Multi-Boot User Guide for Nexus Platform

Technical Note

FPGA-TN-02145-2.5

April 2026

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Contents 3
Glossary 5
1. Introduction 6
2. Resources 8
2.1. Single Bitstream Boot Mode 8
2.2. Dual Boot Mode 8
2.3. Ping-Pong Boot Mode 8
2.4. Multi-Boot Mode 8
3. Dual Boot Mode 9
3.1. Description of the Nexus Device Dual Boot Flow Diagram 10
4. Ping-Pong Boot Mode 11
4.1. Description of the Ping-Pong Boot Flow Diagram 12
4.2. JUMP Table File Structure 12
4.3. Swapping Boot Image Order 13
4.4. Safe Primary Image Update Procedure 13
5. Multi-Boot Mode 15
5.1. MULTIBOOT Primitive 15
5.2. Booting Flow without MULTIBOOT Primitive 16
5.2.1. Drawback 16
5.3. Booting Flow with MULTIBOOT Primitive 16
5.3.1. Advantage 16
5.3.2. Implementation of Multi-Boot Feature Using MULTIBOOT Primitive 17
6. Creating a PROM File 21
6.1. Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File 21
6.2. Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File 26
6.3. Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File 31
7. Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device 38
8. Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the Internal Flash 41
9. Corrupting Primary Image to Test Dual Boot, Ping-pong Boot, or Multi-Boot 44
9.1. Corrupting Generated .mcs File using Deployment Tool 44
9.2. Example of Corrupting Preamble of Primary Image in Dual Boot 45
10. Use Case Restrictions 47
10.1. Ping-pong Boot Limitation 47
10.2. Soft Error Detection and Correction (SEDC) Use Case 47
References 48
Technical Support Assistance 49
Revision History 50

Figures

Figure 3.1. Nexus Device Dual Boot Flow Diagram 9
Figure 4.1. Ping-Pong Boot Flow Diagram 11
Figure 4.2. Example JUMP Table in Binary Format 13
Figure 5.1. MULTIBOOT Primitive, OSC IP, CONFIG_LMMI Primitive and Lmmi Host Connection 17
Figure 5.2. Implementation of Multi-Boot Feature Using MULTIBOOT Primitive Flow 18
Figure 5.3. Launching the Programming File Utility Tool 19
Figure 5.4. Opening the Control Register 0 Editor Window 19
Figure 5.5. Control Register 0 Window 19
Figure 5.6. Default SPIM Bit Setting 20
Figure 5.7. Saving Control Register 0 Bit Settings 20
Figure 6.1. Creating New Deployment for Dual Boot PROM Hex File 21
Figure 6.2. Select Input Files Window 22
Figure 6.3. Dual Boot Options Window 23
Figure 6.4. Select Output File Window 24
Figure 6.5. Generate Deployment Window 25
Figure 6.6. Creating New Deployment for Ping-Pong Boot PROM Hex File 26
Figure 6.7. Select Input Files Window 27
Figure 6.8. Ping-Pong Boot Options Window 28
Figure 6.9. Select Output File Window 29
Figure 6.10. Generate Deployment Window 30
Figure 6.11. Creating New Deployment for Multi-Boot 31
Figure 6.12. Select Input File Window 32
Figure 6.13. Advanced SPI Flash Options – Options Tab Window 34
Figure 6.14. Advanced SPI Flash Options – Multiple Boot Tab Window 35
Figure 6.15. Select Output File Window 36
Figure 6.16. Generate Deployment Window 37
Figure 7.1. Radiant Programmer – Getting Started Window 39
Figure 7.2. Radiant Programmer – Device Properties Window 40
Figure 8.1. Radiant Programmer – Device Properties Window 43
Figure 9.1. Reading Device Status Register using Radiant Programmer 45
Figure 9.2. Device Status Register Value after Fall Back to Golden Image 46

Tables

Table 1.1. Supported Device Families and Parts 6
Table 1.2. Maximum Supported Flash Frequency 7
Table 3.1. Control Register 1 [3:2] – Master Preamble Timer Retry Value 10
Table 4.1. Control Register 1 [3:2] – Master Preamble Timer Retry Value 12
Table 4.2. JUMP Table File Structure[1] 13
Table 8.1. Programming Options for Direct FLASH Programming Mode in Radiant Programmer 41

Glossary

A glossary of terms used in this document.

Acronym	Definition
Alternative Boot	After the FPGA device has been configured, this pattern is loaded when the PROGRAMN pin is toggled or the Refresh instruction is issued. Up to four Alternative Boot patterns are possible.
Binary Hex Data File (.bin File)	The data image of the Hex data file in binary format. All Hex data files are converted into this format prior to consumption. This type of file is not printable.
Bitstream Data File (.bit File)	The configuration data file, for a single FPGA device, in the format that can be loaded directly into the FPGA device to configure the SRAM cells. The file is expressed in binary Hex format. The file is not printable.
Configure	Write the pattern into the SRAM fuses of the FPGA device and wake up.
Dual Boot	The device has two patterns, a Primary pattern and a Golden pattern, to choose to load.
EBR	Embedded Block RAM
FD-SOI (Fully Depleted Silicon On Insulator)	A process that uses an ultra-thin buried oxide layer.
Flash Lock	The feature provides protection to the Flash fuses against accidental erase or corruption. Most of the SPI Flash devices support Soft Lock. Lock choices include: • Whole device • Bottom half • Bottom quarter • Last sector Details can be found in the SPI Flash device data sheet.
Golden Boot	The guaranteed good pattern loaded into the FPGA device when booting failure occurs. It is also known as the root boot. Only one Golden Boot pattern is allowed.
Hex Data File (.exo, .mcs, .xtek Files)	The data record files that are in the format commonly known as Intel Hex, Motorola Hex or Extended Tektronix Hex. They are also known as addressed record files. The advantages include its small size and it is printable, and thus good for record keeping. This type of file is not directly consumable by the utilities supporting it.
LRAM	Large RAM
Multi-Boot Multiple Boot	The device has more than two patterns, a Primary pattern, a Golden pattern and some Alternative patterns, to choose to load.
Primary Boot	Upon power cycling, the FPGA device loads this pattern in first. Only one Primary pattern is allowed.
Program	Writes into the selected Flash cells state a logical zero (0) (close fuse).
RAM	Random Access Memory
Refresh	The action loads the pattern from a non-volatile source to configure the FPGA device.
Sector (Block)	The smallest number of bytes of Flash fuses can be erased at the same time by the erase command.
SPI	Stands for the Serial Peripheral Interface defined originally by Motorola.
SRAM	Static Random Access Memory

1. Introduction

CrossLink™-NX, Certus™-NX, CertusPro™-NX, and MachXO5™-NX families of low-power FPGAs can be used in a wide range of applications and are optimized for the bridging and processing needs in the Embedded Vision space. It is built on Lattice Nexus FPGA platform, using low-power 28-nm FD-SOI technology. For the subsequent part of this document, the Nexus devices refer to all CrossLink-NX, Certus-NX, CertusPro-NX, and MachXO5-NX device families.

The Nexus devices support various booting options for loading the configuration SRAM from a non-volatile memory for configuration flexibility and fail-safe configuration. CrossLink-NX, Certus-NX and CertusPro-NX families use an external memory while MachXO5-NX families only support internal flash memory for storage of configuration bitstreams. See Table 1.1 for details of the supported device families and parts.

Table 1.1. Supported Device Families and Parts

Device Family	Parts included in the Device Family
CrossLink-NX	LIFCL-17, LIFCL-33, LIFCL-33U, and LIFCL-40
Certus-NX	LFD2NX-9, LFD2NX-17, LFD2NX-15, LFD2NX-25, LFD2NX-35, LFD2NX-65, LFD2NX-28, and LFD2NX-40
CertusPro-NX	LFCPNX-50 and LFCPNX-100
MachXO5-NX[1]	LFMXO5-25, LFMXO5-55T, LFMXO5-100T, LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T

Note:

1. This user guide is not applicable to MachXO5-NX Root-of-Trust (RoT) devices. Refer to FAQ 6848 to request for the relevant user guide to learn the RoT device dual boot implementation.

The Nexus devices support various configuration boot modes to mitigate risk during the field upgrade process and to allow flexibility of executing different patterns. Field upgrade disruptions may occur due to power disruption, communication interruption or bitstream pattern corruption. The Nexus devices support the following boot modes:

- Dual Boot mode – Switches to load from the second known good (Golden) pattern when the first pattern becomes corrupted.
- Ping-Pong Boot mode – Switches between two bitstream patterns based on your choice. If the system fails to boot from one of the bitstreams, it automatically boots from the second bitstream.
- Multi-Boot mode – Allows the system to dynamically switch between two to five bitstream patterns while still being protected with a Golden (sixth) pattern. Note that the MachXO5-NX family supports up to 3 bitstream patterns only inclusive of the Golden pattern.

The Nexus devices support these boot modes by combining all the bitstream patterns into a single boot image and storing it in a single external SPI Flash device (internal flash for MachXO5-NX families). This solution decreases cost, reduces board space, and simplifies field upgrades.

For the device to be able to fall back to the golden/secondary image reliably after failing to configure from primary image, it is important to understand the maximum flash frequency setting, MCCLK_FREQ for external flash, FLASH_CLK_FREQ for internal flash, in the Global tab of Lattice Radiant Device Constraint Editor. Refer to Table 1.2 for the maximum supported flash frequency for respective parts.

Table 1.2. Maximum Supported Flash Frequency

Parts	Maximum Supported Flash Frequency
LFMXO5-25, LFMXO5-35, LFMXO5-35T, LFMXO5-65 and LFMXO5-65T	112.5 MHz
LFMXO5-55T, LFMXO5-100T	75 MHz[1]
LFD2NX-15, LFD2NX-25, LIFCL-33, LIFCL-33U, LFD2NX-35, LFD2NX-65	Refer to the MCCLK_FREQ calculation in sysCONFIG User Guide for Nexus Platform (FPGA-TN-02099)
LFCPNX-50, LFCPNX-100, LFD2NX-9, LFD2NX-17, LIFCL-17, LFD2NX-28, LFD2NX-40 and LIFCL-40	75 MHz[1], or refer to the MCCLK_FREQ calculation in sysCONFIG User Guide for Nexus Platform (FPGA-TN-02099), whichever lower.

Note:
1. The CR1[20] RX_EDGE bit must be set to 1 to enable flash frequencies up to 75 MHz. With the default value (RX_EDGE = 0), the maximum supported frequency is 56.2 MHz. See sysCONFIG User Guide for Nexus Platform (FPGA-TN-02099) for CR1 programming details.

Important Note: To enable the Transparent Field Reconfiguration (TransFR™) feature with any of the supported boot modes, the Master SPI port must be persisted or enabled as a configuration port after the device enters user mode. These settings can be set in Lattice Radiant Device Constraint Editor on the Global tab.

2. Resources

The Nexus devices are SRAM-based FPGAs. The volatile SRAM configuration memory must be loaded from a non-volatile memory that can store all the configuration data. The size of the configuration data is based on the amount of logic available in the FPGA, number of pre-initialized Embedded Block RAM (EBR) components and number of pre-initialized Large RAM (LRAM) Block components. A design using the largest device, with every EBR and LRAM pre-initialized with unique data values and generated without compression requires the largest amount of storage.

The minimum SPI Flash densities required to support different configuration boot modes can be calculated based on formula and examples below. The available SPI Flash densities are 2^n Mb, where n is from 2 to 11.

Refer to sysCONFIG User Guide for Nexus Platform (FPGA-TN-02099) for the bitstream size of each device.

2.1. Single Bitstream Boot Mode

The minimum SPI Flash density needs to be larger than the single bitstream size.

Example:

- Device: LFD2NX-9
- Scenario: MAX LRAM, MAX EBR
- Bitstream size: 4.722 Mb
- Minimum SPI Flash density: 8 Mb

2.2. Dual Boot Mode

The total bitstream size = 2 * single bitstream size + 256 byte (backup Jump command).

Example:

- Device: LIFCL-33
- Scenario: MAX LRAM, No EBR
- Total bitstream size: 2 * 7.15 Mb + 256*8b = 14.3 Mb
- Minimum SPI Flash density: 16 Mb

2.3. Ping-Pong Boot Mode

The total bitstream size = 2 * single bitstream size + 65536 byte (Jump Table) + 256 byte (backup Jump command).

Example:

- Device: LFD2NX-65
- Scenario: MAX LRAM, MAX EBR
- Total bitstream size: 2 * 14.543 Mb + 65536*8b + 256*8b = 29.59 Mb
- Minimum SPI Flash density: 32 Mb

2.4. Multi-Boot Mode

The total bitstream size = n * single bitstream size + 65536 byte (Jump Table) + 256 byte (backup Jump command).

Where n = total number of bitstream, minimum 3 and up to 6.

Example:

- Device: LFCPNX-100
- Scenario: MAX LRAM, MAX EBR
- Number of bitstream: 6
- Total bitstream size: 6 * 22.333 Mb + 65536*8b + 256*8b = 134.5 Mb
- Minimum SPI Flash density: 256 Mb

3. Dual Boot Mode

The Nexus Device Dual Boot mode supports booting from two configuration patterns that reside in an external SPI Flash device (internal flash for MachXO5-NX family). One pattern is designated as the Primary pattern, and the second pattern is designated as the Golden pattern. When the device boots up, it attempts to boot from the Primary pattern. If loading of the Primary pattern fails, the device boots from the Golden pattern.

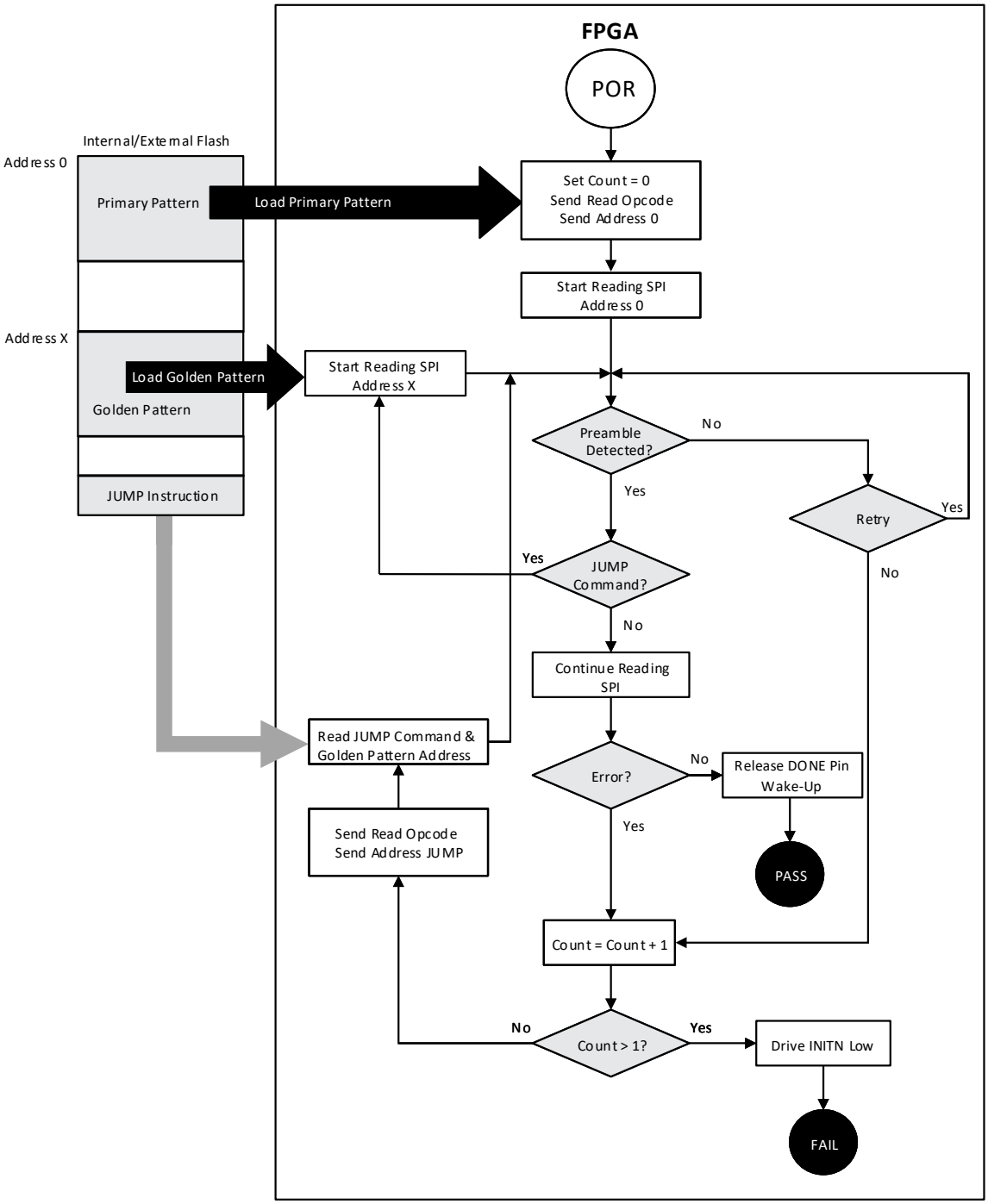


Figure 3.1. Nexus Device Dual Boot Flow Diagram

3.1. Description of the Nexus Device Dual Boot Flow Diagram

This flow is triggered either by power cycle, the PROGRAMN pin being toggled, or by the REFRESH instruction being received.

When the Dual Boot mode is selected, in addition to the standard CRC check, a time-out check is performed while reading the Primary pattern, the Golden pattern, and the JUMP command.

- Time-Out Check – the device searches for the preamble code 0xBDCD (0xBDB3 when Byte Wide Bit Mirror is enabled) from the Primary Pattern as part of the configuration protocol. The number of retries the device attempts is defined in Control Register 1 [3:2] (Table 3.1).
- Data Corruption Check – After the detection of the preamble code, the CRC engine is turned on to detect whether or not the bitstream is corrupted. This determines whether the Flash device has a corrupted Primary pattern or Golden pattern due to Flash program disruption or data loss.

Table 3.1. Control Register 1 [3:2] – Master Preamble Timer Retry Value

	Bit 3	Bit 2
No retry	0	0
Retry 1 time	0	1
Retry 3 times	1	0
Reserved	1	1

If the Primary pattern fails one of the two checks above, the device knows that the Primary pattern is not valid. It drives the INITN pin LOW briefly to indicate an error and resets the configuration engine. After clearing all the SRAM fuses, it drives the INITN pin HIGH, and reads the JUMP command that directs it to the location of the Golden pattern in the Flash.

If the JUMP command is corrupted, it also causes a configuration failure. It is important to note that a corrupted Golden pattern is not the only possible cause for Dual Boot configuration failure.

If the JUMP command is valid, the device stops the SPI clock, drives the INITN pin LOW, resets the configuration engine, and performs a Clear All operation. The device then drives the INITN pin HIGH after the completion of the Clear All action, restarts the SPI clock, and reads the Golden pattern from the Flash address designated in the JUMP command.

The device performs the same time-out check and CRC check when searching for the preamble code from the Golden pattern. If the Golden pattern is also corrupted, configuration fails. The device stops driving the SPI clock, and the INITN pin is driven LOW.

4. Ping-Pong Boot Mode

The Nexus Device Ping-Pong Boot mode supports booting from two configuration patterns that reside in an external SPI Flash device (internal flash for MachXO5-NX family). One pattern is designated as the Primary pattern and the second pattern is designated as the Secondary pattern. The device boots from the pattern assigned in the Jump table. The Jump table allows the device to boot from either the Primary pattern or the Secondary pattern without changing the physical location of the patterns within the Flash. Only the Jump table needs to be updated to change the boot pattern. The other pattern, by default, becomes the Golden pattern.

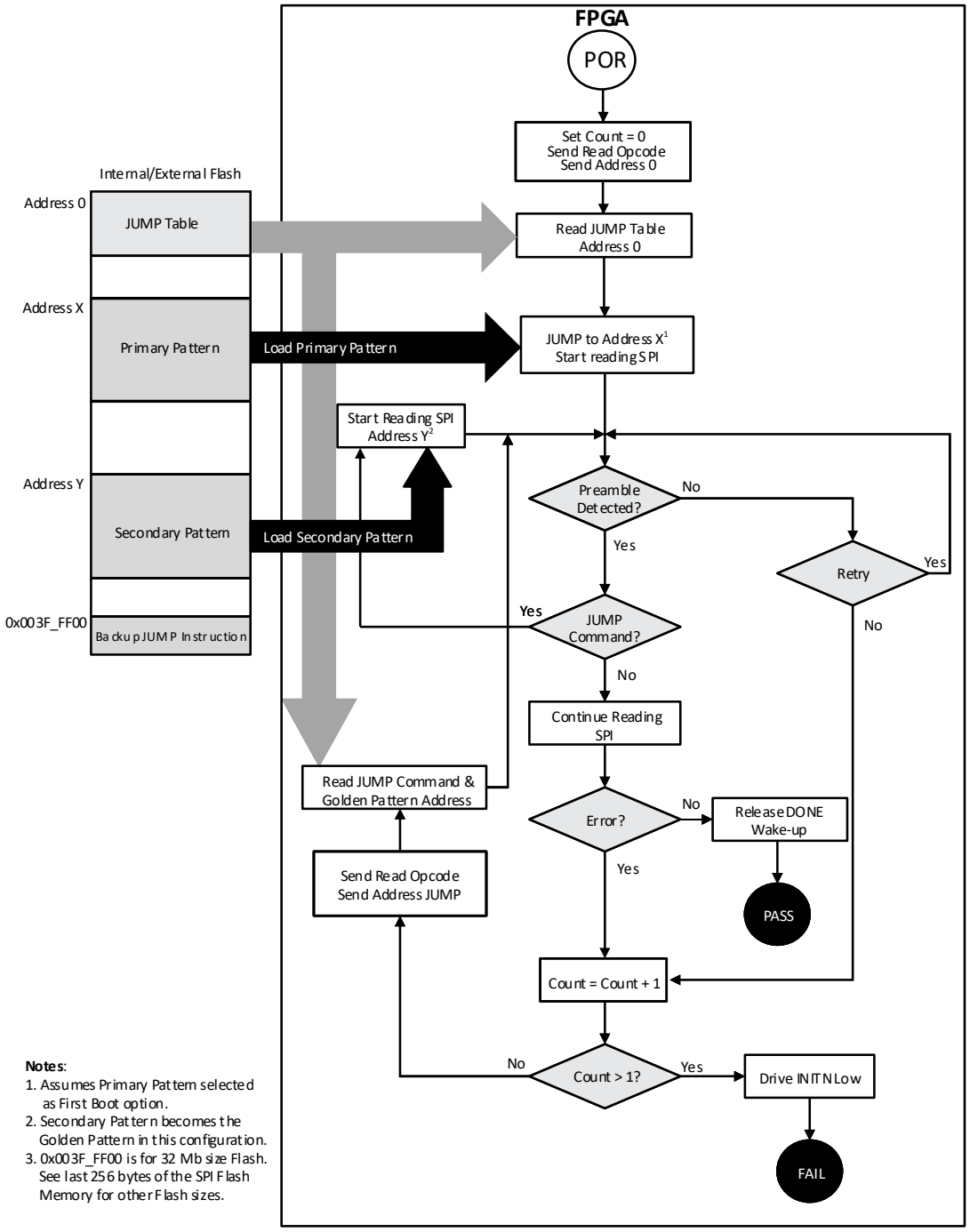


Figure 4.1. Ping-Pong Boot Flow Diagram

4.1. Description of the Ping-Pong Boot Flow Diagram

This flow is triggered either by power cycle, the PROGRAMN pin being toggled, or by the REFRESH instruction being received.

When Ping-Pong Boot mode is selected, in addition to the standard CRC checking, a time-out check is performed while reading the Primary pattern, the Secondary pattern, and the JUMP command.

- Time-Out Check – the device searches for the preamble code 0xBDCD (0xBDB3 when Byte Wide Bit Mirror is enabled) from the pattern designated as “First Boot” selection as part of the configuration protocol. The number of retries the device attempts is defined in Control Register 1 [3:2] (Table 4.1).
- Data Corruption Check – After the detection of the preamble code, the CRC engine is turned on to detect whether the bitstream is corrupted. This determines whether the Flash device has a corrupted Primary or Secondary Pattern due to Flash program disruption or data loss.

Table 4.1. Control Register 1 [3:2] – Master Preamble Timer Retry Value

	Bit 3	Bit 2
No retry	0	0
Retry 1 time	0	1
Retry 3 times	1	0
Reserved	1	1

If the “First Boot” pattern fails one of the two checks above, the device knows that the pattern is not valid. It drives the INITN pin LOW briefly to indicate an error and resets the configuration engine. After clearing all the SRAM fuses, it drives the INITN pin HIGH, and reads the JUMP command that directs it to the location of the other pattern, acting as the Golden pattern, in the Flash.

If the JUMP command is corrupted, it also causes a configuration failure. It is important to note that a corrupted Golden pattern is not the only possible cause for Ping-ping mode configuration failure.

If the JUMP command is valid, the device stops the SPI clock, drives the INITN pin LOW, resets the configuration engine, and performs a Clear All operation. The device then drives the INITN pin HIGH after the completion of the Clear All action, restarts the SPI clock, and reads the Golden pattern from the Flash address designated in the JUMP command.

The device performs the same time-out check and the CRC check when searching for the preamble code from the Golden Pattern. If the Golden Pattern is also corrupted, configuration fails, stops driving the SPI clock, and the INITN pin is driven LOW.

4.2. JUMP Table File Structure

In Ping-Pong Boot mode, the Nexus device determines the configuration image to load based on a JUMP table, rather than a fixed Primary or Secondary image location. The JUMP table specifies:

- the image selected as First Boot;
- the address of the fallback image, which acts as the Golden image.

This mechanism allows switching between two images without changing their physical locations in the internal or external Flash.

In Lattice Radiant software, the JUMP table is generated as a separate *_header.mcs file. This file contains configuration commands only and does not include any bitstream data.

The JUMP table itself is not fixed to 65536 bytes in logical size. The apparent 64 KB size referenced elsewhere in this document reflects 64 KB flash sector erase alignment (0x10000).

The actual JUMP table content is much smaller, and for Ping-Pong Boot mode it can be as small as 76 bytes, including required padding.

Table 4.2 describes the logical content of the JUMP table for Ping-Pong Boot mode. Address offsets are relative to the start of the JUMP table region in Flash. Figure 4.2 shows the example JUMP table in decoded binary format.

Table 4.2. JUMP Table File Structure[1]

Description	Value	Address offset	Size (bytes)
LSCC Signature	0x4C53 4343	0x00	4
Padding Bytes	0xFF (all bytes)	0x04	16
Preamble	0xFFFF BDB3	0x14	4
Padding Bytes	0xFF (all bytes)	0x18	20
LSC_PROG_SEC_BOOT command and Operand	0x7F00 0000	0x2C	4
Secondary Image Boot Address	Secondary image start address (hex)	0x30	4
JUMP Command and Operand	0x7E followed by 3-byte operand[2]	0x34	4
Primary Image Boot Address	Primary image start address (hex)	0x38	4
Padding Bytes	0xFF (all bytes)	0x3C	16

Notes:

1. The JUMP Table .mcs file is stored in Intel Hex format. The bit order within each byte in the .mcs file is swapped relative to the actual data programmed into Flash. This conversion is handled automatically by the programming tool and should be considered when manually inspecting the file contents.
2. JUMP command is sharing the operand definition with LSC_REFRESH command, refer to FPGA-TN-02099 sysCONFIG User Guide for Nexus Platform for detail.

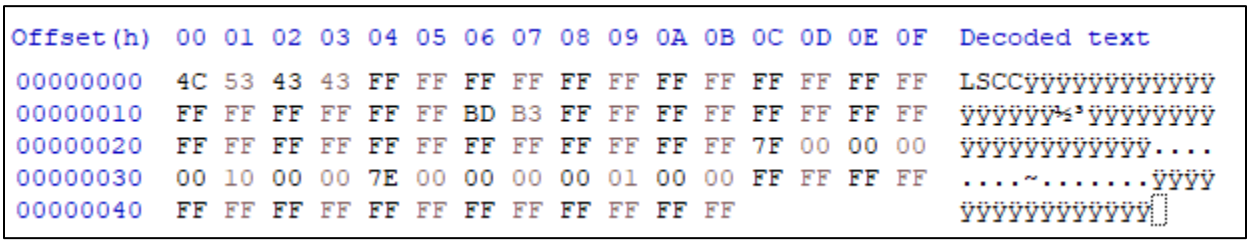


Figure 4.2. Example JUMP Table in Binary Format

4.3. Swapping Boot Image Order

In the event that you need to swap the image boot order, you can reconstruct the JUMP table by configuring the secondary image and primary image addresses, erase JUMP table with 64 KB or lower granularity sector erase, then reprogram with the new JUMP table.

4.4. Safe Primary Image Update Procedure

In Ping-Pong Boot mode, the device relies on a valid image preamble to initiate configuration. As described in the Ping-pong Boot Limitation section, if the 32-bit preamble of the Primary image is corrupted, the configuration engine may not reliably load the Secondary image. In this condition, both BSE Error Code and BSE Error 1 Code indicate a preamble error, and device recovery requires reprogramming through the JTAG or SSPI port.

This behaviour introduces additional risk during Primary image updates, particularly for remote or field upgrade scenarios where power interruption or incomplete Flash programming may occur.

To minimise the risk of device recovery due to Primary image corruption, the following update sequence is recommended when replacing the Primary image in Ping-Pong Boot mode:

1. Switch First Boot to the Known-Good Image
   Reconstruct the JUMP table such that the Secondary image is selected as the First Boot image, as described in the Swapping Boot Image Order section.

Program only the updated JUMP table to Flash and optionally reconfigure or power-cycle the device to confirm successful boot from the Secondary image.

2. Update the Primary Image
   Program the new Primary image to its assigned Flash address without modifying the JUMP table.
   If the update operation is interrupted or the Primary image is corrupted, the device continues to boot from the Secondary image selected in Step 1.
3. Restore Primary Image as First Boot (Optional)
   After validating correct operation of the updated Primary image, the JUMP table may be reconstructed again to select the Primary image as First Boot.
   Program only the updated JUMP table to complete the update process.

5. Multi-Boot Mode

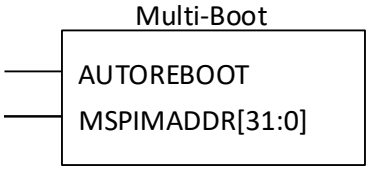
The Nexus device Multi-Boot supports booting from up to six patterns that reside in an external SPI Flash device (up to three patterns for MachXO5-NX internal flash memory). The patterns include a Primary pattern, a Golden pattern, and up to four Alternate patterns, designated as Alternate pattern 1 to Alternate pattern 4.

The device boots by loading the Primary pattern from the internal or external Flash, depending on the device family. If loading of the Primary pattern fails, the device attempts to load the Golden pattern. In static mode, when a reprogramming of the bitstream is triggered through the toggling of the PROGRAMN pin or receiving a REFRESH command, always Alternate pattern 1 is loaded. Subsequent PROGRAMN/REFRESH event loads the next pattern defined in the Multi-Boot configuration. The bitstream pattern sequence, target address of the Golden pattern, and target addresses of the Alternate patterns are defined during the Multi-Boot configuration process in the Lattice Radiant™ Deployment Tool. The Multi-Boot flow is similar to the Dual Boot flow (Figure 3.1). Each PROGRAMN/REFRESH event becomes a Dual Boot event with the addresses being different depending on the pattern being loaded.

By using MULTIBOOT primitive, it allows the device to operate in dynamic mode. It allows the system to dynamically switch to any of the alternate pattern after the device boots up from the Primary pattern while still being protected by a Golden pattern.

5.1. MULTIBOOT Primitive

Wrapper for Interface for multi-boot functionality



Input Ports

Name	Range	Description
AUTOREBOOT		
MSPIMADDR	31:0	

Parameters

Name	Values	Description
MSPIADDR	0'b00000000000000000000000000000000 *(default)*	
SOURCESEL	"DIS" *(default)* "EN"	

Example of code to instantiate and enable the MULTIBOOT primitive.

```
MULTIBOOT
#(
  .MSPIADDR(),               //not necessary to connect
  .SOURCESEL("EN")
) u_multiboot
(
  .AUTOREBOOT(1'b0),          //no need to set this (tie to 1'b0)
  .MSPIMADDR(boot_addr)
);
```

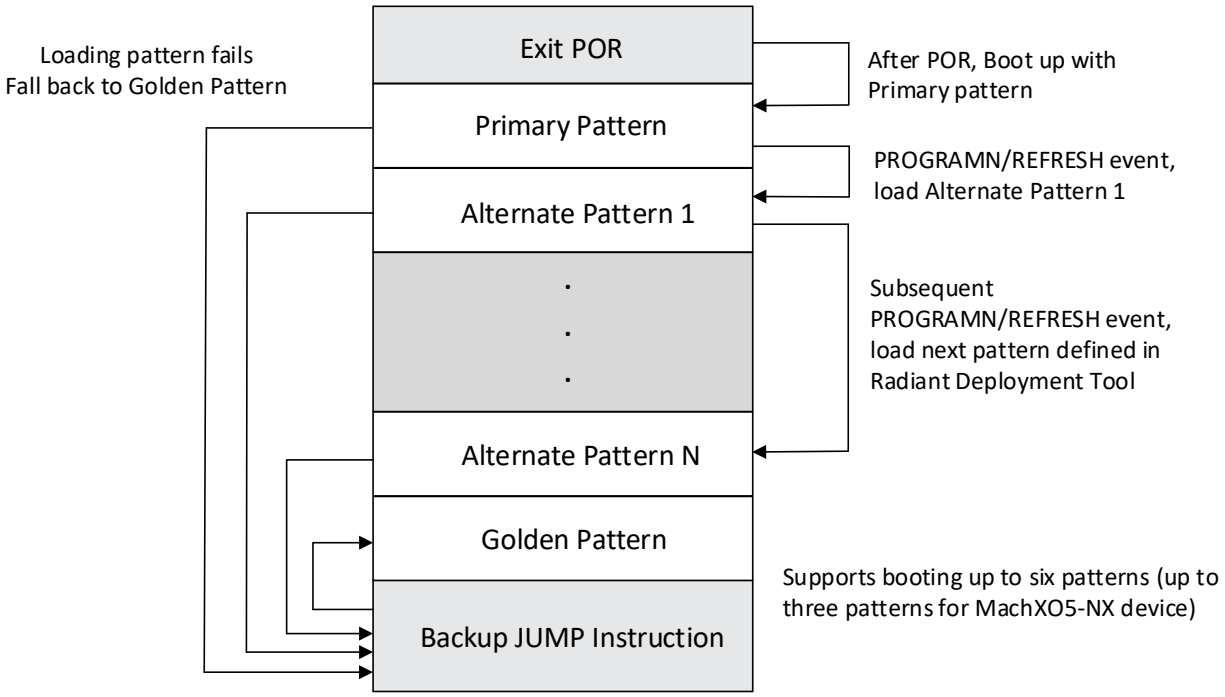
Notes:

- The AutoReboot port is an unused input port and is recommended to tie it to 0.
- Once this primitive is instantiated, the use model of loading the next pattern defined in the multi-boot configuration using Radiant Deployment Tool does not work.
- MSPIMADDR is a 32-bit wide input for you to supply the boot address.

5.2. Booting Flow without MULTIBOOT Primitive

5.2.1. Drawback

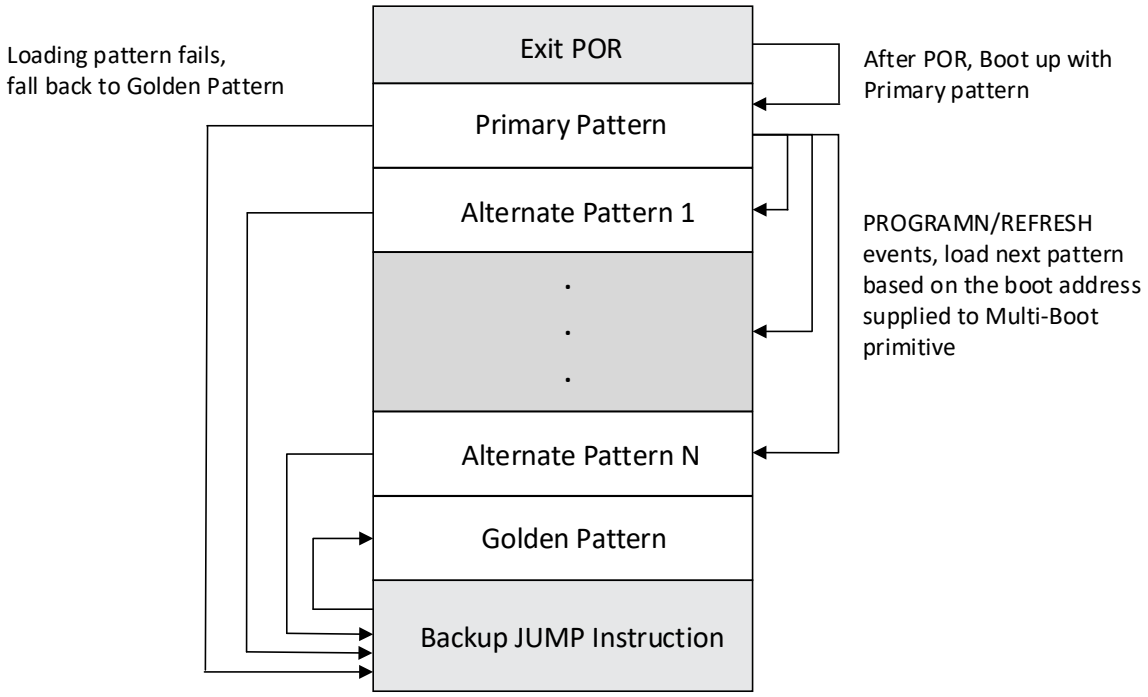
Less flexibility to boot to the desired image freely.



5.3. Booting Flow with MULTIBOOT Primitive

5.3.1. Advantage

Free to load any pattern stored in external/internal SPI flash in any sequence.



5.3.2. Implementation of Multi-Boot Feature Using MULTIBOOT Primitive

In a user design, besides the MULTIBOOT primitive, OSC IP and CONFIG_LMMI primitive are required and should be connected as shown in Figure 5.1 to implement the Multi-Boot feature.

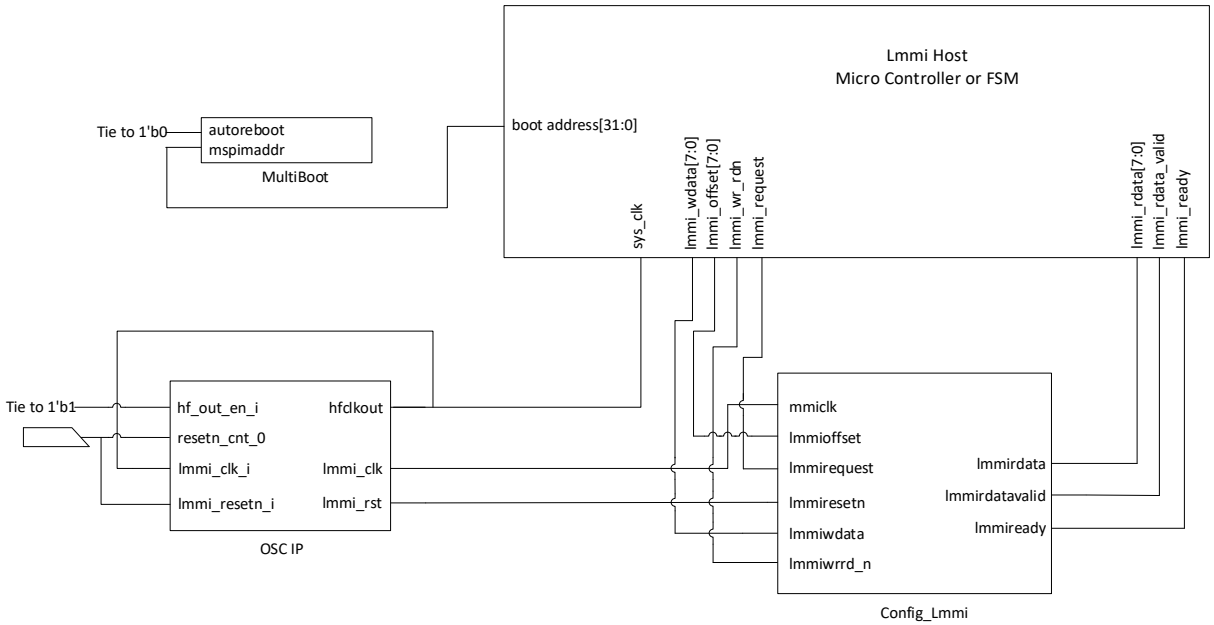


Figure 5.1. MULTIBOOT Primitive, OSC IP, CONFIG_LMMI Primitive and Lmmi Host Connection

- MULTIBOOT primitive: This primitive is a wrapper for the interface to perform the multi-boot functionality. It enables the booting to load the desired Alternate pattern through sending the Refresh command to CONFIG_LMMI block.

- OSC IP: This IP is an oscillator module. It generates clock sources, sys_clk and lmmi_clk, to the Lmmi host controller and CONFIG_LMMI primitive. Refer to the OSC Module - Lattice Radiant Software User Guide for more information.
- CONFIG_LMMI: Lattice Memory Mapped Interface (LMMI) interfaces to the configuration block. Refer to the Config_Lmmi page in Lattice Radiant Software User Guide for more information.
- Lmmi host controller: This controller implements a state machine controller to send the necessary commands to the CONFIG_LMMI block, and sends the boot address to the MULTIBOOT primitive to boot the desired Alternate pattern stored in the internal or external SPI flash. The controller performs the following sequences. Refer to flow diagram below (Figure 5.2) for more details.
  - Start sending 32-bit boot address to MULTIBOOT primitive.
  - Execute ISC_ENABLE_X – Similar to ISC_ENABLE, this command puts the device into the transparent mode. Executing this command is essential to enable the device to execute the next command, the LSC_PROG_CNTRL0 command.
  - Execute LSC_PROG_CNTRL0 – Set the SPIM bit in Control Register 0 to 1. When this bit is set to 1, and once the REFRESH command is executed, it enables the device to boot from the image stored in the external SPI flash according to the boot address sent to MULTIBOOT primitive. Else, the device boots from address zero.

    Note: An alternative method to set the SPIM bit is by modifying the .bit file. Refer to the Setting SPIM Bit Using Programming File Utility Tool section for more information. Use this method only if you encounter difficulties implementing the LSC_PROG_CNTRL0 command to set the SPIM bit. Otherwise, setting the SPIM bit through the LMMI host is recommended to avoid relying on setting the SPIM bit in the .bit file.
  - Execute ISC_DISABLE – Exit Transparent mode.
  - Execute LSC_REFRESH – Equivalent to toggling the PROGRAMN pin. Once this command is executed, the device starts to load the desired alternate pattern from the external SPI flash according to the boot address sent to MULTIBOOT primitive. If loading image fails, the device falls back to load the Golden pattern.

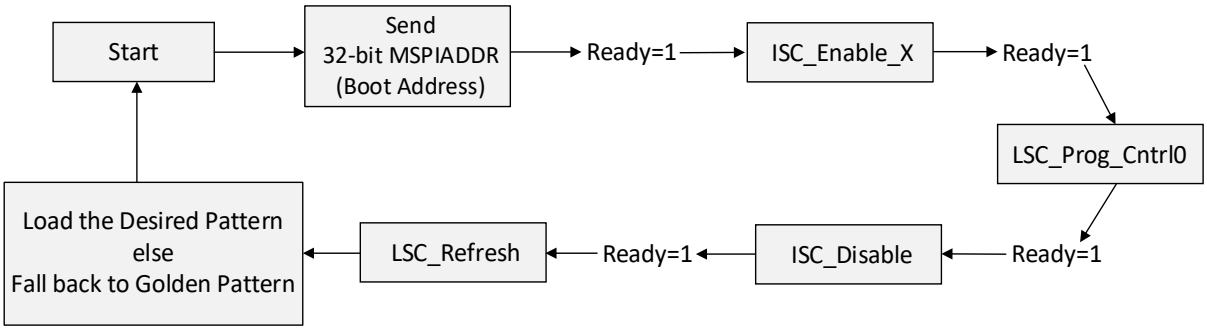


Figure 5.2. Implementation of Multi-Boot Feature Using MULTIBOOT Primitive Flow

Refer to the Lattice Nexus Device Multi-Boot Reference Design (FPGA-RD-02294) for details of the reference design.

5.3.2.1. Setting SPIM Bit Using Programming File Utility Tool

If you need to update or replace an alternate pattern in the configuration memory but the LMMI host cannot be used to set the SPIM bit to 1 during the multi-boot flow, the following is an alternative method to set the SPIM bit to 1 in the .bit file using the Radiant software Programming File Utility tool:

4. Launch Radiant Programmer and select **Tools > Programming File Utility**.

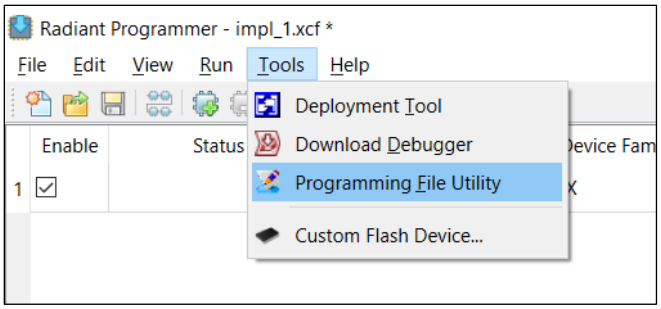


Figure 5.3. Launching the Programming File Utility Tool

5. In the Programming File Utility, select **Tools > Control Register0 Editor…**.

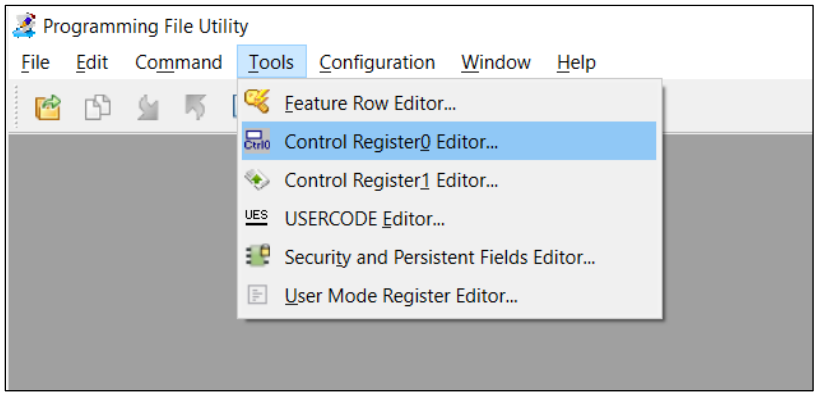


Figure 5.4. Opening the Control Register 0 Editor Window

6. In the Control Register 0 window, click the **…** button, navigate to and select the .bit file, then click **Open.**

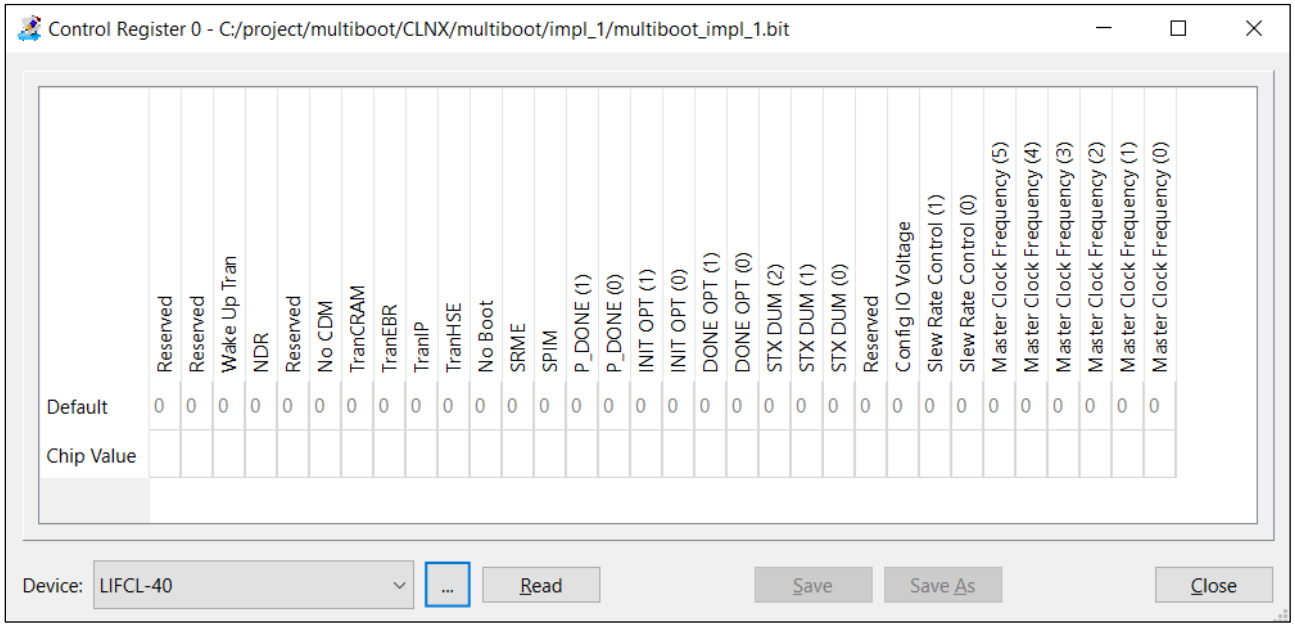


Figure 5.5. Control Register 0 Window

7. Click **Read** to read the Control Register 0 bit settings. By default, the SPIM bit is set to 0 as shown in the following figure.

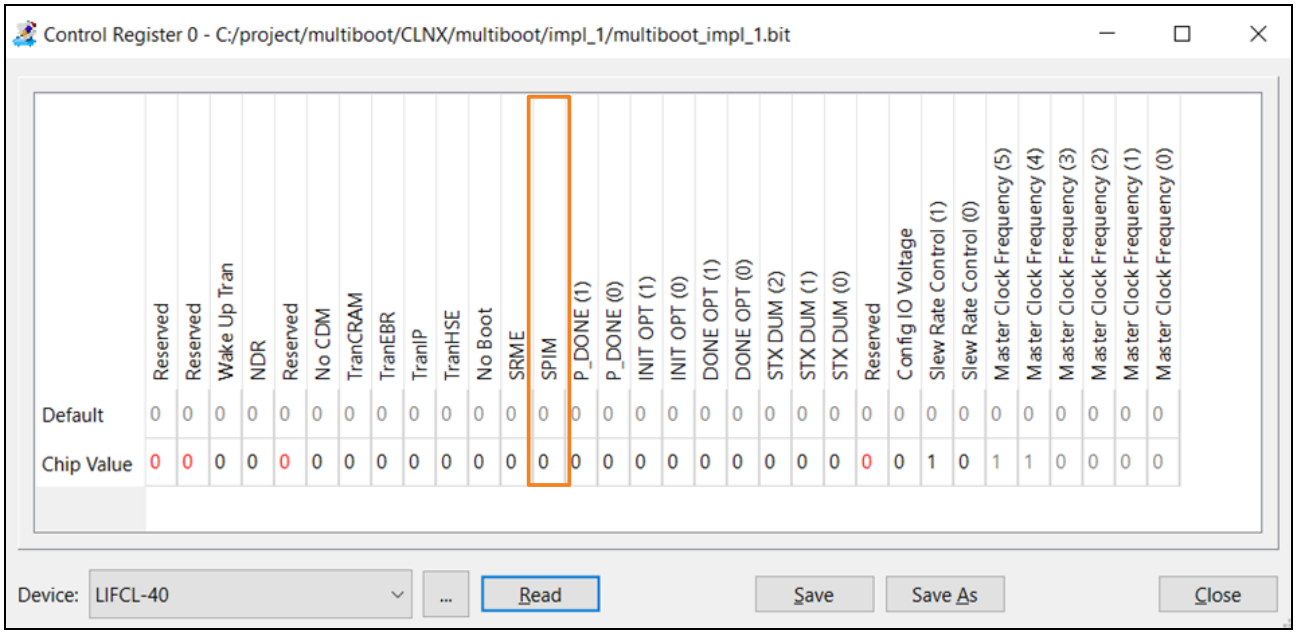


Figure 5.6. Default SPIM Bit Setting

8. Click the chip value associated with the SPIM bit and set it to 1.
9. Click **Save** to save the settings in the selected .bit file or **Save As** to save the settings in a new .bit file. The write to file successful dialog box appears.

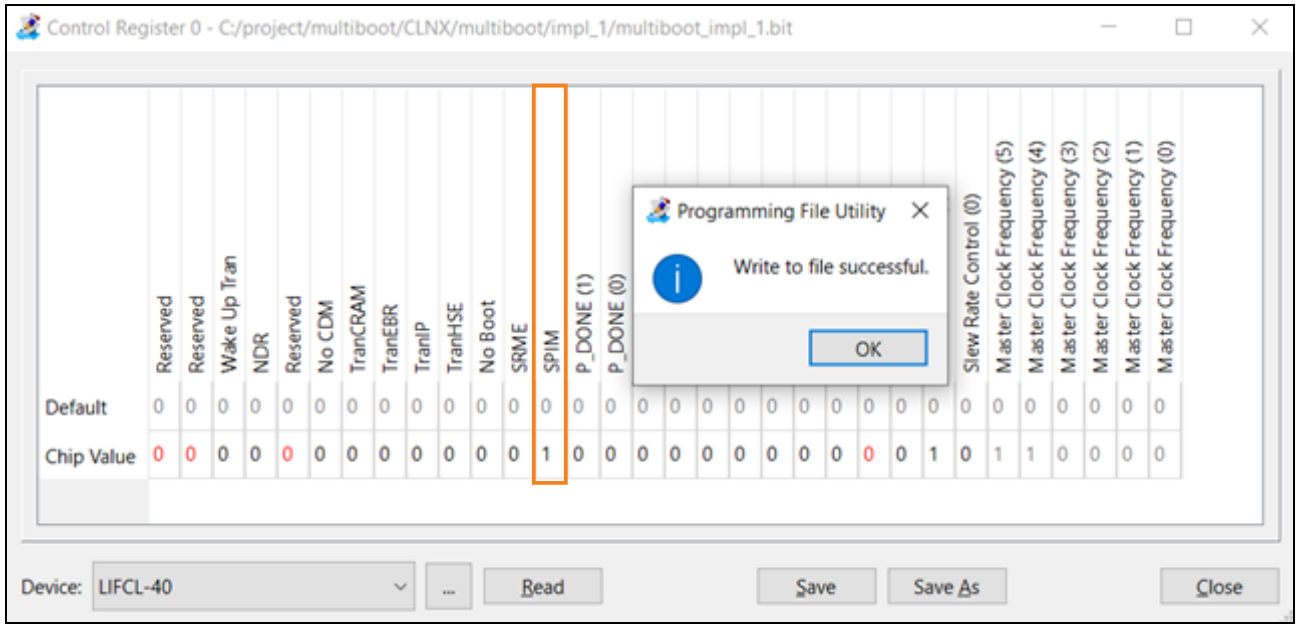


Figure 5.7. Saving Control Register 0 Bit Settings

10. Click **OK** to continue.
11. Read the modified .bit file to confirm that the SPIM bit is set to 1.
12. Proceed to program the configuration memory with the modified .bit file to update the alternate pattern.

6. Creating a PROM File

The various boot features on the Nexus devices are simple, requiring only one external SPI Flash device (MachXO5-NX uses internal flash), and flexible, due to the intelligent use of the JUMP command or table. The Lattice Radiant software provides a turn-key solution to implement this feature. The Lattice Deployment Tool, part of Lattice Radiant Software, merges the different patterns and the JUMP command and table into one PROM hex file with the .mcs file extension. The PROM hex file can later be programmed into the internal or external Flash device using Radiant Programmer or a third-party programmer.

6.1. Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File

The following steps provide the procedure for generating a Dual Boot PROM hex file using the Radiant Deployment Tool.

1. Generate the Golden and Primary bitstream files in Lattice Radiant Software.
2. Invoke **Lattice Radiant Deployment Tool** from **Start > Lattice Radiant Programmer > Deployment Tool**.
3. In the **Radiant Deployment Tool** window, select **External Memory** as the **Function Type** and select **Dual Boot** as the **Output File Type** (Figure 6.1).
   - Note that the **External Memory** selection is also applicable to MachXO5-NX device that uses internal flash memory.
4. Select **OK**.

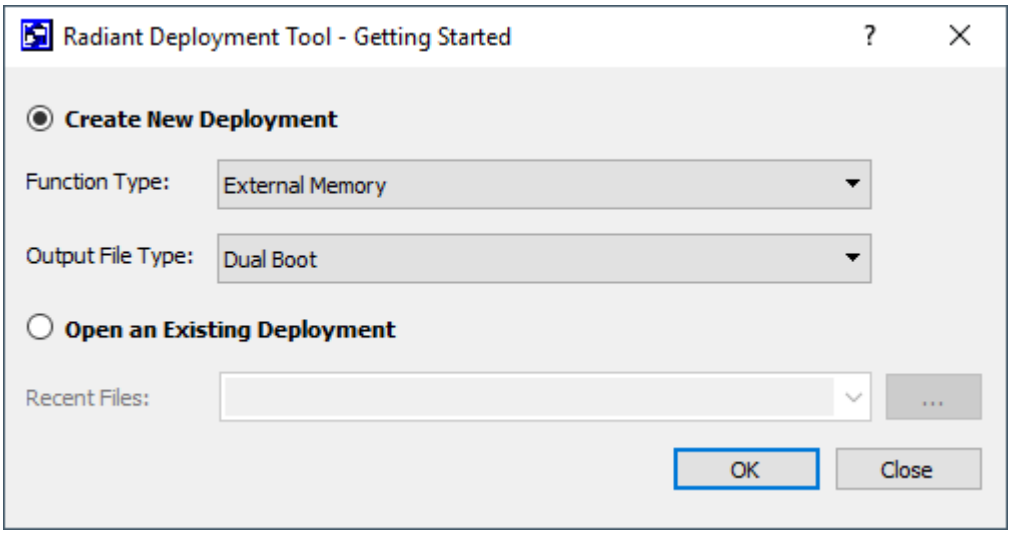


Figure 6.1. Creating New Deployment for Dual Boot PROM Hex File

Step 1 of 4: Select Input File(s) window (Figure 6.2)

- Click the **File Name** fields to browse and select the two bitstream files to be used to create the PROM hex file.
- The **Device Family** and **Device** fields auto populate based on the bitstream files selected.
- Select **Next**.

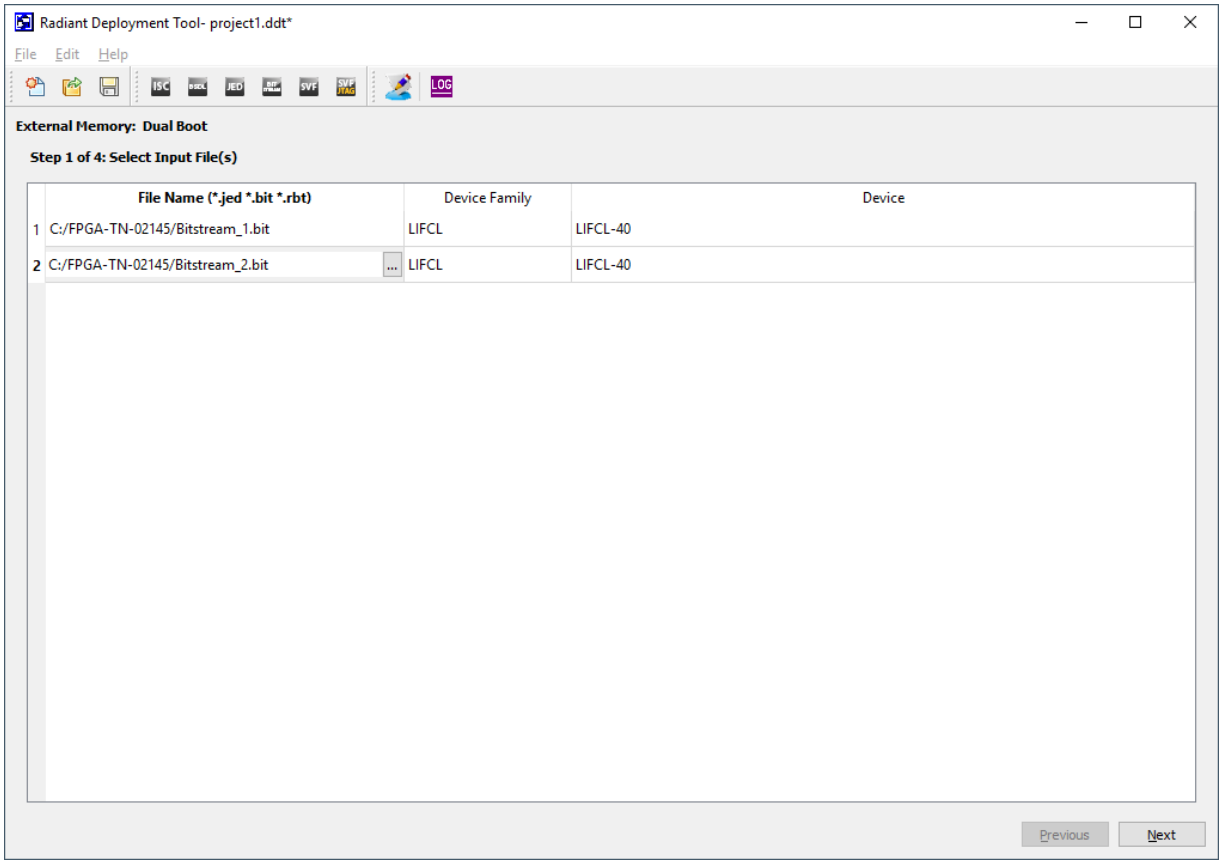


Figure 6.2. Select Input Files Window

Step 2 of 4: Dual Boot Options window (Figure 6.3)

- Select the **Output Format** (Intel Hex, Motorola Hex, or Extended Tektronix Hex).
- Select the **SPI Flash Size** (4, 8, 16, 32, 64, 128, 256, 512, or 1024 Mb).
- Select **SPI Flash Read Mode** (Standard Read, Fast Read, Dual I/O SPI Flash Read or Quad I/O SPI Flash Read).
- The **Radiant Deployment Tool** automatically assigns the bitstream files selected in Step 1 to be used for the Golden pattern and Primary pattern.
  - Change the pattern options by clicking on the drop-down menu of the respective fields.
  - The Starting Address of the Golden pattern is automatically assigned.
  - Change the Starting Address of the Golden pattern by clicking on the drop-down menu.
- Select the following options as required.
  - **Byte Wide Bit Mirror** – Flips each byte in Intel, Extended Tektronix, or Motorola hexadecimal data files.

    For example, 0xCD (b1100 1101) can become 0xB3 (b1011 0011) when this is selected. You do not need to enable this setting if you program the .mcs file using Radiant Programmer. If you are using a third-party programmer, check with your vendor to understand if the byte wide bit mirror is needed.
  - **Retain Bitstream Header** – By default, Radiant Deployment Tool replaces the bitstream header information (name, version number, and date of the file) with 0xFF values.

    Selecting this option retains the header information that was generated as the header.
- Select **Next**.

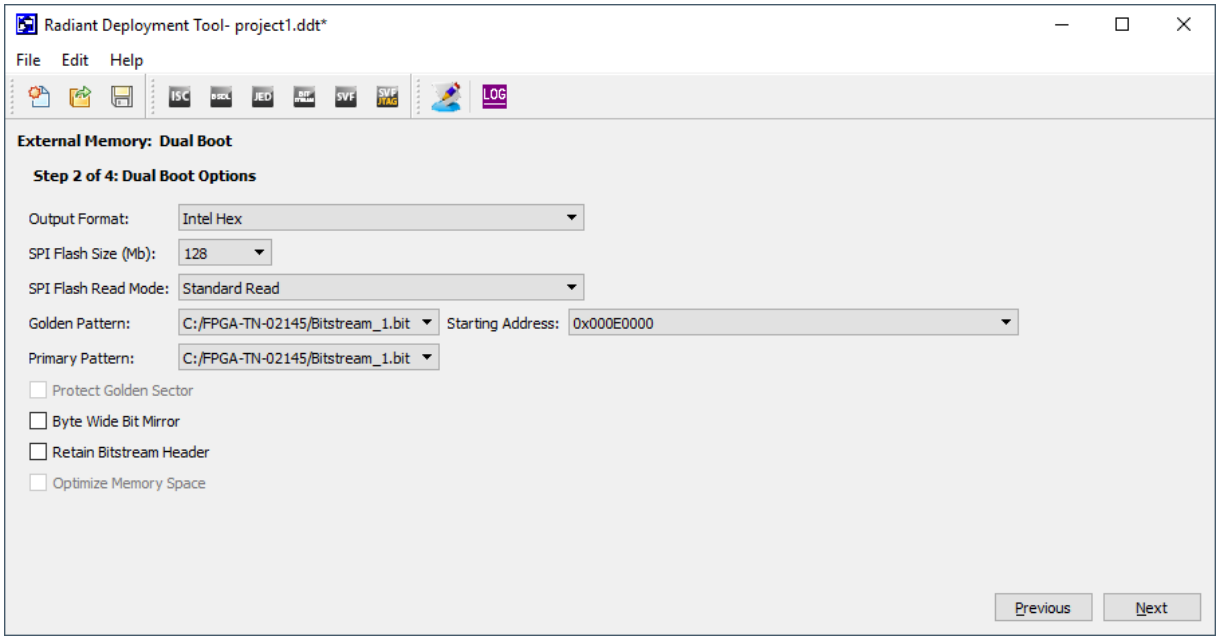


Figure 6.3. Dual Boot Options Window

Step 3 of 4: Select Output File(s) window (Figure 6.4)

- Specify the name of the output PROM hex file in the **Output File 1** field.
- Select **Next**.

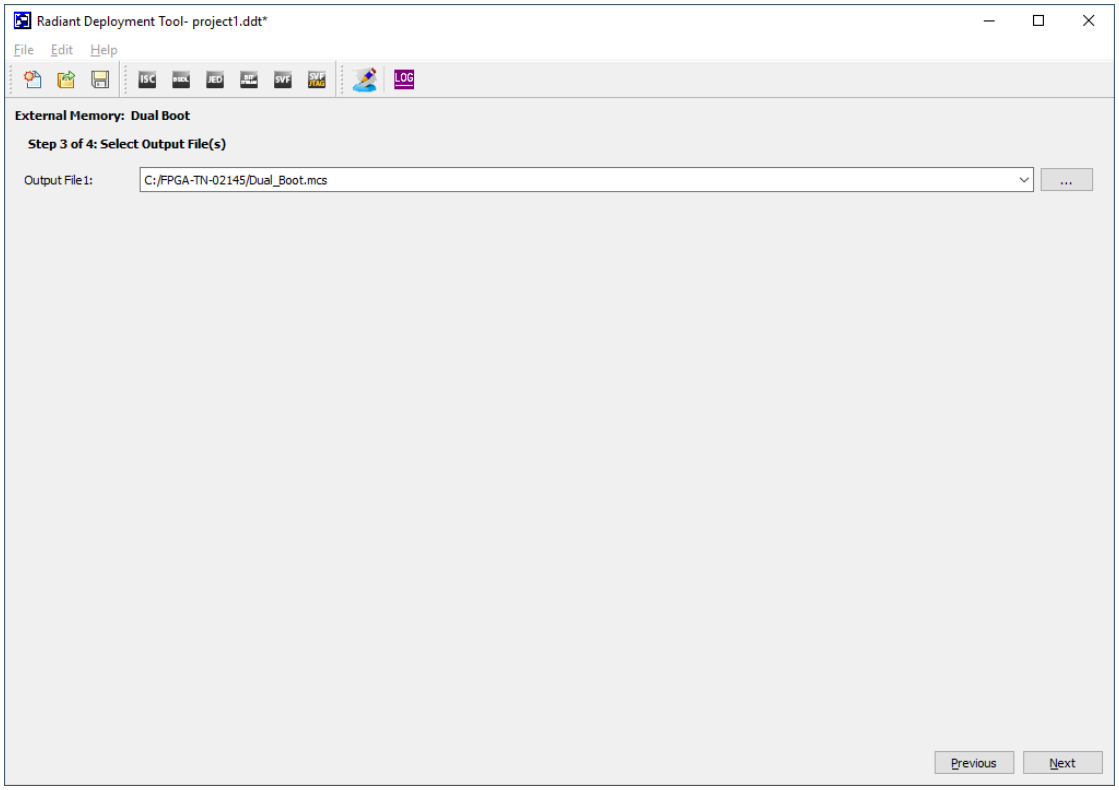


Figure 6.4. Select Output File Window

Step 4 of 4: Generate Deployment window (Figure 6.5)

- Review the summary information.
- If everything is correct, click the **Generate** button.
- The **Generate Deployment** pane should indicate that the PROM file was generated successfully.
- Save the deployment settings by selecting **File > Save**.
- To exit, select **File > Exit**.

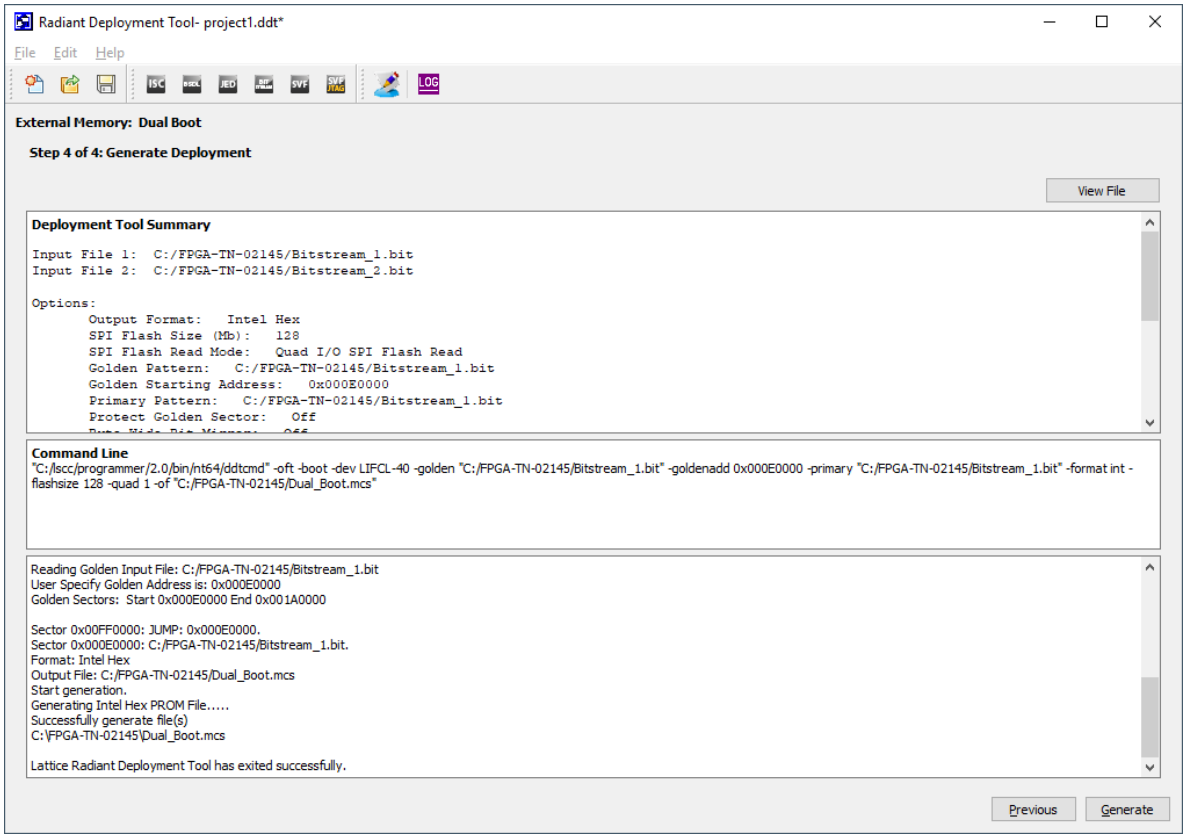


Figure 6.5. Generate Deployment Window

6.2. Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File

The following steps provide the procedure for generating a Ping-Pong boot PROM hex file using the Radiant Deployment Tool.

1. Generate the Primary and Secondary bitstream files in Lattice Radiant software.
2. Invoke **Lattice Radiant Deployment Tool** from **Start > Lattice Radiant Programmer > Deployment Tool**.
3. In the **Radiant Deployment Tool** window, select **External Memory** as the **Function Type** and select **Ping-Pong Boot** as the **Output File Type** (Figure 6.6).
   - Note that the **External Memory** selection is also applicable to MachXO5-NX device that uses internal flash memory.
4. Select **OK**.

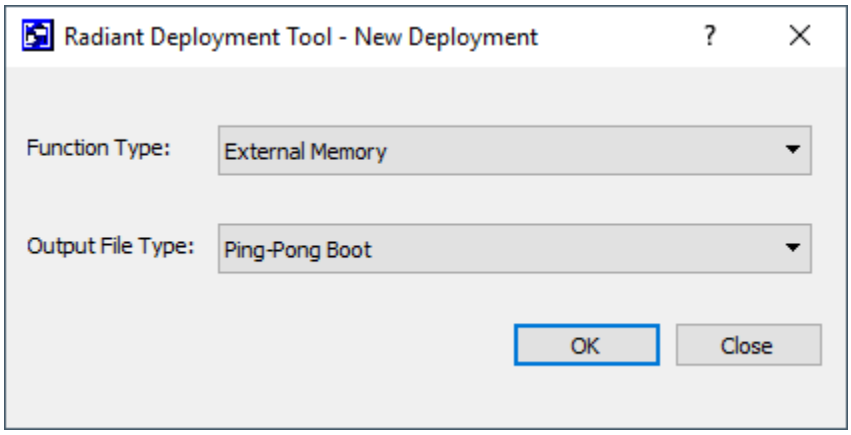


Figure 6.6. Creating New Deployment for Ping-Pong Boot PROM Hex File

Step 1 of 4: Select Input File(s) window (Figure 6.7)

- Click the **File Name** fields to browse and select the two bitstream files to be used to create the PROM hex file.
- The **Device Family** and **Device** fields auto-populate based on the bitstream files.
- Select **Next**.

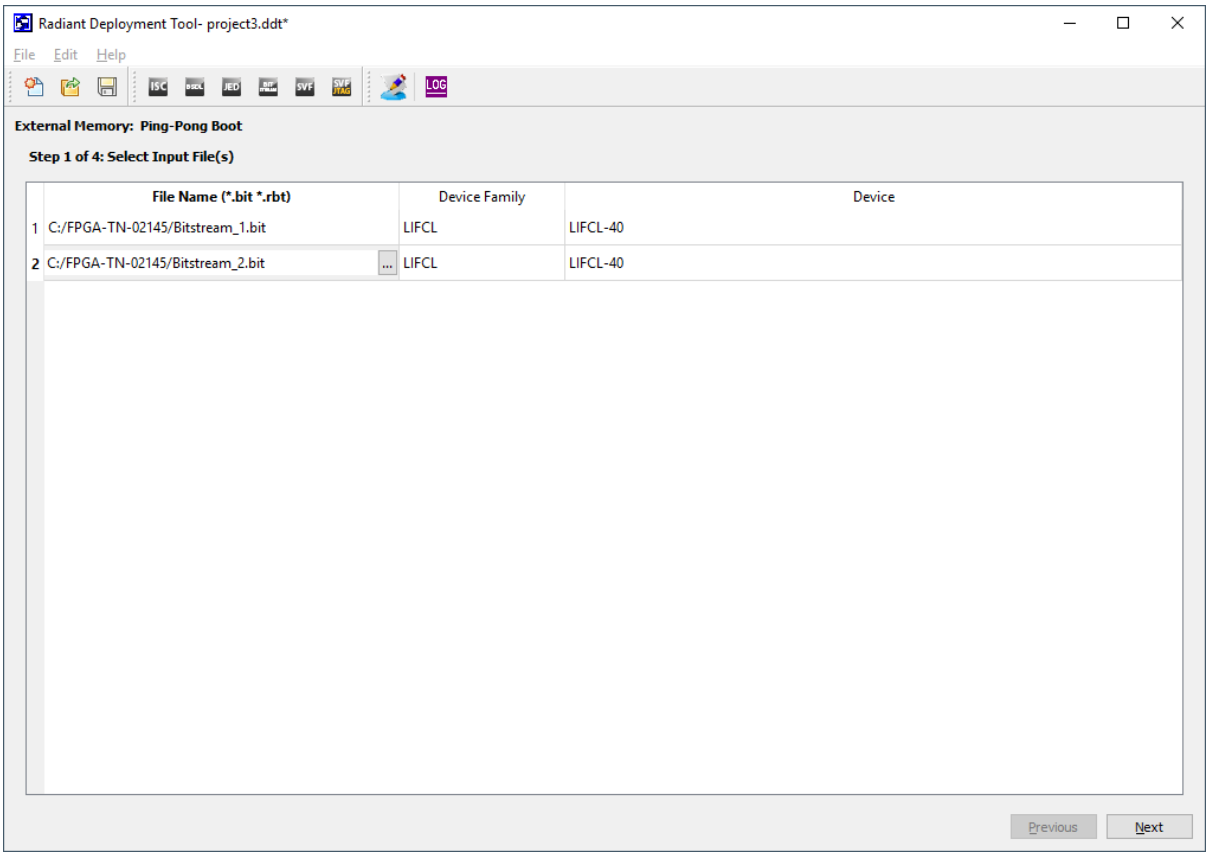


Figure 6.7. Select Input Files Window

Step 2 of 4: Ping-Pong Boot Options window (Figure 6.8)

- Select the **Output Format** (Intel Hex, Motorola Hex, or Extended Tektronix Hex).
- Select the **SPI Flash Size** (4, 8, 16, 32, 64, 128, 256,512, or 1024 Mb).
- Select **SPI Flash Read Mode** (Standard Read, Fast Read, Dual I/O SPI Flash Read, or Quad I/O SPI Flash Read).
- The **Radiant Deployment Tool** automatically assigns the bitstream files selected in Step 1 to be used for Primary and Secondary Patterns.
  - Change the pattern options by clicking on the drop-down menu of the respective field.
  - The Starting Address of the Primary Pattern is automatically assigned and can be modified by clicking on the drop-down menu.
  - The Starting Address of the Secondary Pattern is automatically assigned and can be modified by clicking on the drop-down menu.
- Select the following options as required.
  - **Generate Jump Table Only** – Generates a JUMP table to select an image for booting without changing the physical location of the images in the internal or external SPI Flash.

    For example, a JUMP table file can be created to attempt to load Bitstream_2.bit file first. This new JUMP table file can be programmed to the internal or external Flash to overwrite the previous JUMP table file.

    The JUMP table is in .mcs format that carry the SPI Flash Read Mode setting, the Primary image boot address and the Secondary/Golden image boot address.
  - **Byte Wide Bit Mirror** – Flips each byte in Intel, Extended Tektronix, or Motorola hexadecimal data files.

    For example, 0xCD (b1100 1101) becomes 0xB3 (b1011 0011) when this is selected. You do not need to enable this setting if you program the .mcs file using Radiant Programmer. If you are using a third-party programmer, check with your vendor to understand if the byte wide bit mirror is needed.
  - **Retain Bitstream Header** – By default, Radiant Deployment Tool replaces the bitstream header information (name, version number and date of the file) with 0xFF values.

    Selecting this option retains the header information that was generated as the header.
- Select **Next**.

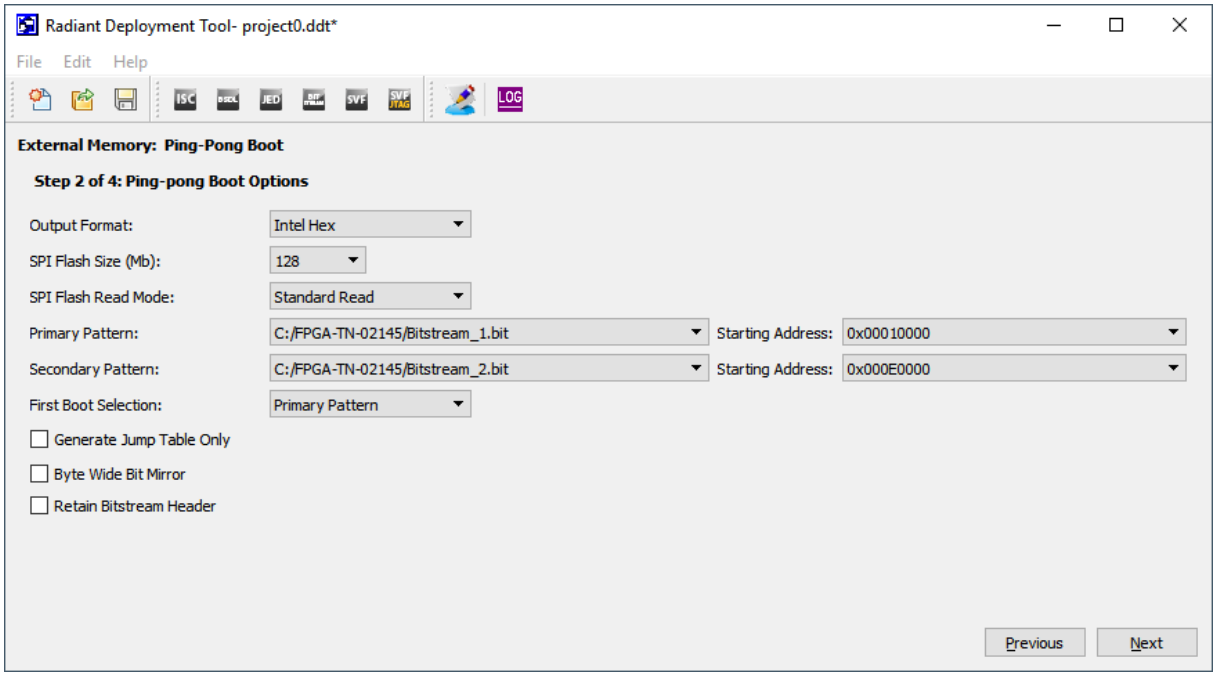


Figure 6.8. Ping-Pong Boot Options Window

Step 3 of 4: Select Output File(s) window (Figure 6.9)

- Specify the name of the output PROM hex file in the **Output File 1** field.
- Select **Next**.

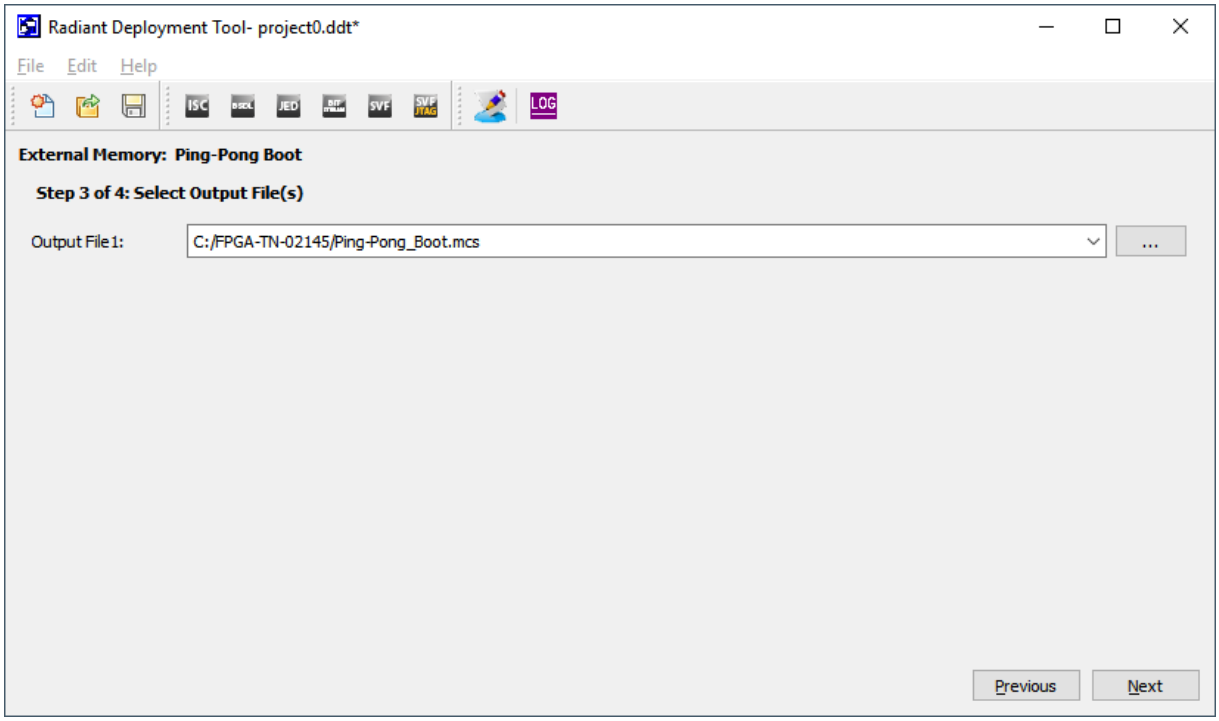


Figure 6.9. Select Output File Window

Step 4 of 4: Generate Deployment window (Figure 6.10)

- Review the summary information.
- If everything is correct, click the **Generate** button.
- The **Generate Deployment** pane should indicate the PROM file was generated successfully.
- Save the deployment settings by selecting **File > Save**.
- To exit, select **File > Exit**.

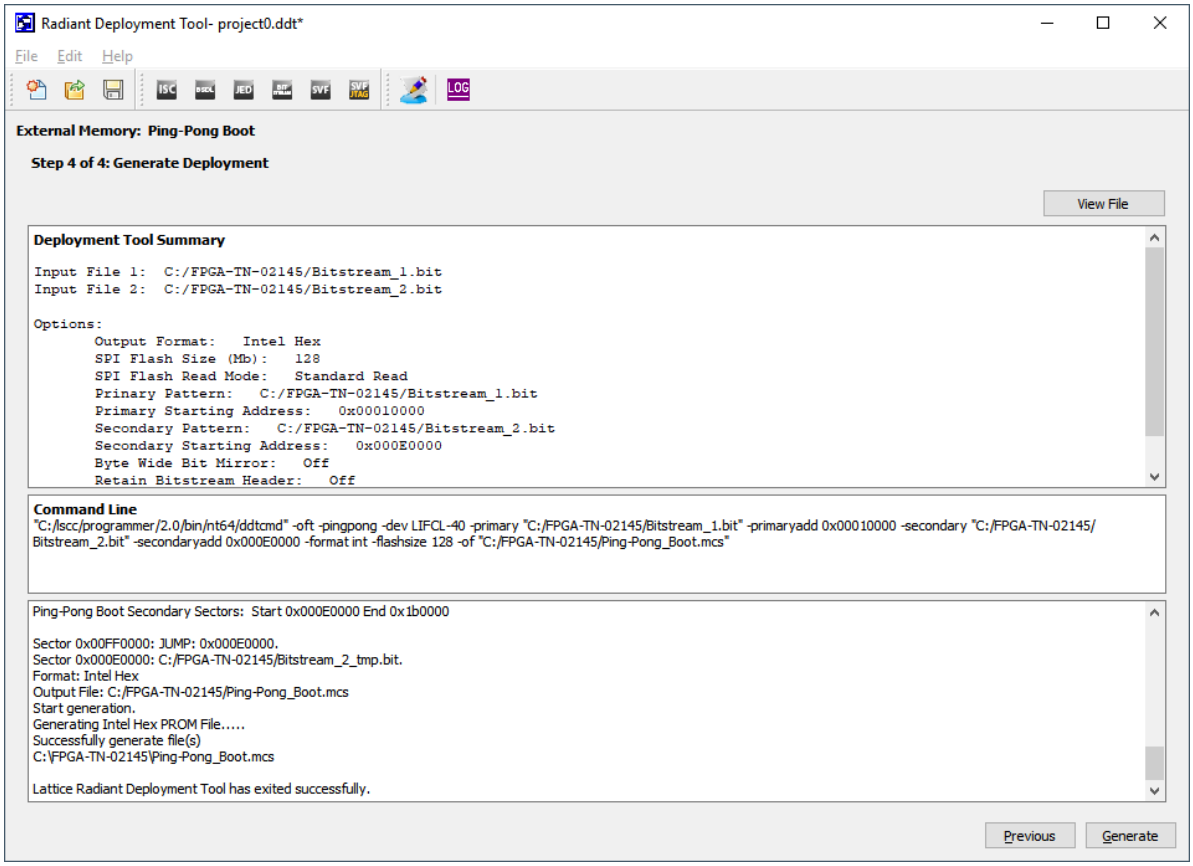


Figure 6.10. Generate Deployment Window

6.3. Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File

The following steps provide the procedure for generating a Multi-Boot PROM hex file using the Radiant Deployment Tool. This procedure is an example for four total bitstreams, Primary Pattern, Golden Pattern, Alternate Pattern 1, and Alternate Pattern 2.

1. Generate all the bitstream files needed in Lattice Radiant Software.
2. Invoke **Lattice Radiant Deployment Tool** from **Start > Lattice Radiant Programmer > Deployment Tool**.
3. In the Radiant Deployment Tool window, select **External Memory** as the **Function Type** and select **Advanced SPI Flash** as the **Output File Type** (Figure 6.11).
   - Note that the **External Memory** selection is also applicable to MachXO5-NX device that uses internal flash memory.
4. Select **OK**.

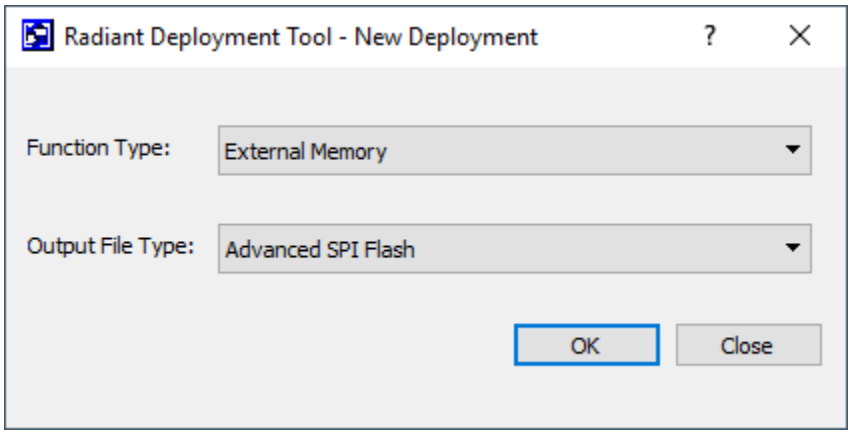


Figure 6.11. Creating New Deployment for Multi-Boot

Step 1 of 4: Select Input File(s) window (Figure 6.12)

- Click the **File Name** field to browse and select the primary bitstream file to be used to create the PROM hex file.
- The **Device Family** and **Device** fields auto populates based on the bitstream files selected.
- Select **Next**.

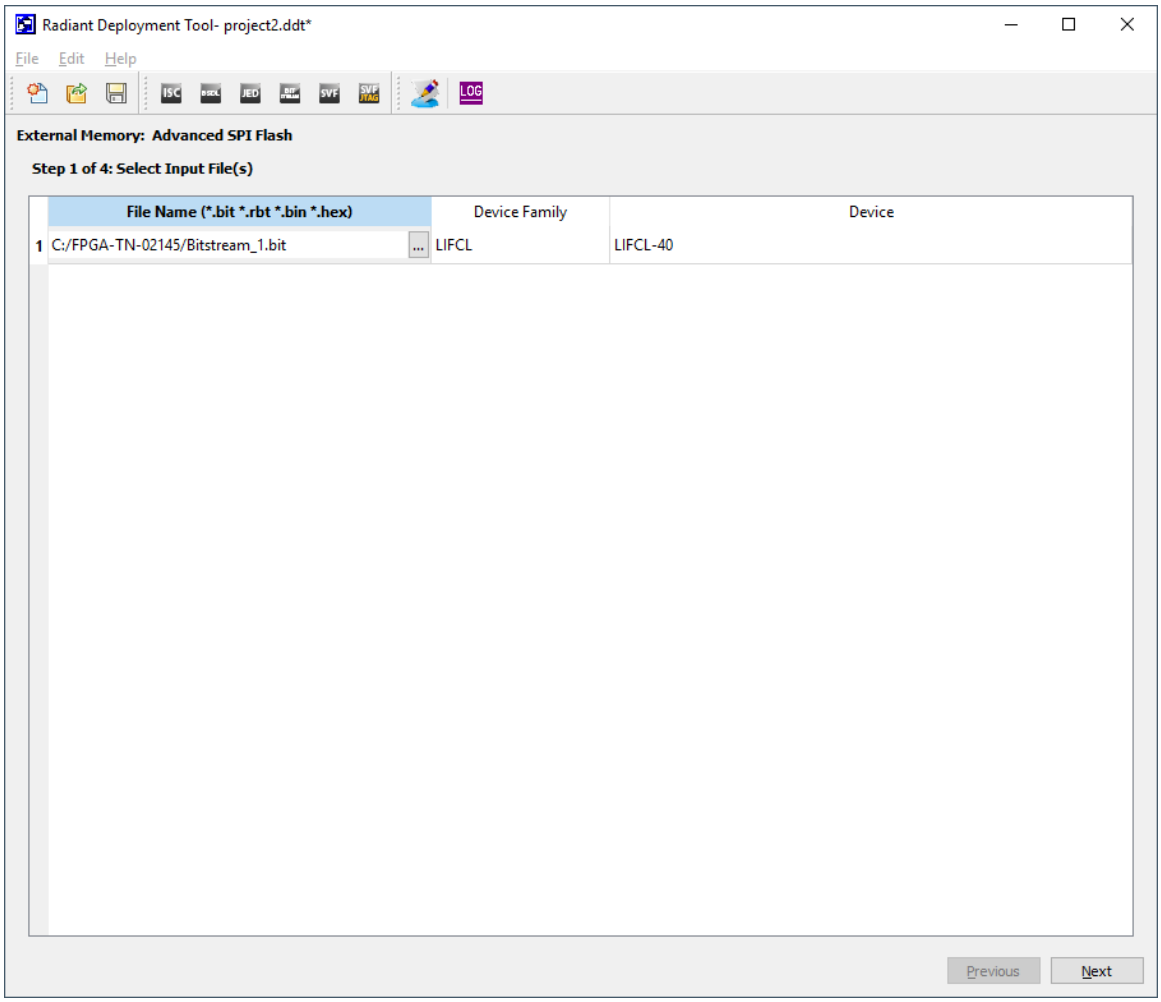


Figure 6.12. Select Input File Window

Note: Figure 6.12 shows the step to select the Primary pattern.

Step 2 of 4: Advanced SPI Flash Options window (Figure 6.13)

- Go to the **Options** tab.
- Select the **Output Format** (Intel Hex, Motorola Hex, or Extended Tektronix Hex).
- Select the **SPI Flash Size** (4, 8, 16, 32, 64, 128, 256, 512, and 1024 Mb).
- Select **SPI Flash Read Mode** (Standard Read, Fast Read, Dual I/O SPI Flash Read, or Quad I/O SPI Flash Read).
- Select the following options as required:
  - **Byte Wide Bit Mirror** – Flips each byte in Intel, Extended Tektronix, or Motorola hexadecimal data files.

    For example, 0xCD (b1100 1101) becomes 0xB3 (b1011 0011) when this is selected. You do not need to enable this setting if you program the .mcs file using Radiant Programmer. If you are using a third-party programmer, check with your vendor to understand if the byte wide bit mirror is needed.
  - **Retain Bitstream Header** – By default, Radiant Deployment Tool replaces the bitstream header information (name, version number, and date of the file) with 0xFF values. Selecting this option retains the header information that is generated as the header.
  - **Optimize Memory Space** – By default, the Radiant Deployment Tool uses the worst case file size for SPI Flash memory space allocation.
    a. Worst case size is an uncompressed bitstream with maximum EBR and PCS. This allows maximum flexibility for field upgrades. If a new Primary Pattern file size grows significantly due to less compression or adding EBR blocks, it is guaranteed to fit in the sectors already allocated for Primary Pattern.
    b. When this option is selected, the Radiant Deployment Tool uses the actual file size for the address allocation. This reduces wasted SPI Flash space and may allow for a smaller Flash device. If one or more of the new patterns have smaller compression ratio or more EBR/PCS, the new pattern(s) can encroach into another pattern bitstream memory space. If this occurs, the entire SPI Flash needs to be erased/ re-programmed with a new Hex file.
- Go to the **Multiple Boot** tab (Figure 6.14).
- Select the **Multiple Boot** option.
- Click on the **Golden Pattern** browse button to select the Golden Pattern bitstream.
  - The Starting Address of the Golden Pattern is automatically assigned. Change the Starting Address of the Golden Pattern by clicking on the drop-down menu.
- Select the following option as required:
  Protect Golden Sector – By default, the golden sector, where the Golden Pattern is stored, is located immediately after the primary sector to save SPI Flash space. When this option is selected, the Golden Pattern location is moved to the first sector in the upper half of the SPI Flash. The new location is reflected in the Golden Patten Starting Address field. This protects the Golden Pattern from accidental erase/reprogram by protecting the upper half of the SPI Flash when it is programmed.
- In the **Number of Alternate Patterns** field, select the number of alternate patterns to include through the drop-down menu.
- In the **Alternate Pattern 1** field, click on the browse button to select the first alternate pattern.
  - The Starting Address of Alternate Pattern 1 is automatically populated. You can change the Starting Address of Alternate Pattern 1 by clicking on the drop-down menu.
- The **Next Alternate Pattern to Configure** field is automatically populated.
  - This is the pattern that is loaded during the next PROGRAMN/REFRESH event. You can change the pattern by clicking on the drop-down menu.

    Note: You can select any available option. Once the MULTIBOOT primitive is instantiated in your design, any option you select does not affect the next pattern loading.
- In the **Alternate Pattern 2** field, click on the browse button to select the second alternate pattern.
  - The Starting Address of Alternate Pattern 2 is automatically populated. You can change the Starting Address of Alternate Pattern 2 by clicking on the drop-down menu.

- The **Next Alternate Pattern to Configure** field is automatically populated.
  - This is the pattern that is loaded during the next PROGRAMN/REFRESH event. You can change the pattern by clicking on the drop-down menu.

    Note: You can select any available option. Once the MULTIBOOT primitive is instantiated in your design, any option you select does not affect the next pattern loading.
- Select **Next**.

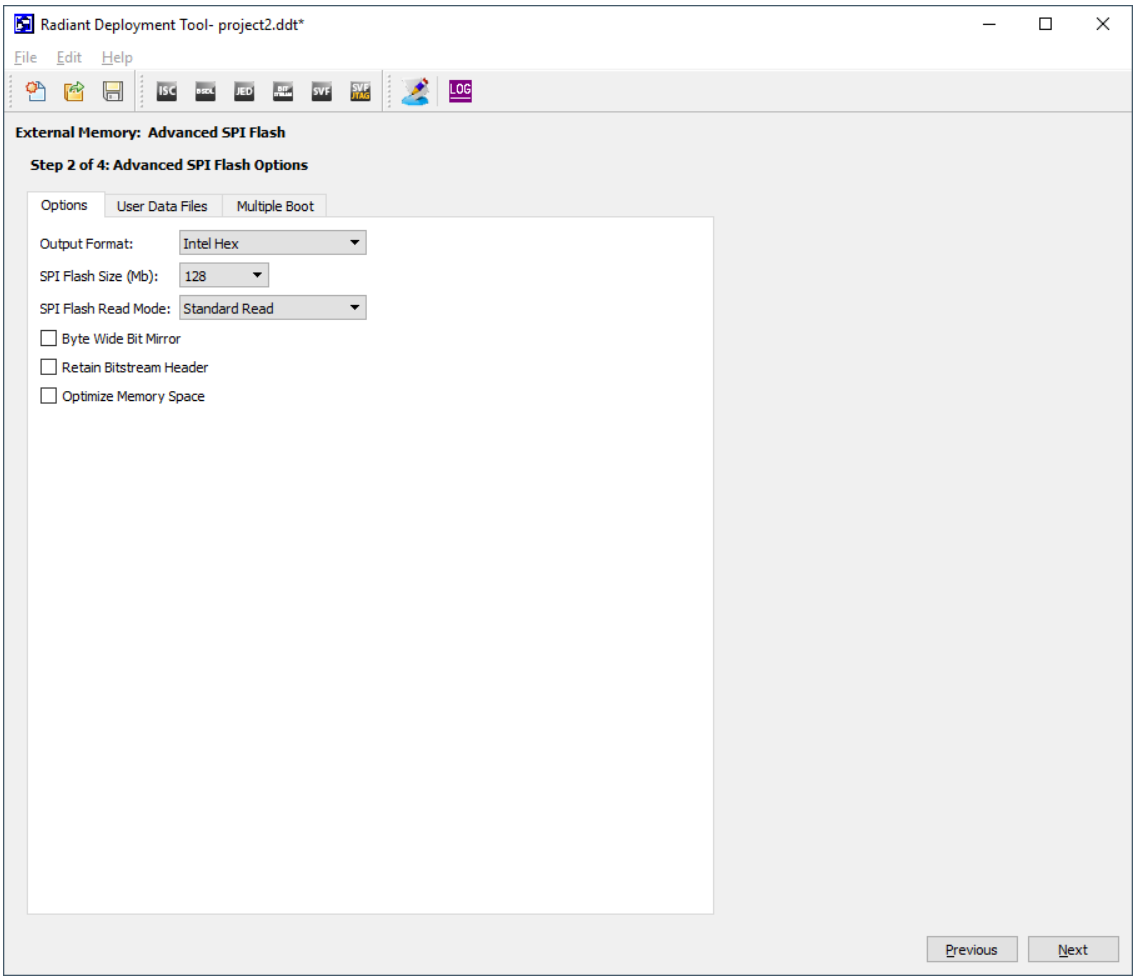


Figure 6.13. Advanced SPI Flash Options – Options Tab Window

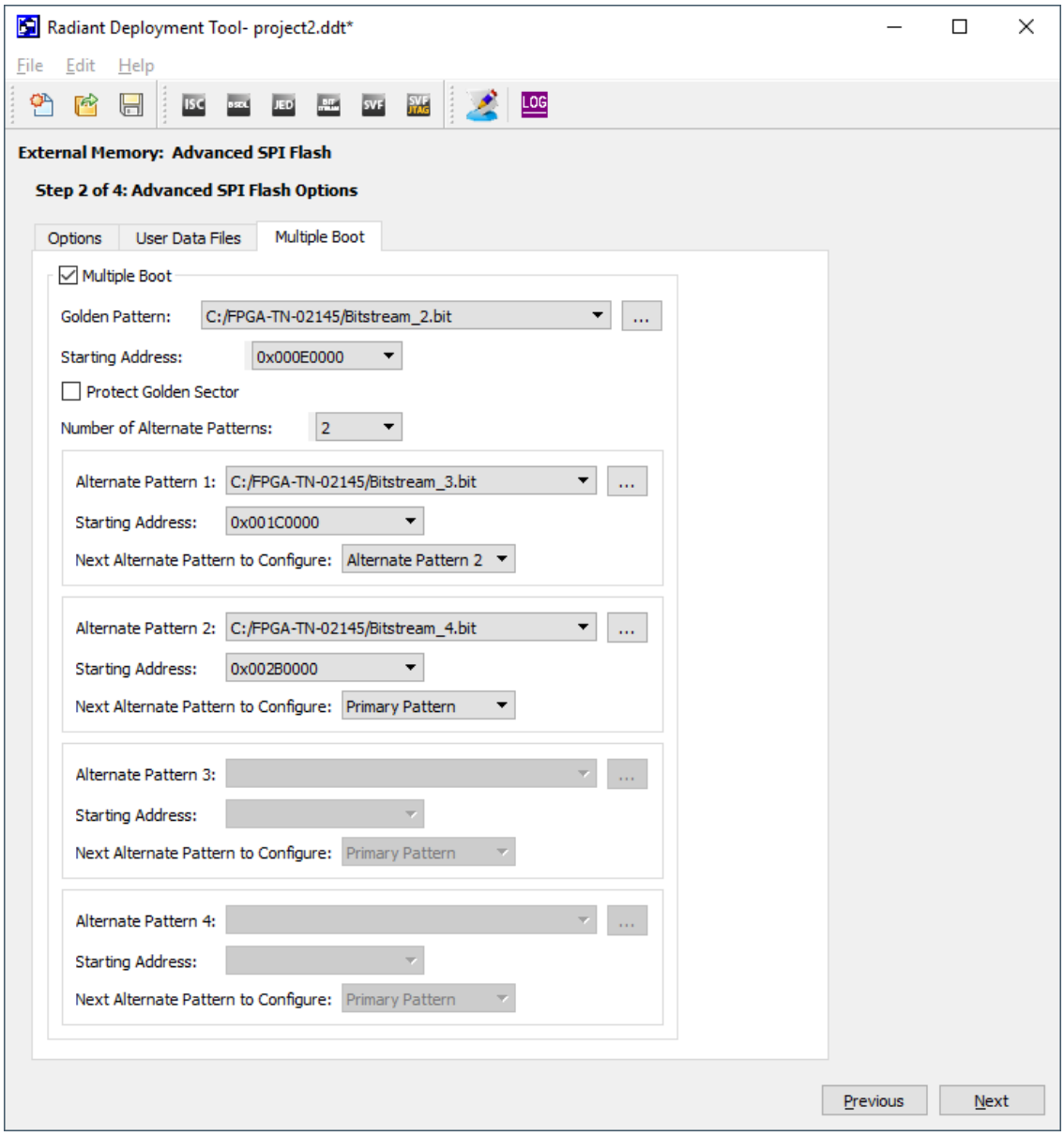


Figure 6.14. Advanced SPI Flash Options – Multiple Boot Tab Window

Step 3 of 4: Select Output File(s) window (Figure 6.15)

- Specify the name of the output PROM hex file in the **Output File 1** field.
- Select **Next**.

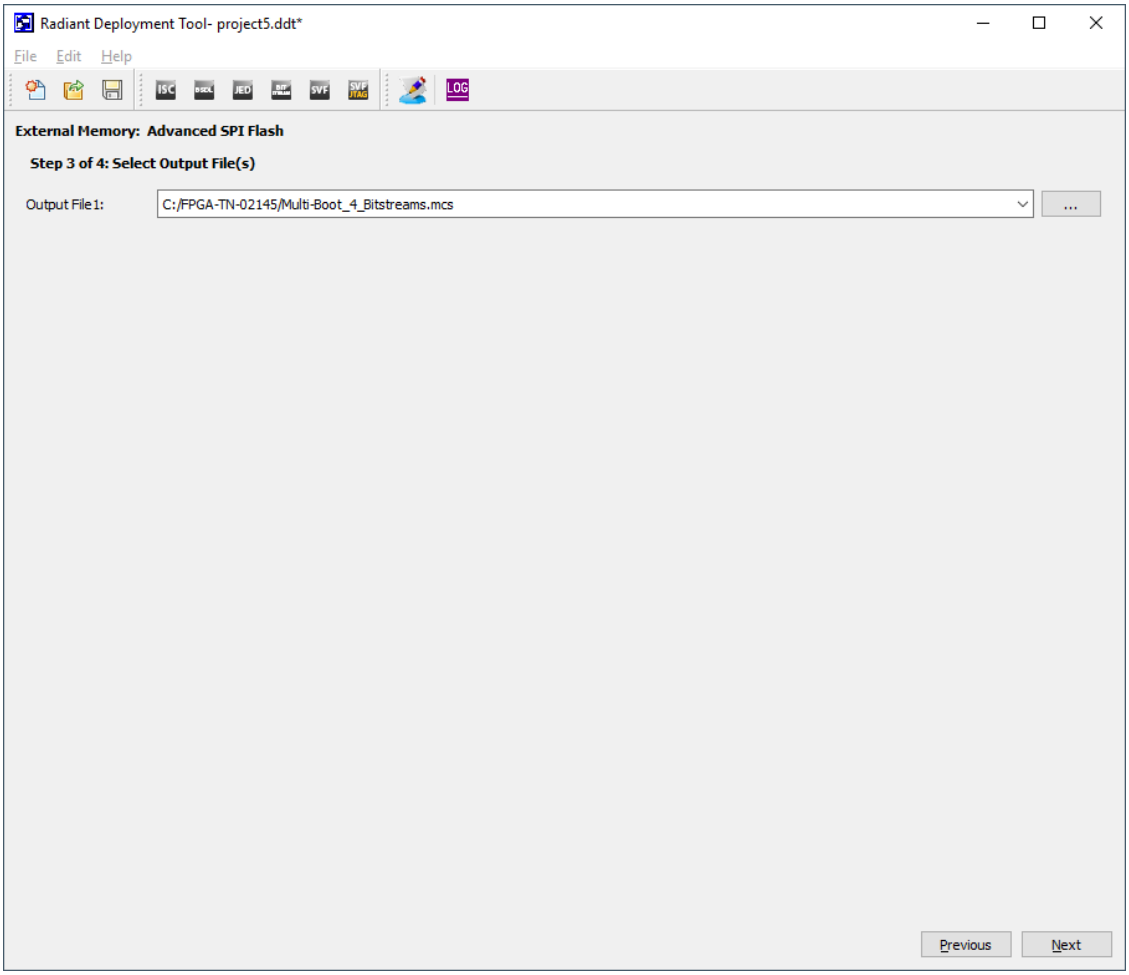


Figure 6.15. Select Output File Window

Step 4 of 4: Generate Deployment window (Figure 6.16)

- Review the summary information.
- If everything is correct, click the **Generate** button.
- The **Generate Deployment** pane should indicate the PROM file is generated successfully.
- Save the deployment settings by selecting **File > Save**.
- To exit, select **File > Exit**.

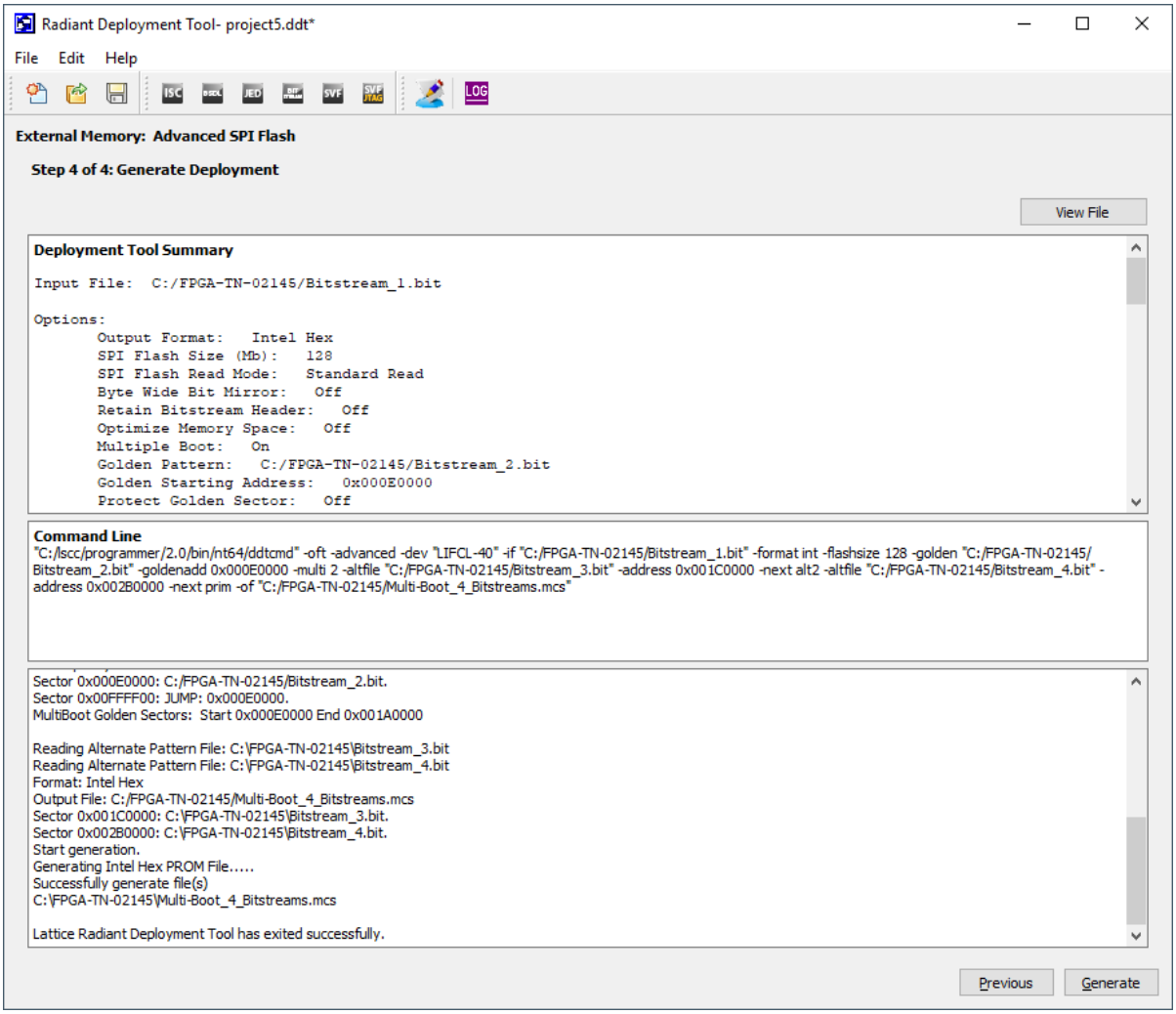


Figure 6.16. Generate Deployment Window

Refer to Lattice Nexus Multi-Boot Reference Design (FPGA-RD-02294) for more details of the reference design.

7. Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device

The following procedure is for programming a Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the SPI Flash Device using Radiant Programmer:

1. Connect power to the board and connect a download cable from the board to the PC.
2. Invoke Radiant Programmer using one of the following methods:
   - In Radiant Software window, select **Tools > Programmer**;
   - In Radiant Software window, select the **Programmer** icon ( ) in the Radiant toolbar;
   - In the Windows Start menu, select **Start > Lattice Radiant Programmer > Radiant Programmer**;
   - In the Windows Start menu, select **Start > Lattice Radiant Software > Radiant Programmer**.
3. Radiant Programmer – Getting Started window opens (Figure 7.1).
   - Select **Create a New Project from a Scan**, or **Create a new blank project**, or Select **Open an existing programmer project**.
   - Select **Detect Cable** to scan the PC to determine what cable is connected. Or, manually select the type of Cable and Port.
   - Select **OK**.
4. Select the Operation field by moving the cursor over it and double clicking the left mouse button.
5. The **Device Properties** window opens (Figure 7.2).
   - For **Target Memory**, select **External SPI Flash Memory (SPI Flash)**.
   - For **Port Interface**, select **JTAG2SPI**.

     CrossLink-NX/Certus-NX/CertusPro-NX and Radiant Programmer automatically takes care of the details to connect the JTAG port pins to the SPI interface pins and to program the external SPI Flash device via the JTAG port.
   - For **Access Mode**, select **Direct Programming**.
   - For **Operation**, select **Erase, Program, Verify**.
   - For Programming File, browse to select the .mcs file.
   - In the **SPI Flash Options** field, specify the **Family**, **Vendor**, **Device**, and **Package** of the Flash device used on the board.
   - For **Data File Size (Bytes)**, click on the **Load from File** button.
   - Click the **OK** button.
6. Program the external Flash device with one of the following methods:
   - In the **Radiant Programmer** window, select **Run > Program Device**.
   - In the **Radiant Programmer** window, click on the **Program** Device icon ( ) in the toolbar.

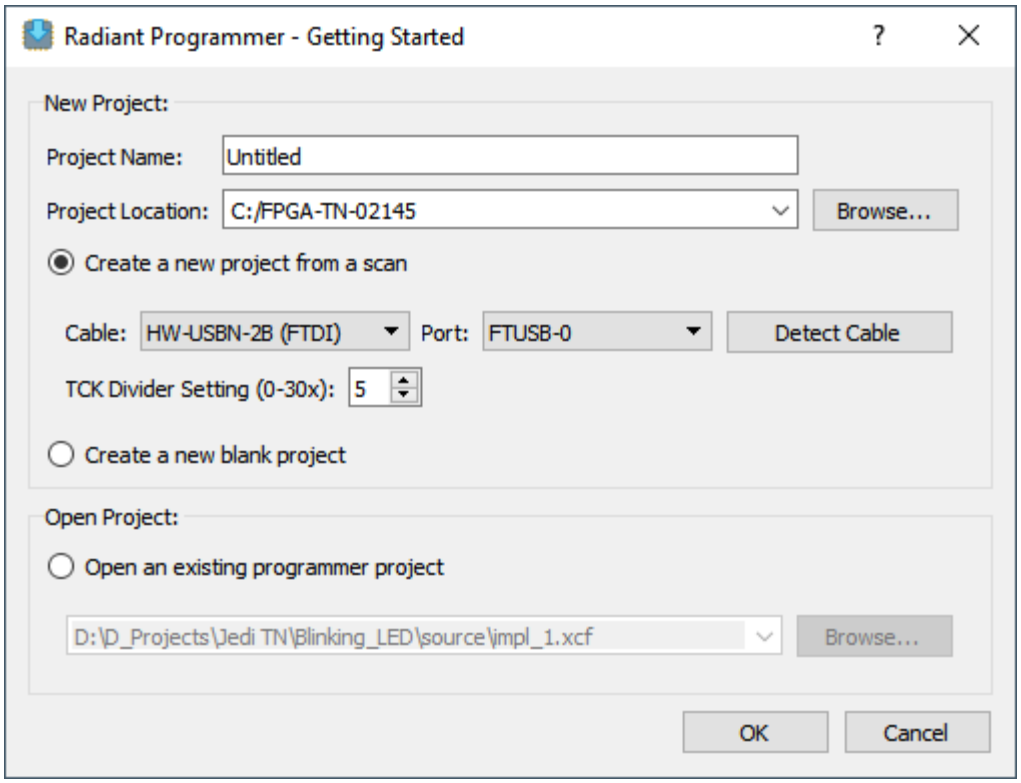


Figure 7.1. Radiant Programmer – Getting Started Window

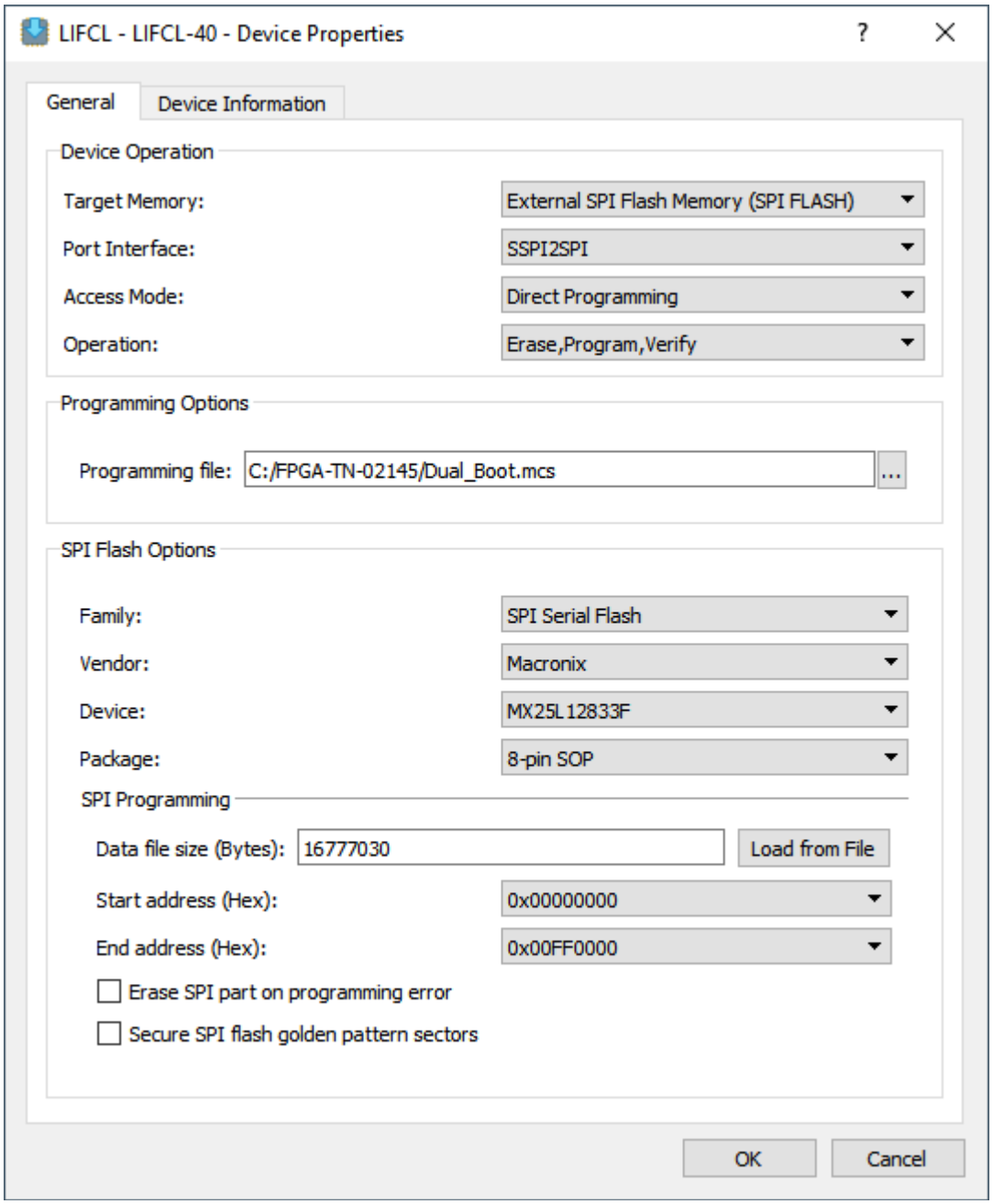


Figure 7.2. Radiant Programmer – Device Properties Window

8. Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the Internal Flash

Radiant Programmer provides the flexibility for you to program various programming files at the different sections of the internal flash for respective purposes. Table 8.1 shows the programming options available for Direct FLASH Programming mode in Radiant Programmer.

Table 8.1. Programming Options for Direct FLASH Programming Mode in Radiant Programmer

Programming Options	Input File	Description
Flash Header/Dual Image[1]	.mcs	There are 2 type of .mcs files allowed: • The *_header.mcs that consists of only JUMP table for Ping-pong boot. Radiant automatically generate *_header.mcs when running Export Bitstream File in a MachXO5-NX project. The primary and secondary location of the *_header.mcs is based on the PRIMARY_BOOT and SECONDARY_BOOT settings in Device Constraint Editor. • The .mcs file generated for Dual Boot, Ping-pong Boot or Multi-Boot using Deployment Tool as described in Section 6. When you select this type of .mcs file, the Programmer will disable all other programming options automatically.
CFG0	.jed	The *_0.jed configuration bitstream file that is generated by Radiant automatically when running Export Bitstream File, provided the CUR_DESIGN_BOOT_LOCATION in Device Constraint Editor is set to IMAGE_0.
UFM0	.jed	The *_u0.jed bitstream file that is generated by Radiant automatically when running Export Bitstream File, provided the UFM0 is initialized in Flash Access IP in the design.
CFG1	.jed	The *_1.jed configuration bitstream file that is generated by Radiant automatically when running Export Bitstream File, provided the CUR_DESIGN_BOOT_LOCATION in Device Constraint Editor is set to IMAGE_1.
UFM1	.jed	The *_u1.jed bitstream file that is generated by Radiant automatically when running Export Bitstream File, provided the UFM1 is initialized in Flash Access IP in the design.
CFG2	.jed	The *_2.jed configuration bitstream file that is generated by Radiant automatically when running Export Bitstream File, provided the CUR_DESIGN_BOOT_LOCATION in Device Constraint Editor is set to IMAGE_2.
UFM2	.jed	The *_u2.jed bitstream file that is generated by Radiant automatically when running Export Bitstream File, provided the UFM2 is initialized in Flash Access IP in the design.
UserData0-7[2]	.mcs	The *_ud0-7.mcs file that is generated by Radiant automatically when running Export Bitstream File, provided the USERDATAx is initialized in Flash Access IP in the design.
JUMP Command[1]	.mcs	The *_tail.mcs is generated by Radiant automatically when running Export Bitstream File. The JUMP address is following the SECONDARY_BOOT setting in the Device Constraint Editor.

Notes:
1. This option is available for Radiant Programmer version 2023.2 and onwards.
2. In Radiant Programmer 2023.1 and older version, UserData is from 0 to 8.

The following procedure is for programming various programming files into the MachXO5-NX device internal flash using Radiant Programmer:

1. Connect power to the board and connect a download cable from the board to the PC.
2. Invoke Radiant Programmer using one of the following methods:
   - In Radiant Software window, select **Tools > Programmer**;
   - In Radiant Software window, select the **Programmer** icon ( ) in the Radiant toolbar;
   - In the Windows Start menu, select **Start > Lattice Radiant Programmer > Radiant Programmer**;
   - In the Windows Start menu, select **Start > Lattice Radiant Software > Radiant Programmer**.
3. **Radiant Programmer – Getting Started** window opens (Figure 7.1).
   - Select **Create a New Project from a Scan**, or **Create a new blank project**, or **Select Open an existing programmer project**.
   - Select **Detect Cable** to scan the PC to determine what cable is connected. Or, manually select the type of Cable and Port.
   - Click **OK**.
4. Select the Operation field by moving the cursor over it and double clicking the left mouse button.
5. The **Device Properties** window opens (Figure 8.1).
   - For **Target Memory**, select **Flash Configuration Memory**.
   - For **Port Interface**, select **JTAG**.
   - For **Access Mode**, select **Direct FLASH Programming**.
   - For **Operation**, select **Erase, Program, Verify**.
   - For **Other Programming Options**, select the appropriate files according to Table 8.1.
   - Click the **OK** button.
   - Refer to MachXO5-NX Programming and Configuration User Guide (FPGA-TN-02271) for designation of primary boot and secondary boot.
6. Program the internal Flash device with one of the following methods:
   - In the Radiant Programmer window, select **Run > Program** Device.
   - In the Radiant Programmer window, click on the **Program** Device icon ( ) in the toolbar.

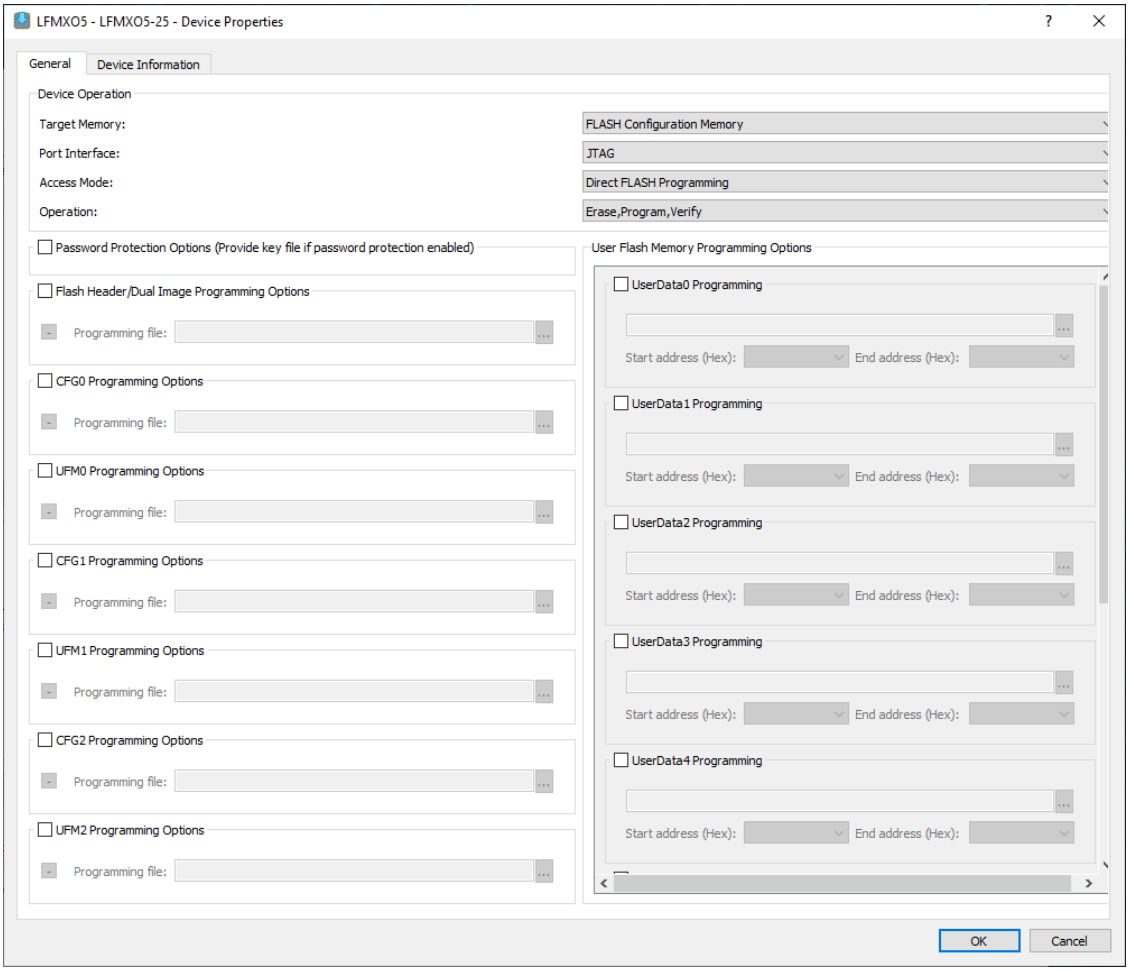


Figure 8.1. Radiant Programmer – Device Properties Window

9. Corrupting Primary Image to Test Dual Boot, Ping-pong Boot, or Multi-Boot

There are several ways to corrupt the Primary image to test if the device will fall back to the Golden image correctly:

- You can corrupt the .mcs file that is generated using Deployment Tool as described in Creating a PROM File section.
- You can use JTAG2SPI bridge in the Radiant Programmer to write junk data to the address space where Primary image is located in the external flash.
- In the MachXO5-NX design, you can use the Flash Access IP to write a byte in the CFG page where Primary image is located.

9.1. Corrupting Generated .mcs File using Deployment Tool

This section describes the first method of corrupting the .mcs file before programming it to the internal or external flash of Nexus devices.

1. Open the .mcs file. The .mcs file is in Intel Hex format, you can open it with any text editor, each line in an Intel Hex file has the same basic pattern as this:

   :NNAAAATT[DDDDDDDDDD]CC

   where,

:	Start of a line marker
NN	Number of data bytes on the line
AAAA	Address in bytes
TT	Type: • 00 indicates data type • 01 indicates end of file • Other types like 02 and 04 for extended address line
DD	Data bytes, the number of bytes depend on the NN value
CC	Checksum (2s-complement of number of bytes+address+type+data)

2. Identify the address location that you want to corrupt in the .mcs file and modify it. For example, below line is the 16 bytes at location 0x01A0, with the checksum 0x88. You can modify any data bytes in the line and recalculate the checksum after the modification.

   :1001A000**000000FFFFFFFF4700000080F00EC244**88 → Original line in .mcs file.

   For example, if you corrupt the location 0x01AF from 0x44 to 0xFF, the checksum is calculated by:

   a. Summing up every byte except checksum:
   b. 0x10 + 0x01 + 0xA0 + 0xFF + 0xFF + 0xFF + 0xFF + 0x47 + 0x80 + 0xF0 + 0x0E + 0xC2 + **0xFF (corrupted value)** = 0x833, omit the carry bit, the balance become 0x33.
   c. Invert all the bit of 0x33, the value is 0xCC.
   d. Adding 1 to 0xCC = 0xCD, this is the new calculated checksum.

3. Replace the original line with the new line that has the corrupted byte.

   :1001A000**000000FFFFFFFF4700000080F00EC2FF**CD → New line to replace the original line in .mcs file.

4. Then, you can proceed to program the corrupted .mcs file to the external or internal flash by following the instructions in Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device and Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the Internal Flash sections.

5. After the programming is successful, reconfigure or power cycle the FPGA. You should be able to observe the FPGA is now loading the Golden image instead of the Primary image. You can identify which image is loaded to the FPGA by observing the functionality of the design or using the Radiant Programmer to read the Status Register as shown in Figure 9.1.

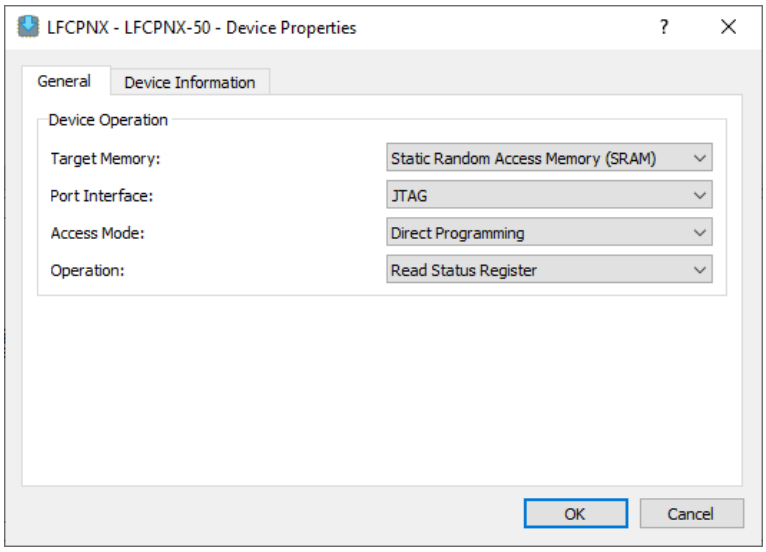


Figure 9.1. Reading Device Status Register using Radiant Programmer

9.2. Example of Corrupting Preamble of Primary Image in Dual Boot

This section provides the example of corrupting the preamble of the Primary image in Dual Boot .mcs, and the expected Status Register value after the device falls back to Golden image upon corruption of Primary image is detected during configuration.

The line below shows the preamble value of 0xFFFFBDB3, note that 0xFFFFBDCD in the .mcs is due to the reversed bit order of every byte, this is the expected .mcs format adopted by Radiant Programmer.

:10017000FFFFFFFFFFFFFFFFFFFFFFFF**FFFFBDCD**FF03 → Original line in .mcs file.

Corrupting the preamble:

:10017000FFFFFFFFFFFFFFFFFFFFFFFF**FFFFFFFF**FF8F → New line to replace the original line in .mcs file.

1. Program the corrupted .mcs file to the device as described in Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device and Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the Internal Flash sections.
2. Reconfigure or power cycle the FPGA.
3. Read the Status Register using Programmer as shown in Figure 9.1.

Figure 9.2 shows the Status Register value after the device falls back to Golden image. The **INITN** and **DONE** set to **1** indicate that the device is in user mode. The **BSE Error 1 Code** is **b0100**, which indicates the preamble error in previous bitstream execution. The **SPIm Fail** set to **1** indicates the failure to load the Primary image from the SPI flash due to the corruption.

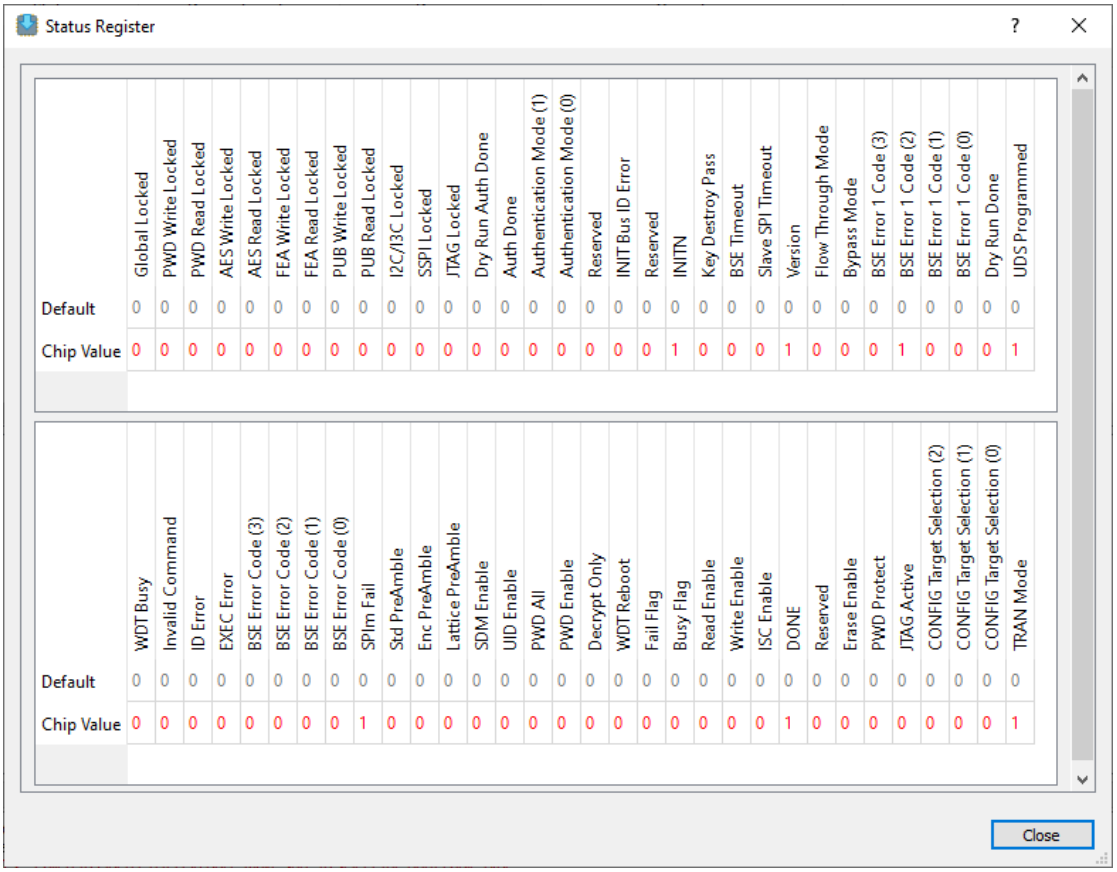


Figure 9.2. Device Status Register Value after Fall Back to Golden Image

10. Use Case Restrictions

10.1. Ping-pong Boot Limitation

If the bitstream corruption happens at the 32-bit preamble of the primary image, the configuration engine will not load the secondary image correctly, both the BSE Error Code and BSE Error 1 Code in Device Status Register will show a preamble error. You can recover it by reprogramming the external or internal flash memory with a good image via the JTAG or SSPI port. It is recommended that you use Dual-Boot configuration if your application requires you to update the user image remotely.

10.2. Soft Error Detection and Correction (SEDC) Use Case

If your design is using the SEDC feature, you need to ensure all images are implementing the SEDC feature. Mixing images with and without the SEDC feature in your design (for Dual Boot, Ping-pong Boot or Multi-Boot application) is not supported and will result in failure when reconfiguring the FPGA with another image that is triggered by the user or due to image corruption.

Refer to SED/SEC User Guide for Nexus Platform (FPGA-TN-02076) for more detail.

References

For more information, refer to:

- CrossLink-NX Family Devices Web Page
- Certus-NX Family Devices Web Page
- CertusPro-NX Family Devices Web Page
- MachXO5-NX Family Devices Web Page
- Lattice Nexus Platform Web Page
- MachXO5-NX Programming and Configuration User Guide (FPGA-TN-02271)
- SED/SEC User Guide for Nexus Platform (FPGA-TN-02076)

For Boards, Demos, IP Cores, and Reference Designs for Lattice Nexus Devices, refer to:

- Boards, Demos, IP Cores, and Reference Designs for CrossLink-NX Devices
- Boards, Demos, IP Cores, and Reference Designs for Certus-NX Devices
- Boards, Demos, IP Cores, and Reference Designs for CertusPro-NX Devices
- Boards, Demos, IP Cores, and Reference Designs for MachXO5-NX Devices

Other References:

- Lattice Insights for Training Series and Learning Plans
- Lattice Radiant Software Web Page

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 2.5, April 2026

Section	Change Summary
Introduction	Added a description of the maximum supported flash clock frequency and added Table 1.2. Maximum Supported Flash Frequency.
Resources	Updated the Total Bitstream Size to the current in the Ping-Pong Boot Mode and the Multi-Boot Mode sections.
Ping-Pong Boot Mode	Added the JUMP Table File Structure, Swapping Boot Image Order, and Safe Primary Image Update Procedure sections.
Creating a PROM File	Deleted the description about MCCLK_FREQ and FLASH_CLK_FREQ from: • Step 1 of the Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File section, • Step 1 of the Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File section, and • Step 1 of the Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File section.

Revision 2.4, August 2025

Section	Change Summary
Introduction	Table 1.1. Supported Device Families and Parts: • removed LFMXO5-15D and LFMXO5-55TD parts; • newly added note to the table.
Resources	• Removed the original Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Mode, and Table 2.3. Maximum Configuration Bitstream Size – Multi-Boot Mode. • Newly added Single Bitstream Mode, Dual Boot Mode, Ping-Ping Boot Mode, and Multi-Boot Mode related contents.

Revision 2.3, July 2025

Section	Change Summary
Introduction	Table 1.1. Supported Device Families and Parts: • added LFD2NX-15, LFD2NX-25, LFD2NX-35 and LFD2NX-65 to Certus-NX device family; • added LFCPNX-50 to CertusPro-NX device family; • added LFMXO5-35, LFMXO5-35T, LFMXO5-65 and LFMXO5-65T to MachXO5-NX device family.
Resources	• Updated Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode globally, adding LFD2NX-15, LFD2NX-25, LFD2NX-35 and LFD2NX-65, and LFCPNX-50 support. • Updated Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode globally, adding LFD2NX-15, LFD2NX-25, LFD2NX-35 and LFD2NX-65, and LFCPNX-50 support. • Updated Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode globally, adding LFD2NX-15, LFD2NX-25, LFD2NX-35 and LFD2NX-65, and LFCPNX-50 support.

Revision 2.2, December 2024

Section	Change Summary
Multi-Boot Mode	In the Implementation of Multi-Boot Feature Using MULTIBOOT Primitive section: • Added note on alternative method for setting the SPIM bit. • Added the Setting SPIM Bit Using Programming File Utility Tool section.

Revision 2.1, October 2024

Section	Change Summary
Introduction	Added Important Note at the end of this section.
Multi-Boot Mode	Newl added the MULTIBOOT Primitive, Booting Flow without MULTIBOOT Primitive, and Booting Flow with MULTIBOOT Primitive sections.
Creating a PROM File	• Added *For LFMXO5-55T and LFMXO5-100T parts, the maximum supported FLASH_CLK_FREQ of the primary bitstream file is 56.2 MHz. This limitation is not applicable to the single boot feature* to the Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File, Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File, and Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File sections. • Added *FLASH_CLK_FREQ* to *MCCLK_FREQ can be configured using the Global tab of the Device Constraint Editor in Lattice Radiant software* in the Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File, Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File, and Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File sections. • MCCLK_FREQ and FLASH_CLK_FREQ can be configured using the Global tab of the Device Constraint Editor in Lattice Radiant software. • Added Note to Step 1 of 4 and Step 2 of 4 in the Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File section.

Revision 2.0, September 2024

Section	Change Summary
Ping-Pong Boot Mode	Updated Note 3 in Figure 4.1. Ping-Pong Boot Flow Diagram.

Revision 1.9, August 2024

Section	Change Summary
Introduction	Table 1.1. Supported Device Families and Parts: • added LFD2NX-9 and LFD2NX-28 to Certus-NX family; • added LFMXO5-15D and LFMXO5-55TD to MachXO5-NX family.

Resources	• Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode:
	  • added LFD2NX-9 and LFD2NX-28 to related existing devices;
	  • added LFMXO5-15D, LFMXO5-55TD, and their related information.
	• Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode:
	  • added LFD2NX-9 and LFD2NX-28 to related existing devices;
	  • added LFMXO5-15D, LFMXO5-55TD, and their related information.
	• Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode:
	  • added LFD2NX-9 and LFD2NX-28 to related existing devices;
	  • added LFMXO5-15D, LFMXO5-55TD, and their related information;
	  • updated Note 1 and added Note 5.

Section	Change Summary
Ping-Pong Boot Mode	Figure 4.1. Ping-Pong Boot Flow Diagram: • added 0x003F_FF00; • added Note 3.

Revision 1.8, February 2024

Section	Change Summary
All	• Updated the mentions of *CrossLink-NX, Certus-NX, CertusPro-NX, and MachXO5-NX device families* as *Nexus devices* or *devices*. • Updated the mentions of *CrossLink-NX/Certus-NX/CertusPro-NX/MachXO5-NX family* as *Nexus device* or *device*. • Updated the mentions of *CrossLink-NX/Certus-NX/CertusPro-NX/MachXO5 NX part* to *device*. • Made editorial fixes.
Introduction	• Added the sentence: *For the subsequent part of this document, the Nexus devices refer to all CrossLink-NX, Certus-NX, CertusPro-NX, and MachXO5-NX device families* in the first paragraph of this section. • Added the sentence: *Note that the MachXO5-NX family supports up to 3 bitstream patterns only inclusive of the Golden pattern* to the Multi-Boot Mode's description.
Resources	• Removed *an external* from the sentence: *The volatile SRAM configuration memory must be loaded from an external non-volatile memory that can store all the configuration data*. • Removed *or embedded flash* from the sentence: *The minimum SPI Flash densities or embedded flash required to support the different configuration boot modes are listed in* Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Mode, and Table 2.3. Maximum Configuration Bitstream Size – Multi-Boot Mode.
Dual Boot Mode	Updated section header names and figure caption in this section.
Ping-Pong Boot Mode	Updated section header names and figure caption in this section.
Multi-Boot Mode	Updated section header names in this section.
Creating a PROM File	• Added the phrase *(MachXO5-NX uses internal flash)* to the sentence: *The various boot features on the Nexus devices are simple, requiring only one external SPI Flash device (MachXO5-NX uses internal flash), and flexible, due to the intelligent use of the JUMP command or table.* • Added the phrase *with the .mcs file extension* to the sentence: *The Lattice Deployment Tool, part of Lattice Radiant Software, merges the different patterns and the JUMP command and table into one PROM hex file with the .mcs file extension.* • Updated steps *1* and *3* of Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File, Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File, and Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File sections. • Updated the description of *Byte Wide Bit Mirror* in Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File, Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File, and Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File sections. • Replaced *the Lattice* to *Radiant* in Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File section header. • Updated the description of *Generate Jump Table Only* in Using Radiant Deployment Tool to Create a Ping-Pong Boot PROM Hex File section.

Section	Change Summary
Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the Internal Flash	• Replaced *Embedded Flash* with *Internal Flash*. • Added the sentences: *Radiant Programmer provides the flexibility for you to program various programming files at the different sections of the internal flash for respective purposes. Table 8.1 shows the programming options available for Direct FLASH Programming mode in Radiant Programmer* to this section. • Added Table 8.1. Programming Options for Direct FLASH Programming Mode in Radiant Programmer. • Replaced the phrases *a Dual Boot, Ping Pong Boot, or Multi-Boot Pattern* with *various programming* files and *embedded Flash, such as a MachXO5-NX device* with *MachXO5-NX device internal flash* in the sentence: *The following procedure is for programming various programming files into the MachXO5-NX device internal flash using Radiant Programmer*. • Replaced the sentence: *For CFGx Programming Files, browse to select the .jed files* with *For Other Programming Options, select the appropriate files according to Table 8.1.* • Updated Figure 8.1.
Corrupting Primary Image to Test Dual Boot, Ping-pong Boot, or Multi-Boot	Added this section.
Use Case Restrictions	Added this section.
References	Added references to MachXO5-NX Programming and Configuration User Guide (FPGA-TN-02271) and SED/SEC User Guide for Nexus Platform (FPGA-TN-02076).

Revision 1.7, December 2023

Section	Change Summary
Disclaimers	Updated this section.
Inclusive Language	Newly added this section.
Creating a PROM File	• In the Using Radiant Deployment Tool to Create a Dual Boot PROM Hex File section, added 1024 Mb support for SPI Flash Size in Step 2 of 4: Dual Boot Options window. • In the Using the Lattice Deployment Tool to Create a Ping-Pong Boot PROM Hex File section, added 1024 Mb support for SPI Flash Size in Step 2 of 4: Ping-Pong Boot Options window. • In the Using Radiant Deployment Tool to Create a Multi-Boot PROM Hex File section, added 1024 Mb support for SPI Flash Size in Step 2 of 4: Advanced SPI Flash Options window.

Revision 1.6, August 2023

Section	Change Summary
Introduction	Added LIFCL-33U to Table 1.1. Supported Device Families and Parts.
Resources	• Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode: added LIFCL-33U and its data. • Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode: added LIFCL-33U and its data. • Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode: added LIFCL-33U and its data.
References	Newly added section

Revision 1.5, March 2023

Section	Change Summary
Introduction	Added support to the LFMXO5-55T and LFMXO5-100T parts in the description.
Resources	• Removed MachXO5-NX from the SRAM support. • Added embedded flash support for different configuration boot modes. • Added LFMXO5-55T and LFMXO5-100T device support in Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode, and Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode.
Creating a PROM File	Removed MachXO5-NX family from the various boot features support.
Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device	• Updated the section title adding external. • Removed the MachXO5-NX family support from this section.
Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the Internal Flash	Newly added section.
Technical Support Assistance	Added the frequently asked questions website link.

Revision 1.4, June 2022

Section	Change Summary
Introduction	Added CrossLink-NX-33 (LIFCL-33) device support.
Resources	• Added CrossLink-NX-33 device support. • Added CrossLink-NX-33 (LIFCL-33) device and its related data to Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode, and Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode.

Revision 1.3, March 2022

Section	Change Summary
All	Changed the document title to *Multi-Boot User Guide for Nexus Platform*.
Introduction	Added MachXO5-NX family support.
Resources	• Added MachXO5-NX family support. • Added MachXO5-NX device and its related data to Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode, and Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode.
Dual Boot Mode	• Added MachXO5-NX device family support. • Updated Figure 3.1. Nexus Device Dual Boot Flow Diagram changing to Internal/External Flash.
Ping-Pong Boot Mode	• Added MachXO5-NX device family support. • Updated Figure 4.1. Ping-Pong Boot Flow Diagram changing to Internal/External Flash.
Multi-Boot Mode	Added MachXO5-NX family support.
Creating a PROM File	
Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device	

Revision 1.2, May 2021

Section	Change Summary
Introduction	Added support for CertusPro-NX device family.
Resources	
Dual Boot Mode	
Ping-Pong Boot Mode	
Multi-Boot Mode	
Creating a PROM File	
Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device	
Resources	Added resources details for CertusPro-NX device family to Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode, and Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode.

Revision 1.1, May 2020

Section	Change Summary
All	Changed the document title to “Multi-Boot Usage Guide for Nexus Platform”.
Introduction	Added support for the Nexus platform including Certus-NX and CrossLink-NX device families.
Resources	
Dual Boot Mode	
Ping-Pong Boot Mode	
Multi-Boot Mode	
Creating a PROM File	
Programming the Dual Boot, Ping-Pong Boot, or Multi-Boot Pattern into the External SPI Flash Device	
Resources	Added resources details for Certus-NX device family in Table 2.1. Maximum Configuration Bitstream Size – Single Bitstream Boot Mode, Table 2.2. Maximum Configuration Bitstream Size – Dual Boot Mode/Ping-Pong Boot Mode, and Table 2.3 Maximum Configuration Bitstream Size – Multi-Boot Mode.

Revision 1.0, January 2020

Section	Change Summary
All	Initial release.

www.latticesemi.com