Lattice Cable Connection Stability Usage Guidelines

Application Note

FPGA-AN-02085-1.0

February 2024

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 REDUDANCIES, 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.

Contents

Figures

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
FPGA	Field Programmable Gate Array
GPIO	General purpose input and outputs
I2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver and Transmitter
USB	Universal Serial Bus

1. Introduction

When working with any FPGA, one of the most important things is to ensure that the connection between your host system and target FPGA is stable and consistent. Without a connection between your host PC and target device, common operations such as device programming and on-chip debugging will not be possible. Aside from that, it is also important that this connection stays stable, to prevent loss of data or functionality depending on what you are trying to do.

One of the first considerations you should make when working with Lattice FPGAs is the type of programming cable you want to use. There are a few different options depending on your target device and intended usage, ranging from common USB cables such as USB-A to mini-USB, to proprietary Lattice programming devices such as the HW-USBN-2B. For more information on each of these programming cables and the best one to use for your application, refer to FPGA-UG-02042-26.6 Programming Cables.

2. Programming and Debugging Lattice Devices

In most cases, the process for programming and debugging a device can be summarized into two main stages: setting up the cable and its cable drivers, followed by the actual programming operations between a device and its host system. Oftentimes, the initial setup process is very brief and does not require much effort. However, one common pitfall occurs when users do not double check their cable setup before trying to program their device.

This and the following sections of the document will briefly describe the process for setting up programming cables for use with Lattice FPGA devices, as well as some tips to help avoid common pitfalls.

2.1. Cable Setup

Depending on the type of programming cable that you are using, as well as your target device, the initial process for setting up your cable and device for programming may vary slightly. Some devices use their built-in FTDI chip to enable easy cable setup and programming, allowing users to simply plug in a micro-USB or mini-USB cable into their target device from their host system and proceed with programming with little other effort.

In some other cases users may want to directly program their device by directly connecting to its GPIO to perform a generic programming operation such as JTAG, SPI, or I2C. Oftentimes this requires the use of a programming cable such as the HW-USBN-2B, which contains individual wires which are connected to the GPIO headers of a target device.

Figure 2.1. Lattice HW-USBN-2B programming cable

Depending on your programming operation of choice, the set of wires from the HW-USBN-2B programming cable that you would need to connect to your device will vary. For example, to program a device through JTAG you would need to connect the TCK, TMS, TDI, and TDO from the programming cable to your device. While for other programming operations such as I2C you would need to connect SCL and SDA instead. For more information on this portion of the cable setup process as well as supported programming operations, refer to FPGA-UG-02042-26.6 Programming Cables.

Additionally, the specific programming methods which are supported may vary from device-to-device, so it is also important that you refer to the datasheet for your target device beforehand as well.

Once your device is connected to your host system using one of the cable connection methods described above, the next step in the setup process would be to ensure that you have the correct drivers installed. In many cases these drivers will be installed automatically on the host operating system's side, however it is still useful to try this to ensure that everything is setup correctly.

To do this in Windows, proceed to the data directory in the Lattice tool installation directory for the software tool you are currently using (e.g., /lscc/radiant/2023.2/data). Within this folder, find the executable called **ispdriverinstl64.exe** and run it with administrator privileges. From the list of dropdown options, select **All Drivers**, and then click **Install** to proceed with the driver installation. If there are any issues, or you encounter any issues later on down the line, it may be worth trying to **Uninstall** any existing drivers before installing using the steps described above.

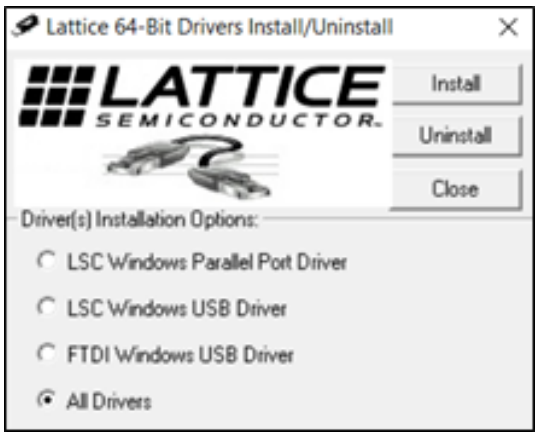

Figure 2.2. Lattice programming cable driver installation and setup

The process for setting up cable drivers in Linux is a little less straightforward than the same process in Windows, as no executable will handle the driver setup for users automatically. Instead, users will need to verify whether there are any existing drivers already by typing “lsusb” into their command terminal. If cable drivers are set up and the device is recognizable, you should see some message indicating the connected device’s bus and target chip (e.g., Future Technology Devices International, LTD FT2232C...). If this check does not work or returns an error, then it is likely that libusb is not installed. To download and install the most recent version of libusb, go to: https://libusb.sourceforge.net.

2.1.1. JTAG Chains

When working with multiple devices simultaneously, a daisy chained setup is needed in order to program and debug the devices in the chain. A daisy chain involves JTAG programing and allows a single host system to target multiple devices in a JTAG chain separately.

To setup a daisy chain, there are four common signals which must be hooked up between each device and the host programming cable. Each device in the JTAG chain should have its TCK and TMS signals connected to the same source programming cable. The setup for the following TDO and TDI signals requires a little more thought, with the TDI from the host programming cable being connected to the first device in the JTAG chain. From then on, the TDO of the Nth device, should be connected to the TDI of the (N+1)th device in the chain until the last device in the chain is connected. For the last device in the chain, the TDO signal should be connected back directly to the source programming cable. Figure 2.3 below depicts how this type of daisy chained setup works in practice.

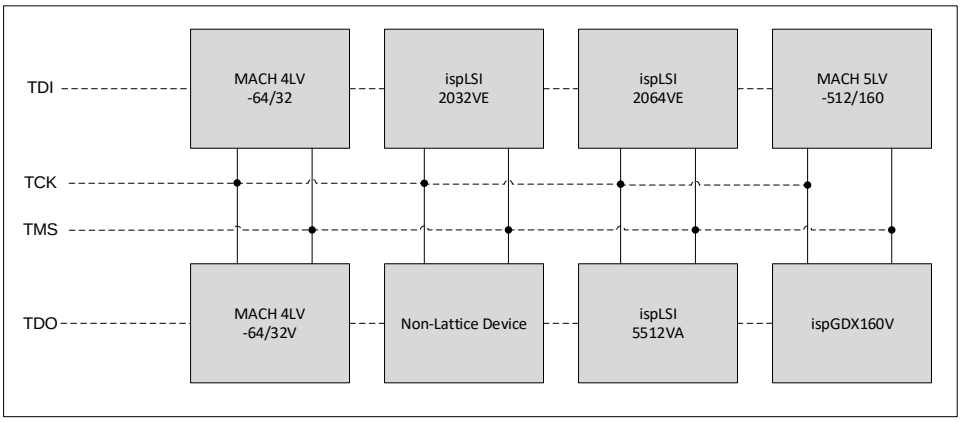

Figure 2.3. Daisy chain setup for programing multiple devices

2.2. Common Pitfalls

Although the process for setting up and using Lattice programming cables may seem straightforward as described in the previous section, there are still several areas which users commonly run into issues. This section of the document describes some of these common pitfalls, as well as some potential steps that can be taken to help resolve them.

Pitfall: When using a Linux-based operating system, the programming operations fail intermittently depending on the USB port the programming cable is connected to.

Keywords: Programmer, Reveal, Diamond, Radiant, Propel

Suggestions:

- Try adjusting the TCK divider if there is a setting. Larger divider values slow down the JTAG clock which may allow the target device to be detected if it's a performance issue.
- Enable the CABLESERVER_VERBOSE_DEBUG variable by setting it to 1.
  - This will generate a debug log called **cableserver_debug_logfile.log** in the **/bin/nt64/** directory of the active Lattice software tool.
- If you are using multiple FTDI devices, it is possible the correct drivers are not being loaded depending on the USB port your target device is connected to.
  - In this case, you will need to either remove the **ftdi_sio** module from the active kernel or create a UDEV to unbind the sysfs interface to disable devices as they are connected.
  - Refer to the **Installing and Configuring USB Cables** section of the Linux installation guide for Radiant, Propel, or Diamond for more details.

Pitfall: In Reveal, I cannot scan for my target device. Or when it does scan, I see core0's incorrect pattern readout, and I cannot debug with the Reveal Analyzer or Controller.

Keywords: Programmer, Reveal, Diamond, Radiant

Suggestions:

- Ensure that your port selection is correct. It should match the same one selected in Programmer.
- Try adjusting the TCK divider if there is a setting. Larger divider values slow down the JTAG clock, which may allow the target device to be detected if it's a performance issue.
- Try connecting to a different USB port on your system to see if you can reproduce.
- Look for the **reveal_debug.log** file in your project's implementation directory.
  - This log contains a history of all the JTAG commands that were issued by Reveal when attempting to communicate with your target device.
  - Search for a line containing a pattern readout. If the pattern readout = 0, it indicates that your Reveal sample clock was stopped.
    - When using Reveal Analyzer, your sample clock must be continuous and free running, so if it is held in reset or locked, then there is likely an issue with the sample clock.
    - If the pattern readout =0, but does not match the expected value, it could be an issue with the selected oscillator on-device, or the cable itself.
      - In this case, try using a different oscillator to see if the problem persists.
- Ensure that your source Reveal Inserter project file (*.RVL) matches the one that is currently inserted in your project.
    - The currently inserted Reveal inserter project file will be **bolded** in your project's file list.
    - If there is a mismatch between the target inserter file and the one that is currently inserted, Reveal will not be able to detect debug insertion on your target device.

Pitfall: The port that the device is connected to is constantly changing

Keywords: Programmer, Reveal, Diamond, Radiant, Propel

Suggestions:

- There are several common operations that may cause the USB port to which your device is connected to change.
  - Anytime a new device is connected, your system restarts, you restart your device, or anything like that may cause your USB port enumerations to update.
  - Unfortunately, there is nothing that can be done on the user side to alleviate this. You need to be wary of this behavior and always double check before trying to program or debug.

Pitfall: In Reveal, I cannot select both hard JTAG and soft JTAG for separate debug core types. Or I cannot select soft JTAG when trying to use Reveal with a RISC-V project.

Keywords: Programmer, Reveal, Diamond, Radiant, Propel

Suggestions:

- It is a known limitation that both hard JTAG and soft JTAG cannot be used simultaneously. You need to select either one or the other.
- By default, Lattice RISC-V processors utilize a hard JTAG for on-chip debugging, so if you intend to use Reveal, you must also use a hard JTAG as the interface type.
  - Some newer processor versions support soft or hard JTAG. For these processors, ensure that whatever you select in the IP configuration matches the JTAG hub type that you want to implement in Reveal.

Pitfall: In Propel SDK, when I try to scan for my device with OpenOCD, I see a '+' symbol and cannot proceed with on-chip debugging.

Keywords: Propel

Suggestions:

- Try adjusting the TCK divider if there is a setting. Larger divider values slow down the JTAG clock, which may allow the target device to be detected if it is a performance issue.
- Try changing the target port and then rescanning for the device. Between debug sessions, the way the target ports are enumerated may change, so it's important to always double-check before debugging.
  - Whichever port you used to program the target device is oftentimes the same one that should be selected for debugging.

Pitfall: I can program my target device, detect it in Propel's on chip debugger, and step through code. However, I cannot see any output in Propel's serial terminal from my UART.

Keywords: Propel

Suggestions:

- Double-check your target device's datasheet.
  - Ensure that there are no jumpers needed to enable UART communication for your device.
  - Double-check your constraints to make sure your UART component's RXD and TXD signals are constrained to the correct pins.
  - Double check the UART settings of your component's source IP and serial terminal to ensure that the baud rate, number of bits, and stop bit all match.
- Try enabling semihosting to debug further. This is functionally the same as a virtual UART and allows you to print statements from your project's UART directly to Propel's console output.

References

- FPGA-UG-02042-26.6 Programming Cables
- FPGA-AN-02060-1-0 Debugging with Reveal Usage Guidelines and Tips
- Lattice Radiant FPGA design software
- Lattice Diamond FPGA design software
- Lattice Propel FPGA design software
- 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, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, February 2024

Section	Change Summary
All	Production release.

www.latticesemi.com