Lattice Avant Device Multi-Boot

Preliminary Reference Design

FPGA-RD-02321-0.80

August 2025

Disclaimers

Inclusive Language

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
FPGA	Field Programmable Gate Array
IP	Intellectual Property
LED	Light-Emitting Diode
LMMI	Lattice Memory Mapped Interface
OTP	One-Time Programmable
RTL	Register Transfer Level
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

1. Introduction

This reference design showcases the Multi-Boot mode supported in Lattice Avant™ devices. The Multi-Boot mode supports booting from up to eighteen patterns that reside in an external SPI flash device. The patterns include a primary pattern, a golden pattern, and up to sixteen alternate patterns, designated as Alternate Pattern 1 to Alternate Pattern 16.

The device boots by loading the primary pattern from the external flash. In static mode, when a toggling of the PROGRAMN pin or receiving a REFRESH command, Alternate Pattern 1 is always loaded. Subsequent PROGRAMN/REFRESH event loads the next pattern defined in the Multi-Boot configuration. If the loading of the primary pattern or any alternate pattern fails, the device attempts to load the golden pattern. 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.

By using the CONFIG_LMMIC primitive, it allows the device to operate in the dynamic mode. It allows the system to dynamically switch to any of the alternate patterns after the device boots up from the primary pattern while still being protected by a golden pattern. This reference design implements the dynamic mode, which allows the system to dynamically switch between two to three bitstream patterns using the CONFIG_LMMIC primitive. By using the Multi-Boot mode, you can combine all the bitstream patterns into a single bitstream image and store it in a single external SPI flash device. This solution decreases cost, reduces board space, and simplifies field upgrades. Note that this reference design is developed using the Avant-X70 device, but the design can be ported to other Avant devices.

Refer to Lattice Avant Multi-Boot User Guide (FPGA-TN-02314) for more information.

1.1. Quick Facts

Download the reference design files from the Lattice reference design web page.

Table 1.1. Summary of the Reference Design

General	Target Devices	Avant-X70 (LAV-AT-X70-3LFG1156I)
	Source code format	Verilog
Simulation	Functional simulation	Not performed
	Timing simulation	Not performed
	Test bench	Not available
	Test bench format	Not available
Software Requirements	Software tool and version	2024.2 SP1
	IP version (if applicable)	OSC 2.1.0
Hardware Requirements	Board	Avant-G/X Versa Board
	Cable	USB-A to Mini-B Programming Cable

1.2. Feature

Key feature of the Multi-Boot reference design is it allows the system to dynamically switch between two and up to three bitstream patterns using the CONFIG_LMMIC primitive while still being protected with a golden pattern.

1.3. Naming Conventions

1.3.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.3.2. Signal Names

- *_n* are active low signals, asserted when value is logic 0.
- *_i* are input signals.
- *_o* are output signals.

2. Directory Structure and Files

Figure 2.1 shows the directory structure.

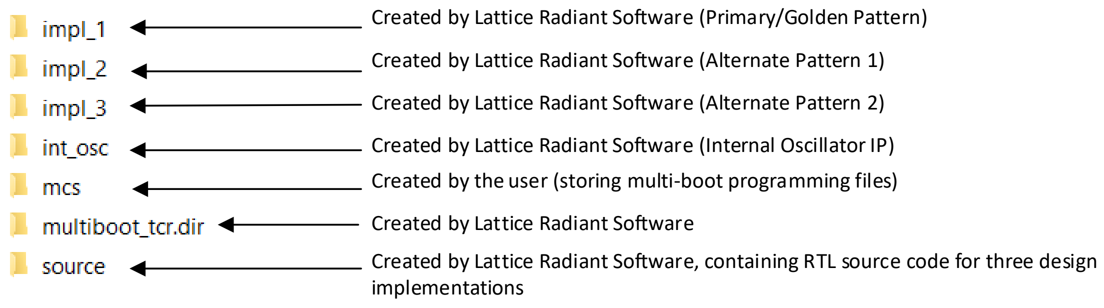

impl_1
Created by Lattice Radiant Software (Primary/Golden Pattern)
impl_2
Created by Lattice Radiant Software (Alternate Pattern 1)
impl_3
Created by Lattice Radiant Software (Alternate Pattern 2)
int_osc
Created by Lattice Radiant Software (Internal Oscillator IP)
mcs
Created by the user (storing multi-boot programming files)
multiboot_tcr.dir
Created by Lattice Radiant Software
source
Created by Lattice Radiant Software, containing RTL source code for three design implementations

Figure 2.1. Directory Structure

Table 2.1 shows the list of files included in the reference design package.

Table 2.1. File List

Attribute	Description
<Component name>.ipx	This file contains the information on the files associated with the generated IP.
<Component name>.cfg	This file contains the parameter values used in the IP configuration.
component.xml	Contains the ipxact: component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.v	This file provides an example RTL top file that instantiates the module.
rtl/<Component name>_bb.v	This file provides the synthesis closed box.
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	These files provide instance templates for the module.

3. Functional Description

Figure 3.1 shows the top-level block diagram of the reference design. The blocks shown in Figure 3.1 are the fundamental blocks that appear in all the four design implementations for Multi-Boot operations.

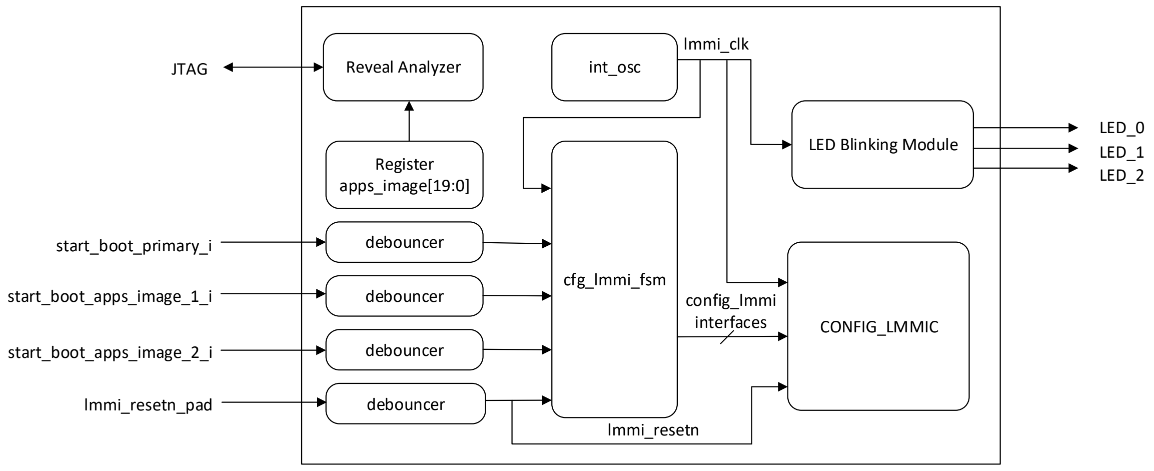


Figure 3.1. Reference Design Block Diagram

3.1. Design Components

The Avant Multi-Boot reference design includes the following blocks:

- Int_osc: This is an oscillator module. It generates the clock source (lmmi_clk) to the cfg_lmmi_fsm module, CONFIG_LMMIC primitive, and the LED blinking module. The lmmi_clk clock is running at 25 MHz. Refer to Lattice Avant OSC Module User Guide (FPGA-IPUG-02184) for more information.
- LED blinking module: Implements a counter to blink the three LEDs on board.
- Reveal Analyzer: Reveal Analyzer is used to determine which alternate pattern is running on the device.
- Debouncer: This block is to eliminate the noise or fluctuations, also known as bouncing, in a digital input signal, which can occur when a mechanical switch is pressed or released.
- CONFIG_LMMIC: This is the Lattice Memory Mapped Interface (LMMI) interface to the configuration block. Refer to the CONFIG_LMMIC section in Lattice Radiant Software Help for more information.
- cfg_lmmi_fsm: Implements a state machine controller to send the necessary commands to the CONFIG_LMMIC primitive to boot the desired alternate pattern stored in an external SPI flash. The controller performs the following sequences. Refer to Figure 3.2 for more details.
  - IDLE: Idle and waiting for either one of these input signals (start_boot_primary_i, start_boot_apps_image_1_i, or start_boot_apps_image_2_i) to perform multi-boot operations.
  - LMMI_WRITE_NOOP: Executes 32 bytes of NOOP command.
    - LMMI Offset = 8’h00
    - LMMI_CFG_DATA[15:0] = 16’h0000
  - PORT_REQUEST_FORCE: Executes the port request with force request for exclusive access to the configuration engine.
    - LMMI Offset = 8’h02 and LMMI_CFG_PORT_REQUEST[31:16]=16’h7A00
    - LMMI Offset = 8’h01 and LMMI_CFG_PORT_REQUEST[15:0]=16’h0002
      Note: Write LMMI_CFG_PORT_REQUEST[15:0] first followed by LMMI_CFG_PORT_REQUEST[31:16]. Failure to follow this sequence might cause the LMMI port to become inactive, thus requiring power cycling of the device to recover the LMMI port.
  - PORT_STATUS: Reads out the port status, LMMI_CFG_PORT_STATUS[15:0], for the CONFIG_LMMI interface. Proceed to the next state once the exclusive access is claimed. If the port status is available, go back to the PORT_REQUEST_FORCE state to claim exclusive access. If the port is inactive or disable, go back to the idle state.

    - LMMI Offset = 8’h03
- MSPI_MULTIBOOT_ADDR: Sends the address through the CONFIG_LMMIC primitive. The address refers to the start address where you store the bitstream in the external SPI flash and that is the bitstream or pattern you desire the device to boot for configuring the device.
    - LMMI Offset = 8’h05 and LMMI_CFG_MSPI_MULTIBOOT_ ADDR [15:0] = boot_address_15_to_0
    - LMMI Offset = 8’h06 and LMMI_CFG_MSPI_MULTIBOOT_ ADDR [31:16] = boot_address_31_to_16
- REFRESH Command – 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 in the MSPI_MULTIBOOT_ADDR state. If the loading of the image fails, the device falls back to load the golden pattern. In case the REFRESH command is not executed properly, proceed to the next state to release the exclusive access to the configuration engine.
    - LMMI Offset = 8’h00 and LMMI_CFG_DATA[15:0] = 16’h8000
    - LMMI Offset = 8’h00 and LMMI_CFG_DATA[15:0] = 16’h0A00
- PORT_REQUEST_RELEASE: Ensures the flow terminates gracefully by releasing exclusive access to the configuration engine, especially in cases where the previous REFRESH command may not have executed correctly.
    - LMMI Offset = 8’h02 and LMMI_CFG_PORT_REQUEST[31:16]=16’h7A00
    - LMMI Offset = 8’h01 and LMMI_CFG_PORT_REQUEST[15:0]=16’h0003

Refer to Table D.2. LMMI CFG Offset List and Table 6.12. Target Configuration Commands in the Avant sysCONFIG User Guide (FPGA-TN-02299) for more information.

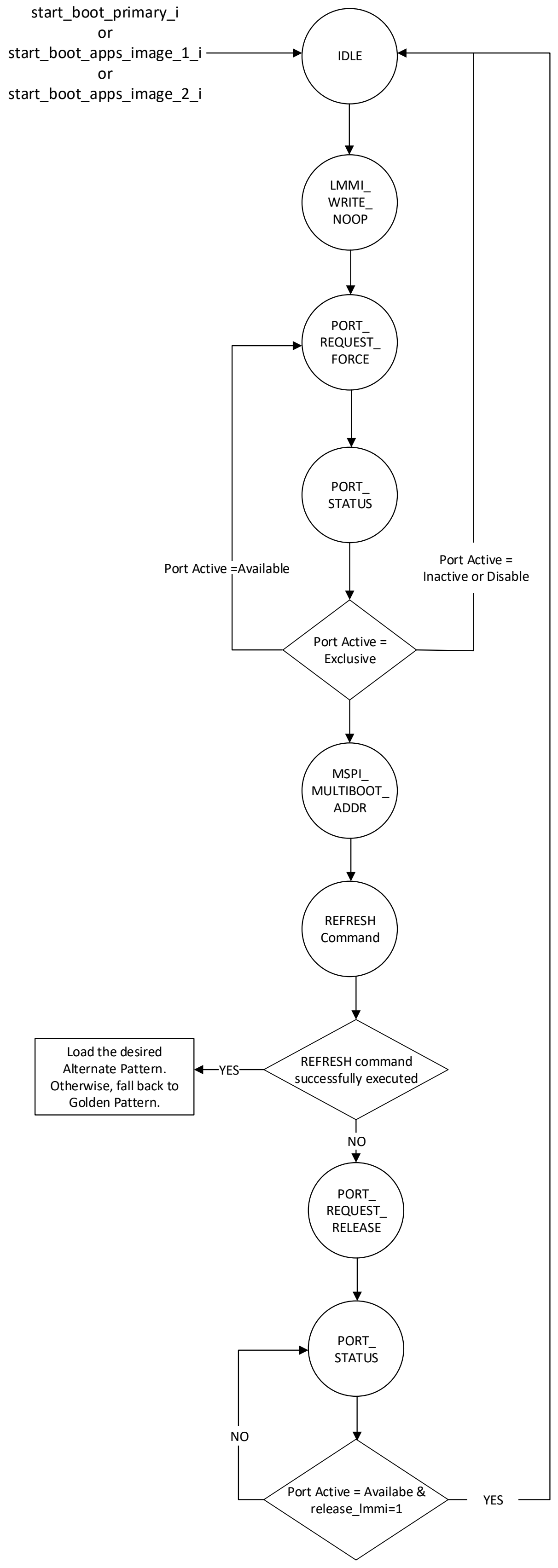

start_boot_primary_i
or
start_boot_apps_image_1_i
or
start_boot_apps_image_2_i
IDLE
LMMI_ WRITE_ NOOP
PORT_ REQUEST_ FORCE
PORT_ STATUS
Port Active =Available
Port Active = Inactive or Disable
Port Active = Exclusive
MSPI_ MULTIBOOT_ ADDR
REFRESH Command
Load the desired Alternate Pattern. Otherwise, fall back to Golden Pattern.
YES
REFRESH command successfully executed
NO
PORT_ REQUEST_ RELEASE
PORT_ STATUS
NO
Port Active = Availabe & release_lmmi=1
YES

Figure 3.2. State Diagram of Refresh Controller Instance

4. Reference Design Parameter Description

The Multi-Boot reference design includes parameters shown in Table 4.1. You can modify the parameters defined in the cmd_list.v file according to the start address where you stored the primary or alternate patterns.

Table 4.1. Parameters in cmd_list.v

Parameter	Default Value	Description
BOOT_ADDRESS_15_TO_0_PRIMARY	16'h0000	Boot address for Primary Pattern. Lower 16 bits of the 32-bit address.
BOOT_ADDRESS_31_TO_16_PRIMARY	16'h0000	Boot address for Primary Pattern. Upper 16 bits of the 32-bit address.
BOOT_ADDRESS_15_TO_0_APPS_IMAGE_1	16'h0000	Boot address for Alternate Pattern 1. Lower 16 bits of the 32-bit address.
BOOT_ADDRESS_31_TO_16_APPS_IMAGE_1	16'h0180	Boot address for Alternate Pattern 1. Upper 16 bits of the 32-bit address.
BOOT_ADDRESS_15_TO_0_APPS_IMAGE_2	16'h0000	Boot address for Alternate Pattern 2. Lower 16 bits of the 32-bit address.
BOOT_ADDRESS_31_TO_16_APPS_IMAGE_2	16'h0240	Boot address for Alternate Pattern 2. Upper 16 bits of the 32-bit address.

5. Signal Description

The input/output interface signals for the top.v are shown in Table 5.1.

Table 5.1. Primary I/O

Port Name	I/O	Width	Description
LED_0	Out	1	Output to blink LED D45
LED_1	Out	1	Output to blink LED D46
LED_2	Out	1	Output to blink LED D47
start_boot_primary_i	In	1	Active low input port. Toggle the SW13 push button to trigger the boot for primary pattern.
start_boot_apps_image_1_i	In	1	Active low input port. Toggle the SW14 push button to trigger the boot for Alternate Pattern 1.
start_boot_apps_image_2_i	In	1	Active low input port. Toggle the SW15 push button to trigger the boot for Alternate Pattern 2.
lmmi_resetn_pad	In	1	An active low asynchronous reset port. Toggle the SW12 push button to assert the reset.

6. Running the Reference Design

This section describes how to run the Multi-Boot reference design using the Lattice Radiant software. For more details on the Lattice Radiant software, refer to the Lattice Radiant Software User Guide.

The project consists of three design implementations that can be used to distinguish the pattern running on the device. They differ in a unique 20-bit apps_image register value and the output port(s) to blink LED(s) on board. Below are the apps_image value for each design implementation.

- impl_1: apps_image=0xcafe0 and output port to blink one LED (D45)
- impl_2: apps_image=0xcafe1 and output ports to blink two LEDs (D45 and D46)
- impl_3: apps_image=0xcafe2 and output ports to blink three LEDs (D45, D46, and D47)

6.1. Compiling the Reference Design

The reference design file includes the pre-compiled files and bitstreams in .bit and .mcs file formats, for you to start quickly. However, you can recompile the reference design in newer Lattice Radiant versions after adding or modifying the user logics if you choose to do so. To do that, you can set the design implementation as active implementation, make the desired changes, and recompile the design through the standard compilation flow.

Note: mcs. files are 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 its printable feature, which make it good for record-keeping. This type of file is not directly consumable by the utilities supporting it.

6.2. Generating the Bitstream File

This section provides the procedure of creating your FPGA bitstream file using the Lattice Radiant Software.
To create the FPGA bitstream file, follow steps below.

1. Open the Lattice Radiant software, as shown in Figure 6.1.

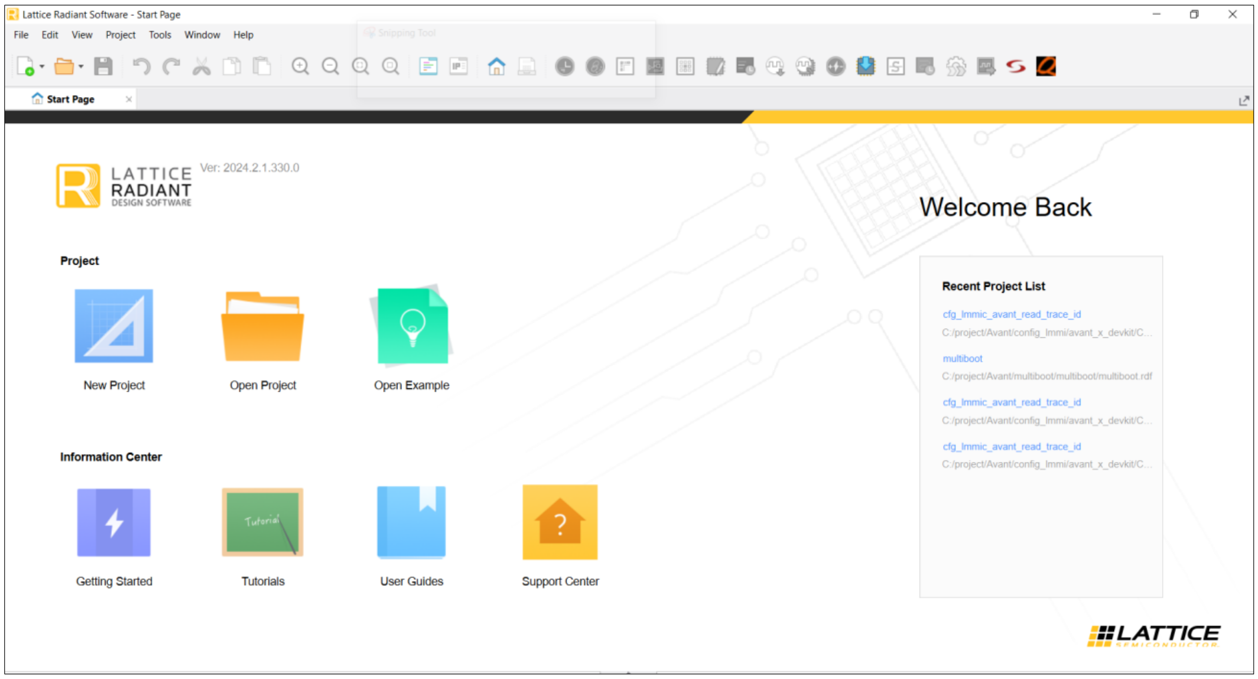
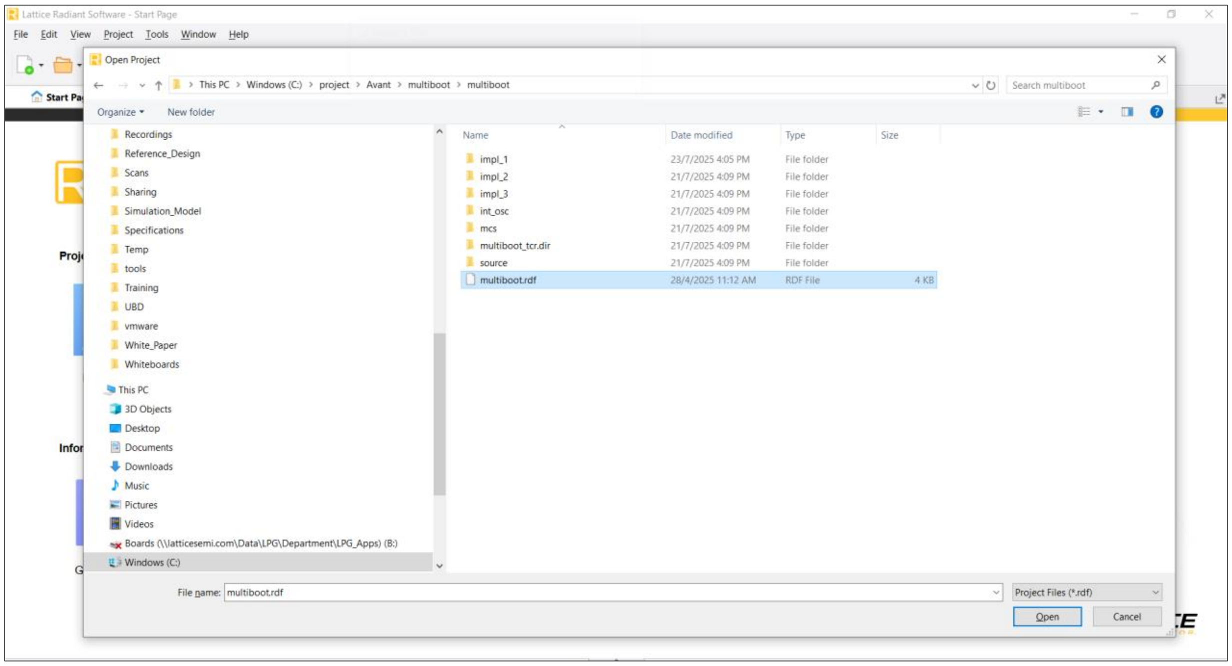

Lattice Radiant Software - Start Page
File
Edit
View
Project
Tools
Window
Help
Start Page
LATTICE RADIANT DESIGN SOFTWARE
Ver: 2024.2.1.330.0
Welcome Back
Project
New Project
Open Project
Open Example
Information Center
Getting Started
Tutorials
User Guides
Support Center
Recent Project List
cfg_lmmic_avant_read_trace_id
C:/project/Avant/config_lmmi/avant_x_devkit/C...
multiboot
C:/project/Avant/multiboot/multiboot/multiboot.rdf
cfg_lmmic_avant_read_trace_id
C:/project/Avant/config_lmmi/avant_x_devkit/C...
cfg_lmmic_avant_read_trace_id
C:/project/Avant/config_lmmi/avant_x_devkit/C...

Figure 6.1. Lattice Radiant Software

2. Click **File > Open Project**. From the project database, browse to the reference design project, the multiboot folder and open the Lattice Radiant project file, lmmi2mspi.rdf, as shown in Figure 6.2.

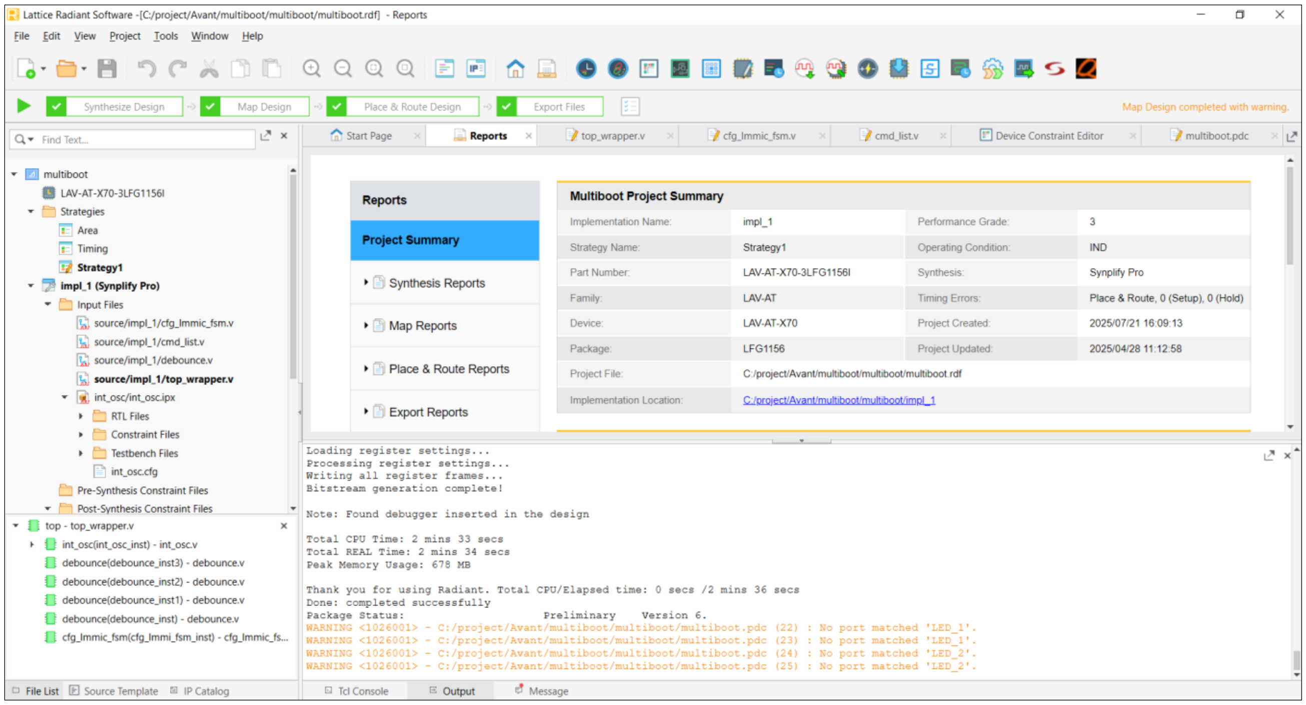


File name: multiboot.rdf

Name	Date modified	Type	Size
impl_1	23/7/2025 4:05 PM	File folder	
impl_2	21/7/2025 4:09 PM	File folder	
impl_3	21/7/2025 4:09 PM	File folder	
int_osc	21/7/2025 4:09 PM	File folder	
mcs	21/7/2025 4:09 PM	File folder	
multiboot_tcr.dir	21/7/2025 4:09 PM	File folder	
source	21/7/2025 4:09 PM	File folder	
multiboot.rdf	28/4/2025 11:12 AM	RDF File	4 KB

Figure 6.2. Open Project File

3. Click **Export Files** to generate the bit file (Figure 6.3). View the log message in the Export Reports folder for the generated bitstream. Once the compilation is successful, the generated bit file is in the project implementation folder, for example, multiboot\impl_1\multiboot_impl_1.bit. Repeat this step for others design implementation if you need to recompile the design.



Map Design completed with warning.

Multiboot Project Summary

Implementation Name:	impl_1	Performance Grade:	3
Strategy Name:	Strategy1	Operating Condition:	IND
Part Number:	LAV-AT-X70-3LFG1156I	Synthesis:	Synplify Pro
Family:	LAV-AT	Timing Errors:	Place & Route, 0 (Setup), 0 (Hold)
Device:	LAV-AT-X70	Project Created:	2025/07/21 16:09:13
Package:	LFG1156	Project Updated:	2025/04/28 11:12:58
Project File:	C:/project/Avant/multiboot/multiboot/multiboot.rdf		
Implementation Location:	C:/project/Avant/multiboot/multiboot/impl_1		

```
Loading register settings...
Processing register settings...
Writing all register frames...
Bitstream generation complete!

Note: Found debugger inserted in the design

Total CPU Time: 2 mins 33 secs
Total REAL Time: 2 mins 34 secs
Peak Memory Usage: 678 MB

Thank you for using Radiant. Total CPU/Elapsed time: 0 secs /2 mins 36 secs
Done: completed successfully
Package Status:                     Preliminary    Version 6.
WARNING <1026001> - C:/project/Avant/multiboot/multiboot/multiboot.pdc (22) : No port matched 'LED_1'.
WARNING <1026001> - C:/project/Avant/multiboot/multiboot/multiboot.pdc (23) : No port matched 'LED_1'.
WARNING <1026001> - C:/project/Avant/multiboot/multiboot/multiboot.pdc (24) : No port matched 'LED_2'.
WARNING <1026001> - C:/project/Avant/multiboot/multiboot/multiboot.pdc (25) : No port matched 'LED_2'.
```

Figure 6.3. Generated Bitstream Log

6.3. Generating Initial Multi-Boot Programming File

This section describes how to generate the initial Multi-Boot programming file.
Note: You can use the provided Multi-Boot programming file located in the \multiboot\mcs\multiboot.mcs directory and skip the steps below.

Follow steps below to generate the single Multi-Boot programing file(.mcs).

1. Launch the Radiant Deployment Tool from Radiant Programmer by clicking **Tools** > **Deployment Tool** (Figure 6.4).

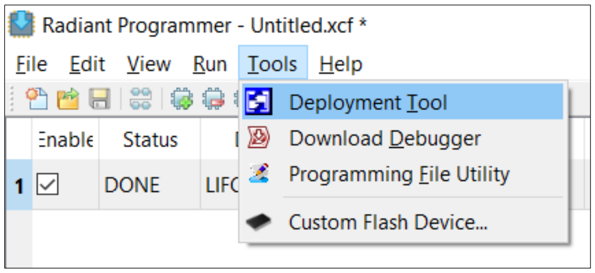
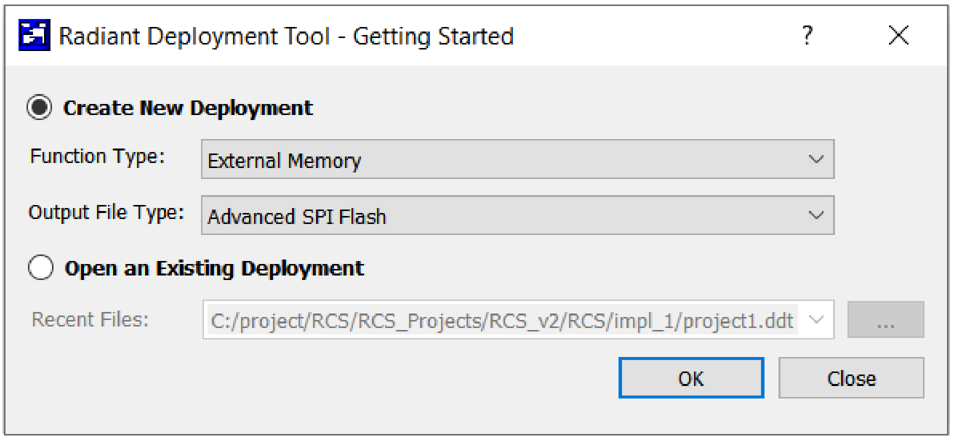


Figure 6.4. Launch Radiant Deployment Tool from Radiant Programmer

2. In the Radiant Deployment Tool window, select External Memory for **Function Type** and Advanced SPI Flash for **Output File Type**. Click **OK** (Figure 6.5).


Radiant Deployment Tool - Getting Started

Create New Deployment

Function Type: External Memory

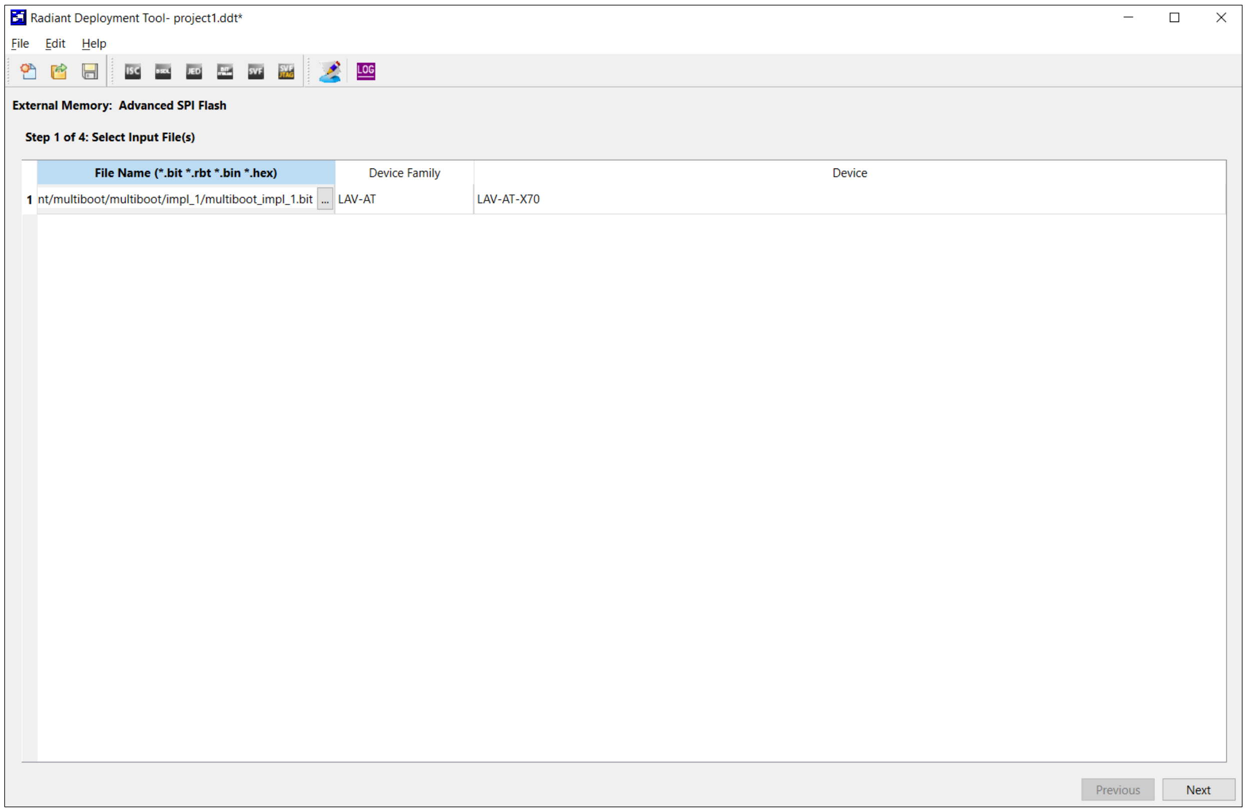
Output File Type: Advanced SPI Flash

Open an Existing Deployment

Recent Files: C:/project/RCS/RCS_Projects/RCS_v2/RCS/impl_1/project1.ddt

Figure 6.5. Radiant Deployment Tool – Getting Started

3. A four-step wizard opens, showing: External Memory: Advanced SPI Flash.
   a. In Step 1 of 4: Select Input File(s), select a .bit file for **File Name** and click **Next** (Figure 6.6). This is the Primary Image.



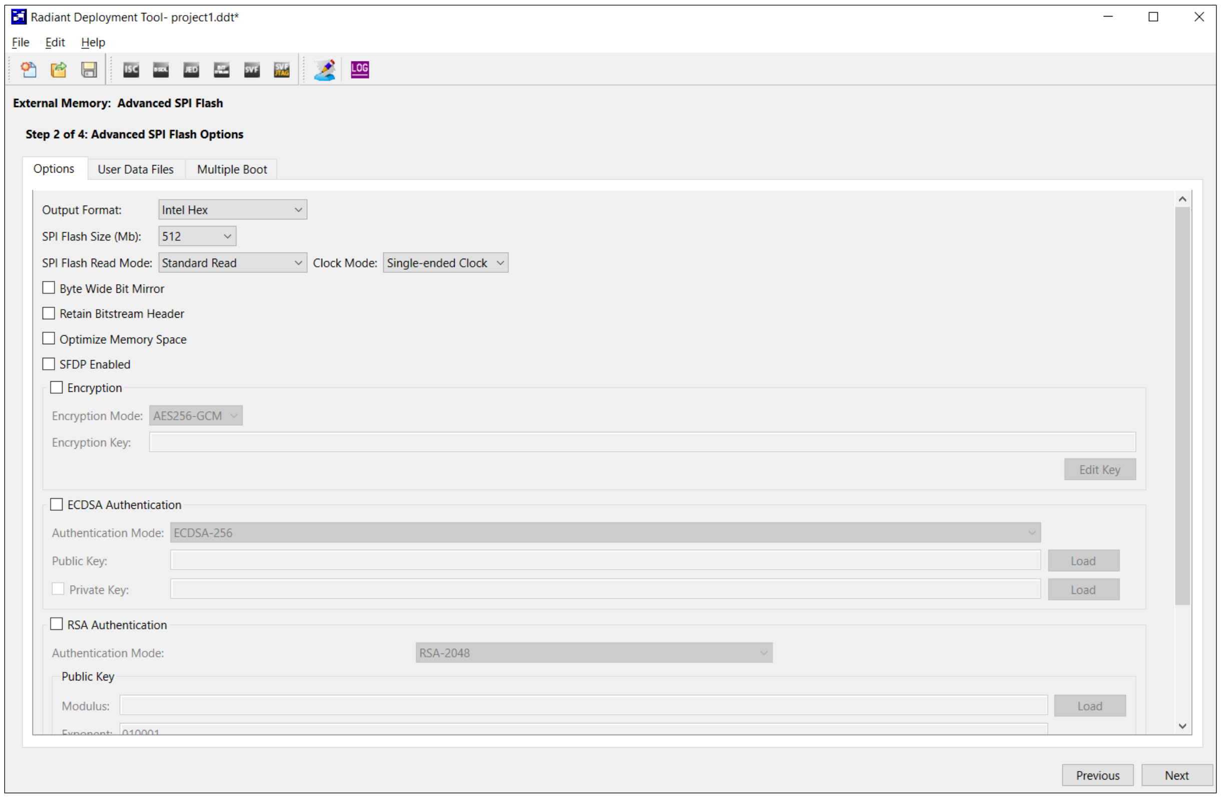
External Memory: Advanced SPI Flash

Step 1 of 4: Select Input File(s)

	File Name (*.bit *.rbt *.bin *.hex)	Device Family	Device
1	nt/multiboot/multiboot/impl_1/multiboot_impl_1.bit	LAV-AT	LAV-AT-X70

Figure 6.6. Step 1 of 4: Select Input File(s)

b. In Step 2 of 4: Advanced SPI Flash Option, under the **Options** tab, select the following options (Figure 6.7):
   - Output Format: Intel Hex
   - SPI Flash Size (Mb): 512
   - SPI Flash Read Mode: Standard Read
   - Clock Mode: Single-ended Clock


Radiant Deployment Tool- project1.ddt*

File Edit Help

External Memory: Advanced SPI Flash

Step 2 of 4: Advanced SPI Flash Options

Options | User Data Files | Multiple Boot

Output Format: Intel Hex

SPI Flash Size (Mb): 512

SPI Flash Read Mode: Standard Read Clock Mode: Single-ended Clock

- [ ] Byte Wide Bit Mirror
- [ ] Retain Bitstream Header
- [ ] Optimize Memory Space
- [ ] SFDP Enabled
- [ ] Encryption

Encryption Mode: AES256-GCM

Encryption Key:

Edit Key

- [ ] ECDSA Authentication

Authentication Mode: ECDSA-256

Public Key: Load

Private Key: Load

- [ ] RSA Authentication

Authentication Mode: RSA-2048

Public Key

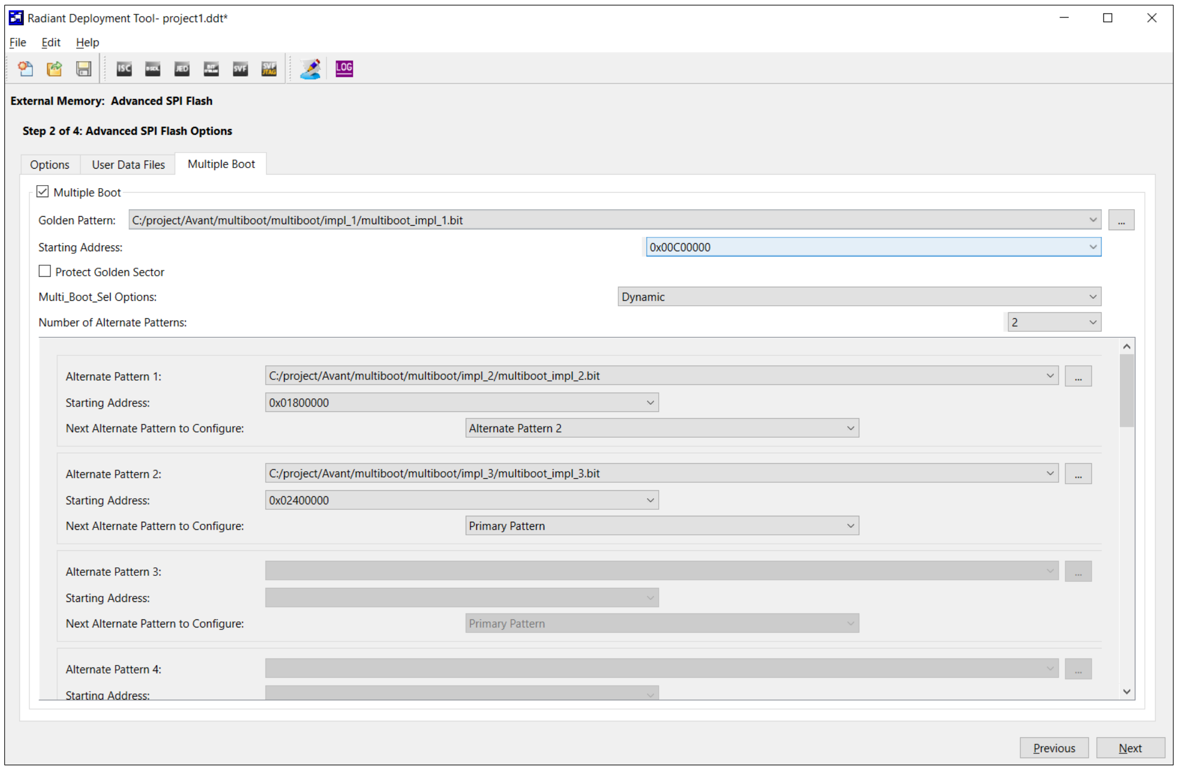
Modulus: Load

Exponent: 010001

Previous Next

Figure 6.7. Options Tab of Step 2 of 4: Advanced SPI Flash Option

c. In Step 2 of 4: Advanced SPI Flash Option, under the **Multiple Boot** tab, select the following options (Figure 6.8):
   - Check Multiple Boot.
   - Golden Pattern:
     Select the multiboot_impl_1.bit file in the multiboot/impl_1 folder.
   - Starting Address: 0x00C00000.
   - Multi_Boot_Sel Options: Dynamic
   - Number of Alternate Patterns: 2 – to program two alternative images in the external SPI flash.
   - Alternate Pattern 1:
     Select the multiboot_impl_2.bit file in the multiboot/impl_2 folder.
   - Starting Address: 0x01800000 – make sure the value is the same as the value defined for parameter BOOT_ADDRESS_15_TO_0_APPS_IMAGE_1 and BOOT_ADDRESS_31_TO_16_APPS_IMAGE_1 in cmd_list.v.
   - Next Alternate Pattern to Configure: Alternate Pattern 2
     Note: You can simply select any available option because it no longer affects the next pattern to load once the Dynamic mode is selected.
   - Alternate Pattern 2:
   - Select the multiboot_impl_3.bit file in the multiboot/impl_3 folder.
   - Starting Address: 0x02400000 – make sure the value is the same as the value defined for parameter BOOT_ADDRESS_15_TO_0_APPS_IMAGE_2 and BOOT_ADDRESS_31_TO_16_APPS_IMAGE_2 in cmd_list.v.
   - Next Alternate Pattern to Configure: Primary Pattern
     Note: You can simply select any available option because it no longer affects the next pattern to load once the Dynamic mode is selected.
   - Click **Next**.



External Memory: Advanced SPI Flash

Step 2 of 4: Advanced SPI Flash Options

☑ Multiple Boot

Golden Pattern: C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit

Starting Address: 0x00C00000

☐ Protect Golden Sector

Multi_Boot_Sel Options: Dynamic

Number of Alternate Patterns: 2

Alternate Pattern 1: C:/project/Avant/multiboot/multiboot/impl_2/multiboot_impl_2.bit

Starting Address: 0x01800000

Next Alternate Pattern to Configure: Alternate Pattern 2

Alternate Pattern 2: C:/project/Avant/multiboot/multiboot/impl_3/multiboot_impl_3.bit

Starting Address: 0x02400000

Next Alternate Pattern to Configure: Primary Pattern

Alternate Pattern 3:

Starting Address:

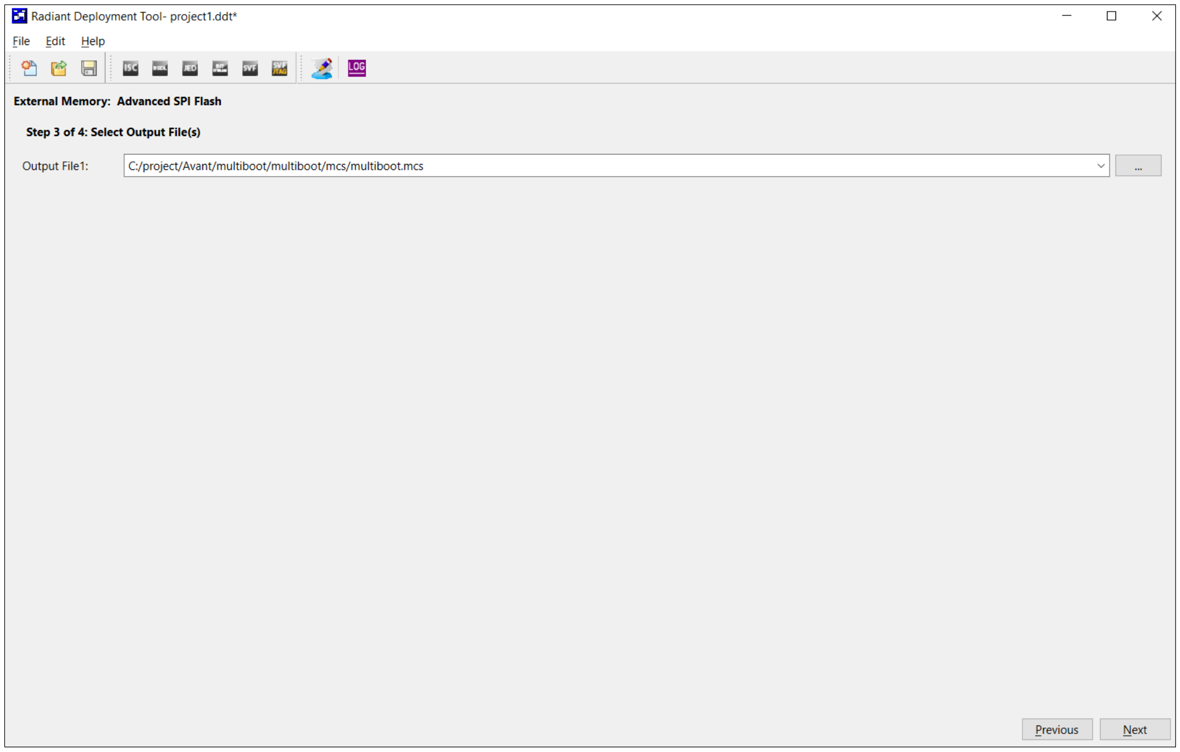
Next Alternate Pattern to Configure: Primary Pattern

Alternate Pattern 4:

Starting Address:

Figure 6.8. Multiple Boot Tab of Step 2 of 4: Advanced SPI Flash Option

d. In Step 3 of 4: Select Output File(s)
   - Browse to the location to store the .mcs file in the Output File1 field and click **Next** (Figure 6.9).



External Memory: Advanced SPI Flash

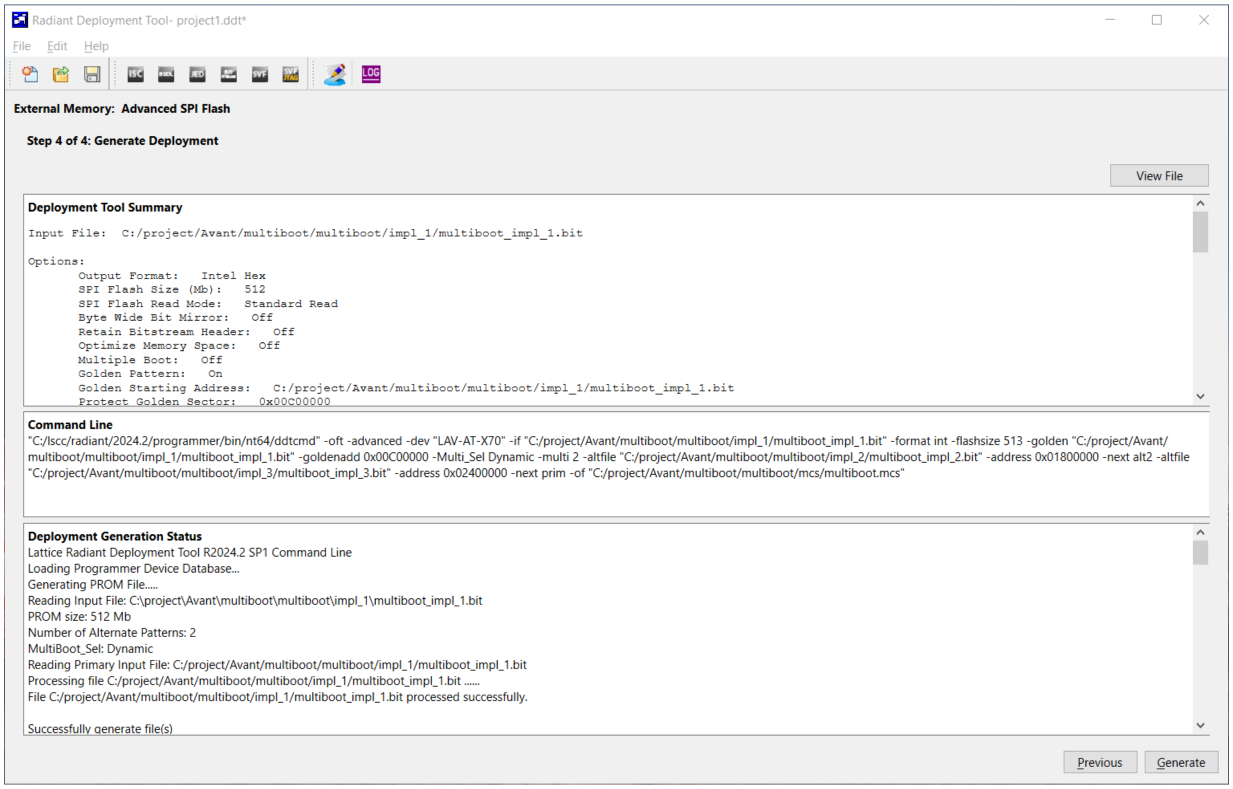
Step 3 of 4: Select Output File(s)

Output File1: C:/project/Avant/multiboot/multiboot/mcs/multiboot.mcs

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

e. In Step 4 of 4: Generate Deployment

- Click **Generate** to generate the .mcs file. You can close the window when the Deployment Tool prompts Lattice Deployment Tool has exited successfully (Figure 6.10).



External Memory: Advanced SPI Flash

Step 4 of 4: Generate Deployment

Deployment Tool Summary

```
Input File:  C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit

Options:
       Output Format:   Intel Hex
       SPI Flash Size (Mb):   512
       SPI Flash Read Mode:   Standard Read
       Byte Wide Bit Mirror:   Off
       Retain Bitstream Header:   Off
       Optimize Memory Space:   Off
       Multiple Boot:   Off
       Golden Pattern:   On
       Golden Starting Address:   C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit
       Protect Golden Sector:   0x00C00000
```

Command Line

```
"C:/lscc/radiant/2024.2/programmer/bin/nt64/ddtcmd" -oft -advanced -dev "LAV-AT-X70" -if "C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit" -format int -flashsize 513 -golden "C:/project/Avant/
multiboot/multiboot/impl_1/multiboot_impl_1.bit" -goldenadd 0x00C00000 -Multi_Sel Dynamic -multi 2 -altfile "C:/project/Avant/multiboot/multiboot/impl_2/multiboot_impl_2.bit" -address 0x01800000 -next alt2 -altfile
"C:/project/Avant/multiboot/multiboot/impl_3/multiboot_impl_3.bit" -address 0x02400000 -next prim -of "C:/project/Avant/multiboot/multiboot/mcs/multiboot.mcs"
```

Deployment Generation Status

```
Lattice Radiant Deployment Tool R2024.2 SP1 Command Line
Loading Programmer Device Database...
Generating PROM File.....
Reading Input File: C:\project\Avant\multiboot\multiboot\impl_1\multiboot_impl_1.bit
PROM size: 512 Mb
Number of Alternate Patterns: 2
MultiBoot_Sel: Dynamic
Reading Primary Input File: C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit
Processing file C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit ......
File C:/project/Avant/multiboot/multiboot/impl_1/multiboot_impl_1.bit processed successfully.

Successfully generate file(s)
```

Figure 6.10. Step 4 of 4: Generate Deployment

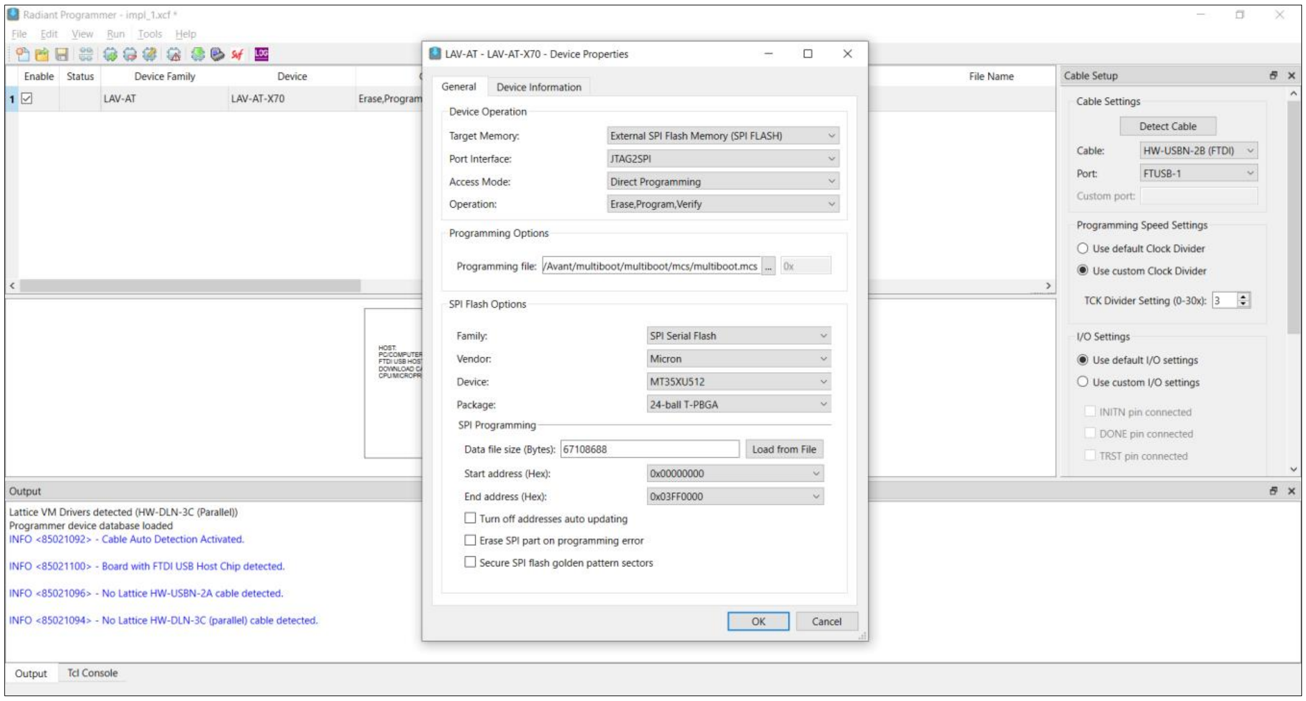
7. Implementing the Reference Design on Board

7.1. Hardware Requirements

- Avant-G/X Versa Board
- USB-A to mini-B Programming Cable

7.2. Programming .mcs into External SPI Flash

1. Launch Radiant Programmer and select **Edit** > **Device Properties**.
2. Select the device properties settings, as shown in Figure 7.1. Click **OK**.
3. In the Cable Settings, select FTUSB-1 for **Port**.
4. Click **Program Device** to start programming the board.


Radiant Programmer - impl_1.xcf *
File Edit View Run Tools Help
Enable Status Device Family Device File Name
1 ☑ LAV-AT LAV-AT-X70 Erase,Program
Output
Tcl Console

LAV-AT - LAV-AT-X70 - Device Properties

General | Device Information

Device Operation
Target Memory: External SPI Flash Memory (SPI FLASH)
Port Interface: JTAG2SPI
Access Mode: Direct Programming
Operation: Erase,Program,Verify

Programming Options
Programming file: /Avant/multiboot/multiboot/mcs/multiboot.mcs

SPI Flash Options
Family: SPI Serial Flash
Vendor: Micron
Device: MT35XU512
Package: 24-ball T-PBGA

SPI Programming
Data file size (Bytes): 67108688
Start address (Hex): 0x00000000
End address (Hex): 0x03FF0000
☐ Turn off addresses auto updating
☐ Erase SPI part on programming error
☐ Secure SPI flash golden pattern sectors

OK | Cancel

Cable Setup
Cable Settings
Detect Cable
Cable: HW-USBN-2B (FTDI)
Port: FTUSB-1
Custom port:

Programming Speed Settings
○ Use default Clock Divider
● Use custom Clock Divider
TCK Divider Setting (0-30x): 3

I/O Settings
● Use default I/O settings
○ Use custom I/O settings
☐ INITN pin connected
☐ DONE pin connected
☐ TRST pin connected

```
Lattice VM Drivers detected (HW-DLN-3C (Parallel))
Programmer device database loaded
INFO <85021092> - Cable Auto Detection Activated.

INFO <85021100> - Board with FTDI USB Host Chip detected.

INFO <85021096> - No Lattice HW-USBN-2A cable detected.

INFO <85021094> - No Lattice HW-DLN-3C (parallel) cable detected.
```

Figure 7.1. Device Properties Window

7.3. Enabling 32-bit MSPI Address and Command Mode

For the multi-boot configuration mode, if the configuration memory is 256 Mb or larger, both the 32-bit SPIM address and the 32-bit SPIM Command option bits must be set to 1.

To do this, first enable the MSPI_ADDRESS_32BIT and MSPI_COMMAND_32BIT settings in the Global tab of the Lattice Radiant software's Device Constraint Editor. These settings are reflected in the .fea file generated after compiling your project in the Lattice Radiant software.

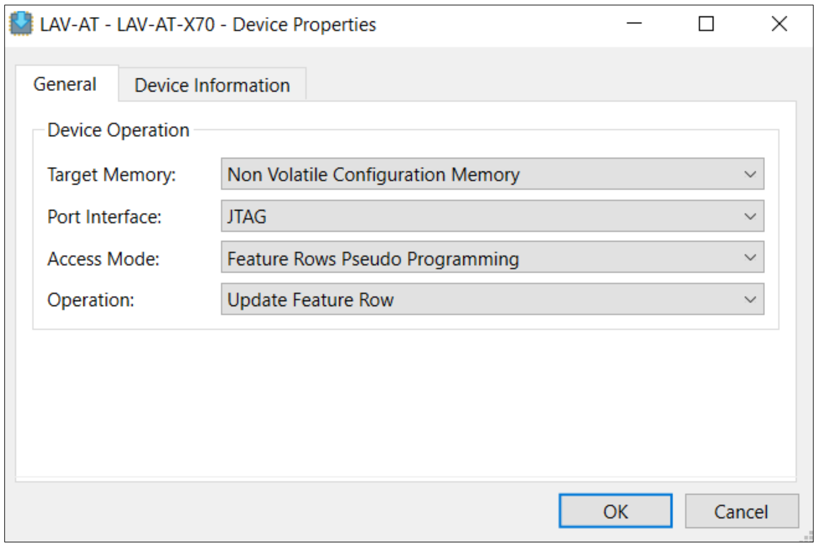
You can then use this .fea file to program the user OTP settings of the Lattice Avant device. If these bits are not set correctly, the device may fail to boot from Alternate Patterns stored beyond 128 Mb, and the fail-safe mechanism may not function as intended.

7.3.1. Setting 32-bit MSPI Address and Command in Non-Volatile Configuration Memory Using Pesudo Programming Mode

This guide allows you to test and ensure that the MSPI configuration from an address beyond 0xFF_FFFF can succeed and therefore the fail-safe mechanism can function properly. The option bits are set in volatile SRAM memory. Setting these option bits is essential to retain the settings in SRAM during reconfiguration. You can perform this test before committing the settings permanently in non-volatile configuration memory.

The following are steps to set the 32-bit MSPI address and 32-bit MSPI commands option bits:

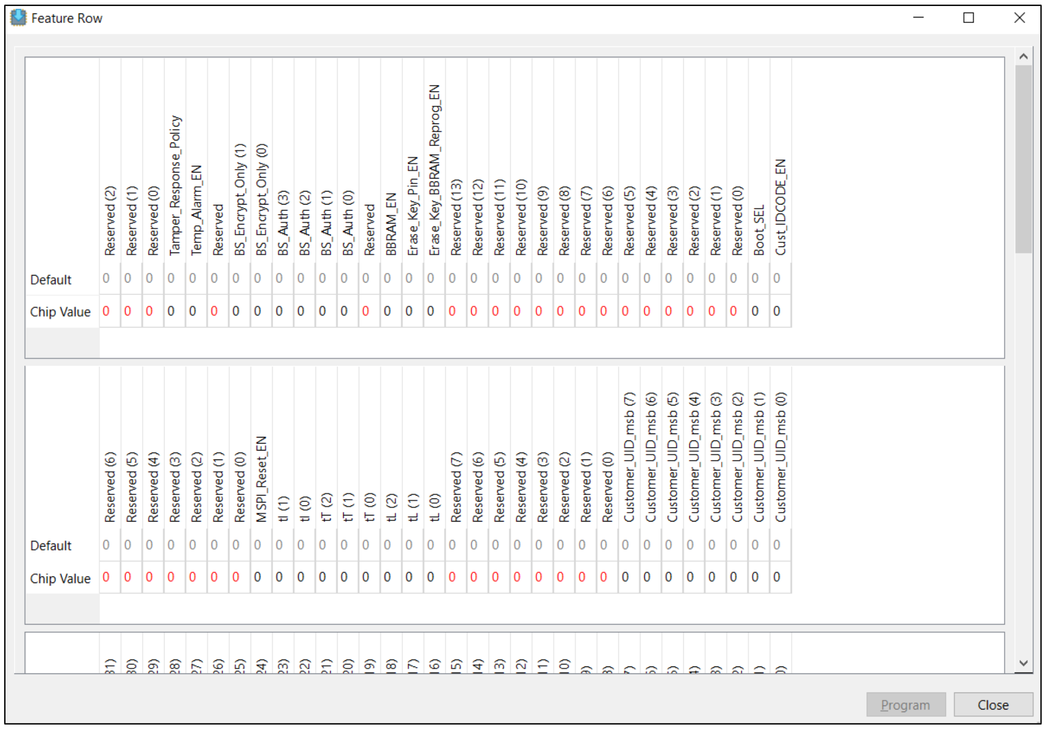
1. Run the Lattice Radiant Programmer.
2. Select the Device Properties settings, as shown in Figure 7.2. Click **OK**.
    - Target Memory: Non Volatile Configuration Memory
    - Port Interface: JTAG
    - Access Mode: Feature Row Pseudo Programming
    - Operation: Update Feature Row



Target Memory: Non Volatile Configuration Memory
Port Interface: JTAG
Access Mode: Feature Rows Pseudo Programming
Operation: Update Feature Row

Figure 7.2. Device Properties Window for Feature Row Pseudo Programming

3. In Cable Settings, select FTUSB-1 for **Port**.
4. Click **Run** > **Program Device**. The Feature Row window appears (Figure 7.3).

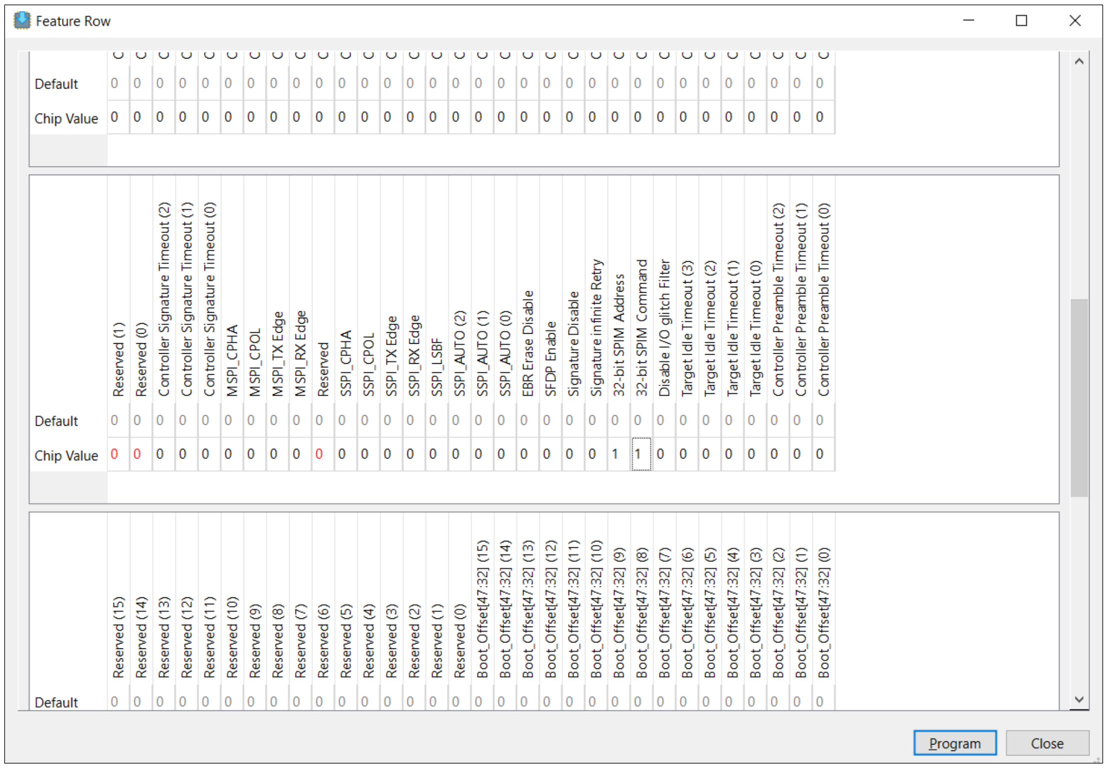


	Default	Chip Value
Reserved (2)	0	0
Reserved (1)	0	0
Reserved (0)	0	0
Tamper_Response_Policy	0	0
Temp_Alarm_EN	0	0
Reserved	0	0
BS_Encrypt_Only (1)	0	0
BS_Encrypt_Only (0)	0	0
BS_Auth (3)	0	0
BS_Auth (2)	0	0
BS_Auth (1)	0	0
BS_Auth (0)	0	0
Reserved	0	0
BBRAM_EN	0	0
Erase_Key_Pin_EN	0	0
Erase_Key_BBRAM_Reprog_EN	0	0
Reserved (13)	0	0
Reserved (12)	0	0
Reserved (11)	0	0
Reserved (10)	0	0
Reserved (9)	0	0
Reserved (8)	0	0
Reserved (7)	0	0
Reserved (6)	0	0
Reserved (5)	0	0
Reserved (4)	0	0
Reserved (3)	0	0
Reserved (2)	0	0
Reserved (1)	0	0
Reserved (0)	0	0
Boot_SEL	0	0
Cust_IDCODE_EN	0	0

	Default	Chip Value
Reserved (6)	0	0
Reserved (5)	0	0
Reserved (4)	0	0
Reserved (3)	0	0
Reserved (2)	0	0
Reserved (1)	0	0
Reserved (0)	0	0
MSPI_Reset_EN	0	0
tI (1)	0	0
tI (0)	0	0
tT (2)	0	0
tT (1)	0	0
tT (0)	0	0
tL (2)	0	0
tL (1)	0	0
tL (0)	0	0
Reserved (7)	0	0
Reserved (6)	0	0
Reserved (5)	0	0
Reserved (4)	0	0
Reserved (3)	0	0
Reserved (2)	0	0
Reserved (1)	0	0
Reserved (0)	0	0
Customer_UID_msb (7)	0	0
Customer_UID_msb (6)	0	0
Customer_UID_msb (5)	0	0
Customer_UID_msb (4)	0	0
Customer_UID_msb (3)	0	0
Customer_UID_msb (2)	0	0
Customer_UID_msb (1)	0	0
Customer_UID_msb (0)	0	0

Figure 7.3. Feature Row Window

5. Scroll down to locate the 32-bit SPIM Address and 32-bit SPIM Command bit.
6. Click on the chip values associated with 32-bit SPIM Address and 32-bit SPIM Command to change the values from 0 to 1 (Figure 7.4).



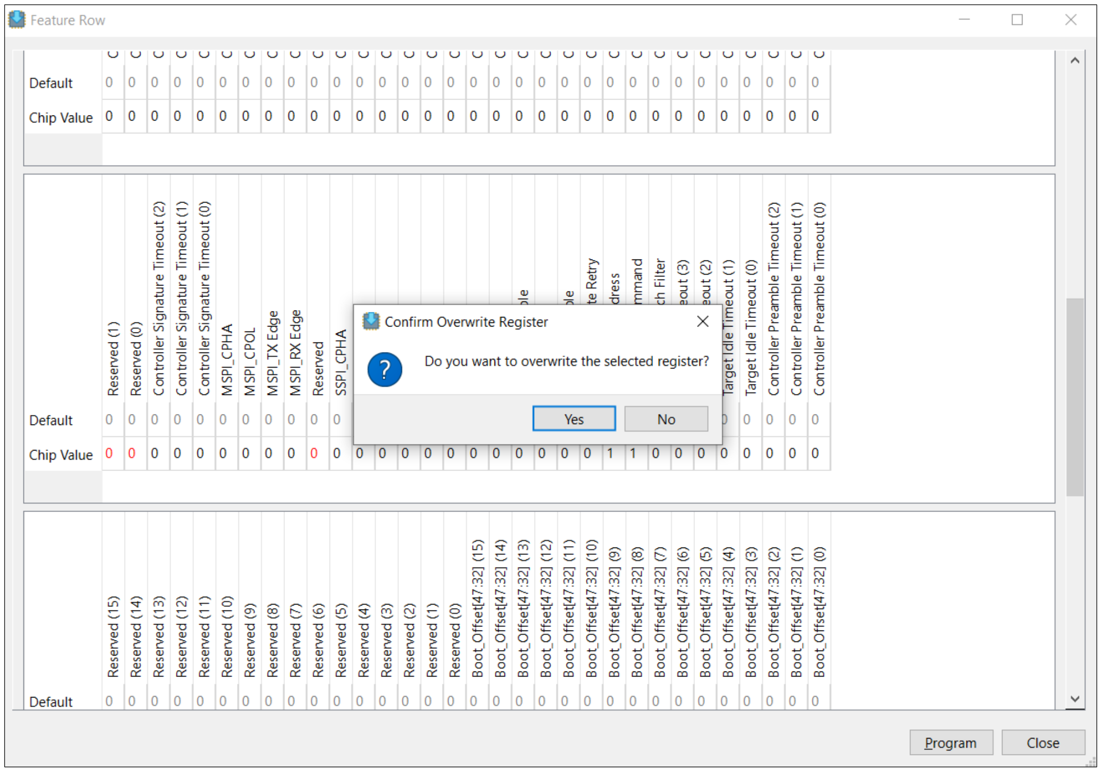
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Chip Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	Default	Chip Value
Reserved (1)	0	0
Reserved (0)	0	0
Controller Signature Timeout (2)	0	0
Controller Signature Timeout (1)	0	0
Controller Signature Timeout (0)	0	0
MSPI_CPHA	0	0
MSPI_CPOL	0	0
MSPI_TX Edge	0	0
MSPI_RX Edge	0	0
Reserved	0	0
SSPI_CPHA	0	0
SSPI_CPOL	0	0
SSPI_TX Edge	0	0
SSPI_RX Edge	0	0
SSPI_LSBF	0	0
SSPI_AUTO (2)	0	0
SSPI_AUTO (1)	0	0
SSPI_AUTO (0)	0	0
EBR Erase Disable	0	0
SFDP Enable	0	0
Signature Disable	0	0
Signature in finite Retry	0	0
32-bit SPIM Address	0	1
32-bit SPIM Command	0	1
Disable I/O glitch Filter	0	0
Target Idle Timeout (3)	0	0
Target Idle Timeout (2)	0	0
Target Idle Timeout (1)	0	0
Target Idle Timeout (0)	0	0
Controller Preamble Timeout (2)	0	0
Controller Preamble Timeout (1)	0	0
Controller Preamble Timeout (0)	0	0

	Default
Reserved (15)	0
Reserved (14)	0
Reserved (13)	0
Reserved (12)	0
Reserved (11)	0
Reserved (10)	0
Reserved (9)	0
Reserved (8)	0
Reserved (7)	0
Reserved (6)	0
Reserved (5)	0
Reserved (4)	0
Reserved (3)	0
Reserved (2)	0
Reserved (1)	0
Reserved (0)	0
Boot_Offset[47:32] (15)	0
Boot_Offset[47:32] (14)	0
Boot_Offset[47:32] (13)	0
Boot_Offset[47:32] (12)	0
Boot_Offset[47:32] (11)	0
Boot_Offset[47:32] (10)	0
Boot_Offset[47:32] (9)	0
Boot_Offset[47:32] (8)	0
Boot_Offset[47:32] (7)	0
Boot_Offset[47:32] (6)	0
Boot_Offset[47:32] (5)	0
Boot_Offset[47:32] (4)	0
Boot_Offset[47:32] (3)	0
Boot_Offset[47:32] (2)	0
Boot_Offset[47:32] (1)	0
Boot_Offset[47:32] (0)	0

Figure 7.4. Change the Values of 32-bit SPIM Address and 32-bit SPIM Command

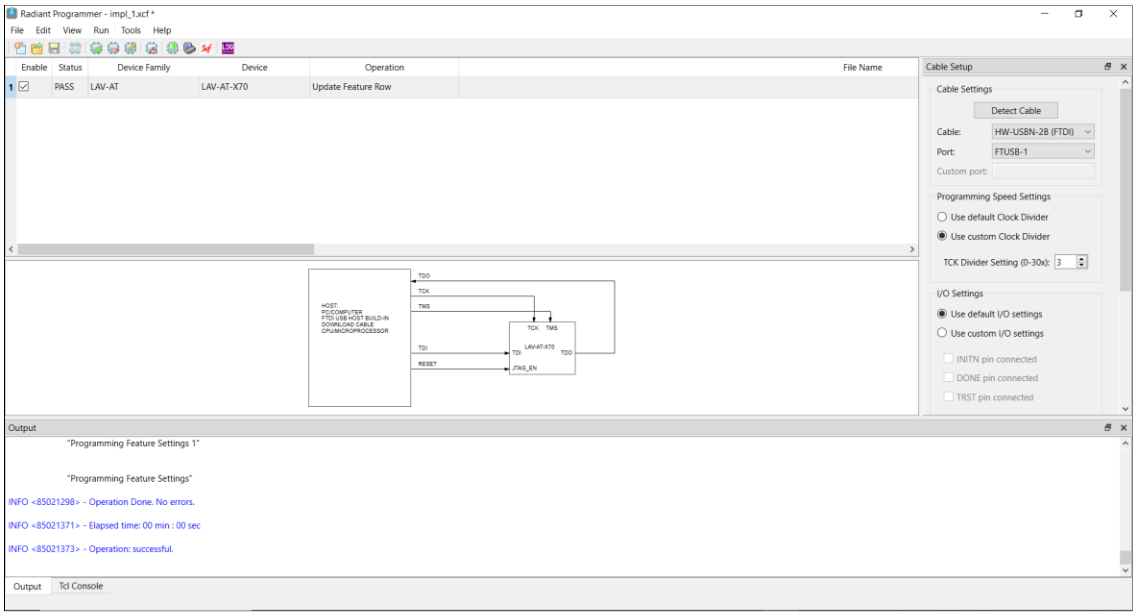
7. Click **Program**.
8. The Confirm Overwrite Register window appears and Click **Yes** (Figure 7.5).



Do you want to overwrite the selected register?

Figure 7.5. Confirm Overwrite Register

9. In the Output Window of Lattice Radiant Programmer, you can see **Operation: successful**, indicating the Feature Row Programming is successful.


Radiant Programmer - impl_1.xcf *
File Edit View Run Tools Help
Detect Cable
HOST: PC/COMPUTER FTDI USB HOST BUILD-IN DOWNLOAD CABLE CPU/MICROPROCESSOR
TDO
TCK
TMS
TDI
RESET
LAV-AT-X70
JTAG_EN
Output
Tcl Console

	Enable	Status	Device Family	Device	Operation	File Name
1	✓	PASS	LAV-AT	LAV-AT-X70	Update Feature Row	

Cable Setup
Cable Settings
Cable: HW-USBN-2B (FTDI)
Port: FTUSB-1
Custom port:
Programming Speed Settings
Use default Clock Divider [✓ Use custom Clock Divider]
TCK Divider Setting (0-30x): 3
I/O Settings
[✓ Use default I/O settings] Use custom I/O settings
INITN pin connected
DONE pin connected
TRST pin connected

```
        "Programming Feature Settings 1"


        "Programming Feature Settings"
INFO <85021298> - Operation Done. No errors.
INFO <85021371> - Elapsed time: 00 min : 00 sec
INFO <85021373> - Operation: successful.
```

Figure 7.6. Output Window Showing Operation: Successful

10. You can repeat steps 4 to 5 to confirm that the 32-bit SPIM Address and 32-bit SPIM Command bits are set to 1.

7.3.2. Setting 32-bit MSPI Address and Command in Non-Volatile Configuration Memory Using Direct Programming Mode

To permanently set the 32-bit SPIM Address and 32-bit SPIM Commands option bits to 1, refer to Appendix C for the steps to do Feature Rows Programming using .fea file in Avant sysCONFIG User Guide (FPGA-TN-02299).

Note: Feature row bits are one-time programmable (OTP). Each bit can only be modified one time. Once the 32-bit SPIM Address and 32-bit SPIM Commands option bits are set to 1, the FPGA can only work with configuration memory of sizes 256 Mb or larger to perform dual-boot, multi-boot, or ping-pong boot.

7.4. Running the Design

7.4.1. Performing Multi-Boot – without Corrupted Pattern

1. The device is configured with Primary Image after the device is powered on and LED D45 blinks once the device is configured successfully. To confirm this:
    a. Ensure the 32-bit SPIM Address and 32-bit SPIM Command bits are set to 1.
    b. Open the multiboot.rdf project located in the multiboot project directory in the Lattice Radiant software.
    c. Launch and run Reveal Analyzer by double-clicking the **multiboot.rva** file (Figure 7.7).

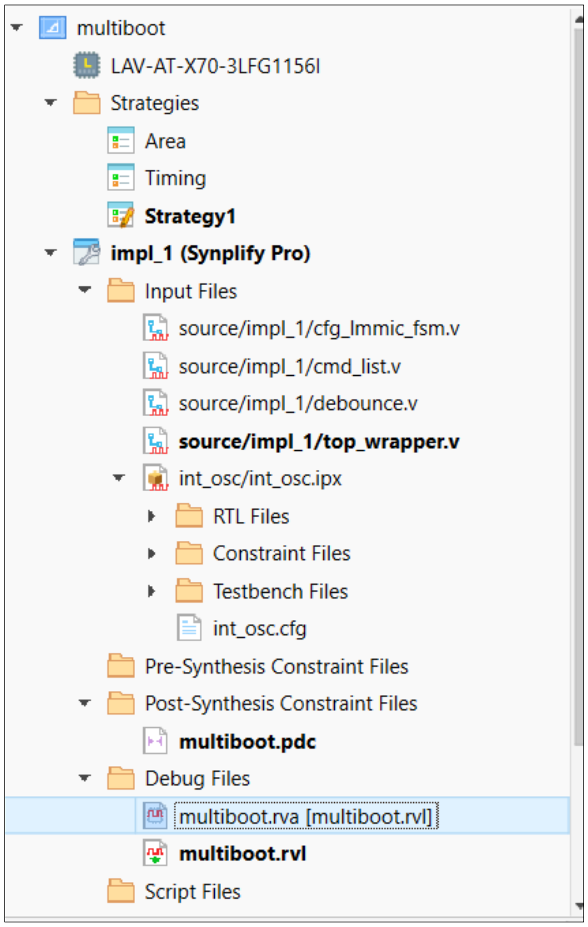
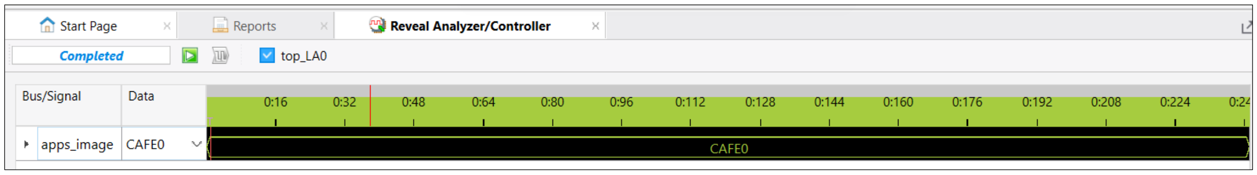


Figure 7.7. multiboot.rva File

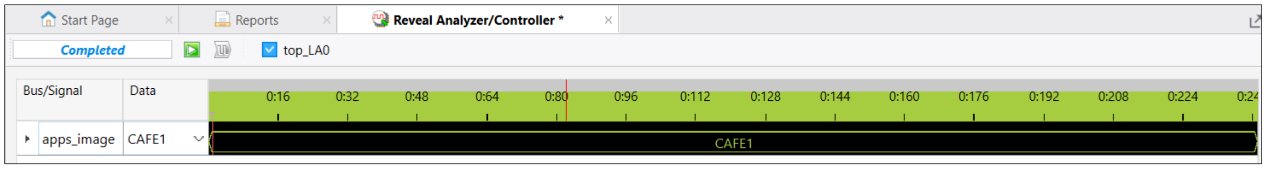
d. Open the waveform tab and click the button. The apps_image displays CAFE0 indicating that Primary Image is running on the device (Figure 7.8).



apps_image CAFE0 CAFE0

Figure 7.8. apps_image Showing CAFE0

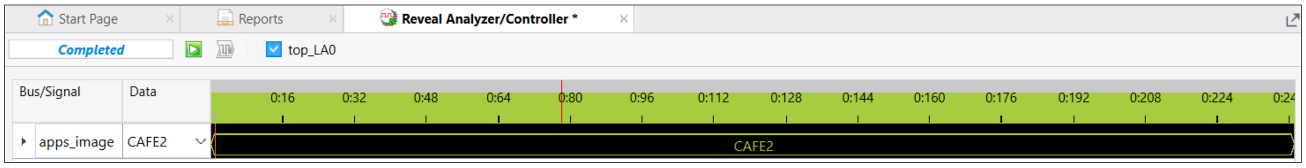
2. Reboot the device with Alternate Pattern 1: Toggle the SW14 push button to reboot or reconfigure the device with Alternate Pattern 1.
3. Run Reveal Analyzer by clicking the button. The apps_image displays CAFE1 and LED D45 and D46 blink, indicating that Alternate Pattern 1 is running on the device (Figure 7.9).



apps_image CAFE1 CAFE1

Figure 7.9. apps_image Showing CAFE1

4. Reboot the device with Alternate Pattern 2: Toggle the SW15 push button to reboot or reconfigure the device with Alternate Pattern 2.
5. Run Reveal Analyzer and the apps_image displays CAFE2. LED D45, D46, and D47 blink, indicating the Alternate Pattern 2 is currently running on the board (Figure 7.10).



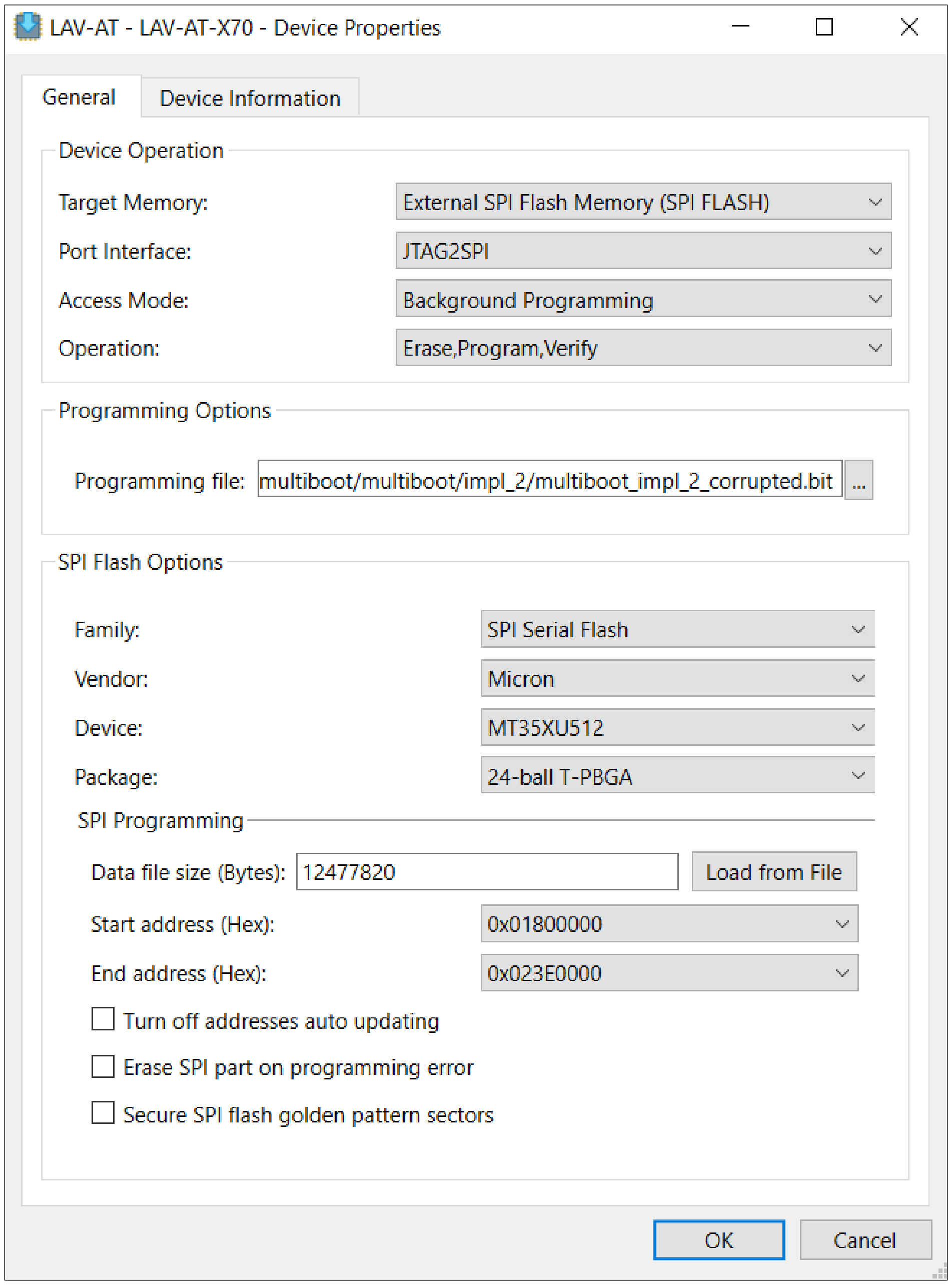
apps_image CAFE2 CAFE2

Figure 7.10. apps_image Showing CAFE2

7.4.2. Performing Multi-Boot – with Corrupted Pattern

To test the fallback to Golden Pattern function, reprogram Alternate Pattern 1 with a corrupted pattern and then boot the board with the corrupted pattern.

1. Close Reveal Analyzer/Controller and open Radiant Programmer.
2. Reprogram a corrupted image, multiboot/impl_2/multiboot_impl_2_corrupted.bit, at Start Address: 0x01800000 (Figure 7.11).


LAV-AT - LAV-AT-X70 - Device Properties

General | Device Information

Device Operation

Target Memory: External SPI Flash Memory (SPI FLASH)
Port Interface: JTAG2SPI
Access Mode: Background Programming
Operation: Erase,Program,Verify

Programming Options

Programming file: multiboot/multiboot/impl_2/multiboot_impl_2_corrupted.bit

SPI Flash Options

Family: SPI Serial Flash
Vendor: Micron
Device: MT35XU512
Package: 24-ball T-PBGA

SPI Programming

Data file size (Bytes): 12477820
Start address (Hex): 0x01800000
End address (Hex): 0x023E0000
☐ Turn off addresses auto updating
☐ Erase SPI part on programming error
☐ Secure SPI flash golden pattern sectors

Figure 7.11. Reprogram Corrupted Image

a. After SPI flash programming is completed, toggle the SW14 push button to reboot or reconfigure the device with Alternate Pattern 1.
b. Run Reveal Analyzer and now the apps_image displays CAFE0 (Figure 7.12). LED D45 blinks, indicating that the device fails to boot from the corrupted Alternate Pattern 1 and falls back to Golden Pattern.

Note: You are seeing the same output as before because the Golden Pattern is set to be the same as the Primary Pattern.

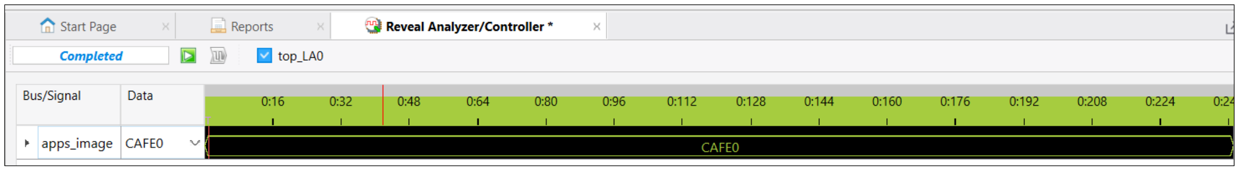


Figure 7.12. apps_image Showing CAFE0

References

- Lattice Avant-E Devices web page
- Lattice Avant-G Devices web page
- Lattice Avant-X Devices web page
- Lattice Avant Platform web page
- Lattice Avant Multi-Boot User Guide (FPGA-TN-02314)
- Avant sysCONFIG User Guide (FPGA-TN-02299)
- Lattice Avant OSC Module User Guide (FPGA-IPUG-02184)
- Lattice Radiant Software Help
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

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

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

Revision History

Revision 0.80, August 2025

Section	Change Summary
All	Preliminary release.