third attempt fails, this history byte is marked `0x00` and the bootstrap logic attempts to load the next bitstream, writing the next history byte in the same way. If the FPGA is successfully configured, the application marks the current history byte as `0x00`. The history record always contains a string of `0x00` values and a single

non-zero value at the end. This last value is 0xFF if there is no history, or 0xnm where *n* identifies the bitstream (1, 2, or 3), and *m* is the coded number of attempts. After the history sector contains 256 entries of 0x00, it is erased, filling the whole sector. The first value now contains 0xFF, which indicates no history.

A history record scenario is demonstrated here. The startup condition is an SPI memory where bitstream 1 contains the highest revision but is corrupted. Bitstream 2 is the next lowest revision and is valid and correct.

At first, the history contains all 0xFFs:

```
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF...
```

Bitstream 1 is attempted once and recorded in the history record:

```
0x1E 0xFF 0xFF 0xFF 0xFF 0xFF...
```

Bitstream 1 is attempted twice and recorded in the history record:

```
0x1C 0xFF 0xFF 0xFF 0xFF 0xFF...
```

Bitstream 1 is attempted a third time and recorded in the history record:

```
0x18 0xFF 0xFF 0xFF 0xFF 0xFF...
```

The bootstrap tries to load the next valid bitstream:

```
0x00 0x2E 0xFF 0xFF 0xFF 0xFF...
```

The load is successful. The application bitstream configures the FPGA, and the application clears the history byte.

```
0x00 0x00 0xFF 0xFF 0xFF 0xFF 0xFF...
```

External Interface and Control

The registers and scratchpad available to the PicoBlaze processor are also available to an external controller or program. The reference design uses a Tcl script running on a PC to provide this access. The Tcl script can communicate to the PicoBlaze processor through the JTAG or the UART port.

The PicoBlaze processor understands the commands shown in [Table 1](#). The SPI memory can be read and written or MultiBoot information can be gathered by using these five commands.

Table 1: External Commands

Command Function	Command Format	Return
Register Read	R <addr>	<data>
Register Write	W <addr> <data>	None
Scratchpad Read	S <addr>	<data>
Register Burst Write	B <addr> <length_h> <length_l> <data_0> ... <data_n>	None
CRC Check	C <bitstream number>	<crc_h> <crc_l>

The demonstration relies on a Tcl script that provides a simplified user interface and issues commands to the PicoBlaze processor based on the user input. The Tcl script handles the complexities of the flash memory structure (addressing, page/block/sector organization, read/write/erase commands, etc.) and the bitstream image (calculating CRC values and adding the header). The Tcl script takes simplified user input (such as E for Erase a bitstream) and issues the necessary PicoBlaze processor commands to perform the function (such as driving SS_b Low and then sending the appropriate bytes to the SPI flash memory to erase the various sectors, blocks and/or pages). An example of how the PicoBlaze processor commands can be combined to write the PROM is shown in the text after [Table 3](#).

The PicoBlaze processor stores some pertinent basic information in its scratchpad. This information can be read using the scratchpad read command. Table 2 shows the scratchpad address map.

Table 2: Scratchpad Address Map

Scratchpad Address	Name	Description
0x00	Spi_addr_hi (PicoBlaze processor)	Reserved for PicoBlaze processor use
0x01	Spi_addr_mid (PicoBlaze processor)	Reserved for PicoBlaze processor use
0x02	Spi_addr_low (PicoBlaze processor)	Reserved for PicoBlaze processor use
0x03	Spi_mfg_ID	SPI device manufacturing ID
0x04–0x07	Unused	Reserved
0x08–0x09	BS #1: Valid/Invalid word	0xFFFF: Bitstream #1 empty 0x00FF: Valid 0x0000: Invalid
0x0A–0x0B	BS #1: Revision	Revision number for bitstream #1
0x0C–0x0D	BS #1: Check value	Check value for bitstream #1
0x0E–0x0F	Unused	
0x10–0x11	BS #2: Valid/Invalid word	0xFFFF: Bitstream #2 empty 0x00FF: Valid 0x0000: Invalid
0x12–0x13	BS #2: Revision	Revision number for bitstream #2
0x14–0x15	BS #2: Check value	Check value for bitstream #2
0x16–0x17	Unused	
0x18–0x19	BS #3: Valid/Invalid word	0xFFFF: Bitstream #3 empty 0x00FF: Valid 0x0000: Invalid
0x1A–0x1B	BS #3: Revision	Revision number for bitstream #3
0x1C–0x1D	BS #3: Check value	Check value for bitstream #3
0x1E–0x1F	Unused	
0x20–0x2F	Unused	
0x30, 0x31, 0x32	Sorted_list	Ordered list of bitstreams based on revisions
0x33	Unused	
0x34	Boot_decision	PicoBlaze processor's decision on what to boot based on revision and validity
0x35	New_history_byte	History byte to be written when booting new bitstream
0x36	History_scratch	Current history byte
0x37	History_ptr_scratch	Pointer to current history byte
0x38	Prompt	% for bootstrap bitstreams > for application bitstreams

The PicoBlaze processor uses register accesses to control its peripherals. It also performs reads and writes to these peripheral registers on behalf of an external control. This mechanism is used primarily to read and write the SPI memory. However, it can be used to do anything that the PicoBlaze processor does, including forcing the boot of an alternate bitstream through the ICAP. Table 3 shows the register address map.

Table 3: Register Address Map

Register Address	Register Access	Name	Description
0x00	Read-Only	FPGA revision LSB	Lowest order byte of FPGA revision
0x01	Read-Only	FPGA revision MSB	Highest order byte of FPGA revision
0x02	Read/Write	Register Control Interface Data	Read and write for register-based control interface
0x03	Read-Only	Register control FIFO status	{tx_pres, 2'b0, tx_full, rx_full, 2'b0, rx_pres}
0x10	Read/Write	JTAG control interface data	Read and write for JTAG-based control interface
0x11	Read-Only	JTAG control FIFO status	{tx_pres, 2'b0, tx_full, rx_full, 2'b0, rx_pres}
0x20	Read/Write	RS-232 control interface Data	Read and write for RS-232-based control interface
0x21	Read-Only	RS-232 control FIFO status	{tx_pres, 2'b0, tx_full, rx_full, 2'b0, rx_pres}
0x30	Read/Write	SPI chip select	{7'h0, SPI_ss_b}
0x31	Read/Write	SPI write/read data	Revision number
0x40	Read/Write	MultiBoot address bits 7:0	Lowest order byte of PROM address for the configuration image
0x41	Read/Write	MultiBoot address bits 15:8	Second byte of PROM address for the configuration image
0x42	Read/Write	MultiBoot address bits 23:16	Third byte of PROM address for the configuration image
0x43	Read/Write	MultiBoot address bits 31:24	Highest order byte of PROM address for the configuration image
0x44	Read/Write	MultiBoot mode	{5'h0, mb_mode}
0x45	Read/Write	MultiBoot VSel	{5'h0, vsel}
0x46	Read/Write	MultiBoot iuse	{7'h0, iuse}
0x47	Write-Only	MultiBoot strobe	Write only (forces ICAP state machine to reconfigure the FPGA)
0x50	Read-Only	Firmware configuration	{4'h0, check_crc, internal/external_b, pause, bootstrap}
0x60	Write-Only	CRC data	Input for the 8-bit CRC-16 shift register
0x61	Read-Only	CRC 7:0	Output of the 8-bit CRC-16 shift register
0x62	Read-Only	CRC 15:8	Output of the 8-bit CRC-16 shift register
0x70	Read-Only	Number of bitstreams	Number of alternate application bitstreams

Table 3: Register Address Map (Cont'd)

Register Address	Register Access	Name	Description
0x71	Read-Only	Bitstream offset 1	Upper byte of 24-bit SPI location for bitstream #1
0x72	Read-Only	Bitstream offset 2	Upper byte of 24-bit SPI location for bitstream #1
0x73	Read-Only	Bitstream offset 3	Upper byte of 24-bit SPI location for bitstream #1

The example shown here reads from the SPI memory starting at 0x080000 using register commands and the control interface:

```

W 30 00 # Drive SPI_ss_b Low
W 31 03 # Shift out SPI_address_byte 00
W 31 08 # Shift out MSB of SPI address byte
W 31 00 # Shift out middle SPI address byte
W 31 00 # Shift out LSB of SPI address byte
W 31 00 # Shift out data 00/shift in miso
R 31    # Read data shifted in from miso
W 31 00 # Shift out data 00/shift in miso
R 31    # Read data shifted in from miso
.
.
.
W 30 FF # Drive SPI_ss_b High

```

Normally, the bootstrap program reads the information it needs from the SPI flash memory and immediately reboots to the application load. The reference design senses the position of SW3 on the Spartan-3AN FPGA Starter Kit board to allow the bootstrap to pause for user interaction or to automatically configure the application load.

The Extended Spartan-3A family configuration logic contains a counter that keeps track of failed configurations. After the configuration logic has counted three configuration attempt failures, the fourth failure halts the FPGA and asserts INIT_B. The counter can only be cleared by asserting the PROG_B pin. This behavior is undesirable for fail-safe MultiBoot operation where the PicoBlaze processor wants to pick an alternate or golden image after three failed configuration attempts. The PicoBlaze controller has an output port, PROG_B, that can be connected to the FPGA PROG_B pin. The PicoBlaze controller asserts this signal to a logic 0 any time an application bitstream is marked invalid. This can occur after three failed configuration attempts from that bitstream or from any mismatch between the stored CRC value in the bitstream header and the value calculated across the bitstream itself. The user logic can assert PROG_B, if needed, under these conditions.

HDL Parameters and Configuration

The PicoBlaze processor firmware provided with the reference design uses a discovery algorithm to determine what external flash memory is used. The discovery algorithm uses a combination of register values set in the HDL parameters and information read from the SPI devices. The algorithm allows the reference design to be configured to use the internal ISF memory of the Spartan-3AN FPGAs or the external flash memory provided on the starter kit board.

For example:

```
parameter SPI_DEVICE = "M25P16";
```

Allowable values are **M25P16**, **AT45DB161D**, **3S50AN**, **3S200AN**, **3S400AN**, **3S700AN**, and **3S1400AN**. Other SPI devices and memory maps can be accommodated by editing the `picoController.v` file.

The `SPI_DEVICE` value also controls the number of bitstreams and the SPI address locations for the bitstreams. These values are hard-coded in the HDL and are shown in [Table 4](#).

Table 4: SPI_DEVICE Parameter Values

SPI_DEVICE	History Location	Number of Application Bitstreams	Bitstream #1 Location	Bitstream #2 Location	Bitstream #3 Location
M25P16	0x070000	3	0x080000	0x100000	0x180000
AT45DB161D	0x3FFC00	3	0x100000	0x200000	0x300000
XC3S50AN	0x03FE00	1	0x020000	N/A ⁽¹⁾	N/A
XC3S200AN	0x0FFE00	1	0x080000	N/A	N/A
XC3S400AN	0x0FFE00	1	0x080000	N/A	N/A
XC3S700AN	0x1FFE00	1	0x100000	N/A	N/A
XC3S1400AN	0x3FFC00	1	0x200000	N/A	N/A

Notes:

1. N/A = not available

The HDL also provides a mechanism for ignoring these hard-coded values and setting them to unique values. This requires setting the `USE_UNIQUE_CFG` parameter and editing the values associated with the `USE_UNIQUE_CFG` section of the HDL within the `picoController.v` and `picoController.vhd` HDL files.

Use of the control interfaces is also controlled by HDL parameters. These parameters allow unused control interfaces to be left out of in the synthesized design, making it smaller. Also, the boundary-scan logic can be used for other purposes when the JTAG interface for PicoBlaze processor communication is not used.

```
parameter USE_REG = "TRUE";
parameter USE_JTAG = "FALSE";
parameter USE_RS232 = "TRUE";
```

Setting these parameter values to `TRUE` instructs the synthesis tool to include the referenced interface. The value `FALSE` instructs the synthesis tool to not include the referenced interface, producing a smaller design.

The PicoBlaze processor can optionally check the entire bitstream image stored in flash memory against the CRC-16 value contained in the header. This function is controlled using the `CHECK_CRC` HDL parameter. A `TRUE` value enables the CRC check. This check protects against errors in the bitstream image that might occur but would not cause `INIT_B` to go Low at the end of a bitstream. The downside of including this check is that it increases the configuration time because the entire application image is read twice; first for the PicoBlaze processor to check against the CRC, and then a second time by the FPGA configuration logic while configuring the FPGA.

The `BOOTSTRAP_FW` parameter determines which version of the firmware is run by the PicoBlaze processor. If the value is set to `TRUE`, the PicoBlaze processor determines that it is running from the initial bootstrap load. It examines the application loads and decides whether or not to jump to the application loads and which application load to jump to. If the value is `FALSE`,

the PicoBlaze processor determines that it is running from an application load. In this case, the PicoBlaze processor clears the most recent, non-zero history byte.

ICAP Module

The ICAP module is contained in the file `source/picoController/picoboot/picoboot.v`. This module instantiates the ICAP primitive and communicates the MultiBoot command to the ICAP primitive. The picoboot module inputs are described in [Table 5](#). There are no outputs.

Table 5: Picoboot Module (`picoboot.v`) Inputs

Port Name	Direction	Description
clk	Input	A synchronous clk for sampling the inputs. This clock also drives the ICAP configuration clock and must meet the AC specifications for the device being used.
reboot	Input	A pulse on the reboot command initiates a state machine that sends the necessary bits to the ICAP primitive to initiate the MultiBoot command.
internal_addr[31:0]	Input	The 32-bit pointer to the SPI flash address that contains the first byte of the MultiBoot bitstream.
internal_use	Input	When this signal is Low, the values in the internal configuration mode register can override the values selected by the I/O pins for MODE and VS bits. Configuration modes different than those specified by the external pins are not supported by Spartan-3A devices. This must be a logic 0.
internal_mode	Input	The 3-bit mode value that is loaded into the configuration mode register to determine the configuration method. Because internal_use must equal 0, this value does not matter.
internal_vsel	Input	The 3-bit vsel value that is loaded into the configuration mode register. Because internal_use must equal 0, this value does not matter.

SPI Module

The SPI module is contained in the file `source/picoController/spi/spi.v`. This module provides logic for a byte-wide interface to communicate with the serial SPI interface. The protocol for building larger SPI commands from these bytes is not handled by the SPI module and must be provided by external logic. This logic is implemented by the PicoBlaze processor for this reference design. The SPI module is controlled by two byte-wide registers described in [Table 6](#).

Table 6: SPI Status and Data Registers

Register	Bit	Name	Description
Status	0 (LSB)	SS_B	When asserted Low (logic 0), the SPI Flash SS_B signal is driven Low.
	1	BUSY	This signal is active High when the SPI module is busy shifting data to and from the parallel data register.
	7:2	–	Not used.
Data	7:0	–	When this register is written, the value written into the register is shifted out MSB-first from the SPI master-out-slave-in (mosi) pin to the SPI flash device. At the same time, data from the SPI master-in-slave-out (miso) pin is shifted into the register. The data can then be read from this register to read data from the SPI device.

The ports of the SPI module are defined by [Table 7](#).

Table 7: SPI Module (spi.v) Ports

Port name	Direction	Description
CLK	Input	Drives the synchronous logic in the module. The spi_clk is half the frequency of this clock.
RESET	Input	This signal synchronously resets the logic in the module.
PICO_ADDR[2:0]	Input	Selects which of the two registers is read or written. Bits 2 and 1 are unused.
PICO_DATAIN[7:0]	Input	Contains the write data for the selected register.
PICO_DATAOUT[7:0]	Output	Contains the read data for the selected register.
PICO_WE	Input	Synchronous write strobe. Causes PICO_DATAOUT[7:0] to be clocked into the selected register. If the PICO_ADDR[0] is also a 1, it causes one byte to be transferred to and from the SPI flash.
PICO_RD	Input	Read strobe. Not used.
miso	Input	Master-in-slave-out data signal for the SPI flash.
mosi	Output	Master-out-slave-in data signal for the SPI flash.
sck	Output	Clock for the SPI flash device.
ss_b	Output	Chip select signal for the SPI flash (active Low).

Demonstration Setup

Before running the demonstration, the software listed in Table 8 must be installed on the demonstration computer. The starter kit, computer hardware requirements, and cables are listed in Table 9. Table 10 lists the required design files.

Hardware, Software, and Design Files

Table 8: Required Software

Description	Comments
ISE WebPACK™ Software ⁽¹⁾	Version 9.2i SP1 or later. Software is included with the Spartan-3AN FPGA Starter Kit.
ActiveTcl Scripting Language ⁽²⁾ version 8.4.x.x	The scripts used by this application are created in ActiveTcl standard distribution, version 8. Note: The ISE software Tcl interface does not work with this reference design. The Tcl script uses a CRC-16 module provided by Active Tcl that is not supported by the ISE software Tcl interface.

Notes:

1. The latest version of free ISE WebPACK software is available at: http://www.xilinx.com/ise/logic_design_prod/webpack.htm
2. The free standard distribution of ActiveTcl is available at: <http://www.activestate.com/Products/activetcl/>

Table 9: Required Hardware

Description	Comments
Xilinx Spartan-3AN FPGA Starter Kit (part number: HW-SPAR3AN-SK-UNI-G ⁽¹⁾)	Contains an XC3S700AN Spartan-3AN FPGA with ISF memory and external SPI memory devices.
USB cable	Included with the Spartan-3AN FPGA Starter Kit. Used to download the initial bitstream.

Table 9: Required Hardware (Cont'd)

Description	Comments
RS-232 9-pin straight-through cable	Not included. Used for communication between the starter kit board and computer.
Personal computer running Microsoft Windows® XP Professional operating system	Requires a USB and RS-232 port.

Notes:

- The Spartan-3AN FPGA Starter Kit is available for online purchase at: <http://www.xilinx.com/products/devkits/HW-SPAR3AN-SK-UNI-G.htm>

Table 10: Required Design Files

Description	Comments
xapp468.zip	Contains design files and other files required by the reference design. The design files for the reference design can be downloaded from: https://secure.xilinx.com/webreg/clickthrough.do?cid=113104
KCPSM3.zip	Contains the PicoBlaze processor files and other supporting files. The free PicoBlaze processor IP core is available at: http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm

Configuring the Design Files

On the computer used for this demonstration:

- Create a temporary directory.
- Extract xapp468.zip to this directory.
- Extract the PicoBlaze processor files in listed in Table 11 to this directory:

Your_Temp_Directory\Multiboot_Spartan3AN_Jtag\source\picoController\picoblze

Table 11: Relocation of PicoBlaze Processor Verilog Files

Source File	Filename
KCPSM3.zip	kcpsm3.v
	kcuart_rx.v
	kcuart_tx.v
	uart_rx.v
	uart_tx.v
	Kcpsm3.exe
	JTAG_Loader_ROM_form.v

- Extract the PicoBlaze processor files listed in Table 12 to this directory:

Your_Temp_Directory\Multiboot_Spartan3AN_Jtag\VHDL\picoController\picoblze

Table 12: Relocation of PicoBlaze Processor VHDL Files

Source File	Filename
KCPSM3.zip	kcpsm3.vhd
	kcuart_rx.vhd
	kcuart_tx.vhd
	uart_rx.vhd
	uart_tx.vhd
	kcpsm3.exe

Configuring the Hardware

Position the starter kit jumpers:

1. Set power switch SW5 to the OFF position.
2. Configure the jumpers on J26 for SPI mode (Figure 5).

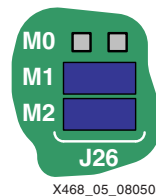


Figure 5: J26 Configured for SPI Mode

3. Set switch SW3 to the logic 1 position. This enables the PAUSE feature on the MultiBoot controller so that the configuration of application loads is not immediate.
4. Configure the boot select jumpers on J1 for the Atmel SPI PROM (Figure 6).

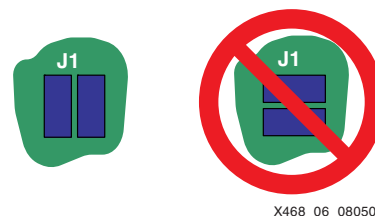


Figure 6: J1 Configured for the Atmel SPI PROM

5. Configure jumpers across J23 and J25 to enable direct SPI programming (Figure 7).

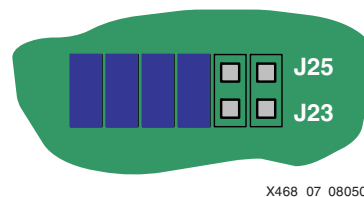


Figure 7: J23 and J25 Configured for Direct SPI Programming

Connect the starter kit board to the computer:

Note: Additional information about connecting the starter kit RS-232 serial port to a computer is available in the *Spartan-3A/3AN FPGA Starter Kit Board User Guide* [Ref 1].

1. Connect the RS-232 cable between J36 on the starter kit board and the serial 9-pin connector on the computer.
2. Connect the USB cable between J3 on the starter kit board and a USB connector on the computer.
3. Connect the power converter cord to J35 on the starter kit board.
4. Turn on the starter kit board by setting switch SW5 to the ON position.

Running the Demonstration

The bootstrap controller must first be loaded into the FPGA. The bootstrap controller is then used to program the bootstrap bitstream image into the SPI PROM.

Upload the Bootstrap Controller to SPI Memory

1. Use the iMPACT software to load the iMPACT project file `bootstrap.ipf` from the bootstrap directory.
2. Click on the **Direct SPI Programming** tab.
3. Right-click on the image of the **SPI PROM** (Figure 8) and select the **Program** command from the menu. Press the PROG_B button on the starter kit board, and while holding it down, click **OK**. Holding the PROG_B button prevents any bitstream that is already in the PROM from actively driving the SPI bus while the iMPACT software is trying to drive the bus.

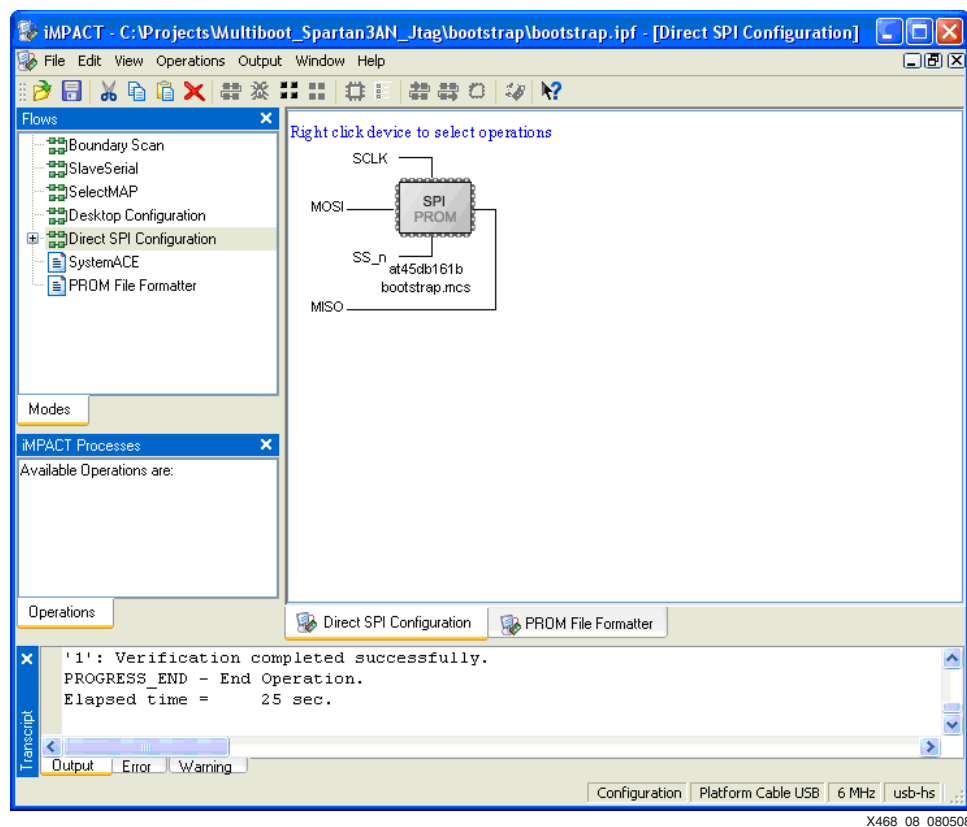


Figure 8: iMPACT Software for Direct SPI Configuration of Bootstrap Bitstream

The SPI memory now contains the bootstrap bitstream, and the FPGA can be configured from the SPI memory. The first application image is programmed next.

Open the Command Window

1. In the taskbar, click **Start**.
2. From the menu select **Run** to open the Run dialog box.
3. In the text field, type `cmd`.
4. Click **OK** to open the Command window.

The command line is used to enter the remaining commands.

Upload Application 1 to SPI Memory

1. Change the directory to the reference design's `Scripts` directory.
2. Enter `tclsh mbScript.tcl` to run the Tcl script. The command window displays the information shown in [Figure 9](#).

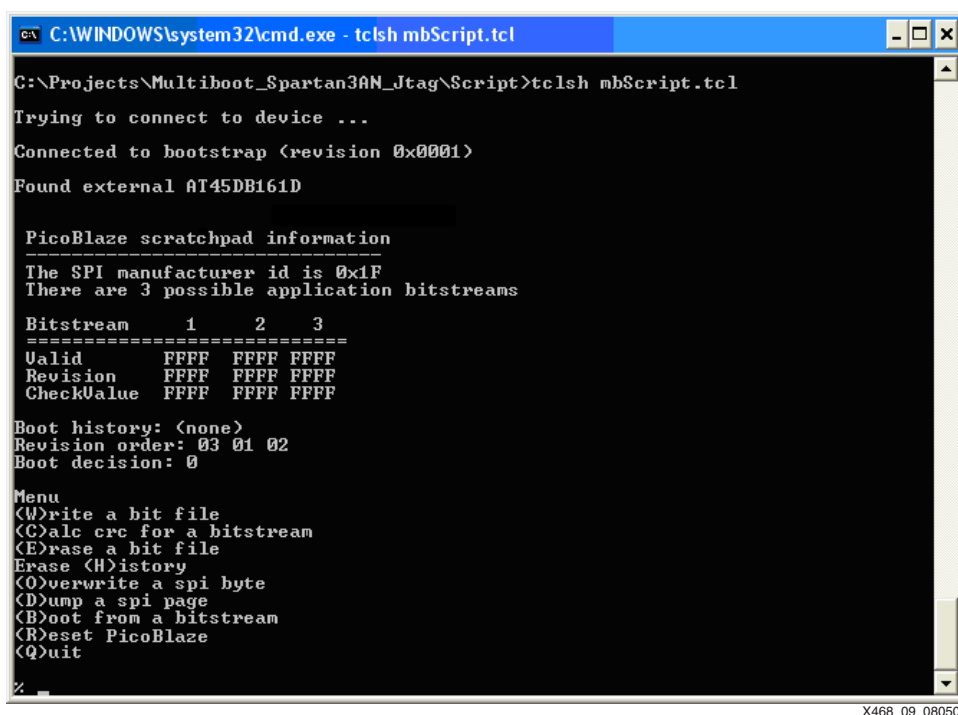
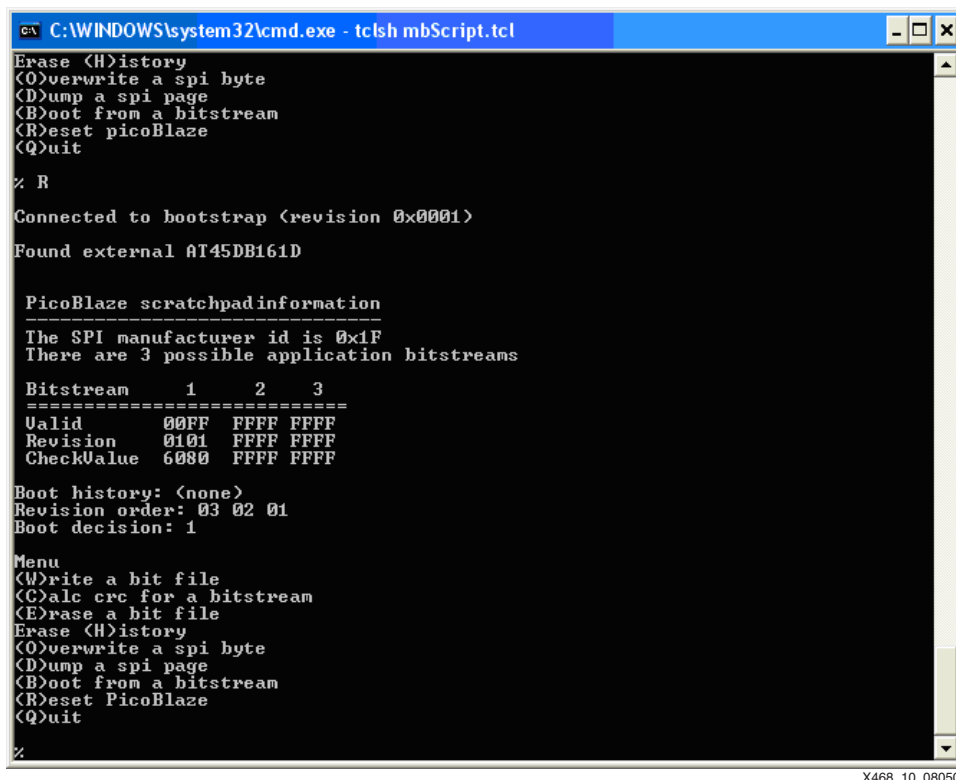


Figure 9: Initial Display after Running `mbScript.tcl`

3. Enter `w`.
4. When asked for the bitstream location, enter `1`. Enter `application1.hex` as the file name.
5. When asked for a revision number, enter `0101`. The file begins to upload, and the display shows the number of kilobytes sent. Wait until the file is completely sent.
6. Enter `R` to reset the PicoBlaze processor and refresh the display. The command window displays the information shown in [Figure 10](#). The bitstream table has changed, and the boot decision now specifies bitstream location `1`.
7. Enter `q` to exit the Tcl script.

The SPI memory now contains the bootstrap bitstream and the bitstream for Application 1.



```

C:\WINDOWS\system32\cmd.exe - tclsh mbScript.tcl
Erase (H)istory
(O)verwrite a spi byte
(D)ump a spi page
(B)oot from a bitstream
(R)eset picoBlaze
(Q)uit

% R
Connected to bootstrap <revision 0x0001>
Found external AT45DB161D

PicoBlaze scratchpad information
-----
The SPI manufacturer id is 0x1F
There are 3 possible application bitstreams

Bitstream      1      2      3
=====
Valid          00FF  FFFF  FFFF
Revision       0101  FFFF  FFFF
CheckValue    6080  FFFF  FFFF

Boot history: <none>
Revision order: 03 02 01
Boot decision: 1

Menu
(W)rite a bit file
(C)alc crc for a bitstream
(E)rase a bit file
Erase (H)istory
(O)verwrite a spi byte
(D)ump a spi page
(B)oot from a bitstream
(R)eset PicoBlaze
(Q)uit

%

```

Figure 10: mbScript.tcl Display after Writing Application Bitstream

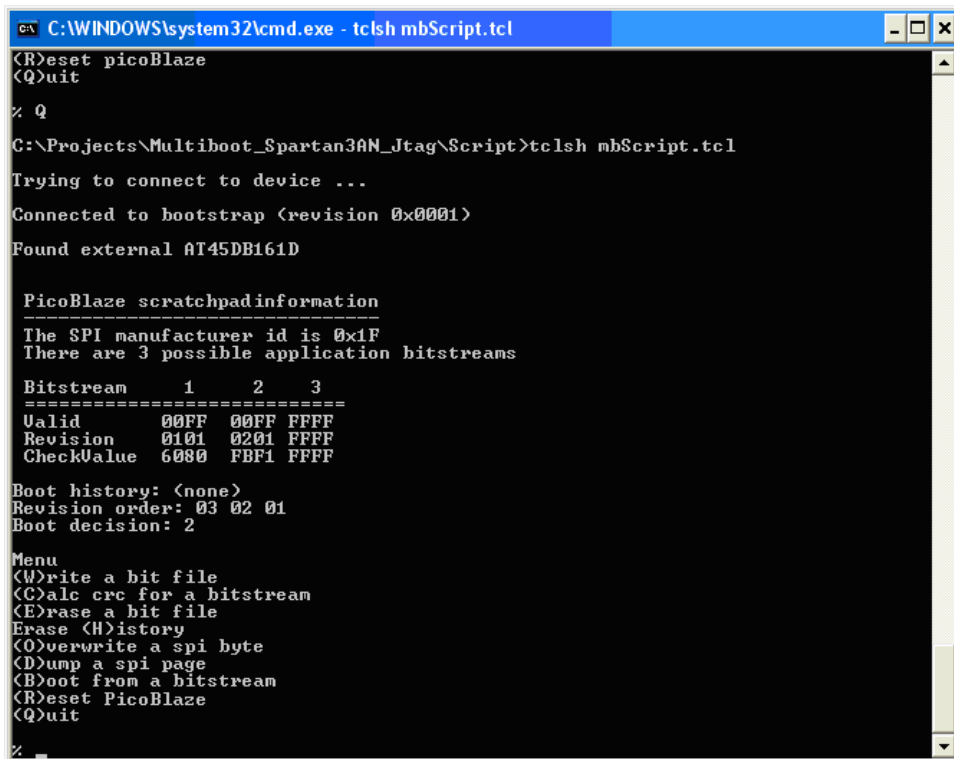
Reconfigure the FPGA Using the Bootstrap

These steps demonstrate the use of the bootstrap bitstream to initially configure the FPGA on the starter kit board and then reconfigure the FPGA with Application 1.

1. Push starter kit BTN1 (PROG_B button) to start configuring the FPGA using the bootstrap program from the SPI memory.
2. LED LD0 flashes, which indicates that the bootstrap program is operating. After a 10-second delay (due to the PAUSE bit on SW3 being a logic 1), the bootstrap loads Application 1.
3. When LEDs LD0 through LD7 begin flashing in sequence from LD0 to LD7, Application 1 is operating.

Upload Application 2 to SPI Memory

1. Enter `tclsh mbScript.tcl` to run the Tcl script.
2. Enter `w`.
3. When asked for the bitstream location, enter 2. Enter `application2.hex` as the file name.
4. When asked for a revision number, enter 0201. The file begins to upload, and the display shows the number of kilobytes sent. Wait until the file is completely sent.
5. Enter `q` to exit the Tcl script.
6. Push starter kit BTN1 (PROG_B button).
7. Enter `tclsh mbScript.tcl` to display the information shown in [Figure 11](#).
The bitstream table has changed and the boot decision now specifies bitstream location 2 because bitstream 2 has a higher revision number than bitstream 1.
8. Enter `q` to exit the Tcl script.



```

C:\WINDOWS\system32\cmd.exe - tclsh mbScript.tcl
(R)eset picoBlaze
(Q)uit
% Q
C:\Projects\Multiboot_Spartan3AN_Jtag\Script>tclsh mbScript.tcl
Trying to connect to device ...
Connected to bootstrap <revision 0x0001>
Found external AT45DB161D

PicoBlaze scratchpad information
-----
The SPI manufacturer id is 0x1F
There are 3 possible application bitstreams

Bitstream      1      2      3
=====
Valid          00FF  00FF  FFFF
Revision       0101  0201  FFFF
CheckValue    6000  FBF1  FFFF

Boot history: <none>
Revision order: 03 02 01
Boot decision: 2

Menu
(W)rite a bit file
(C)alc crc for a bitstream
(E)rase a bit file
Erase (H)istory
(O)verwrite a spi byte
(D)ump a spi page
(B)oot from a bitstream
(R)eset PicoBlaze
(Q)uit
%

```

X468_11_080508

Figure 11: mbScript.tcl Display after Writing Application 2 Bitstream

The SPI memory now contains the bootstrap bitstream and the bitstreams for both Application 1 and Application 2.

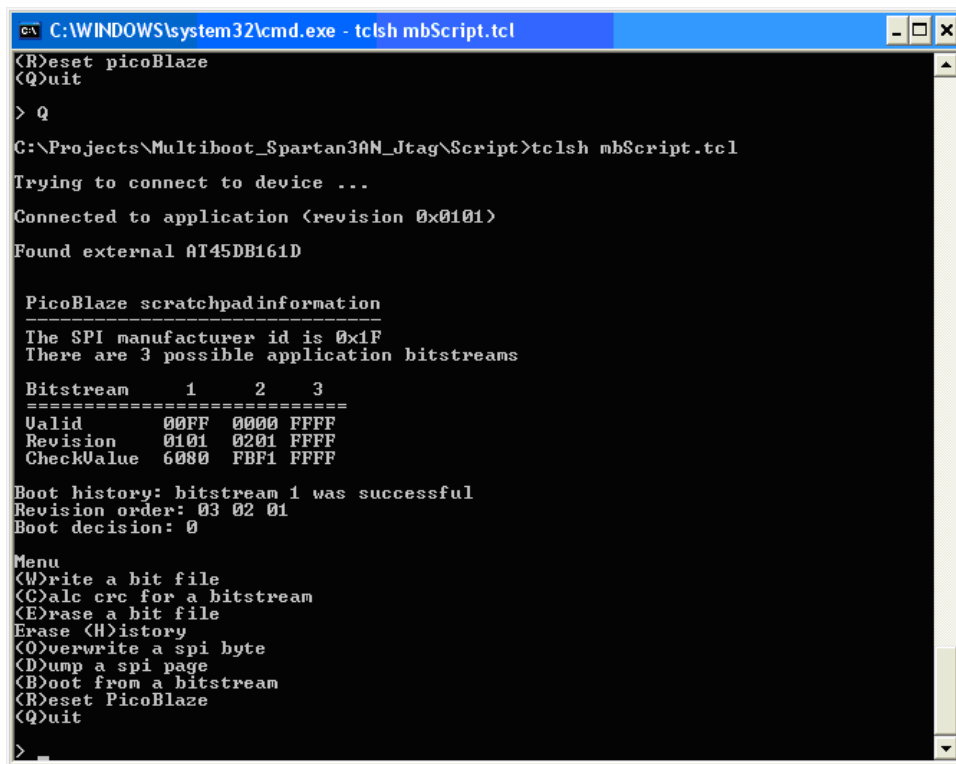
Corrupt the Application 2 Bitstream

1. Push starter kit BTN1 (PROG_B button) to initiate the bootstrap program.
2. Wait until Application 2 is operating as indicated by LEDs LD0 through LD7 flashing in sequence from LD7 to LD0.
3. Enter `tclsh mbScript.tcl` to run the Tcl script.
4. Enter `o` to overwrite a byte in the SPI memory.
5. Enter `201000` as the address to overwrite (simulates a corrupted value).
6. Enter `00` as the data value to write.
7. Enter `q` to exit the Tcl script.

The Application 2 bitstream in SPI memory is now corrupted. The bootstrap logic detects this on the next boot attempt and falls back to loading Application 1.

Fallback Demonstration

1. Push starter kit BTN1 (PROG_B button) to initiate the bootstrap program.
2. Bootstrap operation is indicated by LD0 flashing.
3. After 10 seconds, the bootstrap finds that Application 2 is corrupted and loads Application 1 as indicated when LEDs LD0 through LD7 flash in sequence from LD0 to LD7.
4. Enter `tclsh mbScript.tcl` to display the information shown in [Figure 12](#).
5. Application 2 (indicated as Bitstream 2 in the table on the display shown in [Figure 12](#)) is now marked as invalid (`0x0000`), and the Boot History indicates that bitstream 1 was successfully loaded.



```

C:\WINDOWS\system32\cmd.exe - tclsh mbScript.tcl
(R)eset picoBlaze
(Q)uit
> Q
C:\Projects\Multiboot_Spartan3AN_Jtag\Script>tclsh mbScript.tcl
Trying to connect to device ...
Connected to application (revision 0x0101)
Found external AT45DB161D

PicoBlaze scratchpad information
-----
The SPI manufacturer id is 0x1F
There are 3 possible application bitstreams

Bitstream      1      2      3
=====
Valid          00FF  0000  FFFF
Revision       0101  0201  FFFF
CheckValue    6080  FBF1  FFFF

Boot history: bitstream 1 was successful
Revision order: 03 02 01
Boot decision: 0

Menu
(W)rite a bit file
(C)alc crc for a bitstream
(E)rase a bit file
Erase (H)istory
(O)verwrite a spi byte
(D)ump a spi page
(B)oot from a bitstream
(R)eset PicoBlaze
(Q)uit
>

```

X468_12_080508

Figure 12: mbScript.tcl Display after Reverting to Application 1

Device Utilization

Table 13 shows approximate device utilization in the XC3S700AN FPGA.

Table 13: Device Utilization

Resource	Number Used	Percent of Device Used
Flip-Flops	270	2.3%
4-input Look-Up Tables	610	5.2%
Block RAMs	1	5.0%
Slices	360	6.1%

Conclusion

The MultiBoot capability of the Extended Spartan-3A family provides a means for system architects to implement microprocessor designs inside the FPGA, while maintaining the fallback and fail-safe FPGA reconfiguration mechanisms needed by robust architectures. This implementation requires no additional components. The density and cost advantages provided by SOC designs can be realized in architectures requiring a fail-safe upgrade solution for FPGAs.

Reference Design Files

Links to the reference design files are listed in Table 10, page 13.

The reference design matrix is shown in Table 14.

Table 14: Reference Design Matrix

Parameter	Description
Developer Name	Xilinx
Target Devices (stepping level, ES, production, speed grades)	Extended Spartan-3A family
Source Code Provided	Yes
Source Code Format	Verilog and VHDL
Design Uses Code/IP from an Existing Reference Design/Application Note, Third Party, or CORE Generator™ software	Yes
Simulation	
Functional Simulation Performed	No
Timing Simulation Performed	No
Testbench Used for Functional Simulations Provided	No
Testbench Format	
Simulator Software Used/Version (e.g., ISE software, Mentor, Cadence, other)	ISE software
SPICE/IBIS Simulations	No
Implementation	
Synthesis Software Tools Used/Version	XST
Implementation Software Tools Used/Versions	ISE software, version 10.1, Service Pack 1
Static Timing Analysis Performed	Yes
Hardware Verification	
Hardware Verified	Yes
Hardware Used for Verification	Spartan-3AN FPGA Starter Kit Board

References

This section lists documents referenced in this application note:

1. [UG334](#), *Spartan-3A/3AN FPGA Starter Kit Board User Guide*.

Additional Resources

This section lists additional information useful to this application note:

1. [UG331](#), *Spartan-3 Generation FPGA User Guide*.
2. [UG332](#), *Spartan-3 Generation Configuration User Guide*.
3. [UG333](#), *Spartan-3AN FPGA In-System Flash User Guide*.
4. Spartan-3A/3AN FPGA Starter Kit Schematic
http://www.xilinx.com/support/documentation/boards_and_kits/s3astarter_schematic.pdf
5. PicoBlaze Processor User Lounge
<http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm>
6. Other Spartan-3A FPGA reference designs
http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
11/04/08	1.0	Initial Xilinx release.
07/07/09	1.1	Valid bitstream conditions on page 5 updated per ISE software, version 11. Revised “ Fallback Demonstration ,” page 18 , step 5 . Changed the invalid value from 0x00FF to 0x0000.

Notice of Disclaimer

Xilinx is disclosing this Application Note to you “AS-IS” with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.