JTAG Embedded Programming Demo Using Raspberry Pi User Guide

Reference Design

FPGA-RD-02317-1.0

July 2025

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Contents

Contents 3
Abbreviations in This Document........ 5
1. Introduction........ 6
1.1. Embedded Programming Source Code Architecture 7
1.2. Source Code Directory 9
2. Modifying the Source Code and Writing the Driver 10
2.1. Updating ispvm_ui.c for File-Based Source Code........ 10
2.1.1. Updating main() Function 10
2.2. Updating ivm_eprom.c for EPROM-Based Source Code 11
2.2.1. Updating main() Function 12
2.3. Updating hardware.c........ 13
2.3.1. writePort() Implementation 14
2.3.2. readPort() Implementation 15
2.3.3. sclock() Implementation........ 15
2.3.4. ispVMDelay() Implementation 15
2.3.5. calibration() Implementation 16
2.4. Creating jtagpins.c........ 16
3. Compiling and Running the Demo........ 18
4. Debugging Tips 21
5. Hardware Validation Summary 23
Appendix A. Generation of .vme File........ 24
A.1. Generating .vme File Using Radiant Programmer 24
A.2. Generating .vme File Using Diamond Programmer and Deployment Tool 26
References........ 30
Technical Support Assistance 31
Revision History........ 32

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Figures

Figure 1.1. Raspberry Pi GPIO JTAG Interface with Lattice Device 6
Figure 1.2. Embedded Programming Source Code Architecture 7
Figure 1.3. JTAG VME Hardware Interface with Lattice FPGA 8
Figure 1.4. Embedded Programming Source Code Directories 9
Figure 1.5. Embedded Programming Source Code Folders for File-Based and EPROM-Based Source Codes....9
Figure 3.1. Sample Compilation....18
Figure 3.2. Example of Running the Executable File in File-Based Demo 19
Figure 3.3. Sample Run without Debug Printing....19
Figure 3.4. Sample Run with Debug Printing 20
Figure 4.1. Clock without Added Delay....21
Figure 4.2. Clock with Added Delay 21
Figure 4.3. Running the Calibration Function 22
Figure 4.4. Sample Hardware Run with Calibration....22
Figure 4.5. Two Clock Cycles in Sample Hardware Run with Calibration....22
Figure A.1. Radiant Programmer .xcf File Generation 24
Figure A.2. Embedded Options Window 24
Figure A.3. Generate Embedded Code Button 25
Figure A.4. Generation of .vme File 25
Figure A.5. Generated .vme File in .xcf File Directory for File-Based System 25
Figure A.6. Generated .vme, .c, and .h Files in .xcf File Directory for EPROM-Based System 25
Figure A.7. Diamond Programmer .xcf File Generation....26
Figure A.8. Accessing the Deployment Tool 26
Figure A.9. JTAG VME Embedded Generation in Deployment Tool 27
Figure A.10. Selecting .xcf File 27
Figure A.11. Embedded File Generation Options 27
Figure A.12. Location of .vme File 28
Figure A.13. Generation of JTAG VME Embedded File 28
Figure A.14. Generated .vme File in .xcf File Directory for File-Based System....28
Figure A.15. Generated .vme, .h, and .c Files in .xcf File Directory for EPROM-Based System 29

Tables

Table 1.1. Existing Files Needed for Embedded Programming....7
Table 1.2. Files to be Created for Embedded Programming....7
Table 2.1. Functions to be Modified in hardware.c....13
Table 2.2. Functions to be Created in jtagpins.c....16
Table 5.1. Hardware Validation Results....23

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
CPU	Central Processing Unit
DMA	Direct Memory Access
EPROM	Erasable Programmable Read-Only Memory
FPGA	Field Programmable Gate Array
FTDI	Future Technology Devices International
GPIO	General Purpose Input Output
JTAG	Joint Test Action Group
MSB	Most Significant Bit
PCB	Printed Circuit Board
SRAM	Static Random-Access Memory
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TRST	Test Reset
VME	Virtual Machine Environment

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1. Introduction

This demo interfaces the Raspberry Pi general purpose input output (GPIO) pins to the JTAG embedded source code in the Lattice Radiant™ and Diamond™ software directories. Refer to Source Code Directory for the location of the embedded programming source code. This demo is applicable to Lattice devices including MachXO2™, MachXO3™, and Nexus™-based FPGA devices.

Note: The Lattice embedded source code uses bit banging so GPIO pins are used for JTAG implementation.

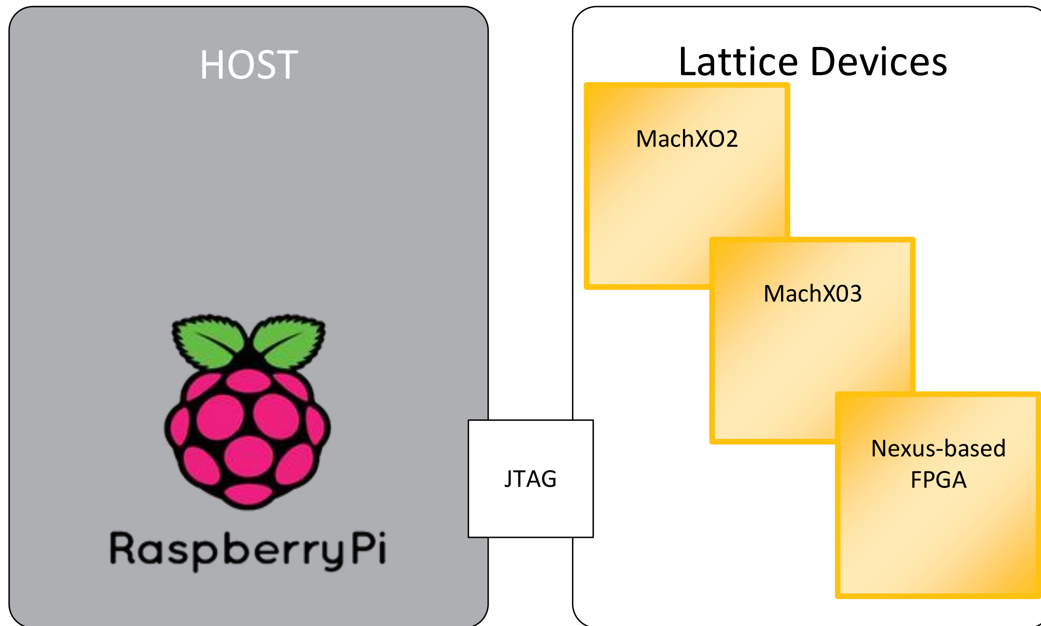

Figure 1.1. Raspberry Pi GPIO JTAG Interface with Lattice Device

The .vme file contains the commands and data to be sent during the programming procedure. Generation of the .vme file is covered in Appendix A.

Download the reference design files from the Lattice JTAG Embedded Programming Demo Using Raspberry Pi Reference Design web page.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1.1. Embedded Programming Source Code Architecture

Table 1.1 and Table 1.2 list the existing files needed and files to be created for embedded programming, respectively. The files to be created are the compiler and driver files.

Table 1.1. Existing Files Needed for Embedded Programming

File	Needs Customization	Description
hardware.c	Yes	JTAG hardware abstraction layer; interfaced with target hosts
ispvm_ui.c	Yes	Contains the main() function for file-based system
ivm_eprom.c	Yes	Contains the main() function for EPROM-based system
ivm_core.c	No	Handles parsing and programming algorithm
vmopcode.h	No	Contains opcodes for programming

Table 1.2. Files to be Created for Embedded Programming

File	Description
Makefile	Makefile that builds the executable jtag-rbpi with the standard make utility
jtagpins.c, jtagpins.h	JTAG pins hardware declaration using Raspberry Pi GPIO pins

Figure 1.2 shows the embedded programming source code architecture**.** For this demo, the executable file will be run using the command terminal. The *ivm_core.c* file handles the parsing of the .vme file and conversion of the file into FPGA programming commands. The *hardware.c* file is the interface of the hardware driver to the FPGA programming commands in the *ivm_core.c* file. The *driver.c* file contains the Raspberry Pi driver functions to be interfaced with the *hardware.c* file.

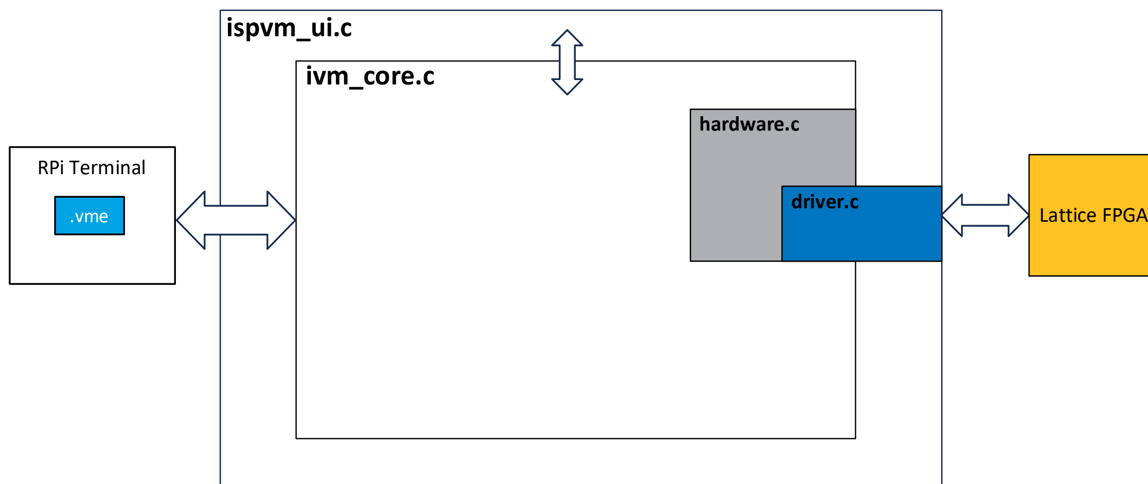

Figure 1.2. Embedded Programming Source Code Architecture

Only two existing source files (*hardware.c* and *ispvm_ui.c* or *ivm_eprom.c*) will be modified as indicated in Table 1.1. Additionally, one source file (*driver.c* or *jtagpins.c*) will be created. Refer to Modifying the Source Code and Writing the Driver for more information. Refer to Compiling and Running the Demo for the Makefile.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

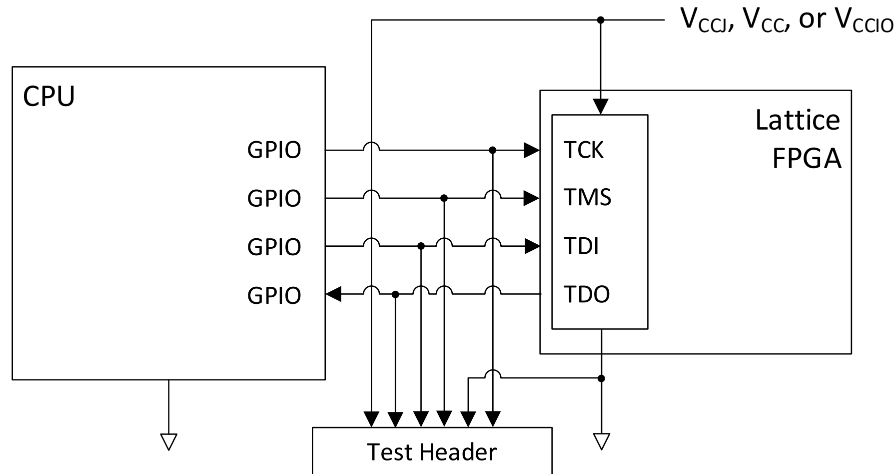

Figure 1.3. JTAG VME Hardware Interface with Lattice FPGA

Figure 1.3 shows GPIO pins acting as JTAG pins. Because the embedded source code uses bit banging, any GPIO pin on the target processor can be used for JTAG communication.

There are two types of embedded source codes:

- **File-based source code** uses file-based data storage, such as on Linux systems. This means that data and algorithms are stored in the .vme file, which can be read and written during runtime. This approach makes modifying the source code easier since you do not need to recompile the code; you only need to update the .vme file during runtime.
- **EPROM-based source code** uses erasable programmable read-only memory (EPROM) for storing data and algorithms, which are compiled into the system. In this case, the .vme file is converted into .c arrays and added to the program during compilation. This method is particularly useful for microcontroller devices but it also means you must recompile the code every time you update the .c arrays containing the .vme data.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1.2. Source Code Directory

Figure 1.4 shows the location of the embedded programming installation directory for the Radiant Programmer and Diamond Programmer. This demo focuses on the vmembedded source code. Figure 1.5 shows the list of folders in the embedded programming source code directory for the Radiant Programmer and Diamond Programmer.

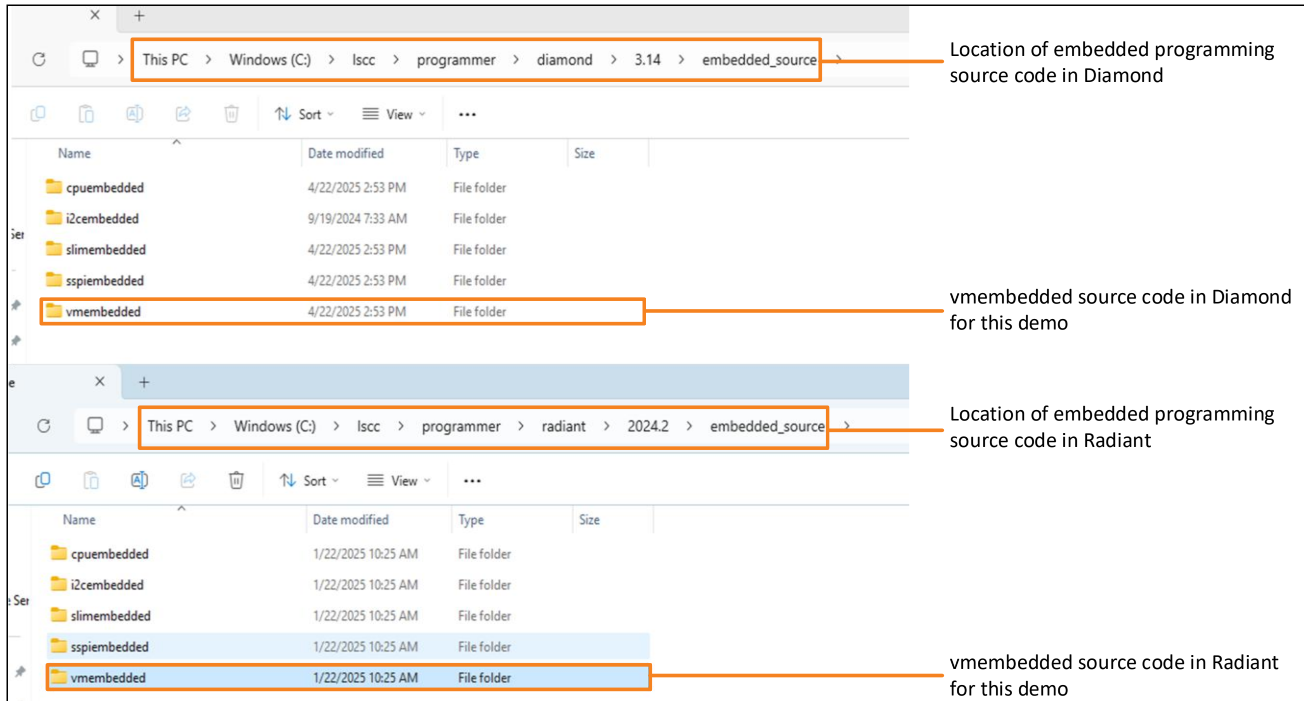

Figure 1.4. Embedded Programming Source Code Directories

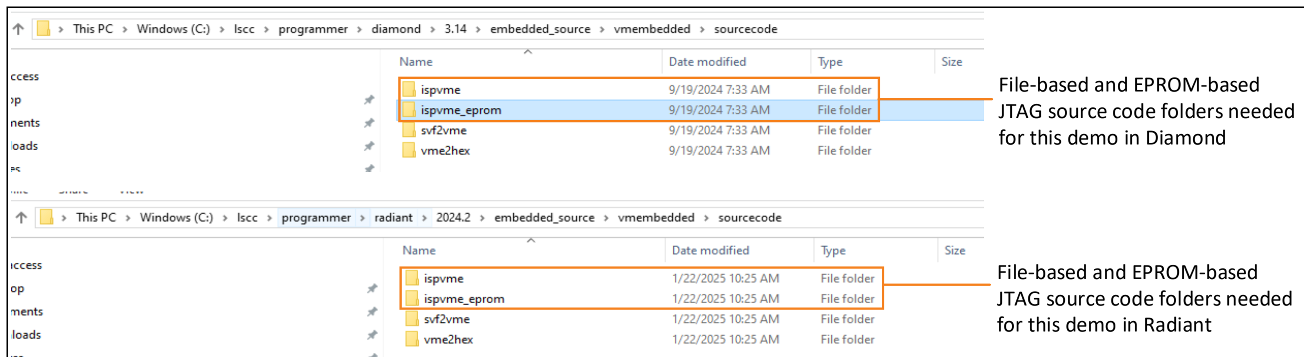

Figure 1.5. Embedded Programming Source Code Folders for File-Based and EPROM-Based Source Codes

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2. Modifying the Source Code and Writing the Driver

This section discusses the modification of the source code files *ispvm_ui.c*, *ivm_eprom.c*, and *hardware.c*. This section also discusses the creation of the driver file and how it is interfaced with *hardware.c*. These examples use the libgpiod library. Install this library in your Raspberry Pi board through the terminal using the following command line:

```
sudo apt-get install gpiod libgpiod-dev
```

2.1. Updating ispvm_ui.c for File-Based Source Code

The *ispvm_ui.c* file contains code for initializing the driver and calling FPGA programming commands. This section discusses the modifications required in *ispvm_ui.c* for file-based source code. Before modifying *ispvm_ui.c*, ensure that the appropriate header files and libraries are included. The following header files and libraries are required in *ispvm_ui.c*:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vmopcode.h"
#include <ctype.h>
#include "jtagpins.h"
#include <time.h>
#include <gpiod.h>
```

2.1.1. Updating main() Function

The following shows the modifications in the main() function:

```
int main( int argc, char * argv[] )
{


      unsigned short iCommandLineIndex  = 0;
      short siRetCode                   = 0;
      char szExtension[ 5 ]             = { 0 };
      char szCommandLineArg[ 300 ]      = { 0 };
      short sicalibrate                 = 0;

      //08/28/08 NN Added Calculate checksum support.
      setup_gpio();
      g_usChecksum = 0;
      g_uiChecksumIndex = 0;

      vme_out_string( "                 Lattice Semiconductor Corp.\n" );
      vme_out_string( "\n             ispVME(tm) V");
      vme_out_string( "\n             Ported By Rhodz \n");
    vme_out_string( VME_VERSION_NUMBER );
    vme_out_string(" Copyright 1998-2011.\n");
      vme_out_string( "\nFor daisy chain programming of all in-system
programmable devices\n\n" );
.
.
.
.
.
.
```

Initialize all GPIO pins to be used for JTAG programming.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
.
.
if ( siRetCode < 0 ) {
            error_handler( siRetCode, szCommandLineArg );
            vme_out_string( "Failed due to ");
            vme_out_string ( szCommandLineArg );
            vme_out_string ("\n\n");
            vme_out_string( "+=======+\n" );
            vme_out_string( "| FAIL! |\n" );
            vme_out_string( "+=======+\n\n" );
      }
      else {
            vme_out_string( "+=======+\n" );
            vme_out_string( "| PASS! |\n" );
            vme_out_string( "+=======+\n\n" );
            //08/28/08 NN Added Calculate checksum support.
            if(g_usChecksum != 0)
            {
                  g_usChecksum &= 0xFFFF;
                  printf("Data Checksum: %.4X\n\n",g_usChecksum);
                  g_usChecksum = 0;
            }
      }
      release_gpio();          Release all GPIO pins to prepare for next run.


      exit( siRetCode );
}
```

The functions setup_gpio() and release_gpio() initialize and close the Raspberry Pi GPIO drivers, respectively. Refer to Creating jtagpins.c for more details on the driver file (*jtagpins.c*). Regardless of platform, it is advisable to initialize the driver in the main() function of *ispvm_ui.c* for easier access and control.

For the file-based source code, the .vme file is an argument of the executable during runtime. If no .vme file is declared during run time, programming will not start. Refer to the Compiling and Running the Demo section for more information.

2.2. Updating ivm_eprom.c for EPROM-Based Source Code

The *ivm_eprom.c* file contains code for initializing the driver and calling FPGA programming commands. This section discusses the modifications required in *ivm_eprom.c* for EPROM-based source code. Before modifying *ivm_eprom.c*, ensure that the appropriate header files and libraries are included. The following header files and libraries are required in *ivm_eprom.c* for this implementation:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vmopcode.h"
#include <ctype.h>
#include "jtagpins.h"
#include <time.h>
#include <gpiod.h>
#include "XO3D_vme.h"          Include the header file for the .vme file to be used.
```

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.2.1. Updating main() Function

The following shows the modifications in the main() function:

```
int main(int argc, signed char *argv[])
{
	signed short siRetCode = 0;
	signed char szCommandLineArg[ 300 ] = { 0 };

	//08/28/08 NN Added Calculate checksum support.
	setup_gpio();
	g_usChecksum = 0;
	g_uiChecksumIndex = 0;


	vme_out_string("\n                 Lattice Semiconductor Corp.\n");
	vme_out_string("\n               ispVME V");
	vme_out_string( VME_VERSION_NUMBER);
	vme_out_string(" Copyright 1998-2011.\n");
	vme_out_string("\n      For Daisy Chain of All In-System Programmable
Devices\n\n");

	if ( argc != 1 ) {
		vme_out_string("\nUsage: vme \n\n");
		vme_out_string("Example: vme\n");
		exit( 1 );
	}
	siRetCode = ispVM();
    if ( siRetCode < 0 ) {
		error_handler( siRetCode, szCommandLineArg );
	  vme_out_string( "Failed due to ");
	  vme_out_string( szCommandLineArg );
	  vme_out_string( "\n\n");
	  vme_out_string( "+=======+\n" );
	  vme_out_string( "| FAIL! |\n" );
	  vme_out_string( "+=======+\n\n" );
	}
	else {
		vme_out_string( "+=======+\n" );
		vme_out_string( "| PASS! |\n" );
	  vme_out_string( "+=======+\n\n" );
		//08/28/08 NN Added Calculate checksum support.
		if(g_usChecksum != 0)
		{
			g_usChecksum &= 0xFFFF;
			printf("Data Checksum: %.4X\n\n",g_usChecksum);
			g_usChecksum = 0;
		}
	}
	release_gpio();
	return ( siRetCode );
}
```

Initialize all GPIO pins to be used for JTAG programming. (annotation on setup_gpio();)

Release all GPIO pins to prepare for next run. (annotation on release_gpio();)

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

The functions setup_gpio() and release_gpio() initialize and close the Raspberry Pi GPIO drivers, respectively. Refer to Creating jtagpins.c for more details on the driver file (*jtagpins.c*). Regardless of platform, it is advisable to initialize the driver in the main() function of *ivm_eprom.c* for easier access and control.

For the EPROM-based source code, the main() function already handles the processing of the .vme converted arrays. Unlike the file-based approach in which the .vme file is an argument provided during runtime, if the proper header files are included, programming should run successfully.

2.3. Updating hardware.c

The only areas requiring modification in *hardware.c* are some functions needed for interfacing with the Raspberry Pi drivers. Table 2.1 lists the functions to be modified in *hardware.c*. Modification involves adding code into the functions (pre-existing functions provide only usage guidelines and contain no code). The *hardware.c* file is the same for file-based and EPROM-based systems so both source codes must have the same modifications.

Table 2.1. Functions to be Modified in hardware.c

Function	Description
writePort()	Applies specified value to the JTAG pins
readPort()	Returns the TDO value from the device
sclock()	Applies a clock pulse; this function controls the clock speed of the programming
ispvVMDelay()	Implements the delay between commands

These functions will be interfaced to the Raspberry Pi GPIO pins. This means that when these functions are called by the FPGA programming algorithm, the GPIO pins should correspondingly behave to implement the JTAG communication with the Lattice FPGA. The following header files and libraries are required in *hardware.c*:

```
#include "vmopcode.h"
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <gpiod.h>
#include "jtagpins.h"
```

Additionally, declare in *hardware.c* the pin number definitions and structs gpiod_line and gpiod_chip.

```
#define TDI_PIN   20
#define TCK_PIN   12
#define TMS_PIN   16
#define TDO_PIN  21
#define ENABLE_PIN 24
#define TRST_PIN  25

struct gpiod_line *line_tdi, *line_tck, *line_tms, *line_tdo ,*line_enable,
*line_trst;
struct gpiod_chip *chip;
```

The pin number definitions are GPIO pin numbers in Raspberry Pi for the JTAG pins. The pins ENABLE_PIN and TRST_PIN are optional pins and can be ignored for most JTAG implementations. The important definitions are TDI_PIN, TCK_PIN, TMS_PIN, and TDO_PIN.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

This JTAG programming implementation needs to use bit addresses of pins for updating pin states. In this example, the addressing used in the source code is maintained.

```
const unsigned char g_ucPinTDI          = 0x01;    /* Bit address of TDI */
const unsigned char g_ucPinTCK          = 0x02;    /* Bit address of TCK */
const unsigned char g_ucPinTMS          = 0x04;    /* Bit address of TMS */
const unsigned char g_ucPinENABLE       = 0x08;    /* Bit address of ENABLE */
const unsigned char g_ucPinTRST         = 0x10;    /* Bit address of TRST */
const unsigned char g_ucPinTDO          = 0x40;    /* Bit address of TDO*/
```

These addresses are used for the writePort() and readPort() functions. The bit addresses used are one-hot vectors, where only one bit is set to 1 for each pin address. This facilitates the use of bitwise AND logic to update the values of multiple pins.

2.3.1. writePort() Implementation

The writePort() function is used to implement specified values to the pins indicated. The following code snippet is a sample implementation:

```
void writePort(unsigned char a_ucPins, unsigned char a_ucValue)
{
    if (a_ucPins & g_ucPinTDI) {
        gpiod_line_set_value(line_tdi, a_ucValue);
    }
    if (a_ucPins & g_ucPinTCK) {
        gpiod_line_set_value(line_tck, a_ucValue);
    }
    if (a_ucPins & g_ucPinTMS) {
        gpiod_line_set_value(line_tms, a_ucValue);
    }
    if (a_ucPins & g_ucPinENABLE) {
        gpiod_line_set_value(line_enable, a_ucValue);
    }
    if (a_ucPins & g_ucPinTRST) {
        gpiod_line_set_value(line_trst, a_ucValue);
    }
}
```

The a_ucPins argument sets the bit flags for the pins that need to be updated. The a_ucValue argument sets the values for the corresponding pins. To implement this function, a bitwise AND operation is used to update multiple pins simultaneously. Note that multiple *if* statements are used to check and update each pin individually. *if-else* statements are not used because checking of other conditions stops once one condition is met, which is not desired. The comments in the following code snippet describe a sample operation with the writePort() function:

```
writePort(unsigned char a_ucPins, unsigned char a_ucValue)
// writePort(0x03, 0x01)
{
    if (a_ucPins & g_ucPinTDI) { //0x03 & 0x01 is true
        gpiod_line_set_value(line_tdi, a_ucValue); // TDI is updated to 0x01
    }
    if (a_ucPins & g_ucPinTCK) { //0x03 & 0x02 is true
        gpiod_line_set_value(line_tck, a_ucValue); // TCK is updated to 0x01
    }
    if (a_ucPins & g_ucPinTMS) { //0x03 & 0x04 is false
        gpiod_line_set_value(line_tms, a_ucValue); // TMS is not updated.
    }
    if (a_ucPins & g_ucPinENABLE) { //0x03 & 0x08 is false.
```

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
        gpiod_line_set_value(line_enable, a_ucValue); // ENABLE is not updated
    }
    if (a_ucPins & g_ucPinTRST) { //0x03 & 0x10 is false
        gpiod_line_set_value(line_trst, a_ucValue); // TRST is not updated
    }
} // TDI and TCK are set to 1
```

2.3.2. readPort() Implementation

The readPort() function returns the value of the TDO pin. The following code snippet is a sample implementation:

```
unsigned char readPort()
{
unsigned char ucRet = 0;
      if (gpiod_line_get_value(line_tdo)){
          ucRet=0x01;
      }
      return ( ucRet );
}
```

This is only a simple function that reads the value of the TDO pin.

2.3.3. sclock() Implementation

The sclock() function generates a pulse for the test clock (TCK), which is essential for managing the clock cycle. The frequency of the clock varies depending on the performance of the processor and the speed of GPIO commands. This function generates one clock cycle. The following code snippet shows a sample implementation:

```
void sclock()
{
      gpiod_line_set_value(line_tck,1);
      usleep(1000);//if need to slow down clock
      gpiod_line_set_value(line_tck,0);
      usleep(1000);//if need to slow down clock
}
```

A delay can be introduced to reduce the clock speed. The usleep() command is used to lengthen the high and low phases of the TCK signal. This is particularly important for debugging complex printed circuit boards (PCBs).

2.3.4. ispVMDelay() Implementation

The ispVMDelay() subroutine provides a delay ranging from 1 millisecond to several hundred milliseconds. Because a_usTimeDelay is defined as an unsigned short (16-bit integer), the maximum delay this subroutine can produce is 64000 microseconds or 32000 milliseconds. The .vme file will not request a delay exceeding these limits. If a longer delay is needed, the .vme file will call this function multiple times to achieve the desired cumulative delay. The following code snippet is a sample implementation:

```
void ispVMDelay(unsigned short a_usTimeDelay) {
    if (a_usTimeDelay & 0x8000) {
        a_usTimeDelay &= ~0x8000;
        usleep(a_usTimeDelay * 1000);  // Millisecond delay
    } else {
        usleep(a_usTimeDelay);  // Microsecond delay
    }
}     return 1;
}
```

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

This implementation must be adhered to as the *if* statement differentiates between microsecond and millisecond delays to ensure the correct delay is applied. If the most significant bit (MSB) of a_usTimeDelay is 1, the delay is in milliseconds; if the MSB is 0, the delay is in microseconds. Follow this implementation to avoid excessively long programming times.

2.3.5. calibration() Implementation

The calibration function is used to check the accuracy of the clock frequency or the delay function. It performs this by sending clock cycles with delays in between. The following code snippet is a sample implementation:

```
void calibration(void)
{
      /*Apply 2 pulses to TCK.*/
      sclock();
      sclock();

      /*Delay for 1 millisecond. Pass on 1000 or 0x8001 both = 1ms delay.*/
      ispVMDelay(0x8001);

      /*Apply 2 pulses to TCK*/
      sclock();
      sclock();
}
```

This implementation sends two clock cycles, introduces a 1-ms delay, and then sends another two clock cycles. This can be used in hardware to check the desired frequency or delay implementation.

2.4. Creating jtagpins.c

The *jtagpins.c* file contains the functions used to initialize and release GPIO control. Additionally, it also contains the function to update the state of the GPIO. It connects the source code to the hardware drivers. Table 2.2 lists the functions contained in *jtagpins.c*.

Table 2.2. Functions to be Created in jtagpins.c

Function	Description
setup_gpio()	Initializes GPIO control
release_gpio()	Releases control of the GPIO pins

The following header files and libraries are required for *jtagpins.c*:

```
#include "vmopcode.h"
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <gpiod.h>
#include "jtagpins.h"
```

Most of the commands used for GPIO control are included in libgpiod so *jtagpins.c* only contains the setup and release functions.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
void setup_gpio() {
    chip = gpiod_chip_open_by_name("gpiochip4");

    line_tdi = gpiod_chip_get_line(chip, TDI_PIN);
    line_tck = gpiod_chip_get_line(chip, TCK_PIN);
    line_tms = gpiod_chip_get_line(chip, TMS_PIN);
    line_tdo = gpiod_chip_get_line(chip, TDO_PIN);
    line_enable = gpiod_chip_get_line(chip, ENABLE_PIN);
    line_trst = gpiod_chip_get_line(chip, TRST_PIN);

    gpiod_line_request_output(line_tdi, "ispvm", 0);
    gpiod_line_request_output(line_tck, "ispvm", 0);
    gpiod_line_request_output(line_tms, "ispvm", 0);
    gpiod_line_request_input(line_tdo, "ispvm");  // TDO is input
    gpiod_line_request_output(line_enable, "ispvm", 0);
    gpiod_line_request_output(line_trst, "ispvm", 0);
}

void cleanup_gpio() {
    gpiod_chip_close(chip);
}

void release_gpio() {
    // Release each line
    if (line_tdi) gpiod_line_release(line_tdi);
    if (line_tck) gpiod_line_release(line_tck);
    if (line_tms) gpiod_line_release(line_tms);
    if (line_tdo) gpiod_line_release(line_tdo);
    if (line_enable) gpiod_line_release(line_enable);
    if (line_trst) gpiod_line_release(line_trst);

    // Close the GPIO chip
    if (chip) gpiod_chip_close(chip);
}
```

The setup_gpio() function initializes the GPIO pins assigned to each JTAG signal and sets all pins to an initial state of 0. The cleanup_gpio() function ends the control for struct gpiod_chip. The release_gpio() function releases control of the pins initialized by the setup_gpio() function.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Compiling and Running the Demo

After modifying the source code and writing the driver, your files are now ready for compiling and running.

1. To compile the file-based demo, a Makefile is used containing the following:

```
jtag-rbpi: $(OBJ)
	gcc -D VME_DEBUG -o rhodz_vme ispvm_ui.c hardware.c ivm_core.c jtagpins.c -lgpiod


jtag-rbpi-nodebug: $(OBJ)
	gcc -o rhodz_vme ispvm_ui.c hardware.c ivm_core.c jtagpins.c -lgpiod
```

   Ensure the Makefile and all source codes including the driver files are in the same folder.

   For the EPROM-based demo, a Makefile is used containing the following:

```
jtag-rbpi: $(OBJ)
	gcc -D VME_DEBUG -o rhodz_vme ivm_eprom.c hardware.c ivm_core.c jtagpins.c XO3D_vme1.c XO3D_vme2.c  XO3D_vme3.c  XO3D_vme4.c  XO3D_vme5.c  XO3D_vme6.c XO3D_vme7.c  XO3D_vme8.c  XO3D_vme9.c  XO3D_vme10.c  XO3D_vme11.c  XO3D_vme12.c XO3D_vme13.c  XO3D_vme14.c  XO3D_vme15.c  XO3D_vme16.c  XO3D_vme17.c XO3D_vme18.c  XO3D_vme19.c  XO3D_vme20.c  XO3D_vme21.c  XO3D_vme22.c XO3D_vme23.c  XO3D_vme24.c  XO3D_vme25.c  XO3D_vme26.c  XO3D_vme27.c XO3D_vme28.c  XO3D_vme29.c  XO3D_vme30.c  XO3D_vme31.c  XO3D_vme32.c XO3D_vme33.c XO3D_vme34.c -lgpiod

jtag-rbpi-nodebug: $(OBJ)
	gcc -o rhodz_vme ivm_eprom.c hardware.c ivm_core.c jtagpins.c XO3D_vme1.c XO3D_vme2.c  XO3D_vme3.c  XO3D_vme4.c  XO3D_vme5.c  XO3D_vme6.c  XO3D_vme7.c XO3D_vme8.c  XO3D_vme9.c  XO3D_vme10.c  XO3D_vme11.c  XO3D_vme12.c  XO3D_vme13.c XO3D_vme14.c  XO3D_vme15.c  XO3D_vme16.c  XO3D_vme17.c  XO3D_vme18.c XO3D_vme19.c  XO3D_vme20.c  XO3D_vme21.c  XO3D_vme22.c  XO3D_vme23.c XO3D_vme24.c  XO3D_vme25.c  XO3D_vme26.c  XO3D_vme27.c  XO3D_vme28.c XO3D_vme29.c  XO3D_vme30.c  XO3D_vme31.c  XO3D_vme32.c  XO3D_vme33.c XO3D_vme34.c -lgpiod
```

   For the EPROM-based implementation, you must include all the .c files generated by the Diamond or Radiant sofware. Refer to the Generation of .vme File section for more information.

2. Run *make jtag-rbpi* if you want to enable debug printing and *jtag-rbpi-nodebug* if you want to disable debug printing as shown in Figure 3.1.

   When compiling for the first time, the list of objects compiled appears in the terminal. After running the command, the executable file to run the demo is generated.

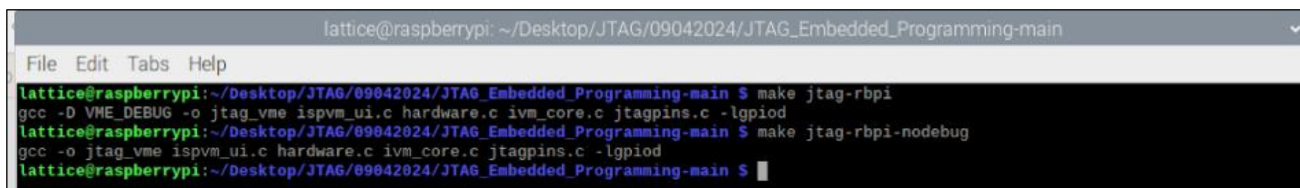

Figure 3.1. Sample Compilation

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Run the executable file.

Figure 3.2. Example of Running the Executable File in File-Based Demo

- The executable file is generated after compiling the source code and the driver. It initiates and ends the JTAG programming.
- The executable for file-based has only one argument, which is the .vme file. The executable for EPROM-based has no arguments as the .vme data is declared as arrays within the Makefile.
- You can slow down the clock by modifying the sclock() function.
- All arguments of the executable file are required to run the demo.

Figure 3.3 shows an example of running the file-based demo with jtag-rbpi-nodebug. There are no logs for sending and receiving commands.

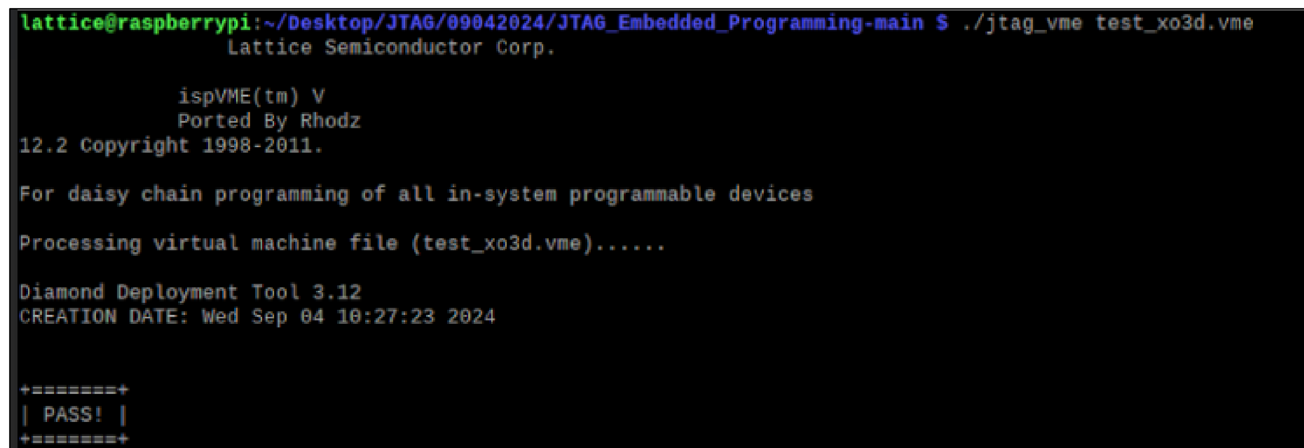

Figure 3.3. Sample Run without Debug Printing

Figure 3.4 shows an example of running the file-based demo with jtag-rbpi, which prints out the sending and receiving of data.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

```
lattice@raspberrypi:~/Desktop/JTAG/09042024/JTAG_Embedded_Programming-main $ ./jtag_vme test_xo3d.vme
                 Lattice Semiconductor Corp.

             ispVME(tm) V
             Ported By Rhodz
12.2 Copyright 1998-2011.

For daisy chain programming of all in-system programmable devices

Processing virtual machine file (test_xo3d.vme)......

// ISPVM EMBEDDED ADDED
STATE RESET;
Diamond Deployment Tool 3.12
CREATION DATE: Wed Sep 04 10:27:23 2024


// MEMSIZE 760
// VENDOR LATTICE
HDR 0;
HIR 0;
TDR 0;
TIR 0;
ENDDR DRPAUSE;
ENDIR IRPAUSE;
FREQUENCY 1.00E+06 HZ;
STATE IDLE;
SIR 8 TDI (E0);
SDR 32 TDI (00000000)
                TDO (212E3043)
                MASK (FFFFFFFF);
RECEIVED TDO (212E3043)

SIR 8 TDI (1C);
SDR 760 TDI (FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF);
SIR 8 TDI (C6);
SDR 8 TDI (00);
STATE IDLE;
RUNTEST 2 TCK;
RUNTEST 1.00E-03 SEC;
SIR 8 TDI (0E);
SDR 16 TDI (0000);
STATE IDLE;
RUNTEST 2 TCK;
RUNTEST 1.00E+00 SEC;
SIR 8 TDI (3C);
STATE IDLE;
RUNTEST 2 TCK;
RUNTEST 1.00E-03 SEC;
SDR 32 TDI (00000000)
                TDO (00000000)
                MASK (00003100);
RECEIVED TDO (00080E50)

SIR 8 TDI (C6);
SDR 8 TDI (08);
STATE IDLE;
RUNTEST 2 TCK;
RUNTEST 1.00E-03 SEC;
```

Figure 3.4. Sample Run with Debug Printing

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. Debugging Tips

The following are tips to help with debugging:

- Ensure that the writePort() function is properly implemented as discussed in writePort() Implementation. Improper implementation of writePort() causes incorrect transmission of data.
- Improper implementation of ispVMDelay() causes long programming times because of the long wait time between transactions. Follow the delay implementation in ispVMDelay() Implementation to optimize delays between transactions.
- During development, enabling debug printing to see transactions sent and received is recommended.
- When programming on complex PCBs and with high-speed signals, verification fail might occur. This is usually caused by the failures in the setup and hold times for the JTAG transactions. To fix the issue, you need to slow down the clock through the sclock() function. The following code snippet shows a sample implementation:

```
void sclock()
{
	gpiod_line_set_value(line_tck,1);
	//usleep(1);//if need to slow down clock
	gpiod_line_set_value(line_tck,0);
	//usleep(1);// if need to slow down clock
}
```

  Figure 4.1 shows a clock with a frequency of approximately 125 kHz generated by the above code with usleep() commented out.

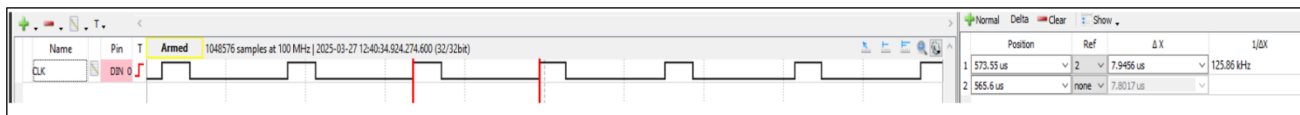

Figure 4.1. Clock without Added Delay

  Adding delay makes the clock frequency slower providing the JTAG transaction enough setup and hold times. Figure 4.2 shows a clock with additional delay added.

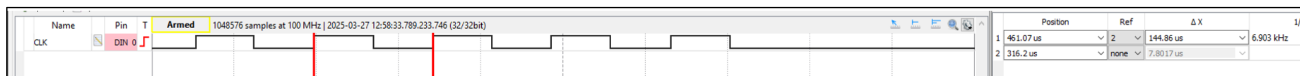

Figure 4.2. Clock with Added Delay

  The driver used for this implementation is slow so only a low clock frequency is observed. On complex PCBs with transactions at the MHz level, use a faster device or direct memory access (DMA) to control the GPIO pins to achieve a higher clock frequency.

- The calibration function described in the calibration() Implementation section is used to verify both the clock frequency and the delay implementation. It works by generating two clock cycles, introducing a 1-ms delay, and then generating another two clock cycles. This sequence enables verification of whether the clock is operating at the correct frequency and whether the delay is accurately implemented. To run the calibration, you must use the -c argument of the executable file as shown in Figure 4.3.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

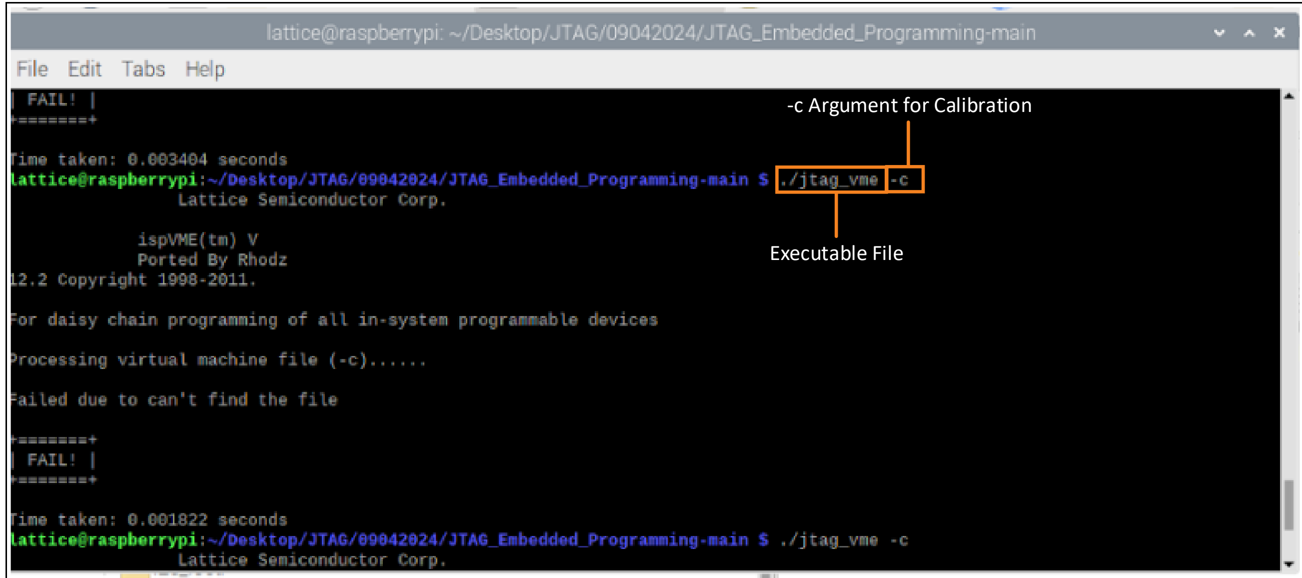

Figure 4.3. Running the Calibration Function

Note that an error will appear in the console but implementation on hardware will proceed.

Figure 4.4 shows a sample hardware run. The measured delay is 1.1324 ms, which is close to the 1-ms delay implemented by the calibration function.

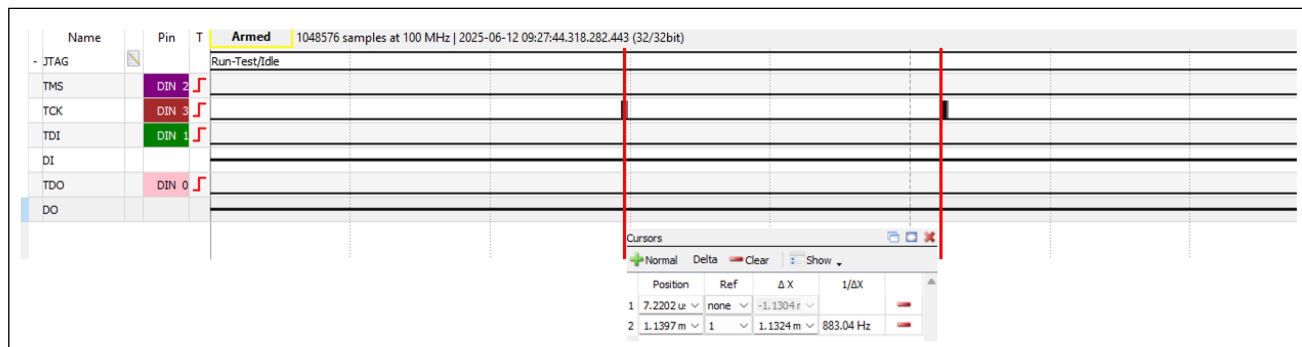

Figure 4.4. Sample Hardware Run with Calibration

Figure 4.5 shows a zoomed in view of the two clock cycles present at both sides of the delay interval. You can use these two clock cycles to check if the target frequency is achieved before running the .vme file. Perform the modifications as described earlier to slow down a clock if necessary, particularly for complex PCBs.

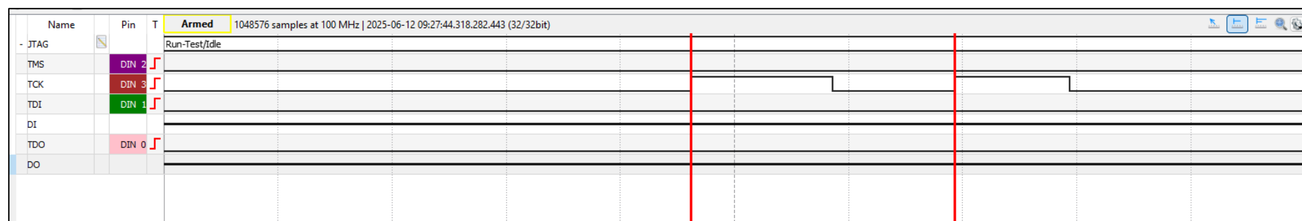

Figure 4.5. Two Clock Cycles in Sample Hardware Run with Calibration

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5. Hardware Validation Summary

Table 5.1 shows the results from validation of the demo on different development boards. This demo is tested on a chain of up to three devices.

Table 5.1. Hardware Validation Results

Number of Devices in Chain	Devices in the Chain	Pass/Fail
1	MachXO3D	Pass
2	MachXO3D and MachXO5-NX	Pass
3	MachXO3D, MachXO5-NX, and MachXO3LF	Pass

Note: When using Lattice development boards for testing, ensure that the FTDI chip of the board is disabled. Check the development board schematic or user guide for guidance on disabling the FTDI chip.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix A. Generation of .vme File

This section describes the steps for generating the .vme file needed for embedded programming. The Radiant Programmer can directly generate the .vme file. For the Diamond Programmer, the Deployment Tool is needed to generate the .vme file.

A.1. Generating .vme File Using Radiant Programmer

1. Open the Radiant Programmer and set up your operation. For example, fast configuration of the MachXO5-NX configuration static random-access memory (SRAM) using the JTAG port.

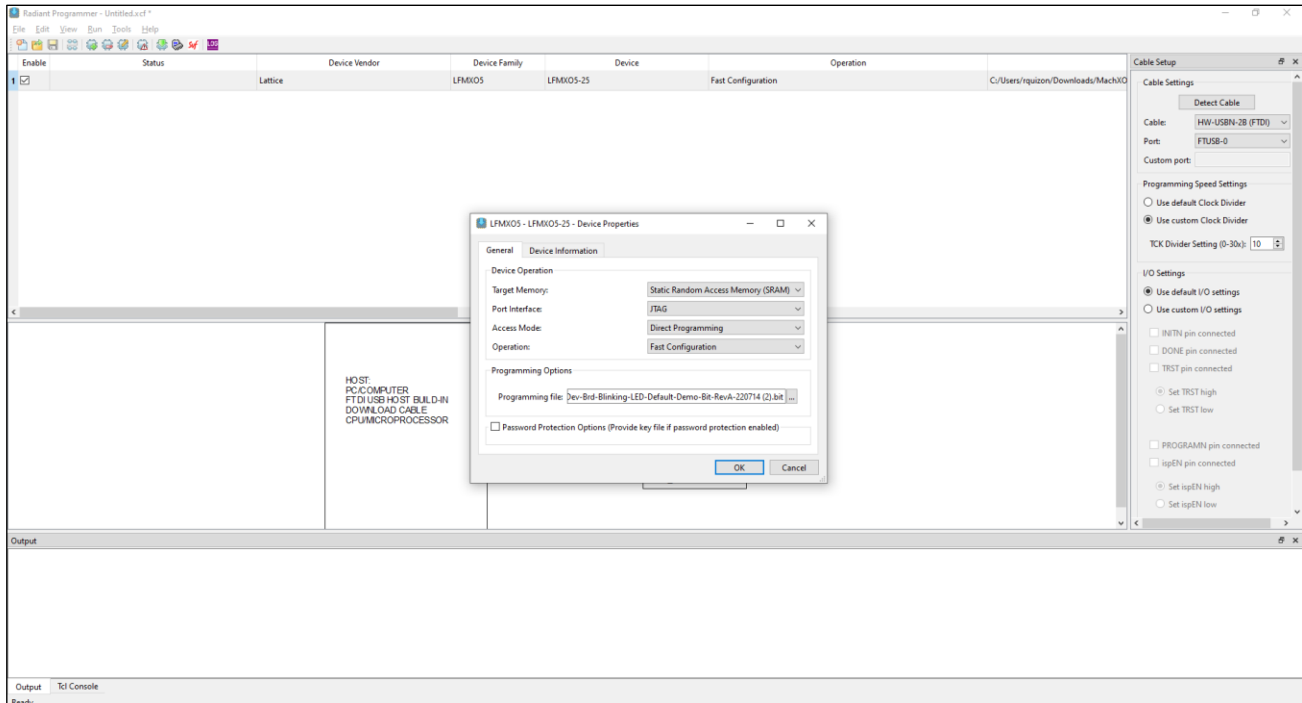

Figure A.1. Radiant Programmer .xcf File Generation

2. For the EPROM-based system, click the **Generate Embedded Code** button to open the Embedded Options window. Check the **Convert VME files to HEX (.c) for Prom-Based Embedded VME** checkbox and then click **OK**.

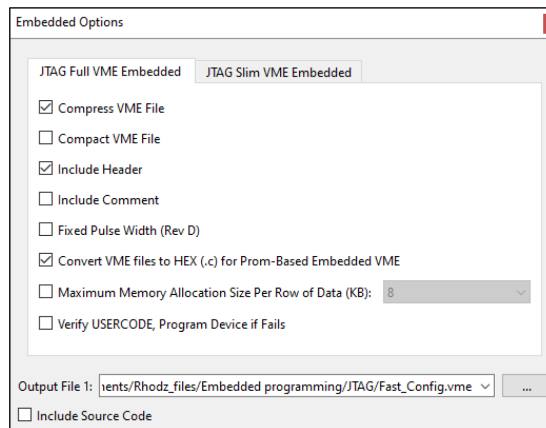

Figure A.2. Embedded Options Window

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Note: The Radiant software generates the header together with the .c arrays

3. Click the **Generate Embedded Code** button.

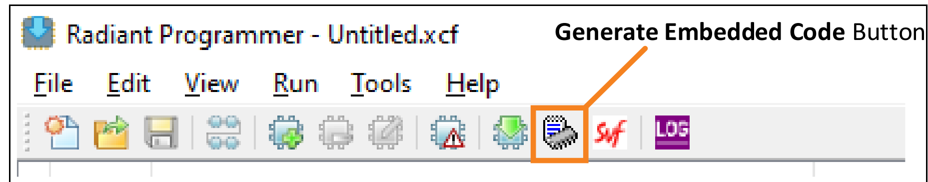

Figure A.3. Generate Embedded Code Button

The .vme file is generated as indicated in the Output window.

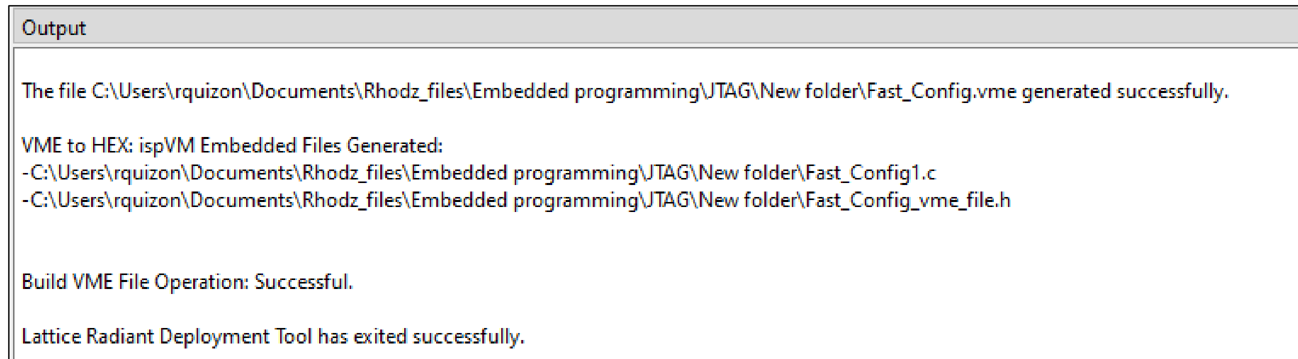

Figure A.4. Generation of .vme File

For both the file-based and EPROM-based systems, the .vme file appears in the .xcf file directory. For the EPROM-based system, .c array and header files also appear in the .xcf file directory.

Name	Date modified	Type	Size
Fast_Config.vme	3/28/2025 9:41 AM	VME File	46 KB
FastConfig_JTAG.xcf	3/28/2025 9:41 AM	XCF File	2 KB

Figure A.5. Generated .vme File in .xcf File Directory for File-Based System

Name	Date modified	Type	Size
Fast_Config.vme	3/28/2025 9:46 AM	VME File	46 KB
Fast_Config_vme_file.h	3/28/2025 9:46 AM	H File	1 KB
Fast_Config1.c	3/28/2025 9:46 AM	C File	281 KB
FastConfig_JTAG.xcf	3/28/2025 9:41 AM	XCF File	2 KB

Figure A.6. Generated .vme, .c, and .h Files in .xcf File Directory for EPROM-Based System

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

A.2. Generating .vme File Using Diamond Programmer and Deployment Tool

1. Open the Diamond Programmer and set up your operation. For example, flash erase, program, and verify for JTAG operation.

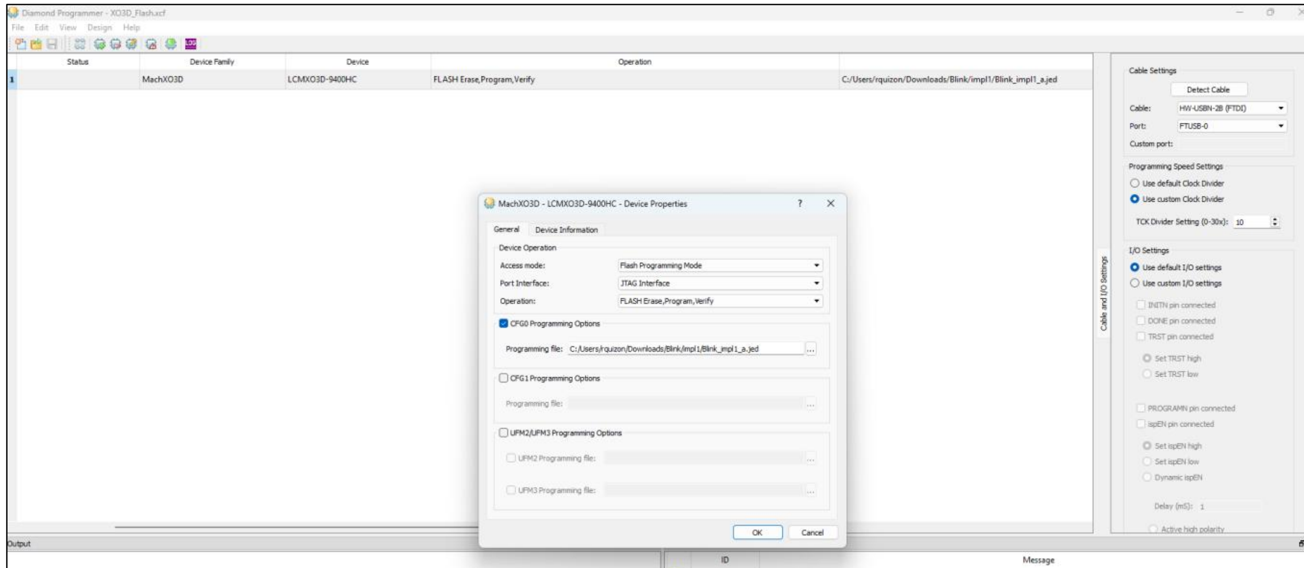

Figure A.7. Diamond Programmer .xcf File Generation

2. Save the .xcf file in a directory.
3. Open the Deployment Tool (**Design** > **Utilities** > **Deployment Tool**).

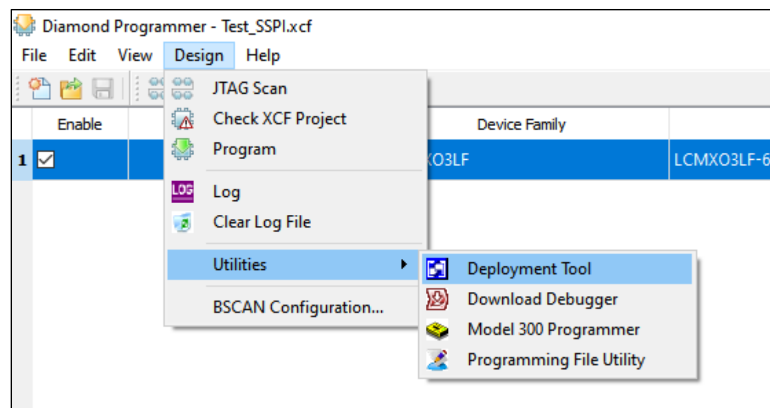

Figure A.8. Accessing the Deployment Tool

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. Select the options as shown to generate the .vme file.

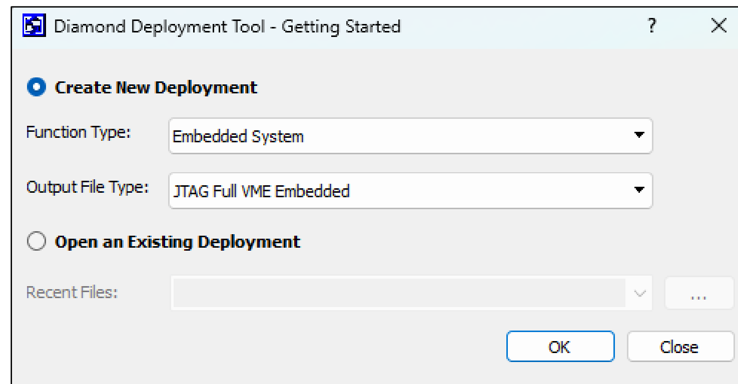

Figure A.9. JTAG VME Embedded Generation in Deployment Tool

5. Check the **Input XCF file** checkbox, select the .xcf file previously saved in step 2, and click **Next**.

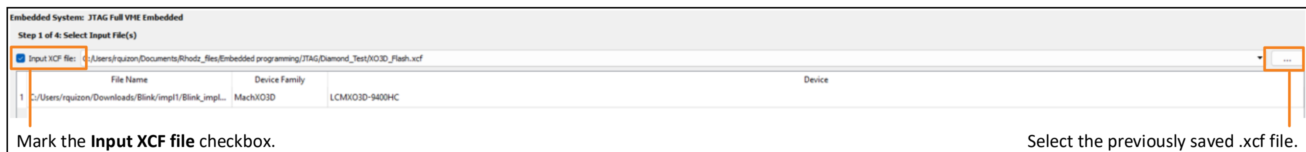

Figure A.10. Selecting .xcf File

6. The settings as shown appear in the next page. Leave the settings as is for this demo. For the EPROM-based system, check the **Convert VME files to HEX (.c) for Prom-Based Embedded VME** checkbox. Click **Next.**

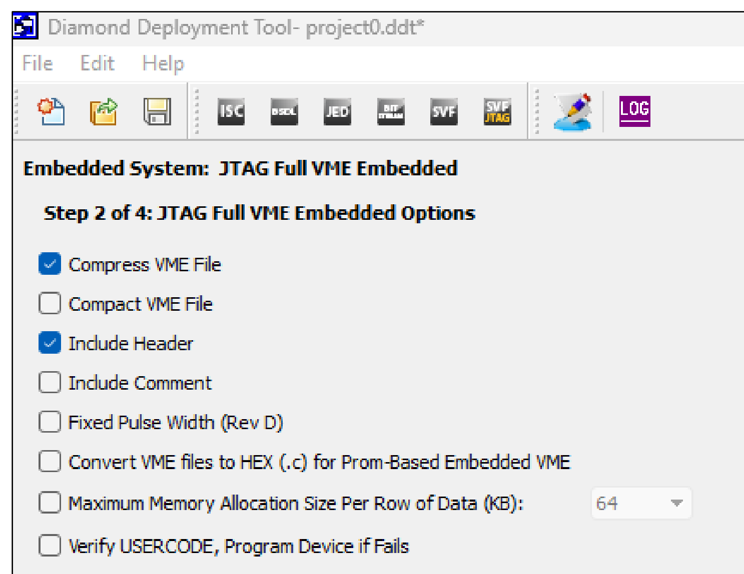

Figure A.11. Embedded File Generation Options

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

7. Confirm the location of the .vme file and click **Next**.

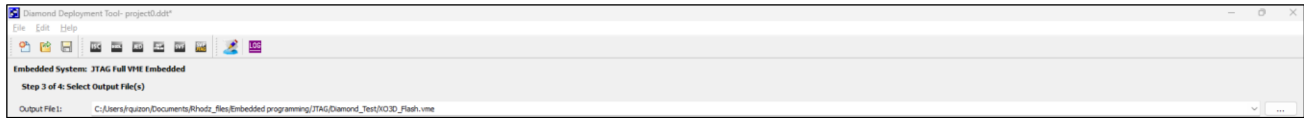

Figure A.12. Location of .vme File

8. Click **Generate**. The .vme file is generated as indicated in the window.

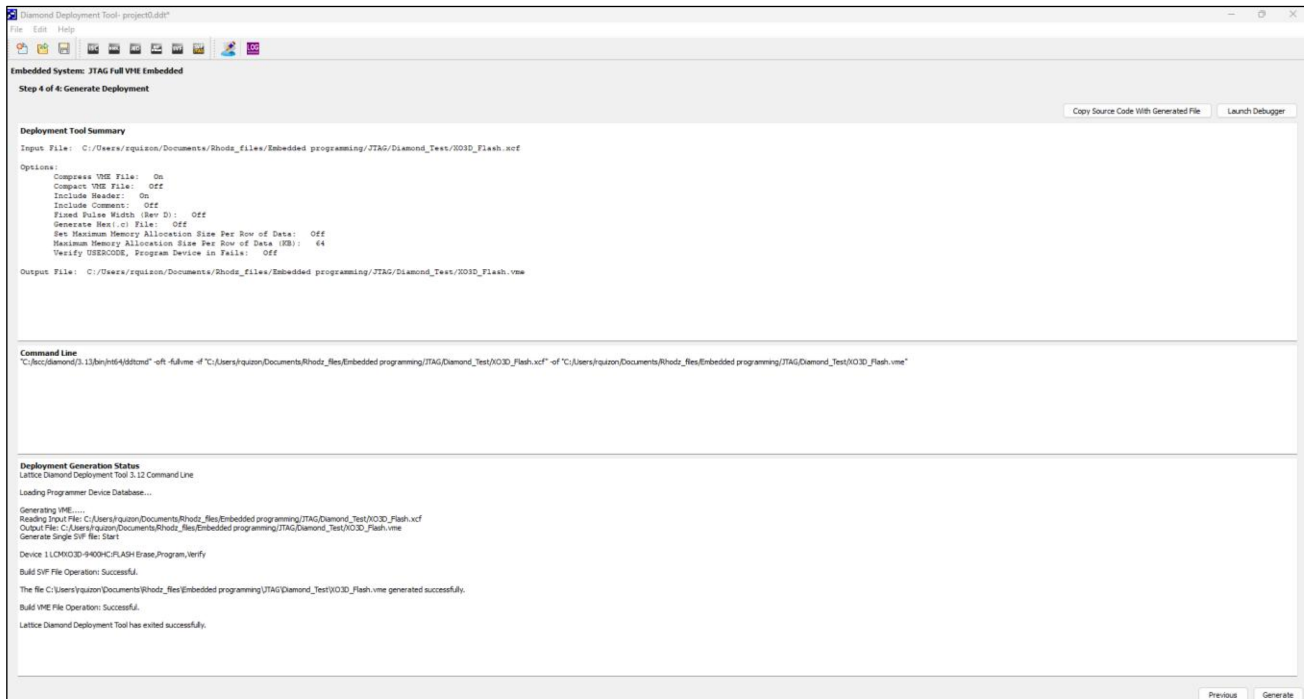

Figure A.13. Generation of JTAG VME Embedded File

For both the file-based and EPROM-based systems, the .vme file appears in the .xcf file directory. For the EPROM-based system, .c array and header files also appear in the .xcf file directory.

	Name	Date modified	Type	Size
.xcf File	XO3D_Flash.xcf	6/11/2025 10:28 AM	XCF File	2 KB
.vme File	XO3D_Flash.vme	6/11/2025 10:31 AM	VME File	977 KB

Figure A.14. Generated .vme File in .xcf File Directory for File-Based System

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

.vme File
.xcf File
Header File
.c Array Files

Name	Date modified	Type	Size
XO3D_Flash.vme	6/11/2025 10:33 AM	VME File	977 KB
XO3D_Flash.xcf	6/11/2025 10:28 AM	XCF File	2 KB
XO3D_Flash_vme_file.h	6/11/2025 10:33 AM	H File	2 KB
XO3D_Flash1.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash2.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash3.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash4.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash5.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash6.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash7.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash8.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash9.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash10.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash11.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash12.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash13.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash14.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash15.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash16.c	6/11/2025 10:33 AM	C File	360 KB
XO3D_Flash17.c	6/11/2025 10:33 AM	C File	238 KB

Figure A.15. Generated .vme, .h, and .c Files in .xcf File Directory for EPROM-Based System

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

References

- Programming Tools User Guide for Radiant Software 2024.2
- Lattice Diamond 3.14 Programming Tools User Guide
- JTAG Embedded Programming Demo Using Raspberry Pi Reference Design web page
- MachXO3D Breakout Board web page
- MachX03LF Starter Kit web page
- MachX05-NX Development Board web page
- Lattice Diamond Software web page
- Lattice Radiant Software web page
- Lattice Solutions Reference Designs web page
- Lattice Insights for Lattice Semiconductor training courses and learning plans

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Revision History

Revision 1.0, July 2025

Section	Change Summary
All	Initial release.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

www.latticesemi.com