



XAPP733 (v1.0) May 24, 2012

# Applying MultiBoot and the LogiCORE IP Soft Error Mitigation Controller

Author: Eric Crabill

## Summary

This document details the implementation of the MultiBoot feature and the LogiCORE™ IP Soft Error Mitigation (SEM) controller for Spartan®-6, Virtex®-6, and 7 series FPGAs. The six supporting reference designs embody key concepts.

## Introduction

The MultiBoot feature supports robust design management and field upgrade capabilities. Designers initiate a MultiBoot event by commanding the FPGA to fully reconfigure itself with an alternate bitstream. The required commands are usually issued via the internal configuration access port (ICAP).

Typically, the SEM controller has exclusive control of the ICAP to meet its functional and performance specifications. Designers who apply MultiBoot and the SEM controller must coordinate ICAP sharing. Designers might also need to organize multiple sets of SEM controller data in addition to multiple sets of FPGA configuration data.

This application note demonstrates how to apply the method documented in *Dual Use of ICAP with SEM Controller* [Ref 1] towards a reusable MultiBoot module compatible with the SEM controller for Spartan-6, Virtex-6, and 7 series FPGAs. Additionally, this application note illustrates how to organize multiple sets of SEM controller data. The supporting reference designs yield implementations for hardware evaluation on the SP605, ML605, and KC705 evaluation kits.

## MultiBoot Overview

The MultiBoot feature provides designers with an inexpensive method to perform dynamic full reconfiguration of an FPGA. Dynamic full reconfiguration has a wide range of applications, including robust design management and field upgrade capability.

Prior to the introduction of MultiBoot, implementation of dynamic, full reconfiguration incurred development costs for a proprietary solution involving additional hardware, software, and design tool components. With MultiBoot, these costs are eliminated. Only the per bit storage cost of FPGA configuration data remains.

Although the details of MultiBoot vary between Spartan-6, Virtex-6, and 7 series FPGAs, the fundamentals of a MultiBoot event are identical. The designer programs the FPGA with the next bitstream location, then commands the FPGA to load the next bitstream. As a result, the FPGA shuts down its programmable logic, loads the next bitstream, and then starts up its programmable logic. The MultiBoot event can be protected by optional error-recovery features to guard against data corruption during FPGA configuration.

### Programming the Next Bitstream Location

In addition to programmable logic, the FPGA contains a block of dedicated circuitry that is responsible for managing the programmable logic. This management function is called the configuration logic and its primary function is to load FPGA configuration data into configuration memory, thereby setting the behavior of the programmable logic.

In preparation for a MultiBoot event, the designer programs the FPGA with the next bitstream location with register write commands to specific registers in the configuration logic. The two recommended approaches that can be used individually or in combination are:

- Using the bitstream generator program (BitGen):
  - The designer specifies option switches with the next bitstream location.
  - BitGen translates the option switches into register write commands inserted in the bitstream. The information is placed into the associated configuration logic registers during the configuration process.
- Using the ICAP:
  - The designer determines the register write commands required to set the next bitstream location.
  - The designer includes functionality to apply the register write commands to the ICAP. The information is placed into the associated configuration logic registers when the register write commands are issued at runtime.

When used individually, the BitGen approach does not support runtime selection of the next bitstream location, nor does it eliminate the need for the ICAP. While the BitGen approach eliminates register write commands required to set the next bitstream location, the ICAP is needed to command the next bitstream load. For advanced applications, the ICAP might also be required to read MultiBoot status. For these reasons, the simplest solution is often to program MultiBoot operations entirely via the ICAP and to use BitGen only for setting optional error recovery features.

**Note:** The user should know which registers are written by BitGen and which ones are written by the ICAP to ensure accurate tracking of MultiBoot status. It is also important to know whether MultiBoot operations are used to implement a safe update application.

## Commanding the Next Bitstream Load

The designer commands the FPGA to load the next bitstream by issuing an IPROG command to the configuration logic. This step is done with a register write of the IPROG command to the command register in the configuration logic.

After an IPROG command is issued, the configuration logic shuts down the programmable logic within a few clock cycles. Therefore, this command terminates activity of the design that issued the IPROG command. The configuration logic then loads the next bitstream from the specified location and subsequently starts up the programmable logic.

## Support Detail

For register-level detail on programming the next bitstream location and commanding the next bitstream load, refer to the configuration user guides for the target FPGAs:

- *Spartan-6 FPGA Configuration User Guide* [Ref 2]
- *Virtex-6 FPGA Configuration User Guide* [Ref 3]
- *7 Series FPGAs Configuration User Guide* [Ref 4]

Successful application of MultiBoot requires information about the mode used to configure the FPGA device and information about the FPGA itself. The supporting reference designs described in this document use the SP605 [Ref 5], ML605 [Ref 6], and KC705 [Ref 7] evaluation kits as context for examples.

## Coordination of ICAP Sharing

Most implementations of MultiBoot use the ICAP. If nothing else in the design requires access to the ICAP, the designer can monopolize the ICAP—even though only a few dozen clock cycles of access are actually required for a MultiBoot event. If other functions require access to the ICAP, the designer must coordinate between all functions to share the ICAP.

The SEM controller generally has exclusive control of the ICAP to meet its functional and performance specifications. Many devices have two ICAPs, with a provision to switch between them. This reference design assumes the use of one ICAP so the general solution is the same for all FPGA families. To share one ICAP, the SEM controller can be idled, temporarily freeing the ICAP for other uses. This sharing method is documented in *Dual Use of ICAP with SEM Controller* [Ref 1] and is ideal for applying MultiBoot and the SEM controller.

A generic sharing sequence can be coordinated by control logic, such as a finite state machine. The following sequence removes manual intervention from the method documented in *Dual Use of ICAP with SEM Controller* [Ref 1] and adapts it to the specific case of MultiBoot:

1. A system-specific MultiBoot next bitstream location is determined.
2. A system-specific MultiBoot trigger occurs.
3. Control logic commands the SEM controller to idle using the SEM controller error injection port.
4. Control logic waits for the SEM controller to idle using the SEM controller status port.
5. Control logic multiplexes the ICAP away from the SEM controller for its own use.
6. Control logic issues the required commands to the ICAP for a MultiBoot event.
7. Wait forever is the final step that prevents further activity on the ICAP.

The IPROG command terminates activity of the design that issued the IPROG command, including the control logic and the SEM controller, within a few cycles. For this reason, the interruption of the SEM controller has no impact on its error mitigation performance. Further action by the control logic to restart the SEM controller is unnecessary.

## Data Organization

For detailed recommendations about the organization of FPGA configuration data for MultiBoot, see the target FPGA configuration user guides [Ref 2], [Ref 3], [Ref 4]. A successful MultiBoot event depends on the next bitstream being properly stored at a known address in a non-volatile device suitably interfaced to the FPGA for configuration. The required amount of storage for FPGA configuration depends on the number of bitstreams to be stored, the bitstream size, and a few second-order considerations described in the FPGA configuration user guides.

Designers might also need to consider the organization of SEM controller data. A SEM controller instance uses external data if one or both of these optional features are enabled:

- Error correction by replace
- Error classification

When these optional features are enabled, the SEM controller example design includes a simplified serial peripheral interface (SPI) master through which the SEM controller accesses data in an SPI flash. The SPI flash is dedicated to the SEM controller and independent from any nonvolatile devices used for FPGA configuration.

When the SPI master is present in the SEM controller example design, the SEM controller gains an additional 32-bit input port named `fetch_tbladdr`. The SEM controller uses the value applied to this port as a pointer to a table in the SPI flash. By default, `fetch_tbladdr` is assigned to zero in the SEM controller example design. The SPI flash programming file created by the SEM controller example design implementation script must be stored in the SPI flash starting at address zero.

The table in the SPI flash contains one or more pointers to blocks of data. These pointers are calculated assuming the blocks of data immediately follow the table and that the table is stored in the SPI flash starting at address zero. Refer to *LogiCORE IP Soft Error Mitigation Controller v3.2 Product Guide* [Ref 8] for additional information on the format of this table.

For MultiBoot, if more than one bitstream contains a SEM controller instance using external data, multiple sets of SEM controller data exist. Multiple sets of SEM controller data cannot all

be stored in the SPI flash starting at address zero. The solution is to concatenate the required sets of SEM controller data and note the starting address for each set. The pointers in the table for each set must be adjusted by the starting address for the set. The adjusted, concatenated information is stored in the SPI flash.

In the design for each applicable bitstream, `fetch_tbladdr` must be set so that the SEM controller instance can find its unique data in the SPI flash. With a minimal amount of pre-planning to develop a memory map of the SPI flash, `fetch_tbladdr` can be set with constant values at design time, or at runtime if necessary, provided it is set prior to the SEM controller entering the observation state.

**Note:** It is critical to maintain consistency between the FPGA configuration data and the SEM controller data. If a bitstream is updated for any reason, any corresponding SEM controller data must also be updated. The SEM controller's usage of stale or incorrect classification data can erode reliability, and usage of stale or incorrect replace data can result in functional failure.

## MultiBoot Application Example

The reference designs are examples of fallback MultiBoot with golden image load at power-up. Each bitstream contains an instance of the SEM controller using external data.

### Getting Started

There are six supporting reference designs in the ZIP file that accompanies this application note (see [Reference Designs](#)). After accepting the license agreement, the user can download and decompress the ZIP file. The `readme.txt` file contains the detailed directory structure and file listing. [Table 1](#) lists the supporting reference designs.

**Table 1: Supporting Reference Design List**

| Family         | Language | Hardware Target   | CGP, XCO, Directory Name |
|----------------|----------|---|--------------------------|
| Spartan-6 FPGA | Verilog  | SP605 (XC6SLX45T-FGG484-2) and 128 Mb SPI flash <sup>(1)</sup>  | spartan6_ver             |
|                | VHDL     |   | spartan6_vhd             |
| Virtex-6 FPGA  | Verilog  | ML605 (XC6VLX240T-FF1156-2) and 128 Mb SPI flash <sup>(1)</sup> | virtex6_ver              |
|                | VHDL     |   | virtex6_vhd              |
| 7 Series FPGAs | Verilog  | KC705 (XC7K325T-FFG900-2) and 256 Mb SPI flash <sup>(1)</sup>   | kintex7_ver              |
|                | VHDL     |   | kintex7_vhd              |

**Notes:**

1. The private SPI flash used by the SEM controller is not included in the evaluation kit. SPI flash is an optional, user-provided resource that enables evaluation of SEM controller data organization.

Each supporting reference design uses these two bitstreams that are generated from the SEM controller example design:

- Implementation A
- Implementation B

The SEM controller and the example design are not included. To generate them from the supplied CGP and XCO files the user follows these steps:

1. Select a supporting reference design to evaluate.
2. Use the CORE Generator™ tool to open the associated CGP and XCO files.
3. Generate the SEM controller (which also generates the SEM controller example design).
4. Exit the CORE Generator tool.

**Note:** The steps can be done interactively in a CORE Generator tool window or through the tool's batch mode:

```
coregen -p <project.cgp> -b <core.xco>
```

The CORE Generator tool overlays the generated output products on the selected supporting reference design directory. To ensure that files are deposited where expected, the user selects the directory where the associated CGP and XCO files exist.

The CGP file contents select the family, language, and hardware target. The XCO file contents set the SEM controller features:

- Enable Correction = True
- Correction Method = Repair
- Enable Classification = True
- Enable Injection = True
- Injection Shim = Pins
- Retrieval Shim = SPI flash

To use the supporting reference designs as provided with the listed hardware targets, the CGP and XCO file contents must not be changed. However, these designs are intended for adaptation, and in general, the only requirement is that the SEM controller has the error injection feature enabled. Control logic in the supporting reference designs commands the SEM controller to idle using the SEM controller error injection port.

## Design Overview

The supporting reference designs introduce a MultiBoot module into the SEM controller example design and also contain additional minor modifications, with the result shown in [Figure 1](#). The blocks drawn in gray only exist for certain configurations of the SEM controller example design. See the *LogiCORE IP Soft Error Mitigation Controller v3.2 Product Guide* [Ref 8] for illustrations of the unmodified SEM controller example design.

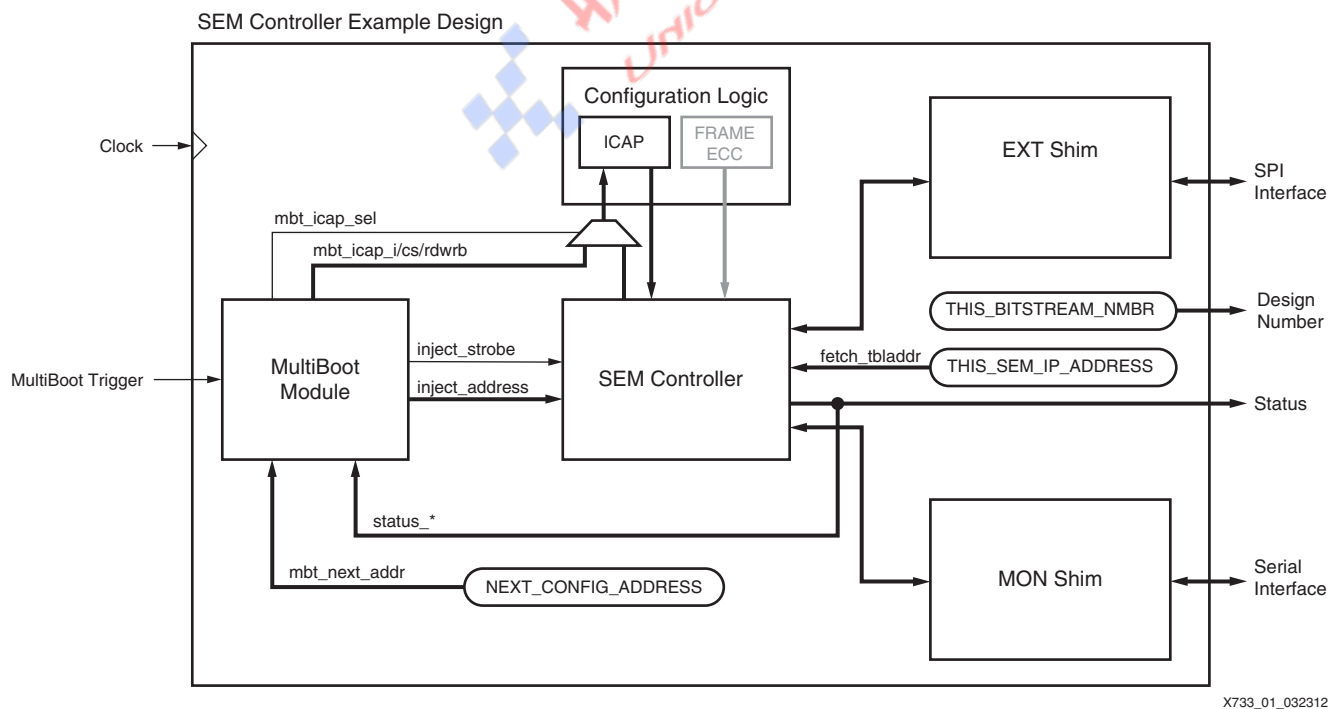


Figure 1: Modified SEM Controller Example Design

The `example_design` subdirectory in each supporting reference design's directory contains the original SEM controller example design source files. The `xapp_design` subdirectory

contains the MultiBoot module and modified source files. The modifications to the source files are:

- The parameter `THIS_BITSTREAM_NMBR` is added to the SEM controller example design and the value applied to a new output port `design_number[2:0]`. This change provides a way to discriminate between multiple bitstreams produced from the same SEM controller example design source files.
- The parameter `THIS_SEM_IP_ADDRESS` is added to the SEM controller example design and the value applied to the internal signal `fetch_tbladdr[31:0]`, which was previously driven to zero. This change provides a way to set arbitrary starting addresses for the SEM controller data.
- The parameter `NEXT_CONFIG_ADDRESS` is added to the SEM controller example design and the value is applied to a new internal signal `mbt_next_addr[31:0]`. This change provides a way to set arbitrary next bitstream addresses for the MultiBoot module.
- The MultiBoot module is instantiated in the SEM controller example design and connected in the following manner:
  - The SEM controller error injection port is removed from the top-level I/O pin connection and interfaced to the MultiBoot module. This change enables the MultiBoot module to issue commands to the SEM controller.
  - The five state signals from the SEM controller status port are interfaced to the MultiBoot module. This change enables the MultiBoot module to observe the state of the SEM controller, primarily to determine when the SEM controller is idle.
  - The signal driven with the next bitstream address is interfaced to the MultiBoot module.
  - A top-level I/O pin is added to provide an external MultiBoot trigger signal to the MultiBoot module.
  - The ICAP multiplexer select, `cs`, `rdwr`, and data signals generated by the MultiBoot module are combined with those from the SEM controller using a multiplexer. As documented in *Dual Use of ICAP with SEM Controller* [Ref 1], this multiplexer must be purely combinational, otherwise the SEM controller fails to operate properly.
- The constraint file is updated to reflect the changes in top-level I/O pins and assign pin locations suitable for the hardware target. The four I/O pins associated with the optional, user-provided SPI flash for private use of the SEM controller have placeholder location constraints. The user can update these placeholder constraints, if desired.
- A modified copy of the SPI Master is supplied and enables 4-byte addressing mode. This modification is necessary when two sets of SEM controller data require the use of a 256 Mb SPI flash.

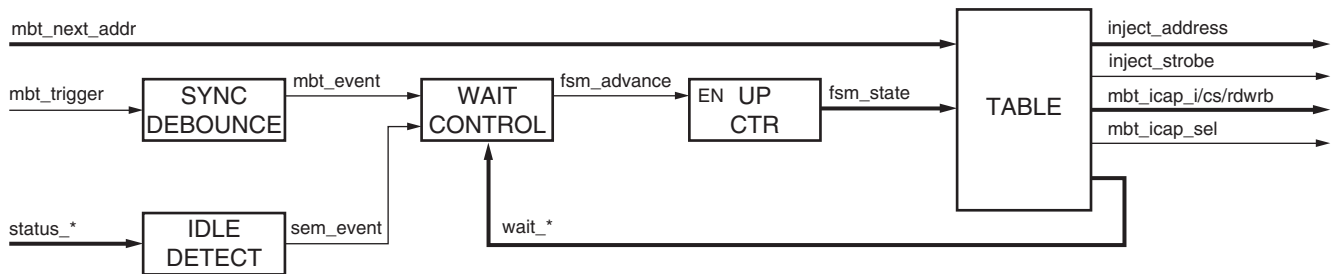
## Design Detail

This section provides details on the MultiBoot module and the parameter value selection. The user can review the MultiBoot module source files and the modified source files, if desired.

### MultiBoot Module

The MultiBoot module contains control logic to coordinate the generic sharing sequence. The sequencing is identical in all implementations, although the register write commands issued to the configuration logic via the ICAP are unique to each FPGA family. The MultiBoot module is constructed from a counter-driven table (see [Figure 2](#)) so that the general solution is the same for all FPGA families, and only the table contents and datapath widths differ.





X733\_02\_032312

Figure 2: MultiBoot Module Block Diagram

The next bitstream location is provided through the `mbt_next_addr` port and the value is used where needed in the table. The trigger is provided through the `mbt_trigger` port. This signal is synchronized and debounced only so that it can be driven by a mechanical pushbutton.

The counter increments when enabled to do so by the `fsm_advance` signal. The wait control logic uses the table `wait_*` feedback signals to determine if it should unconditionally advance, wait for trigger, wait for idle, or unconditionally halt. As the counter increments, it plays back a stored sequence from the remaining table outputs.

The `inject_address` and `inject_strobe` outputs are connected to the SEM controller error injection port. After waiting for the trigger, the control logic commands the SEM controller to idle by providing the “enter idle” command with a three-cycle assertion of the `inject_strobe` output. After this, the control logic waits for the SEM controller to idle, a condition decoded from the SEM controller status port signals `status_*`.

After the SEM controller is idle, no further error detections occur, and the control logic multiplexes the ICAP away from the SEM controller using the `mbt_icap_sel` output. The control logic then presents the required commands to the ICAP for the MultiBoot event using the `mbt_icap_i`, `mbt_icap_cs`, and `mbt_icap_rdwrb` outputs:

- SYNC word.
- Type 1 write to CMD register with NULL command.
- Type 1 write to GENERAL1 and GENERAL2, or WBSTAR (depends on the FPGA architecture).
- Type 1 write to CMD register with IPROG command.

The ICAP activity depends on `mbt_next_addr` and the FPGA architecture. The sequences are based on information presented in the FPGA configuration user guides, modified by requirements for sharing the ICAP with the SEM controller. Refer to the MultiBoot module source files for register-level detail. After the ICAP activity is complete, reconfiguration occurs.

### SP605 Parameter Selection

The MultiBoot module loads `NEXT_CONFIG_ADDRESS` into the Spartan-6 FPGA's GENERAL1 and GENERAL2 registers as the next bitstream location. The GENERAL1 and GENERAL2 settings are based on the SP605 schematic, which shows the FPGA configuration signal A[23] connected to the most significant address pin on the 16M x 16 (256 Mb) parallel flash. The parallel flash is large enough to hold two bitstreams.

The parallel flash is logically divided into two banks, one bank for each bitstream. The selection is made via the FPGA's configuration signal A[23] with the remaining lower address bits always zero. It is assumed that Implementation A is in bank zero, and Implementation B is in bank one.

For Implementation A to reconfigure the FPGA with Implementation B, `NEXT_CONFIG_ADDRESS` is 0x00800000 hex, or 8,388,608 decimal:

- GENERAL2[15:10] = 000000, reserved
- GENERAL2[9:8] = 00, unused

- GENERAL2[7:0] = 10000000
- GENERAL1[15:0] = 0000000000000000

For Implementation B to reconfigure the FPGA with Implementation A, NEXT\_CONFIG\_ADDRESS is 0x00000000 hex, or 0 decimal:

- \* GENERAL2[15:10] = 000000, reserved
- \* GENERAL2[9:8] = 00, unused
- \* GENERAL2[7:0] = 00000000
- \* GENERAL1[15:0] = 0000000000000000

The SEM controller locates the start of its data in the SPI flash using THIS\_SEM\_IP\_ADDRESS. The 16M x 8 (128 Mb) SPI flash is large enough to hold two sets of data considering the SEM controller option settings. Therefore, if the SPI flash is logically divided into two banks, one bank for each set of data, the starting address only differs by the most significant address bit of the SPI flash. It is assumed that Implementation A stores its data in bank zero, and Implementation B stores its data in bank one.

For Implementation A to find its data, THIS\_SEM\_IP\_ADDRESS is 0x00000000 hex, or 0 decimal. For Implementation B to find its data, THIS\_SEM\_IP\_ADDRESS is 0x00800000 hex, or 8,388,608 decimal.

Figure 3 illustrates the resulting memory maps for the flash devices associated with this application example.

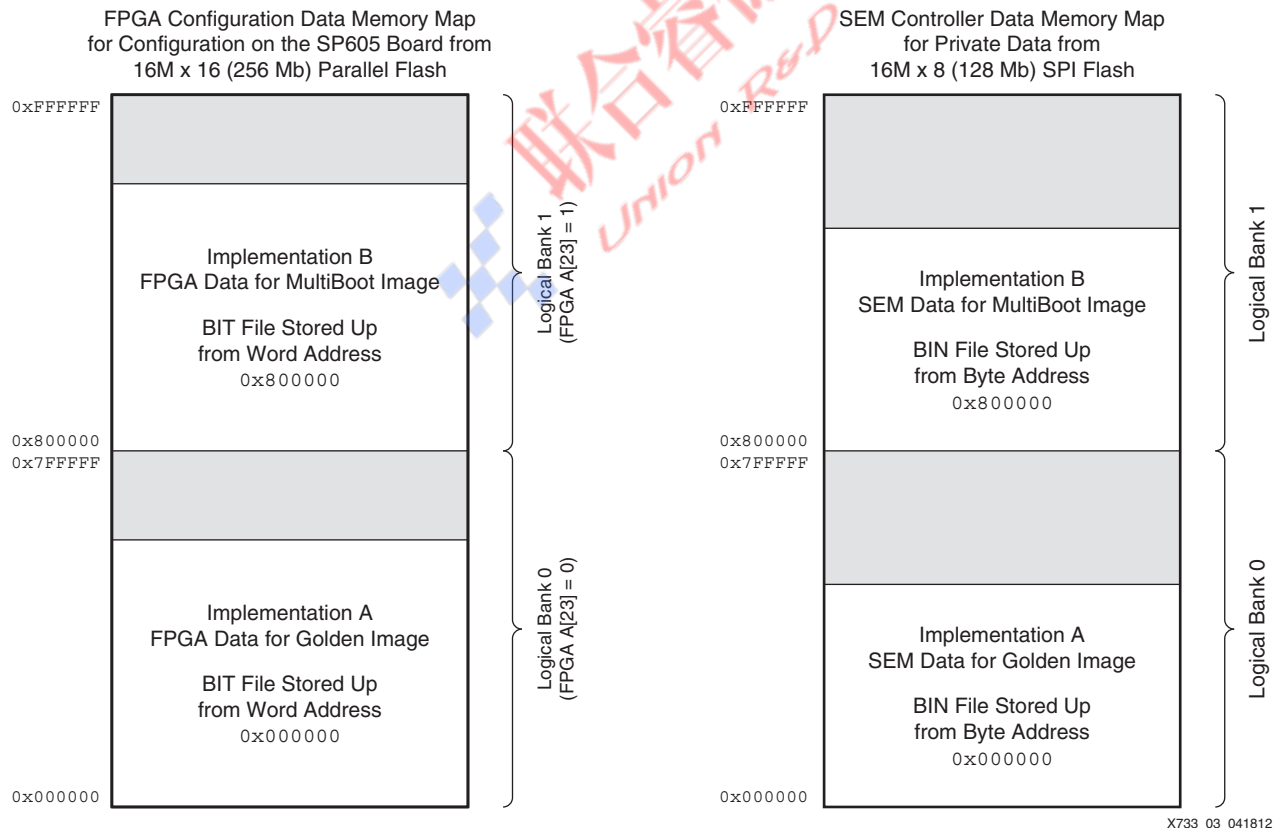


Figure 3: Data Organization for the SP605 MultiBoot Application Example



THIS\_BITSTREAM\_NMBR is the value applied to LEDs identified as DS3, DS4, and DS5 in the SP605 schematic to visually identify the bitstream. For Implementation A, the value used is 2, which results in DS4 turning on and DS3 and DS5 turning off (Figure 6). For Implementation B the value used is 5, which results in DS3 and DS5 turning on and DS4 turning off (Figure 7).

### ML605 Parameter Selection

The MultiBoot module loads NEXT\_CONFIG\_ADDRESS into the Virtex-6 FPGA's WBSTAR register as the next bitstream location. WBSTAR settings are based on the ML605 schematic, which shows FPGA configuration signal RS[0] connected to the most significant address pin on the 16M x 16 (256 Mb) parallel flash. The parallel flash is large enough to hold two bitstreams.

The parallel flash is logically divided into two banks, one bank for each bitstream. The selection is made via RS[0] with the remaining lower address bits always zero. It is assumed that Implementation A is in bank zero, and Implementation B is in bank one.

For Implementation A to reconfigure the FPGA with Implementation B, NEXT\_CONFIG\_ADDRESS is 0x0C000000 hex, or 201,326,592 decimal:

- WBSTAR[31:29] = 000, reserved
- WBSTAR[28:27] = 01, RS[1:0] drive value
- WBSTAR[26] = 1, RS[1:0] driver enable
- WBSTAR[25:0] = 000000000000000000000000

For Implementation B to reconfigure the FPGA with Implementation A, NEXT\_CONFIG\_ADDRESS is 0x04000000 hex, or 67,108,864 decimal:

- WBSTAR[31:29] = 000, reserved
- WBSTAR[28:27] = 00, RS[1:0] drive value
- WBSTAR[26] = 1, RS[1:0] driver enable
- WBSTAR[25:0] = 000000000000000000000000

The SEM controller locates the start of its data in the SPI flash using THIS\_SEM\_IP\_ADDRESS. The 16M x 8 (128 Mb) SPI flash is large enough to hold two sets of data considering the SEM controller option settings. Therefore, if the SPI flash is logically divided into two banks, one bank for each set of data, the starting address only differs by the most significant address bit of the SPI flash. It is assumed that Implementation A stores its data in bank zero, and Implementation B stores its data in bank one.

For Implementation A to find its data, THIS\_SEM\_IP\_ADDRESS is 0x00000000 hex, or 0 decimal. For Implementation B to find its data, THIS\_SEM\_IP\_ADDRESS is 0x00800000 hex, or 8,388,608 decimal.

Figure 4 illustrates the resulting memory maps for the flash devices associated with this application example.

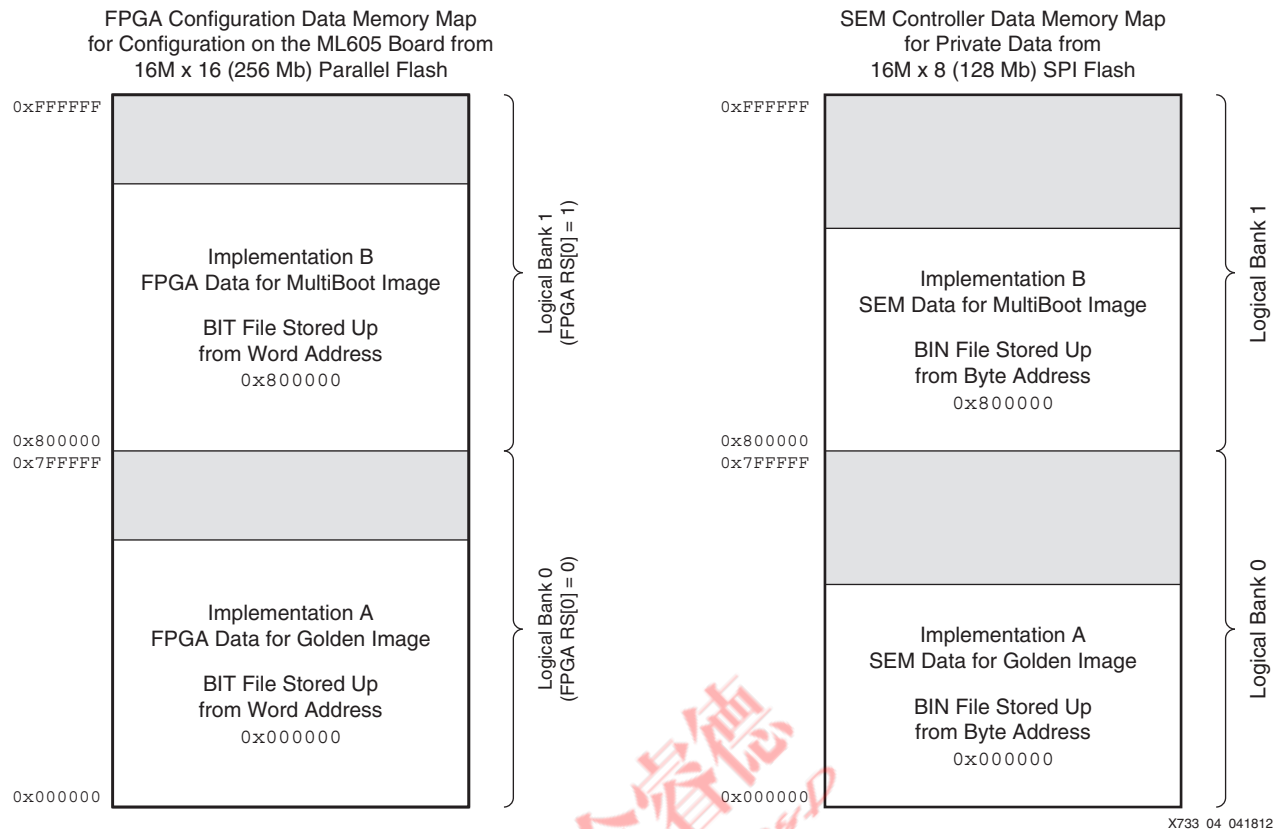


Figure 4: Data Organization for the ML605 MultiBoot Application Example

THIS\_BITSTREAM\_NMBR is the value applied to LEDs identified as DS16, DS17, and DS19 on the ML605 schematic so that the bitstream identity can be identified visually. For Implementation A the value used is 2, which results in DS16 turning on and DS17 and DS19 turning off (Figure 8). For Implementation B the value used is 5, which results in DS17 and DS19 turning on and DS16 turning off (Figure 9).

### KC705 Parameter Selection

The MultiBoot module loads NEXT\_CONFIG\_ADDRESS into the Kintex-7 FPGA's WBSTAR register as the next bitstream location. The WBSTAR settings are based on the KC705 schematic, which shows FPGA configuration signal RS[1:0] connected to the two most significant address pins on the 64M x 16 (1 Gb) parallel flash. The parallel flash is large enough to hold four bitstreams.

The parallel flash is logically divided into four banks, one bank for each bitstream and two unused banks. The selection is made via RS[1:0] with the remaining lower address bits always zero. It is assumed that Implementation A is in bank zero, and Implementation B is in bank one.

For Implementation A to reconfigure the FPGA with Implementation B, the NEXT\_CONFIG\_ADDRESS is 0x60000000 hex, or 1,610,612,736 decimal:

- WBSTAR[31:30] = 01, RS[1:0] drive value
- WBSTAR[29] = 1, RS[1:0] driver enable
- WBSTAR[28:0] = 00000000000000000000000000000000

For Implementation B to reconfigure the FPGA with Implementation A, the NEXT\_CONFIG\_ADDRESS is 0x20000000 hex, or 536870912 decimal:

- WBSTAR[31:30] = 00, RS[1:0] drive value
- WBSTAR[29] = 1, RS[1:0] driver enable

- WBSTAR[28:0] = 00000000000000000000000000000000

The SEM controller locates the start of its data in the SPI flash using THIS\_SEM\_IP\_ADDRESS. The 32M x 8 (256 Mb) SPI flash is large enough to hold two sets of data considering the SEM controller option settings. Therefore, if the SPI flash is logically divided into two banks, one bank for each set of data, the starting address differs only by the most significant address bit of the SPI flash. It is assumed that Implementation A stores its data in bank zero, and Implementation B stores its data in bank one.

For Implementation A to find its data, THIS\_SEM\_IP\_ADDRESS is 0x00000000 hex, or 0 decimal. For Implementation B to find its data, THIS\_SEM\_IP\_ADDRESS is 0x01000000 hex, or 16,777,216 decimal.

Figure 5 illustrates the resulting memory maps for the flash devices associated with this application example.

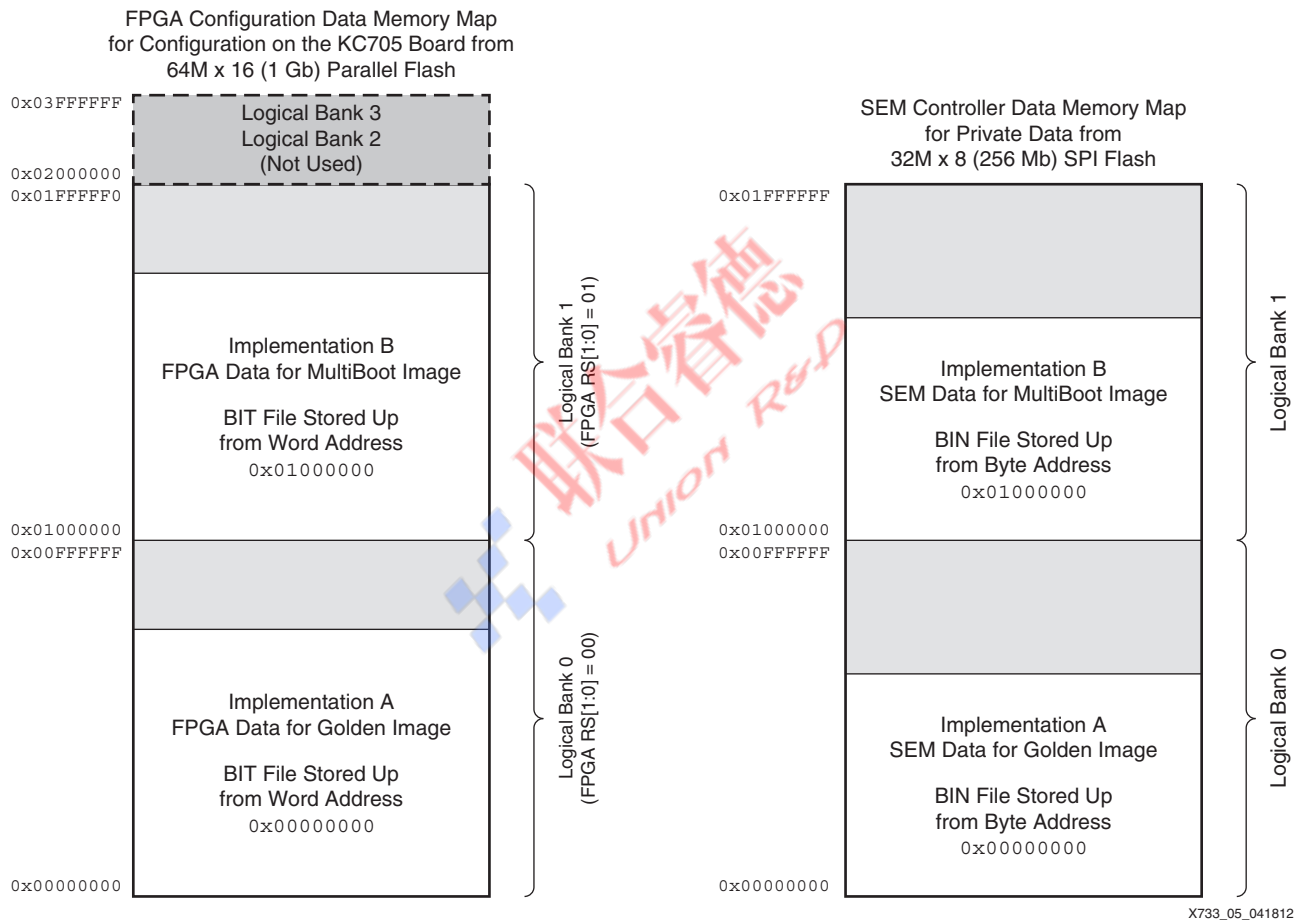


Figure 5: Data Organization for the KC705 MultiBoot Application Example

THIS\_BITSTREAM\_NMBR is the value applied to LEDs identified as DS25, DS26, and DS27 on the KC705 schematic so that the bitstream can be identified visually. For Implementation A the value used is 2, which results in DS26 turning on and DS25 and DS27 turning off (Figure 10). For Implementation B the value used is 5, which results in DS25 and DS27 turning on and DS26 turning off (Figure 11).

## Implementation and Programming

Within the selected supporting reference design directory, the `xapp_implement_a` subdirectory contains scripts to produce the result files for Implementation A. Similarly, the `xapp_implement_b` subdirectory contains scripts to produce the result files for Implementation B. These implementations are derived from the same source files, with top-level parameters specified in `xst.scr` yielding unique physical implementations.

For implementations with `THIS_SEM_IP_ADDRESS` set to a non-zero value, pointers in the table in the SEM controller data need to be adjusted to reflect the new location of the data. The `makedata.tcl` scripts provided with the supporting reference designs are derived from the original `makedata.tcl` script provided with the SEM controller, with minor modifications to support a non-zero value of `THIS_SEM_IP_ADDRESS`.

The `makedata.tcl` scripts provided with the supporting reference designs have also been modified to inject a test pattern in the SEM controller data. This modification enables validation of data set selection by injecting errors at specific locations and observing the error classification result.

The test pattern is located in frame address zero in the word that holds the essential bit data for the error correcting code (ECC) checksum. Normally, the checksum is classified as non-essential, which is represented by zeroes. The modified `makedata.tcl` script replaces these zeroes with `THIS_BITSTREAM_NMBR`. As a result, it is possible to distinguish between the SEM controller data of Implementation A, the SEM controller data of Implementation B, and unmodified SEM controller data.

Known good implementation results are provided in each of the implementation subdirectories. The user can rerun the `implement` script (`implement.sh` or `implement.bat`) in the implementation subdirectories, if desired. The programming script in the `xapp_programming` subdirectory can be used with either the known good or regenerated implementation results.

The programming script (`make_proms.sh` or `make_proms.bat`) checks for the existence of the result files and then merges the Implementation A bitstream and the Implementation B bitstream for programming the parallel flash on the evaluation kit. The script also merges the SEM controller data for programming the SPI flash. After creating the merged data files, the script provides instructions to program the parallel flash and prompts the user to either proceed with programming or skip the programming step.

**Note:** Programming the parallel flash can take a considerable amount of time.

If the optional SPI flash is available as part of the hardware target, the user must obtain and follow the instructions of a third-party programming solution and program the SPI flash separately.

## Validation in Hardware

After the parallel flash (and optional SPI flash) programming is complete, the MultiBoot application example is ready for validation in hardware.

### MultiBoot Operation

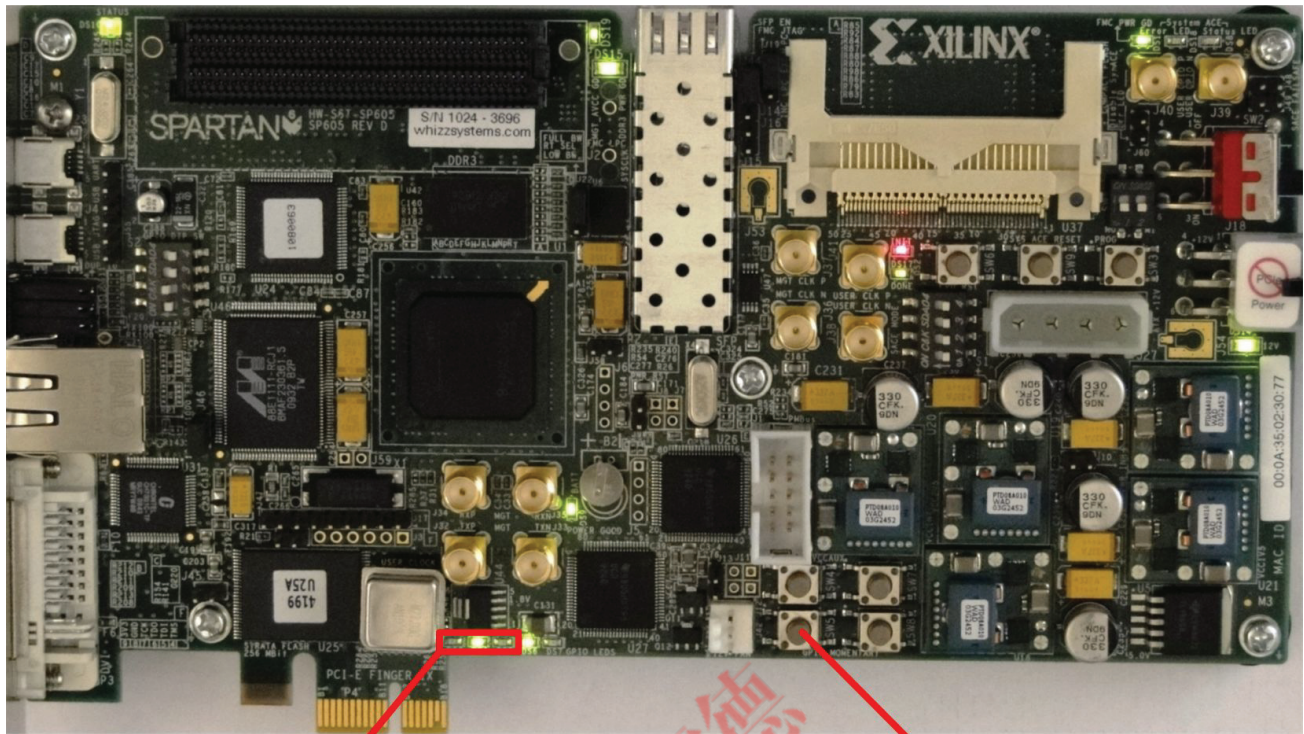
To validate correct operation of MultiBoot:

1. Disconnect all programming hardware and cycle the power.
2. Confirm that the FPGA initially loads Implementation A by observing the LEDs on `design_number`.
3. Press the MultiBoot trigger button.
4. Confirm that the FPGA loads Implementation B by observing the LEDs on `design_number`.
5. Press the MultiBoot trigger button several more times. Each time, confirm that the FPGA cycles between Implementation A and Implementation B.

These figures show the implementations loaded on each hardware target and the locations of the MultiBoot trigger button and the LEDs on `design_number`:

- SP605 Implementation A (Figure 6) and Implementation B (Figure 7)
- ML605 Implementation A (Figure 8) and Implementation B (Figure 9)
- KC705 Implementation A (Figure 10) and Implementation B (Figure 11)



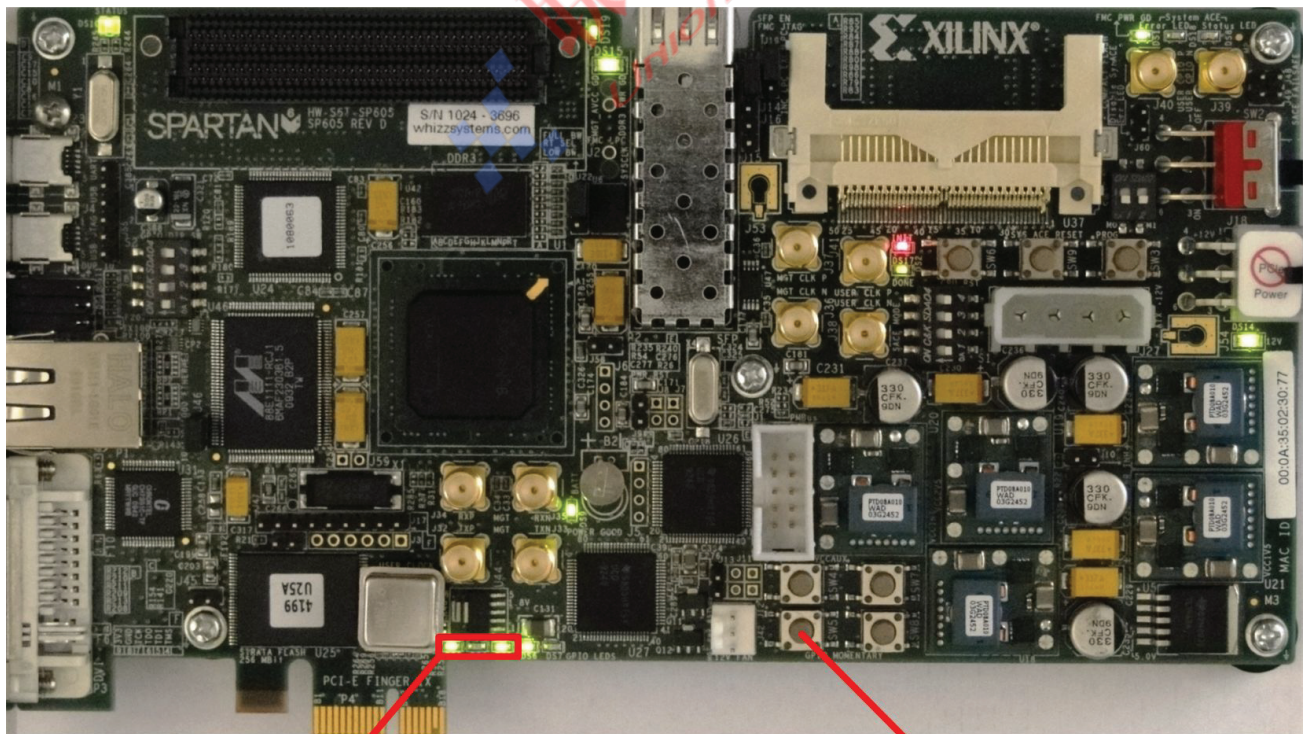


Design Number LEDs

MultiBoot Trigger Button

XAPP733\_006\_030712

Figure 6: SP605 Implementation A with Design Number LEDs = 2



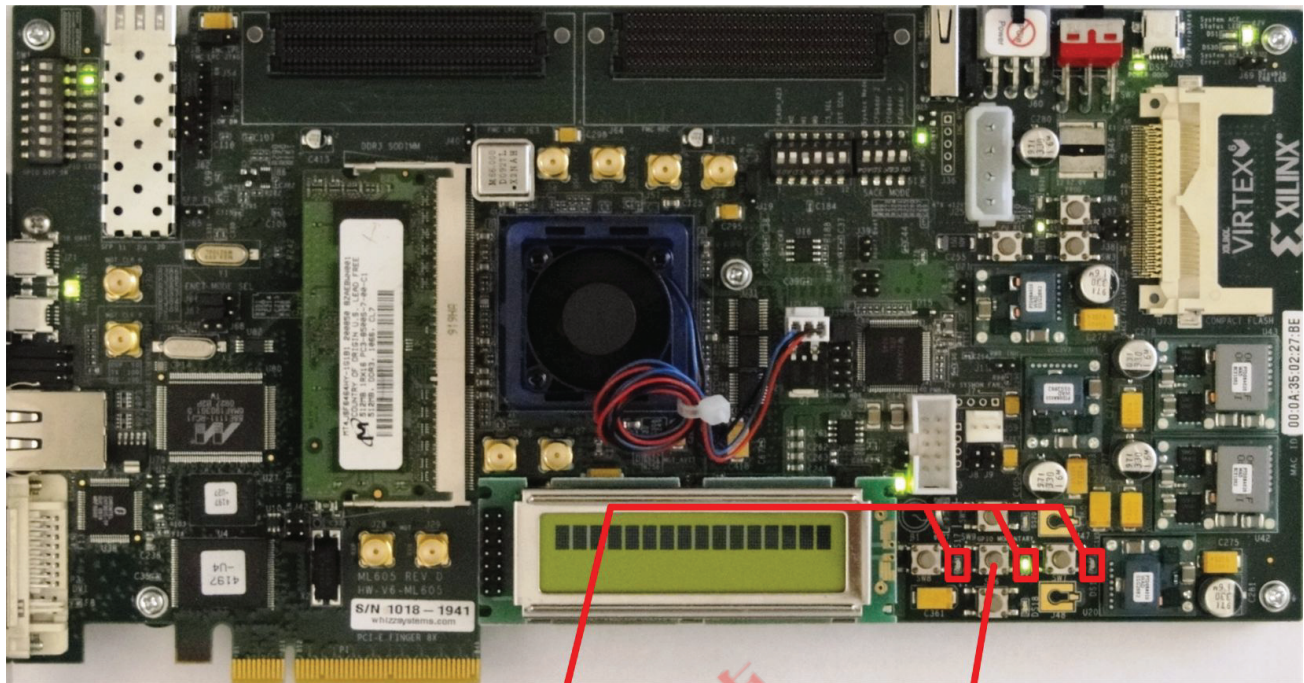
Design Number LEDs

MultiBoot Trigger Button

X733\_07\_030712

Figure 7: SP605 Implementation B with Design Number LEDs = 5



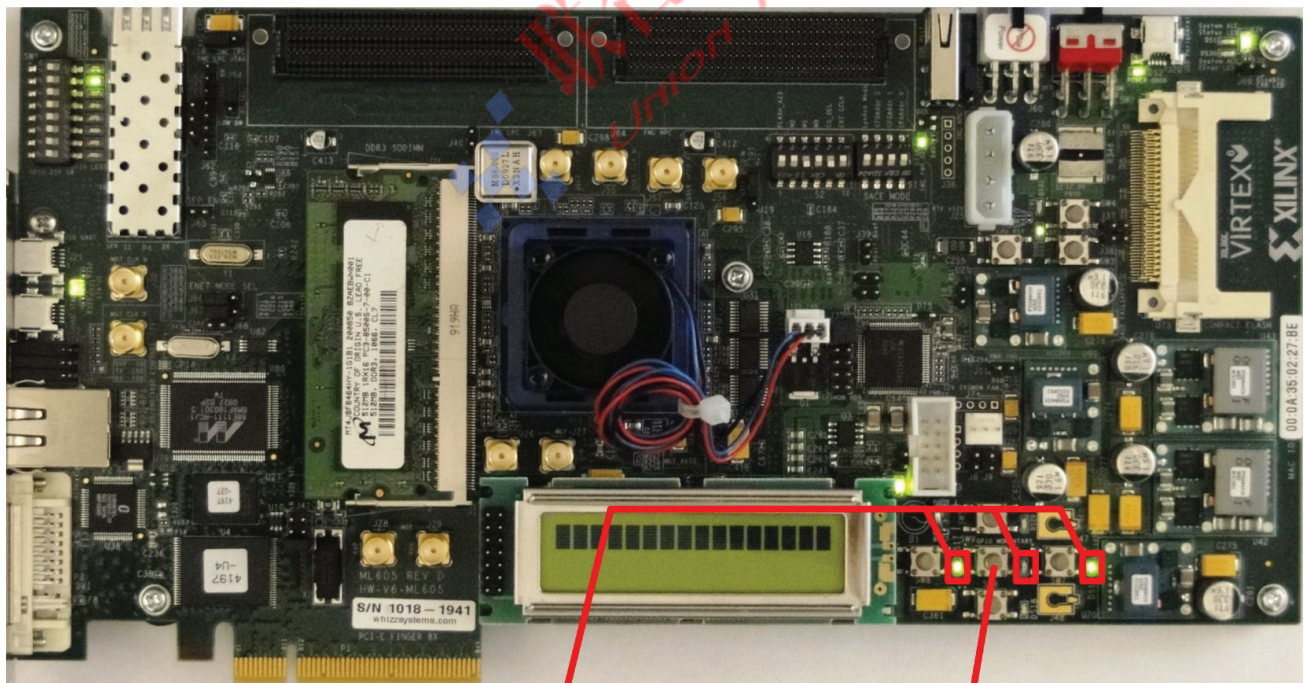


Design Number LEDs

MultiBoot Trigger Button

X733\_08\_030712

Figure 8: ML605 Implementation A with Design Number LEDs = 2



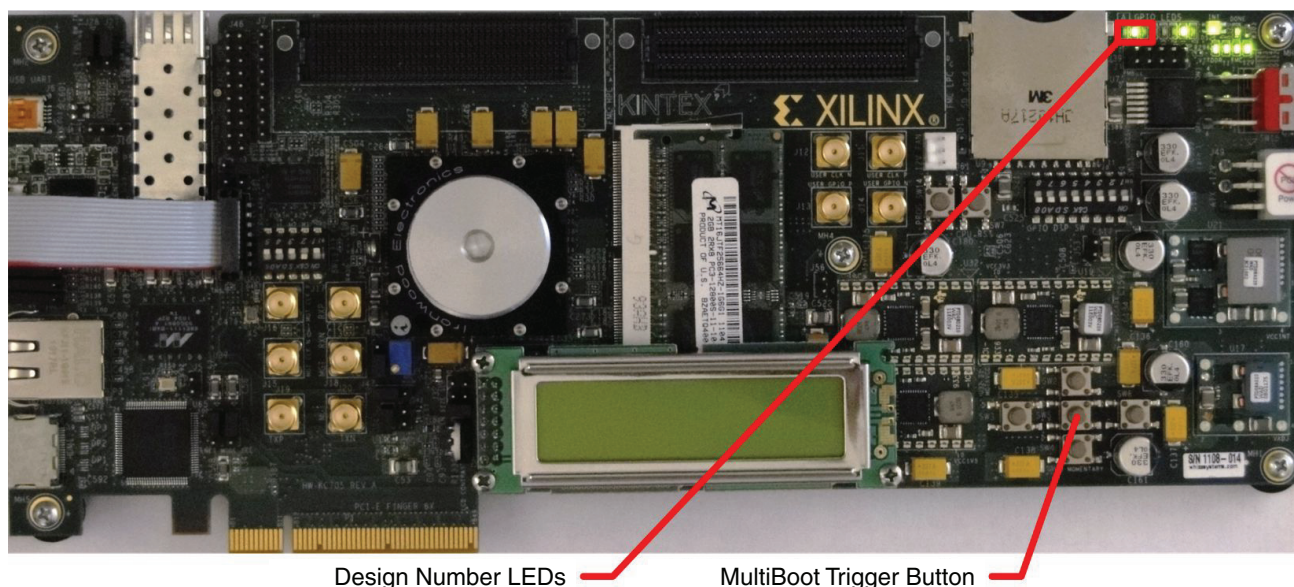
Design Number LEDs

MultiBoot Trigger Button

X733\_09\_030712

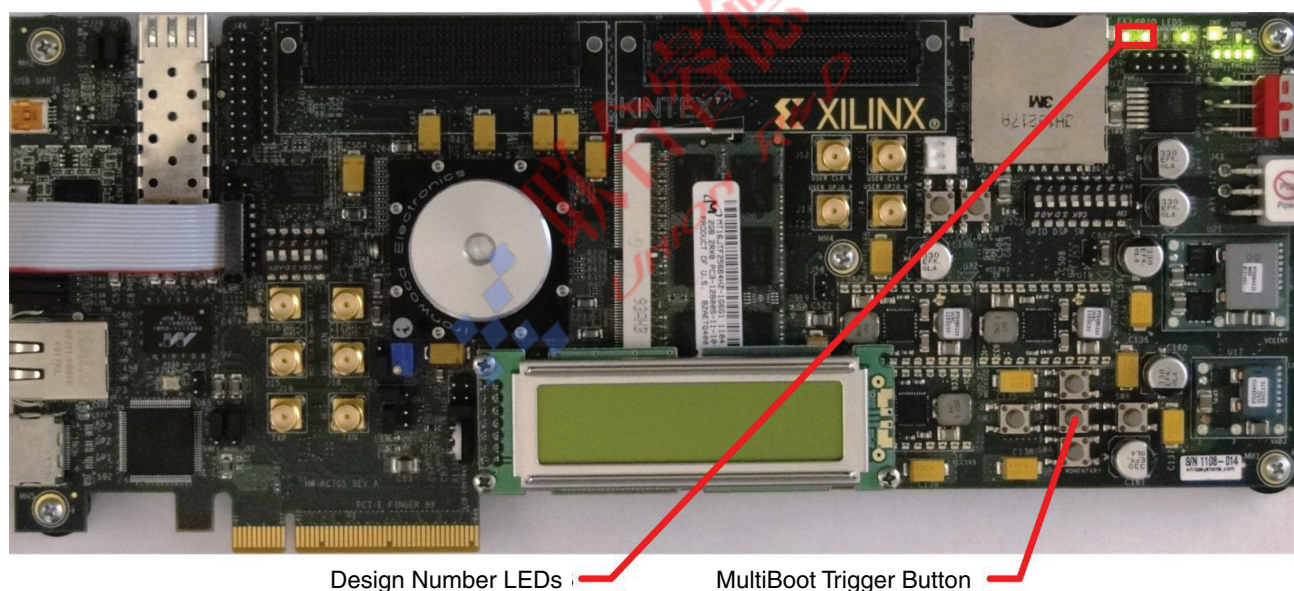
Figure 9: ML605 Implementation B with Design Number LEDs = 5





X733\_10\_030712

Figure 10: KC705 Implementation A with Design Number LEDs = 2



X733\_11\_030712

Figure 11: KC705 Implementation B with Design Number LEDs = 5

## SEM Controller Boot

The USB-to-UART bridge on the hardware target is used to interface the SEM controller to a terminal program running on a PC. The Silicon Labs USB-to-UART Bridge Virtual COM Port (VCP) driver must be installed on the PC, and then the PC cabled to the hardware target. The required drivers are available free of charge on the Silicon Labs driver download site:

<http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>

Refer to the configuration guide for the target FPGA in [References, page 20](#) for more information on installing the driver and connecting the hardware. The terminal program on the PC must have these settings:

- Baud: 9600

- Settings: **8-N-1**
- Flow control: **None**
- Terminal setup: **VT100**
- TX newline: **CR**
- RX newline: **CR+LF**
- Local echo: **No**

The USB-to-UART bridge is typically self-powered (versus bus-powered), therefore the hardware target requires power before the terminal program can identify and set up the associated COM port. Power-cycling the hardware target can cause the terminal session to cease responding.

After application of power and each MultiBoot event, the SEM controller issues an initialization report to confirm that it is present and successfully initialized. A sample initialization report from the KC705 reference design is:

```
X7_SEM_V3_2      Core name and version
SC 01            Enter initialization state
FS 08            Core feature set report
ICAP OK          ICAP communication pass
RDBK OK          Readback initialized
INIT OK          Initialization complete
SC 02            Enter observation state
O>              Observation prompt
```

If desired, the user can compare the observed initialization report against the reference shown in *LogiCORE IP Soft Error Mitigation Controller v3.2 Product Guide* [Ref 8]. This exercise validates SEM controller boot.

## SEM Controller Data

If the optional SPI flash is available as part of the hardware target and the SPI flash has been programmed, the SEM controller data can be validated by using the test pattern in the frame zero ECC checksum. This validation is done by probing the error classification results from errors injected into those addresses.

**Note:** This exercise validates the SEM controller data. If the SPI flash is absent, blank, or contains incorrect data, the SEM controller generates logs that differ from those shown.

From the terminal session, the user moves the SEM controller to idle (“I” command), injects one error (“N” command), and then returns the SEM controller to observation (“O” command). At the specified addresses, the error classification should generate the expected classification.

For SP605 Implementation A, the error injection addresses and expected classifications are:

- **C00000820** (Non-essential, based on THIS\_BITSTREAM\_NMBR[0] = 0)
- **C00000821** (Essential, based on THIS\_BITSTREAM\_NMBR[1] = 1)
- **C00000822** (Non-essential, based on THIS\_BITSTREAM\_NMBR[2] = 0)

For SP605 Implementation B, the error injection addresses and expected classifications are:

- **C00000820** (Essential, based on THIS\_BITSTREAM\_NMBR[0] = 1)
- **C00000821** (Non-essential, based on THIS\_BITSTREAM\_NMBR[1] = 0)
- **C00000822** (Essential, based on THIS\_BITSTREAM\_NMBR[2] = 1)

For ML605 Implementation A, the error injection addresses and expected classifications are:

- **C00000500** (Non-essential, based on THIS\_BITSTREAM\_NMBR[0] = 0)
- **C00000501** (Essential, based on THIS\_BITSTREAM\_NMBR[1] = 1)
- **C00000502** (Non-essential, based on THIS\_BITSTREAM\_NMBR[2] = 0)

For ML605 Implementation B, the error injection addresses and expected classifications are:

- **C00000500** (Essential, based on THIS\_BITSTREAM\_NMBR[0] = 1)
- **C00000501** (Non-essential, based on THIS\_BITSTREAM\_NMBR[1] = 0)
- **C00000502** (Essential, based on THIS\_BITSTREAM\_NMBR[2] = 1)

For KC705 Implementation A, the error injection addresses and expected classifications are:

- **C000000640** (Non-essential, based on THIS\_BITSTREAM\_NMBR[0] = 0)
- **C000000641** (Essential, based on THIS\_BITSTREAM\_NMBR[1] = 1)
- **C000000642** (Non-essential, based on THIS\_BITSTREAM\_NMBR[2] = 0)

For KC705 Implementation B, the error injection addresses and expected classifications are:

- **C000000640** (Essential, based on THIS\_BITSTREAM\_NMBR[0] = 1)
- **C000000641** (Non-essential, based on THIS\_BITSTREAM\_NMBR[1] = 0)
- **C000000642** (Essential, based on THIS\_BITSTREAM\_NMBR[2] = 1)

This is a sample log, with comments, of the terminal session from KC705 Implementation A:

```
O> I
SC 00
I> N C000000640          Inject error into ECC[0]
SC 10
SC 00
I> O
SC 02
O>
SC 04
SED OK                   Single bit error detect
PA 00000000
LA 00000000
WD 32 BT 00
COR                      Begin correction rpt
WD 32 BT 00              Correction, expected
END                      End rpt
FC 00
SC 08
CLA                      Begin classification rpt
END                      End rpt, non-essential, expected
FC 00
SC 02

O> I
SC 00
I> N C000000641          Inject error into ECC[1]
SC 10
SC 00
I> O
SC 02
O>
SC 04
SED OK                   Single bit error detect
PA 00000000
LA 00000000
WD 32 BT 01
COR                      Begin correction rpt
WD 32 BT 01              Correction, expected
END                      End rpt
FC 00
SC 08
CLA                      Begin classification rpt
```

```

WD 32 BT 01          Essential, expected
END                  End rpt
FC 40
SC 02

O> I
SC 00
I> N C0000000642     Inject error into ECC[2]
SC 10
SC 00
I> 0
SC 02
O>
SC 04
SED OK              Single bit error detect
PA 00000000
LA 00000000
WD 32 BT 02
COR                  Begin correction rpt
WD 32 BT 02          Correction, expected
END                  End rpt
FC 40
SC 08
CLA                  Begin classification rpt
END                  End rpt, non-essential, expected
FC 00
SC 02

```

This a sample log, with comments, of the terminal session from KC705 Implementation B:

```

O> I
SC 00
I> N C0000000640     Inject error into ECC[0]
SC 10
SC 00
I> 0
SC 02
O>
SC 04
SED OK              Single bit error detect
PA 00000000
LA 00000000
WD 32 BT 00
COR                  Begin correction rpt
WD 32 BT 00          Correction, expected
END                  End rpt
FC 00
SC 08
CLA                  Begin classification rpt
WD 32 BT 00          Essential, expected
END                  End rpt
FC 40
SC 02

O> I
SC 00
I> N C0000000641     Inject error into ECC[1]
SC 10
SC 00
I> 0
SC 02
O>
SC 04

```

```

SED OK                               Single bit error detect
PA 00000000
LA 00000000
WD 32 BT 01
COR                                 Begin correction rpt
WD 32 BT 01                         Correction, expected
END                                 End rpt
FC 40
SC 08
CLA                                 Begin classification rpt
END                                 End rpt, non-essential, expected
FC 00
SC 02

O> I
SC 00
I> N C000000642                     Inject error into ECC[2]
SC 10
SC 00
I> 0
SC 02
O>
SC 04
SED OK                               Single bit error detect
PA 00000000
LA 00000000
WD 32 BT 02
COR                                 Begin correction rpt
WD 32 BT 02                         Correction, expected
END                                 End rpt
FC 00
SC 08
CLA                                 Begin classification rpt
WD 32 BT 02                         Essential, expected
END                                 End rpt
FC 40
SC 02
O> I
SC 00
I> 0
SC 02

```

## Reference Designs

The reference design files for this application note can be downloaded from:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=185389>

The reference design checklist is shown in [Table 2](#).

**Table 2: Reference Design Checklist**

| Parameter            | Description                             |
|----------------------|---|
| <b>General</b>       |   |
| Developer Name       | Eric Crabill                            |
| Target Devices       | Spartan-6, Virtex-6, and 7 series FPGAs |
| Source Code Provided | Yes                                     |
| Source Code Format   | Verilog and VHDL                        |



Table 2: Reference Design Checklist

| Parameter  | Description  |
|--|--|
| Design Uses Code or IP from Existing Reference Design, Application Note, 3rd Party, or CORE Generator Software | SEM_V3_2 or higher   |
| <b>Simulation</b>  |  |
| Functional Simulation Performed  | N/A  |
| Timing Simulation Performed  | N/A  |
| Testbench Provided for Simulations   | N/A  |
| Testbench Format   | N/A  |
| Simulator Software and Version   | N/A  |
| SPICE/IBIS Simulations   | N/A  |
| <b>Implementation</b>  |  |
| Synthesis Software Tools and Version   | XST, ISE® Design Suite 14.1  |
| Implementation Software Tools and Version  | ISE Design Suite 14.1  |
| Static Timing Analysis Performed   | Yes  |
| <b>Hardware Verification</b>   |  |
| Hardware Verified  | Yes, all six designs   |
| Hardware Platform Used for Verification  | SP605 board and 128 Mb SPI flash<br>ML605 board and 128 Mb SPI flash<br>KC705 board and 256 Mb SPI flash |

The device utilization for the reference design is shown in [Table 3](#).

Table 3: Device Utilization for Logic Added to SEM Controller Example Design

| FPGA      | LUT | Flip-Flop | Block RAM | BUFG |
|-----------|-----|-----------|-----------|------|
| Spartan-6 | 55  | 35        | 0         | 0    |
| Virtex-6  | 70  | 35        | 0         | 0    |
| 7 series  | 70  | 35        | 0         | 0    |

## Conclusion

This application note demonstrates how to apply MultiBoot and the LogiCORE IP Soft Error Mitigation (SEM) controller. Coordination of ICAP sharing is accomplished by asking the permission of the SEM controller, and data organization is fundamentally unchanged even with additional SEM controller data.

The reference designs provided with this document are hardware-validated examples suitable for adaptation to satisfy system-specific design management and field upgrade requirements.

## References

These references provide additional information for this application note:

1. [XAPP517](#), *Dual Use of ICAP with SEM Controller*
2. [UG380](#), *Spartan-6 FPGA Configuration User Guide*
3. [UG360](#), *Virtex-6 FPGA Configuration User Guide*
4. [UG470](#), *7 Series FPGAs Configuration User Guide*
5. [UG525](#), *Getting Started with the Xilinx Spartan-6 FPGA SP605 Evaluation Kit*



6. [UG533](#), *Getting Started with the Xilinx Virtex-6 FPGA ML605 Evaluation Kit*
7. [UG883](#), *Kintex-7 FPGA KC705 Evaluation Kit Getting Started Guide*
8. PG036, *LogiCORE IP Soft Error Mitigation Controller v3.2 Product Guide*
9. [UG526](#), *SP605 Hardware User Guide*
10. [UG534](#), *ML605 Hardware User Guide*
11. [UG810](#), *KC705 Evaluation Board for the Kintex-7 FPGA User Guide*

## Revision History

The following table shows the revision history for this document.

| Date     | Version | Description of Revisions |
|----------|---------|--------------------------|
| 05/24/12 | 1.0     | Initial Xilinx release.  |

## Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.