Lattice PCI Express Demo for Linux 2.4

User's Guide

November 2007
UG03_01.0

Lattice PCI Express Demo Overview

Introduction

This user's guide describes how to install and run the Lattice PCI Express Endpoint IP demo on a Linux system. The demo software runs on a Linux PC using Red Hat Enterprise Linux WS release 3 (Taroon Update 3) Kernel 2.4.21-20.ELsmp served as the development and test platform. Other revisions of Red Hat, or other Linux distributions, may also work. The complete source code for the driver and demo applications, as well as the build environment, are included with the distribution so compiling for other systems is simple.

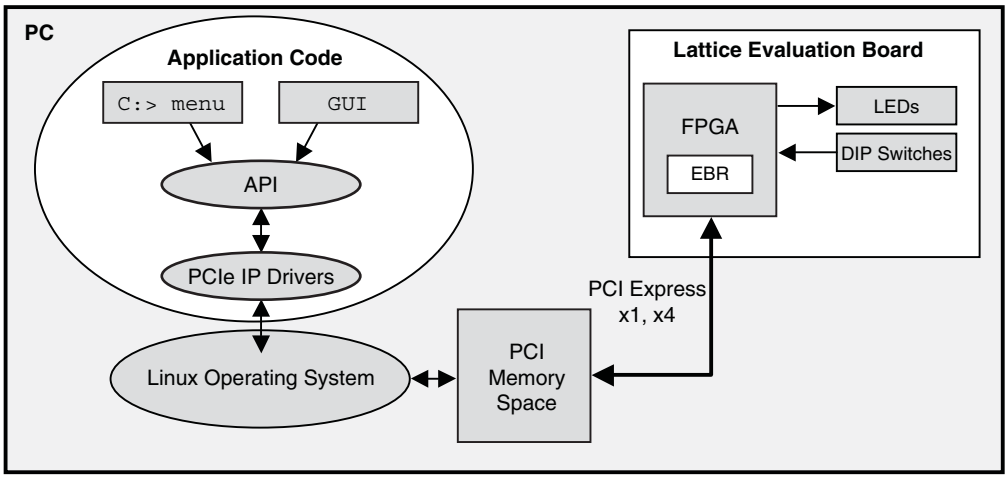
See TN1123, *Lattice PCI Express Demo Users Guide*, for a description of the demo operations and board components. The same LatticeSCM™ and LatticeECP2M™ PCI Express evaluation boards are used in Linux and Windows demos.

This guide covers the theory, installation and use of the Linux device driver and demo application code on a Linux system.

Demo Operations Overview

As mentioned, the demo executes on a standard PC running RedHat Linux OS, and accesses a Lattice PCI Express evaluation board installed in a PCI Express slot. Figure 1 shows the relationship of the hardware and software components of the demo.

Figure 1. PCI Express Block Diagram



The PCI Express IP core, in the Lattice FPGA on the evaluation board, acts as a PCI Express endpoint. A PCI Express endpoint device looks like a regular PCI device to application software executing on a PC. It is a memory-mapped device occupying a certain range(s) of the PCI memory space. When the PC boots, the BIOS and OS probe the PCI Express and PCI buses, detect the devices present on the buses, and assign them ranges in the PCI memory space. The PCI memory space is then mapped into the application software's memory space by the supplied driver and OS system calls. Once the mapping is done, the user-designed IP registers, sitting on top of the PCI Express IP stack, can be read/written as memory locations by the demo application software executing on the PC CPU. The demo software writes to the LED register, reads and displays the DIP switch setting and reads/writes the EBR memory.

The demo software shows that the Lattice PCI Express IP core correctly handles the PCI Express protocol in a PC through its interaction with the devices on the evaluation board. The demo exercises the following functions:

1. Displays operating system information on the detected Lattice evaluation board(s).
2. Displays information about the PCI Express IP core, such as reading and displaying all the pertinent information in the configuration registers, extended capability registers and control registers.

3. Performs General Purpose I/O (GPIO) Register Access: blink LEDs; read DIP switches and display value.
4. Performs Memory Access: write a pattern of values into internal FPGA EBR memory, read back and verify that all accesses are error-free.

Current Limitations

This preliminary release of demo software and demo IP core design does not support some advanced features and applications. The following features are not demonstrated:

- DMA
- Hardware interrupts or MSI
- Multiple VCs or traffic classes

Background Knowledge

This demo assumes the user is familiar with basic PCI Express technology and is comfortable installing new hardware and software packages in a Linux PC. Some experience in these areas is helpful when installing the evaluation board and software.

Installing the driver requires the root password. You may also need to rebuild the driver (and demo application) from the source if your kernel varies from the Red Hat 2.4.21 the binaries were generated with. The kernel header files must be installed on your system and you should be familiar with building kernel drivers from source code.

A good resource is *Linux Device Drivers*, by A. Rubini and J. Corbet. This book details all aspects of Linux driver development.

Installation Guide

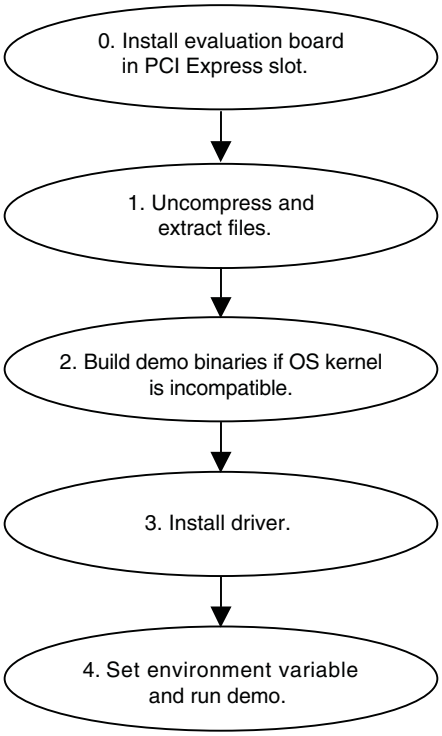
This section discusses the installation of the demo package, including software installation and the evaluation board hardware. Please read this section completely before attempting to install the package so that you understand the steps involved and how they apply to your situation.

The Lattice PCI Express demo package is released as a compressed Linux tar file. The package includes the Linux driver, FPGA bitstreams, Java GUI, and all demo source code. The file must first be uncompressed and then extracted into the final destination directory. A script can then be run to install the device driver and build the specific system files. Once this is complete, you are ready to set up an environment variable and run the executable.

Installation Overview

The following steps are taken to install and run the demo.

Figure 2. Demo Installation Flow



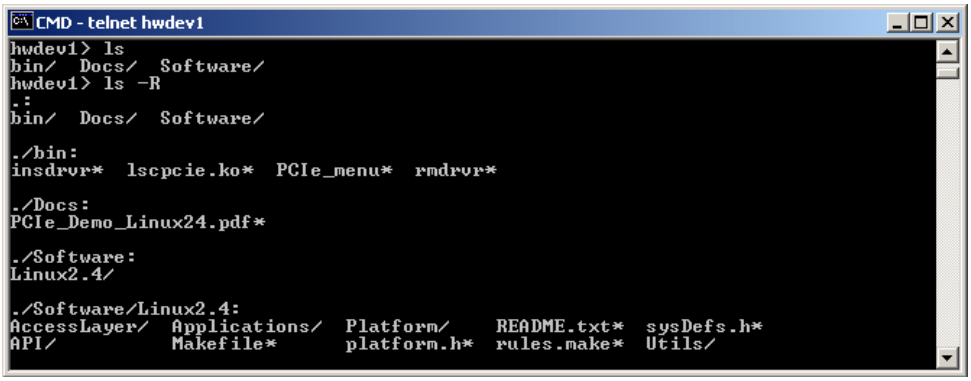
Demo Package Software Installation

The demo requires a Lattice PCI Express evaluation board to communicate with. If you do not have an evaluation board installed in the PC, the demo will not run. The driver will not have anything to open, and the demo application will not have anything to communicate with.

1. Obtain the gzipped tar file and place it in the directory you wish to work from.
2. Uncompress it with: > gunzip PCIeDemo*.tar.gz
3. Untar it with: > tar -xvf PCIeDemo*.tar

The directory structure is as follows:

Figure 3. Directory Structure



The bin/ directory contains pre-built Linux 2.4 binaries and driver installation scripts.

The Docs/ directory has documentation (this file only).

The Software directory contains the source code for the demo and the driver.

1. Change to the bin/ directory of the release.
2. Become root.
3. Install the driver by using the script ./insdrvr.
4. Return to a normal user account (you will not need root privileges anymore).
5. Verify the driver installed by issuing the following commands:
   - ls -l /dev/lscpcie (to see a list of LatticeSCM and LatticeEC2M file names)
   - cat /proc/modules (to see lscpcie.ko listed, near the top)
   - cat /proc/driver/lscpcie (for information about the evaluation boards installed and what BAR resources they have been assigned)

To run the demo:

1. Set the environment variable PCIE_BOARD to the ID of the evaluation board installed. If you have a LatticeECP2M installed, export PCIE_BOARD=EC1. If you have a LatticeSCM board installed, export PCIE_BOARD=SC1.
2. Start the demo with ./PCIe_menu.

If the demo binaries will not run, see Building Demo Binaries from Source Code, later in this document, to produce the demo binaries.

Installing Java for the GUI

1. Go to the Sun Microsystems website and download JRE 1.5.0 (www.java.com/en/download/manual.jsp).
2. Choose the Linux self-extracting file (jre-1_5_0_10-linux-i586.bin), not the RPM. Read the installation instructions located there.
3. Save the JRE installation file into the <demo>/bin directory (or install globally on the machine).
4. chmod 777 on the jre*.bin file.
5. f.) ./jre*.bin and install.
6. Tell the demo GUI where to find the Java JRE: export DEMO_JRE="./jre1.5.0_10".
7. Run the demo GUI with ./rundemo. Choose the installed board to talk to: 1 = EC1.

Un-installing the Software

If you want to remove the software from your system (i.e., to perform a clean installation of a new version), simply run the rmdrvr script.

Delete the /dev/lscpcie/ directory and all its contents.

Delete the directory that the demo release was extracted into.

Environment Setup

This section notes any additional setup that may be necessary before running the demo.

Java

The GUI is written in Java and uses the Java Swing libraries for display. The Java 1.5.0 run-time libraries are included in the release and installed in the demo directory. No additional setup is required and the Java files included with the demo will not interfere with any other Java installations.

Environment Variables

The environment variable PCIE_BOARD needs to be set to indicate the Lattice PCI Express evaluation board the demo is to connect to and operate on. The variable can be set with the bash shell command:

```
> export PCIE_BOARD=SC1
```

The naming convention of the PCIE_BOARD boards is:

```
<BoardType><Instance>
```

Examples:

- SC1 = First LatticeSCM evaluation board installed
- SC2 = Second LatticeSCM evaluation board installed
- EC1 = First LatticeECP2M evaluation board installed

Building Demo Binaries from Source Code

In the event that the binaries in this release are incompatible with the installed Linux distribution, the binaries can be built from the source code.

Prerequisites:

- GCC and other compiler tools
- Kernel source header files

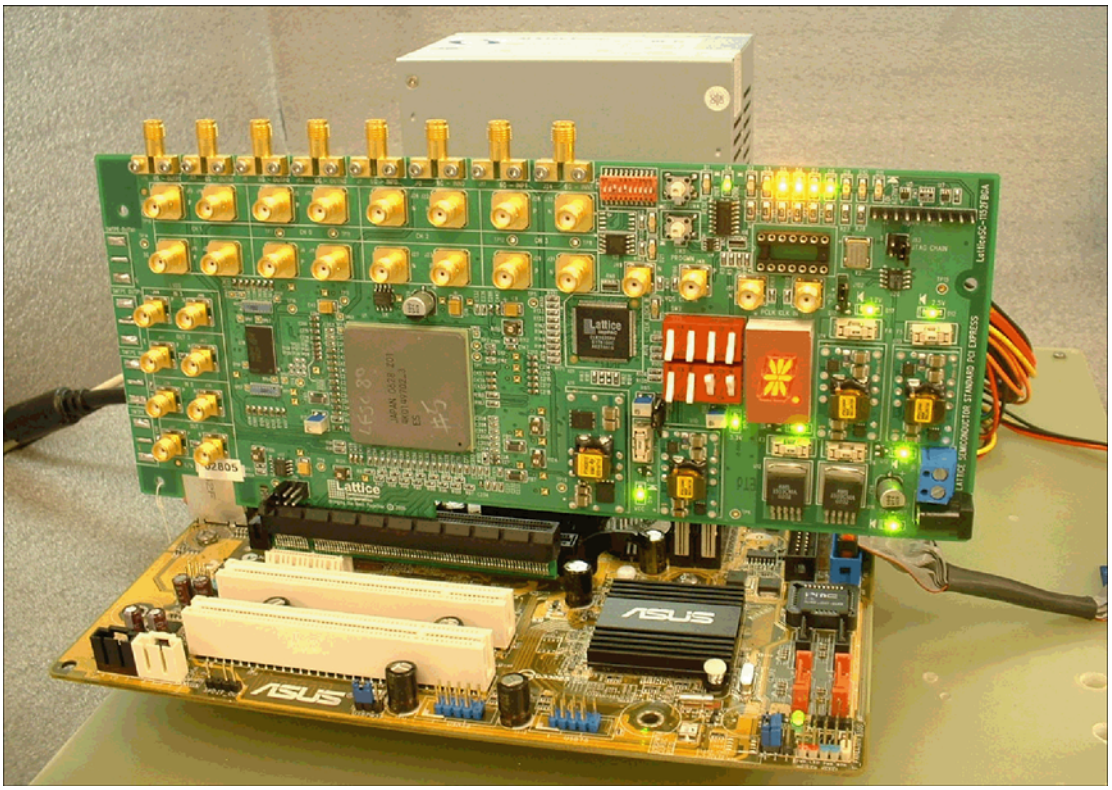
Steps:

1. Change to the Linux2.4 directory (i.e., `cd PCIeDemo/Software/Linux2.4`).
2. Create the PLATFORM_DIR environment variable with: `export `pwd``.
3. Run the make file at the top level to build dependencies: `make depends`.
4. Build the source files into the executable demo: `make`.
5. Build the driver:
   - Change to `AccessLayer/drv/`.
   - Run `make`.
6. Install the driver object file `lscpcie.ko`, as described previously.

Running the Demo

An operational demo is shown in Figure 4.

Figure 4. Demo Setup



This is a LatticeSCM80 evaluation board installed in a standard PC motherboard. Note that the 16-segment display has been set by the demo to display an "*". The four status LEDs at the top right are lit, indicating a functional PCI Express link. The decimal point on the 16-segment display is lit, showing that PCI Express memory reads and writes are taking place over the bus.

Note that this demo functions the same in a Windows PC and a Linux PC. The same application code and demo API source code are used for both systems; only the OS-specific drivers differ.

Demo Setup

When the PC is powered on, the status LEDs will light to indicate a good communication link over the PCI Express bus. All four LEDs on the top right of the board must be lit. Their functions are listed in Table 1.

Table 1. LED Functions

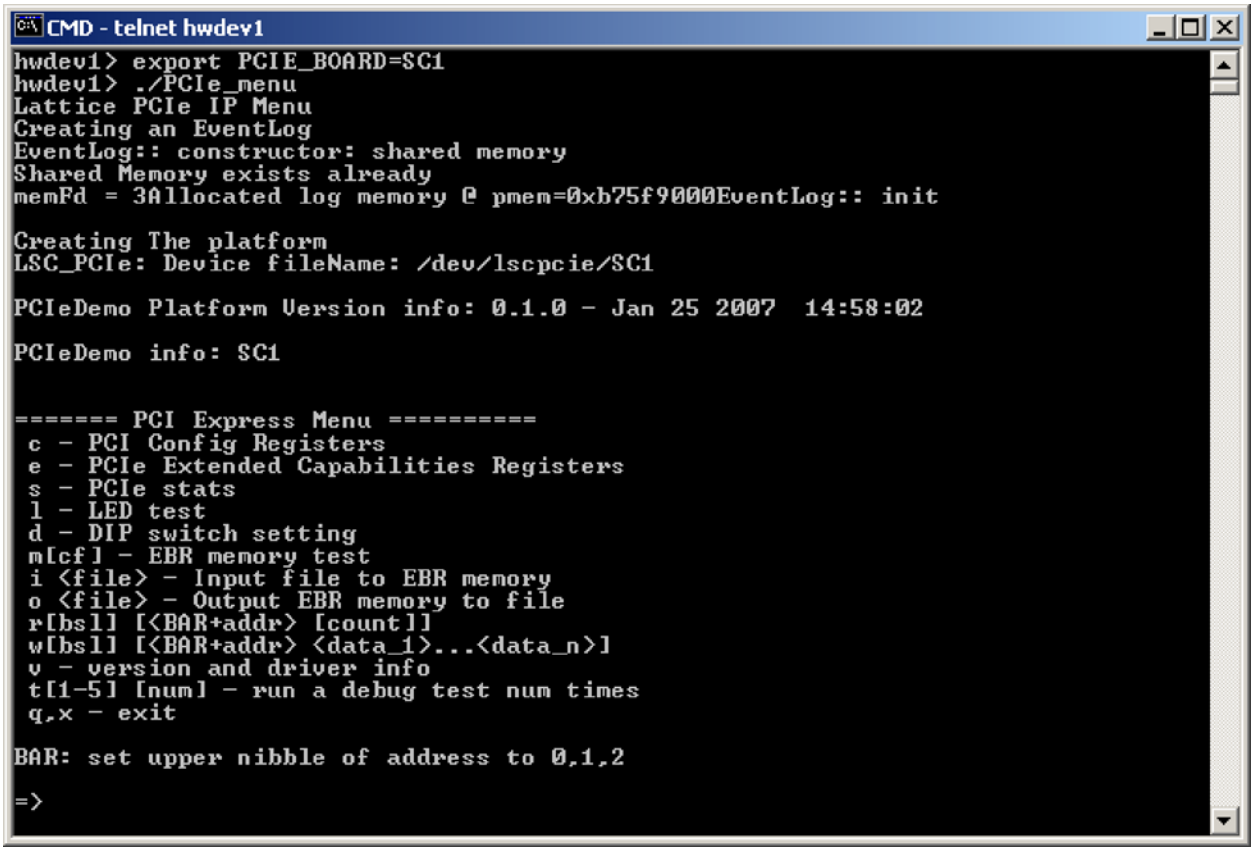
D5 (yellow)	D6 (yellow)	D7 (green)	D8 (green)
PCI-E Clock	LTSSM Polling	L0 state	DL up
PLL has clock form PCI-E slot and IP running	LTSSM completed detect state	Training sequence complete	TL ready for packets

Running the Demo (Menu Mode)

The text menu program can be started from a shell prompt. As stated, the demo uses the environment variable PCIE_BOARD to determine which board to communicate with. If you have a LatticeECP2M board, set PCIE_BOARD = EC1. If you have a LatticeSCM board installed, set PCIE_BOARD = SC1.

Start the demo by executing `./PCIe_demo`.

Figure 5. Running the Demo

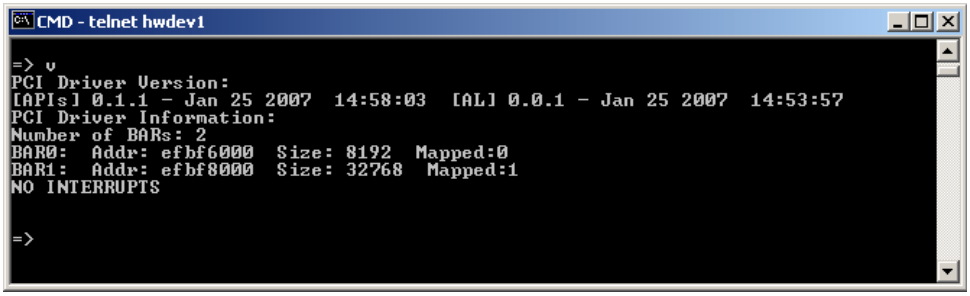


The top lines display information about the board and the driver installation. The lower half of the screen shows the various menu options available. Type the letter, or command string, and press **<Enter>** to execute. **Q** or **X** exits back to the command prompt.

Type **V** to display details about the demo and driver versions, and hardware resources used. This indicates that the evaluation board hardware was recognized as a PCI device by the PC and Linux and assigned address spaces corresponding to the BARs programmed into the FPGA IP core.

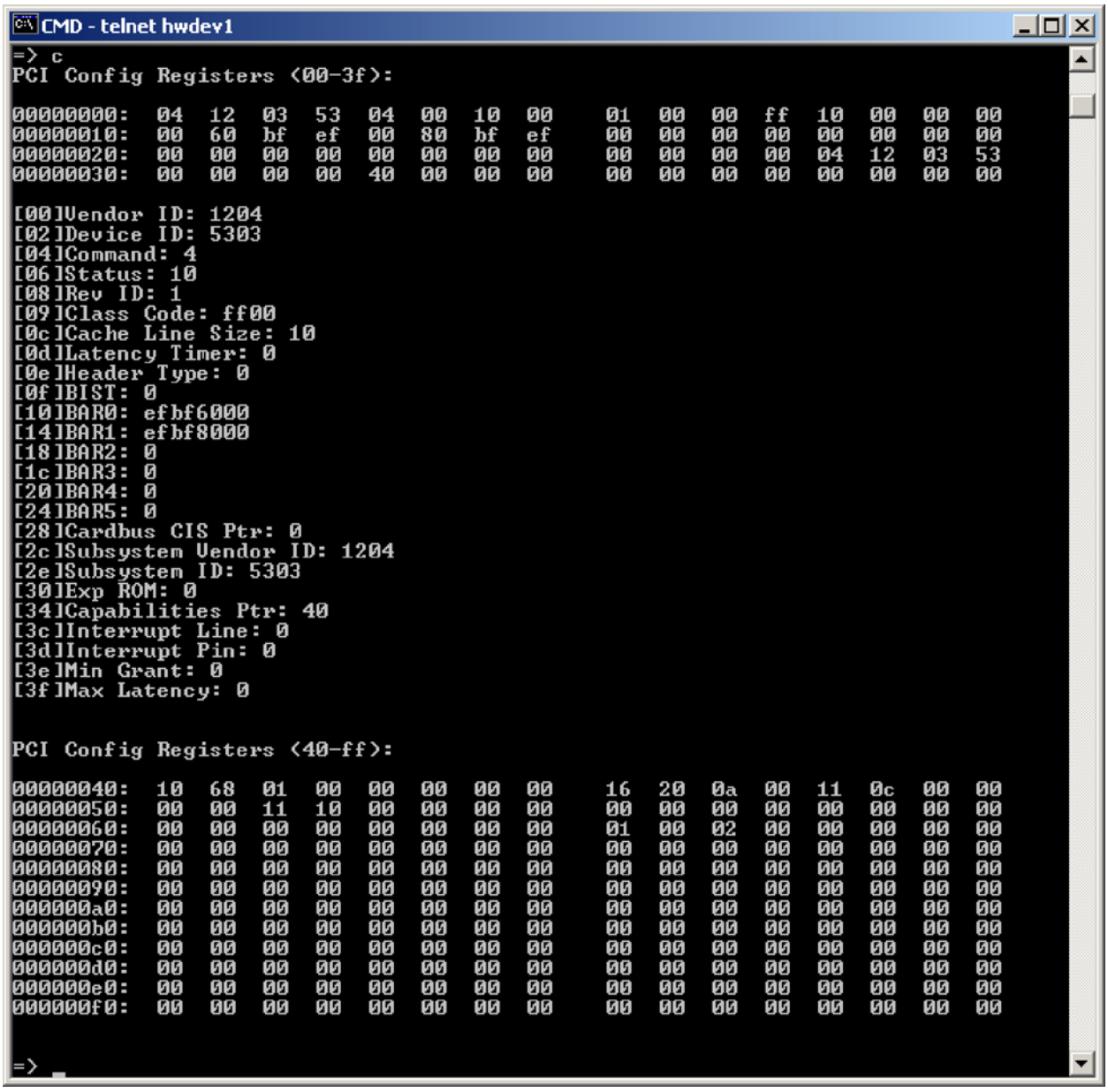
Version and Information Commands

Figure 6. Version Command



Type **C** to display the PCI Config Type 0 registers. The standard 256 bytes of device configuration space are read by the driver and displayed.

Figure 7. Displaying Config Registers



The top portion of the screen displays the standard 64 bytes that contain the required PCI fields. Their values are then displayed in a more readable format. The bottom portion of the screen shows the extended capabilities. This area contains the PCI Express Capabilities structure and Power Capabilities structure. Remember that configuration space is little endian byte order per the PCI specification.

Memory Access Commands

Use the **r** and **w** commands to read and write registers in the user space of the design. Consult the memory map of the demo IP for register addresses. The following command reads back all GPIO registers which are in BAR 1.

Figure 8. Read Memory Command

The **r** and **w** commands can specify 8, 16, or 32-bit accesses using "rb", "rs" and "rl", respectively. The BAR to read from is indicated in the most significant nibble. In the above command, the "1" in 10000000 indicates access offset 0 (the remaining 0000000) in BAR 1. This command format requires typing all eight digits of the 32-bit address. The command "rl 0" reads from BAR0 offset 0.

This command sequence shows writing a 32-bit word (0x44332211) into the scratchpad register and then reading it back:

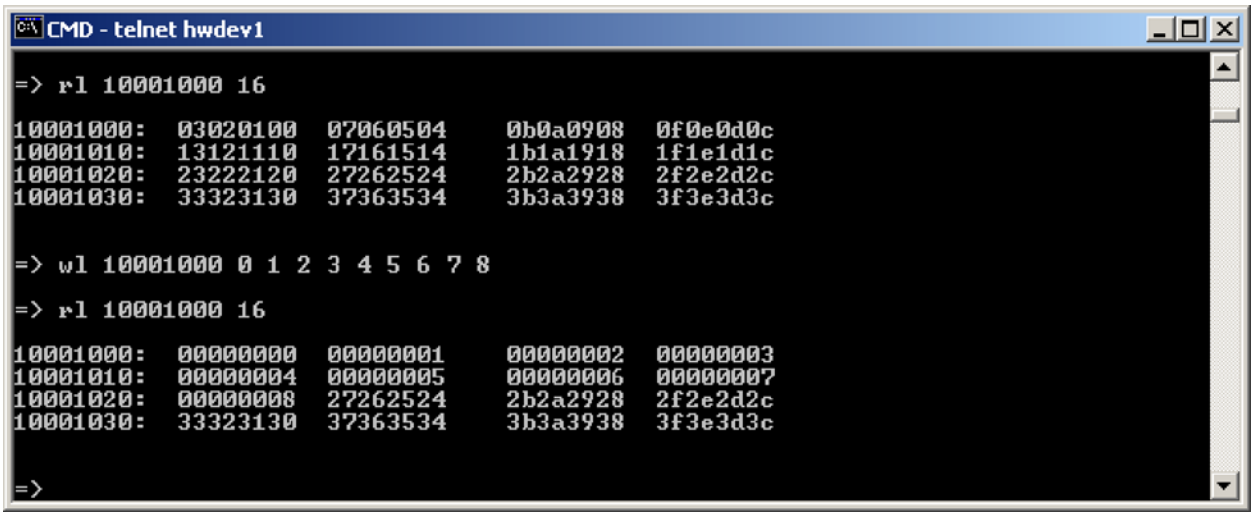
Figure 9. Write Memory Command

The following command sequence toggles the 16-segment LEDs on and off:

Figure 10. Write to LEDs

The following commands show access to the EBR memory, which is located at offset 0x1000 in BAR1. The first command displays the current, uninitialized contents. The second command writes eight 32-bit words into memory, and the third command displays the modified memory locations.

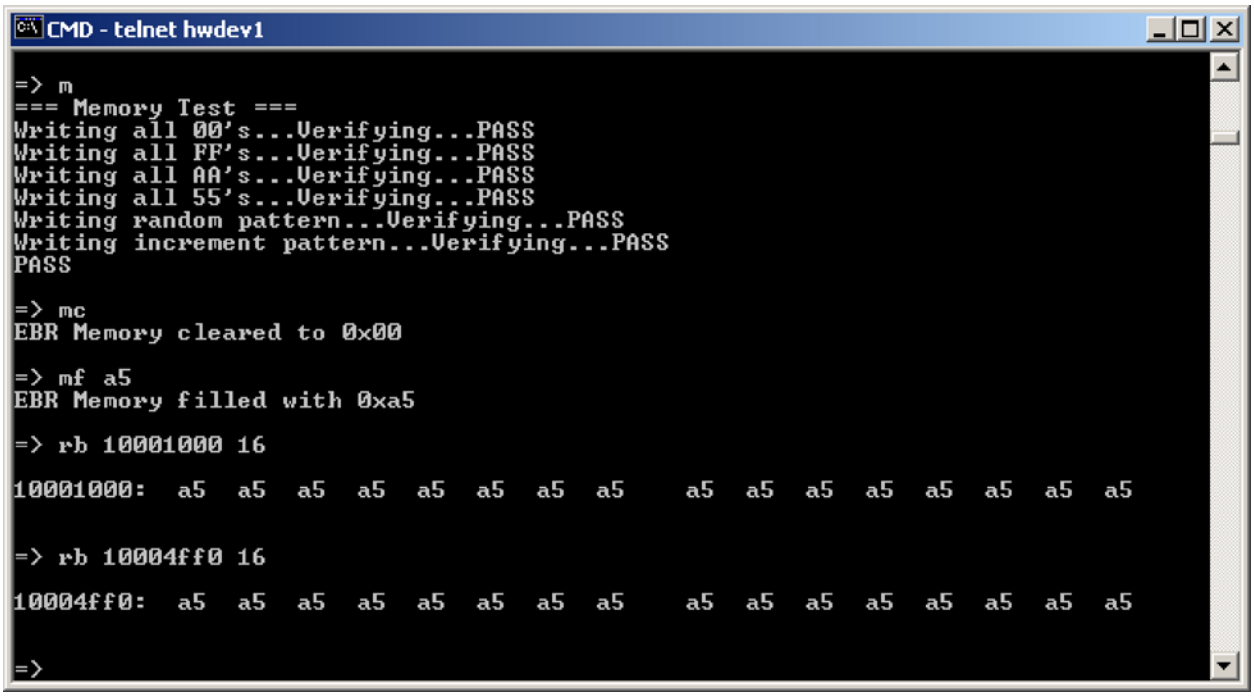
Figure 11. Modifying Memory



Test Commands

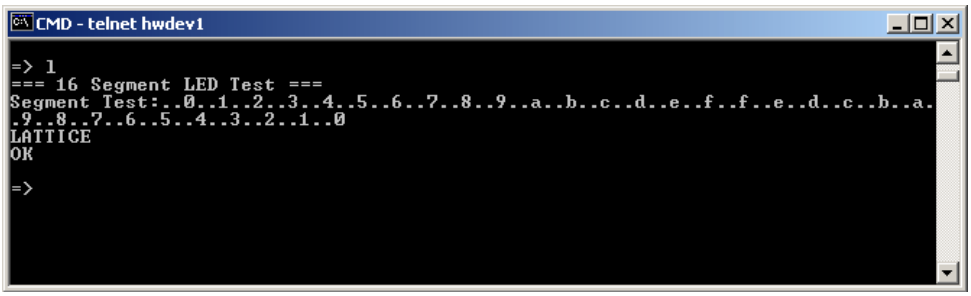
EBR memory testing is performed using the **m** menu command. The **m** command alone runs a series of tests that fill the entire 16kB EBR with various patterns and verifies every location has the correct value. The command **mc** clears all contents to zeros. The command **mf <hex>** fills all memory locations with the same byte value.

Figure 12. Memory Test Commands



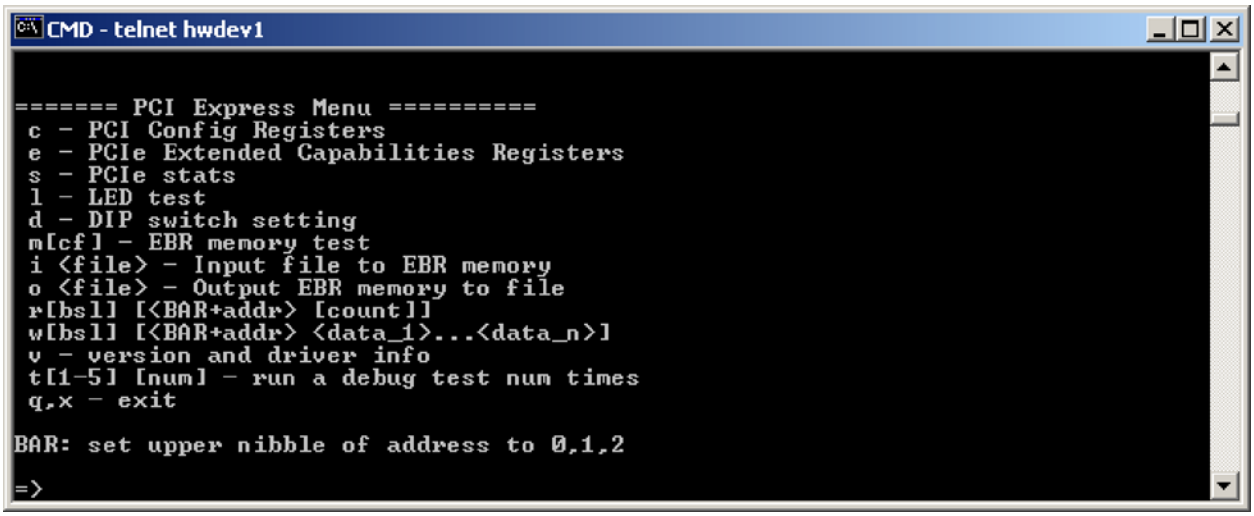
The 16-segment LED (on certain boards) can be exercised using the **L** command. The segments of the LED are lit one at a time and then shut off in reverse order. The letters LATTICE are then displayed, ending with the display of an asterisk (*).

Figure 13. LED Test Command



Other Menu Functions

Figure 14. List of Menu Commands



The **i <file>** and **o <file>** commands are used to read the contents of a file into EBR memory and to store the contents of EBR memory to a file. This allows a specific pattern to be loaded into the EBR and read back to verify all locations are operational.

The **d** command displays the DIP switch settings. It reads the switch 10 times over 10 seconds, allowing you to change the switch positions in real time to see the driver respond.

The **t[1-5]** functions perform various repetitive tests to stress access to the EBRs and registers. These tests are useful to capture TLP transfer anomalies using an analyzer connected to the PCI Express bus. The test specifies an optional count argument that indicates the number of times to repeat the test. If the argument is not given, the default number of 100000 is used. To run the test without stopping, specify -1 for the number (you will need to Ctrl-C to abort).

- **t1 [n]** – Read the demo ID register n times. The value read is compared to 0x53030100. If it differs, the test stops with an error.
- **t2 [n]** – Write a value to the scratchpad register. The same value is written over and over and is not read back. This test can be used to monitor memory write TLPs.
- **t3 [n]** – Write a random 32-bit value to the scratchpad register. Read the value back and compare to what was written. This test can be used to monitor memory write and read TLPs and verify register access to the hardware.
- **t4 [n]** – Write random 32-bit values to sequential memory locations in the EBR. Each value that is written is read back to verify. This test runs the number of times specified, wrapping back to the start of EBR when in reaches the end.

- **t5 [n]** – The same as t4 except the data size is in bytes, to verify byte lanes are operating properly.

Demo Design Details

This section provides technical details of the demo system design to give users a better understanding of how the PCI Express IP core is implemented and demonstrated. The supporting IP around the PCI Express core is described in detail. The software application and driver design are also discussed.

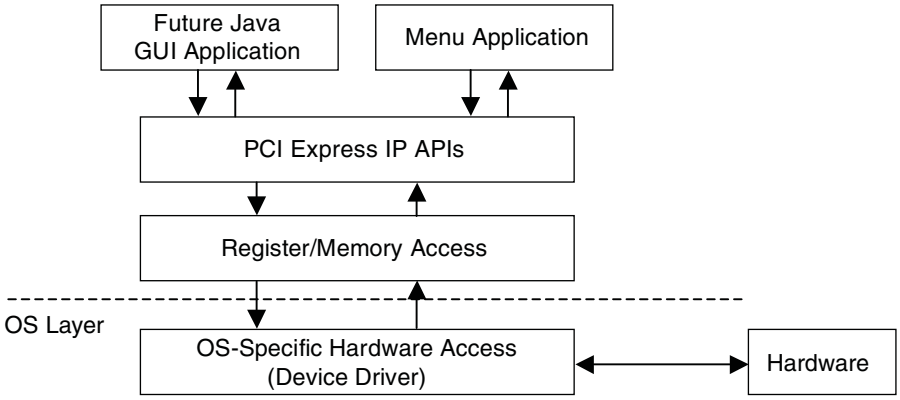
IP Components

Please refer to the corresponding section in TN1123. The hardware and IP are identical for either the Linux or Windows demo.

Demo Software Components Overview

The demo software consists of the command line menu (a Java GUI in the future) and a Linux device driver. The components are divided into the hierarchical layers shown in Figure 15.

Figure 15. Demo Software Components



OS-Specific Hardware Access (Linux Device Driver)
This portion of the software provides the OS driver to allow application software to "open" a device and gain access to the evaluation board. This code is Linux kernel 2.4.x and Lattice-specific in that it will look for the PCI Device ID, Vendor ID, etc. to know that it is talking to the Lattice PCI Express evaluation board. The driver, a Linux 2.4.x kernel module, is loaded with the **insmod** command. The driver provides the mapping between the device node in user space (found in `/dev/lscpcie/`) and the hardware device (the evaluation board). Once the driver is loaded, it searches for known Lattice devices, records device resources (how many BARs, sizes, interrupts, etc.) and maps the BARs into user memory space that the driver code can access. The driver also provides an interface to allow user applications to read or write data to a hardware device. The user application opens the device driver through standard OS system calls (open, close, ioctl) The drive maps the board's BAR into a region of memory. It is always invoked by the user space code to perform a read/write, and the results are returned back to the user code.

Register/Memory Access
This portion of the software provides an OS-independent set of functions to read and write to a hardware device. The upper layers of the application code can then be designed to be OS independent and can be built to run on Windows or Linux. This layer is similar to Windows HAL.

PCI Express IP APIs
These API functions provide simplified access (helper functions) to the standard PCI configuration registers and the Lattice PCI Express demo IP specific registers. The APIs access:

1. PCI configuration registers – BARs, device ID

2. Extended capabilities registers

3. Demo GPIO registers – scratchpad, LEDs, switches, EBR

4. Driver version information

Application Software
The application software is the demo or test code that is used to demonstrate PCI Express operations with a Lattice FPGA. The demo software is either a Java GUI or a simple console-based text menu. Both applications use the APIs provided to access the PCI Express driver to:

1. Perform memory reads/writes to access the LEDs and switches to show real-time control.

2. Stress test the PCI Express interface via high-throughput accesses to the EBR Memory and memory tests to verify that the contents are correct.

The Java GUI application uses JNI methods to invoke functions in a DLL that, in turn, call the APIs. The JNI methods allow the C code to be invoked by Java. The advantage is that most of the elementary demo operations are handled by the API library which is shared by the text menu and the GUI. Functional changes only need to be made in one place, and the user can be assured that the operations in the GUI and menu are equivalent.

Linux Device Driver Design

This section describes the implementation of a Linux kernel module device driver for accessing the registers and memory resources on a PCI Express board. The driver is designed to be generic. The details of the hardware design are deferred to the user space software. The driver provides the most basic services to the user space client: read and write memory. This design is not intended for use in a commercial, production board. For a commercial product, most of the device functionality is encapsulated in the driver, and the user space interacts at a higher level (sending packets, receiving packets, clearing the screen, etc.), and the driver does all this internally.

The driver is one piece in a larger demo infrastructure, designed to be portable and adaptable to many different hardware platforms.

Implementation Decisions
The driver is intended to be a generic interface to the hardware device. The driver does not contain any specific knowledge of the hardware design implemented in the FPGA. It does not know the address of the LED registers or the EBR memory. It is intended to be as basic as possible so that new IP designs and/or demo changes do not require the driver to be modified.

The decision to have all demo knowledge reside at the API and application (menu) level is by design. The most basic function the driver can implement is providing access to the hardware through memory reads and writes. All knowledge of which device is at which address is contained in the API, not the driver.

The units of access are BARs. Each PCI Express device can have up to six BARs defined in the configuration space registers. Each BAR is a memory window into the device. Each BAR is associated with a device minor number.

Memory accesses are done by mapping the device driver's memory windows to the BAR into locked pages of the application space, through the **mmap** system call. This brings the memory mapped registers of the devices into the address space of the application. Page tables are set up in the application's memory space to directly translate the address into the bus address where the BAR resides.

/dev/lscpcie/ Architecture: The user space code needs handles to identify the device it wants to open. This is the common Unix approach to hardware; everything is viewed as a file. Identifying hardware is done through special file names in the `/dev` directory. The Lattice PCI Express demo creates a new directory named `/dev/lscpcie/` and populates it with a number of files that name potential LatticeSCM or LatticeECP2M boards that can be installed in the system. Four boards of each type are created and each board can have up to six BARs. The file names are of the format:

- SC[n]_[bar]
- EC[n]_[bar]
- SC = LatticeSCM PCI Express evaluation board
- EC = LatticeECP2M PCI Express evaluation board
- n = board number 1-4
- bar = PCI BAR number 0-5

Boards are identified by their type (LatticeECP2M or LatticeSCM) and enumeration order. SC1 = the first LatticeSCM board detected in the system. EC2 = the second LatticeECP2M board. If a board does not exist (i.e., there is no LatticeECP2M board installed or there are not two LatticeECP2M boards, only one) then the open() call will fail for that device.

Notes on Board Order: The order is determined by how the OS implements the PCI and PCI Express bus probing on a motherboard. The numbering of buses can be logical, and not physically tied to a slot. Some experimentation may be necessary to determine which of two identical boards is actually the "first". The simplest thing to do is run the demo on SC1 and blink the LEDs to identify which board is physically in SC1.

The LatticeECP2M or LatticeSCM boards are identified by the unique Device ID code in the configuration registers. They each have the same vendor number (Lattice) but different Device IDs.

The `/dev/lscpcie/` device directory is populated with every possible board and BAR configuration, even if only one (or no) board is installed. This provides for future growth (i.e., more BARs used in the demo or more boards installed) without regenerating the device nodes or driver. Stated another way, the `/dev/lscpcie/` nodes provide access to the potential devices installed and does not represent the physical hardware that is installed at any one time.

Device Major Numbers: The device major number is dynamically allocated when the **lscpcie.ko** module is installed. This avoids potential collisions with other device drivers the user may have installed that use a fixed major number. The major number chosen can be seen by cat'ing the /proc/modules file. An install script is used to perform the following tasks:

1. Remove any old device nodes under `/dev/lscpcie/` (their major numbers may change).
2. Install the driver module into the kernel (get new major number) with `insmod`.
3. Read the dynamically assigned major number from the `/proc/modules` file.
4. Generate [SC|EC][n]_[bar] entries in `/dev/lscpcie/` using the new major number.

Device Minor Numbers: The device minor number is used to indicate to the driver the type of device (LatticeECP2M or LatticeSCM), the instance of the device (1-4) and the BAR (0-5). Each device entry has the same major number (all use the same driver, **lscpcie.ko**) and a unique minor number. The minor numbers are encoded as:

Minor_Number = [Device]*100+[Instance]*10+[BAR]
Device: SC = 0, EC = 1
Instance: 1 - 4
BAR: 0 - 5, 9

The following show some example minor numbers encoding and the devices they refer to:

- 010 = LatticeSCM, first one, BAR0
- 122 = LatticeECP2M, second one installed, BAR2
- 019 = Globab PCI resources of LatticeSCM board 1 (configuration registers., IRQ, etc.)

- 000 = Not valid
- 210 = Not valid - no device type = 2
- 100 = Not valid - LatticeECP2M device type, but 0'th one installed is wrong.

The evaluation board reference/numbering system is similar to the approach taken with hard drives. In Linux, sda refers to the first SCSI disk drive as a whole; sda1 refers to partition 1 of that drive; sda2 refers to the second partition, etc. The disk driver knows how to read and write sectors to the disk, and its the file system that implements the real use of the disk and partitions.

Note: Major and minor numbers and the device node are created using the system call mknod. For example: mknod /dev/lscpcie/SC1_0 c 241 10.

This creates a special character device node file in `/dev/lscpcie/` named SC1_0 with a major number of 241 and a minor number of 10. You can see the major/minor numbers assigned to each device by executing an `ls -l /dev/lscpcie`.

Driver Install Script
A shell script is used to install the driver module and handle the administrative tasks of discovering the assigned major number and generating the device nodes using that major number. The script must be run as root because the system calls need administrator privileges. The script is located in the `/bin` directory and the `Software/Linux24/AccessLayer/drvr/` directory.

Driver __init
The entry point of the driver when it is installed using insmod. The driver detects all Lattice evaluation boards in the system and assigns them names based on their type (LatticeSCM or LatticeECP2M) and their discovery order, i.e. SC1, SC2. The init module also records the PCI resources the board has been allocated.

Driver __exit

Driver open()
Opening a device with a sub number (_0, _1) causes the address range of that BAR number to be mapped into the kernel's space. This is done in preparation for mmap()'ing the devices hardware into the user's space through mmap().

Driver close()
Closing a device BAR causes the mapping to be released. Closing a device (SC1 or EC1) causes the PCI to disable the device.

Driver read() and write()
The standard read() and write() system calls are not implemented for this driver. They do not directly map to random access memory map accessible devices. Read and write are intended for file or stream type devices. This implementation is a pure memory mapped, register-based device.

Device memory is instead accessed using a user space pointer that is mmap()'ed to point directly into the BAR address range that is mapped onto the PCI Express bus.

Driver mmap()
Implements the Unix standard mmap() system call for this device (the evaluation boards). This system call is made by the demo code to map the base address of the hardware registers into a C pointer that is then used to read/write all demo IP in the FPGA.

Driver ioctl()
Implements the Unix standard ioctl() system call for this device. The only implemented function at the moment returns a structure containing the PCI resource details of the board. This is used to display information in the menu.

Troubleshooting

This section outlines some debug procedures to follow if you experience trouble installing or running the demo.

Trouble Installing the Demo Software Package

The most likely issue that could arise is the issue of permissions when installing

Trouble with the Board

Ensure the board is installed in a PCI Express slot. It can physically fit into a PCI slot. This could damage the board or PC if power is applied when it is in the wrong type of slot.

Ensure the board has a valid PCI Express bitstream loaded in the SPI Flash and for LatticeECP2M that the Mode DIP switches are set to program from SPI Flash (does not apply to LatticeSCM evaluation boards).

Ensure the four Status LEDs are on, indicating the board is seen as a PCI Express endpoint. If the two yellow LEDs and two green LEDs are not on, the board will not be recognized by the PC BIOS or Windows. You can try installing in a different PCI Express slot to see if that fixes the link-up problem. You can also try pressing the evaluation board's reset button immediately after the PC boots.

Ensure the board is seen by Linux. Check `/proc/pci` to see that a Lattice board is present

Verify the Vendor ID and Device ID are valid.

Trouble with the Driver

You will need root permission to install the device driver module and create the device nodes.

Trouble Running the Demo

The evaluation board must be installed in the PC, and seen by Linux, for the driver to be installed/loaded. The driver must be loaded by Linux in order to run the demo.

Debug Tools

Linux offers some utilities to interrogate the operation of hardware devices. Use these tools to verify the evaluation board device driver is loaded and running.

/proc/pci

/proc/modules

/proc/bus/pci

dmesg

References

The following documents provide more information on topics discussed throughout this user's guide:

- LatticeSCM PCI Express x1, x4 IP Core User's Guide
- LatticeSC Standard x8 PCI Express Evaluation Board Users Guide
- LatticeECP2M PCI Express Evaluation Board User's Guide
- Linux Device Drivers", A. Rubini & J. Corbet, ISBN 0-569-00008-1

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
November 2007	01.0	Initial release.