



Lattice PCI-Express Demo

Linux 2.4 Guide

Revision 0.1
January 26, 2007

For Demo Software 1.0.0
Lattice PCI-E IP Release 1.0
ispLEVER6.1 SP1
Linux 2.4 kernel

Table of Contents

1 Lattice PCI-Express Demo Overview.....	3
1.1 Introduction.....	3
1.2 Demo Operations Overview.....	3
1.3 Current Limitations.....	4
1.4 Background Knowledge.....	4
2 Installation Guide.....	5
2.1 Install Overview.....	5
2.2 Demo Package Software Installation.....	5
2.3 Un-Installing the Software.....	6
2.4 Environment Setup.....	7
2.4.1 Java.....	7
2.4.2 Environment Variables.....	7
2.5 Building the Demo Binaries from the Source.....	7
3 Running the Demo.....	8
3.1 Demo Setup.....	8
3.2 Running the Demo (Menu mode).....	9
3.2.1 Version and Information Commands.....	10
3.2.2 Memory Access Commands.....	11
3.2.3 Test Commands.....	13
3.2.4 Other menu functions.....	14
4 Demo Design Details.....	16
4.1 Demo Hardware Design.....	16
4.2 Demo Software Components Overview.....	16
4.2.1 OS Specific Hardware Access (Linux Device Driver).....	16
4.2.2 Register/Memory Access.....	16
4.2.3 PCI-E IP APIs.....	17
4.2.4 Application Software.....	17
4.3 Linux Device Driver Design.....	17
4.3.1 Implementation Decisions.....	17
4.3.2 Driver Install Script.....	19
4.3.3 Driver <code>__init</code>	19
4.3.4 Driver <code>__exit</code>	19
4.3.5 Driver <code>open()</code>	19
4.3.6 Driver <code>close()</code>	19
4.3.7 Driver <code>read()</code> and <code>write()</code>	19
4.3.8 Driver <code>mmap()</code>	19
4.3.9 Driver <code>ioctl()</code>	19
5 Troubleshooting.....	21
5.1 Trouble Installing the Demo Software Package.....	21
5.2 Trouble with the Board.....	21
5.3 Trouble with the Driver.....	21
5.4 Trouble Running the Demo.....	21
5.5 Debug Tools.....	21
5.5.1 <code>/proc/pci</code>	21
5.5.2 <code>/proc/modules</code>	22
5.5.3 <code>/proc/bus/pci</code>	22
5.5.4 <code>dmesg</code>	22
6 Reference Information.....	23
7 Technical Support Assistance.....	23

1 Lattice PCI-Express Demo Overview

1.1 Introduction

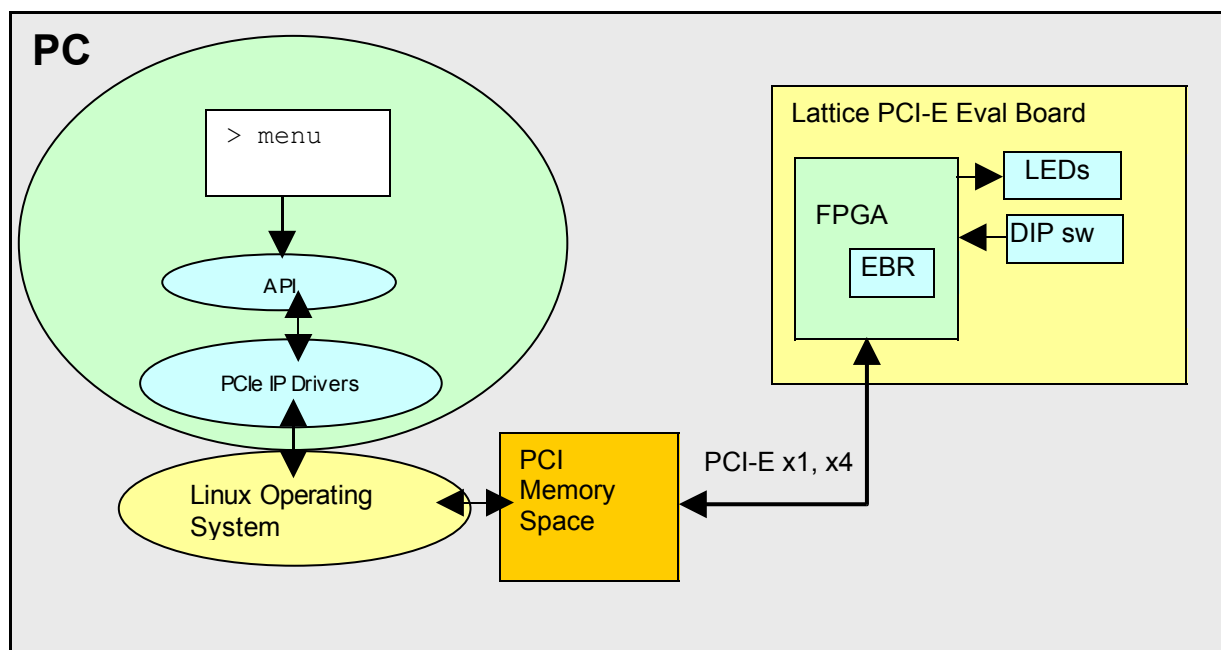
This User's Guide describes how to install and run the Lattice PCI-Express (PCI-E) Endpoint IP demo on a Linux system. The demo software runs on a Linux PC. Red Hat Enterprise Linux WS release 3 (Taroon Update 3) Kernel 2.4.21-20.ELsmp has been the main development and test platform. Other revisions of RedHat, 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 should be simple.

See the Lattice PCI-E Demo Users Guide (TN1123) for a description of the demo operations and board components. The same Lattice SCM and ECP2M 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. The

1.2 Demo Operations Overview

As mentioned, the demo executes on a standard PC running RedHat Linux OS, and accesses a Lattice PCI-E evaluation board installed in a PCI-E slot. The following figure shows the relationship of the hardware and software components of the demo.



The PCI-E IP, in the Lattice FPGA on the eval board, acts as a PCI-E endpoint. A PCI-E endpoint device looks like a regular PCI device to application software executing on a PC. It is just 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 and 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, user design 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-E IP correctly handles the PCI-Express protocol in a PC through the interaction with the devices on the eval 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 LED's; read DIP switches and display value
4. Performs Memory Access: write a pattern of values into internal FPGA EBR memory, read back and verify the all accesses are error-free.

1.3 Current Limitations

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

- DMA
- Hardware interrupts or MSI
- Multiple VC's or traffic classes

1.4 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 to install the evaluation board and the software.

Installing the driver will require the root password. It is also possible that you may need to rebuild the driver (and demo application) from the source if your kernel varies from the RedHat 2.4.21 the binaries were generated with. You will also need the kernel header files installed on your system and should be familiar with building kernel drivers from source.

A great resource is "Linux Device Drivers" by A. Rubini & J. Corbet. This book details all aspects of Linux driver development.

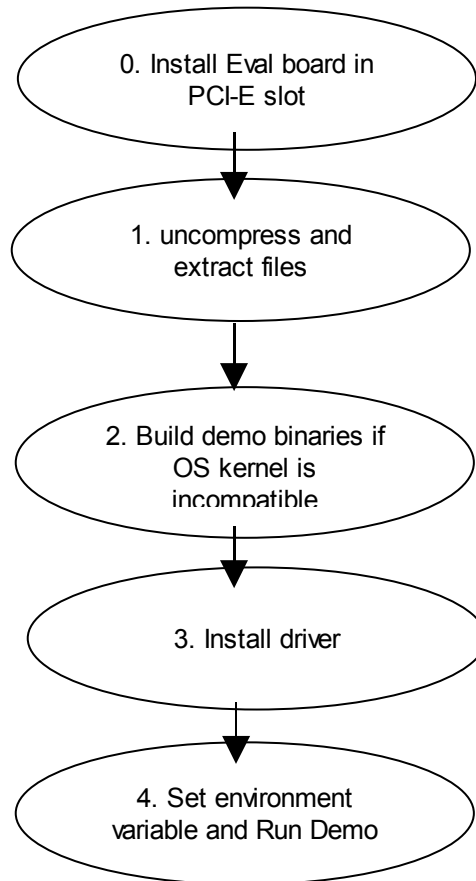
2 Installation Guide

This section discusses installing the demo package. The installation of the software and the eval board hardware are discussed. Please read this section completely before attempting to install the package so that you understand the steps involved and have planned how they apply to your specific situation.

The Lattice PCI-E 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 the final destination directory. A script can then be run to install the device driver and build the specific system files. After that, all that is needed is to setup an environment variable and run the executable.

2.1 Install Overview

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

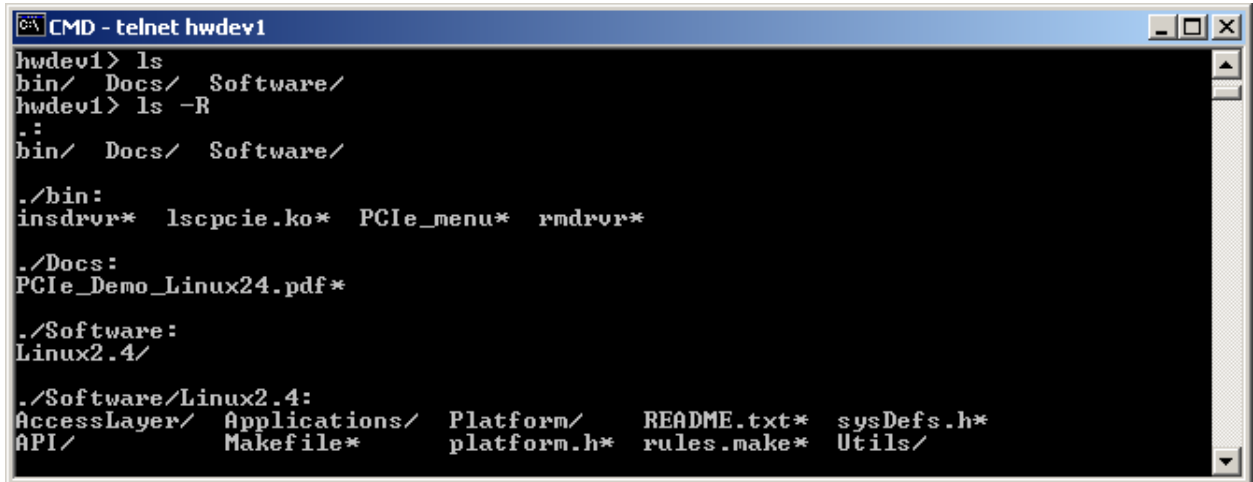


2.2 Demo Package Software Installation

The demo requires a Lattice PCI-E Eval Board to communicate with. If you do not have an eval 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 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:



```

CMD - telnet hwdev1
hwdev1> ls
bin/  Docs/  Software/
hwdev1> ls -R
.:
bin/  Docs/  Software/

./bin:
insdrv*  lscpcie.ko*  PCIE_menu*  rmdrvr*

./Docs:
PCIE_Demo_Linux24.pdf*

./Software:
Linux2.4/

./Software/Linux2.4:
AccessLayer/  Applications/  Platform/  README.txt*  sysDefs.h*
API/          Makefile*       platform.h*  rules.make*  Utils/

```

The bin/ directory contains pre-built Linux 2.4 binaries and driver install 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 **./insdrv**
4. return to a normal user account (don't need root privileges anymore)
5. verify the driver installed by issuing the following commands:
 1. **ls -l /dev/lscpcie** - you should see a list of SC and EC file names
 2. **cat /proc/modules** - you should see lscpcie.ko listed (near the top)
 3. **cat /proc/driver/lscpcie** - you should see info about the Lattice eval boards installed and what BAR resources they have been assigned.

Running the demo:

1. First set the environment variable PCIE_BOARD to the id of the eval board installed. If you have an ECP2M installed: **export PCIE_BOARD=EC1** If you have an SC board installed: **export PCIE_BOARD=SC1**
2. Start the demo with **./PCIE_menu**

If the demo binaries will not run, see section 2.5 on recompiling the source code to produce the demo binaries.

2.3 Un-Installing the Software

In the event you want to remove the software from your system (ie. to perform a clean install 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.

2.4 Environment Setup

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

2.4.1 Java

The Java GUI is not supported in Linux yet.

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.

2.4.2 Environment Variables

The environment variable PCIE_BOARD needs to be set to indicate the Lattice PCI-E 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 eval board installed
- SC2 = second LatticeSCM eval board installed
- EC1 = first LatticeECP2M

2.5 Building the Demo Binaries from the Source

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

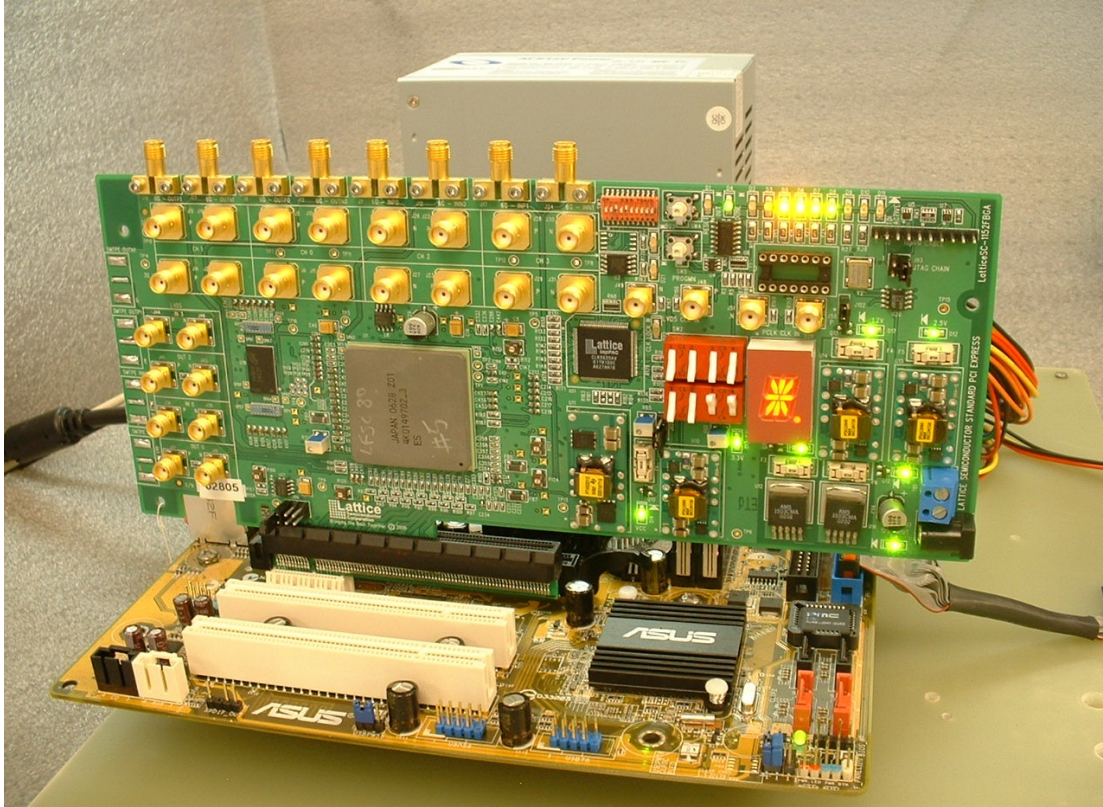
- gcc and other compiler tools
- kernel source header files

Steps:

1. Change to the Linux2.4 directory (ie. `cd PCIDemo/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:
 1. change to `AccessLayer/drv/`
 2. run `make`
6. Install the driver object file `lscpcie.ko` as describe previously.

3Running the Demo

An operational demo looks something like this:



This is a LatticeSCM80 eval board installed in a standard PC motherboard. Note the 16 segment display has been set by the demo to display an “*.”. The 4 status LEDs at the top right are lit, indicating a functional PCI-E link. The decimal point on the 16 Segment display is lit showing PCI-E memory reads and writes are taking place over the bus.

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

3.1 Demo Setup

When the PC is powered on, the status LEDs should light to indicate a good communication link over the PCI-E bus. All four LEDs on the top right of the board must be lit. Their functions are:

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

3.2 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 an ECP2M board, set `PCIE_BOARD = EC1`. If you have an SC board installed, set `PCIE_BOARD = SC1`.

Start the demo by executing `./PCie_demo`

```

CMD - telnet hwdev1
hwdev1> export PCIE_BOARD=SC1
hwdev1> ./PCie_menu
Lattice PCie IP Menu
Creating an EventLog
EventLog:: constructor: shared memory
Shared Memory exists already
memFd = 3Allocated log memory @ pmem=0xb75f9000EventLog:: init

Creating The platform
LSC_PCIE: Device fileName: /dev/lscpcie/SC1
PCieDemo Platform Version info: 0.1.0 - Jan 25 2007 14:58:02
PCieDemo info: SC1

===== PCI Express Menu =====
c - PCI Config Registers
e - PCie Extended Capabilities Registers
s - PCie stats
l - LED test
d - DIP switch setting
m[cf] - EBR memory test
i <file> - Input file to EBR memory
o <file> - Output EBR memory to file
r[bsl] [<BAR+addr> <count>]
w[bsl] [<BAR+addr> <data_1>...<data_n>]
v - version and driver info
t[1-5] [num] - run a debug test num times
q,x - exit

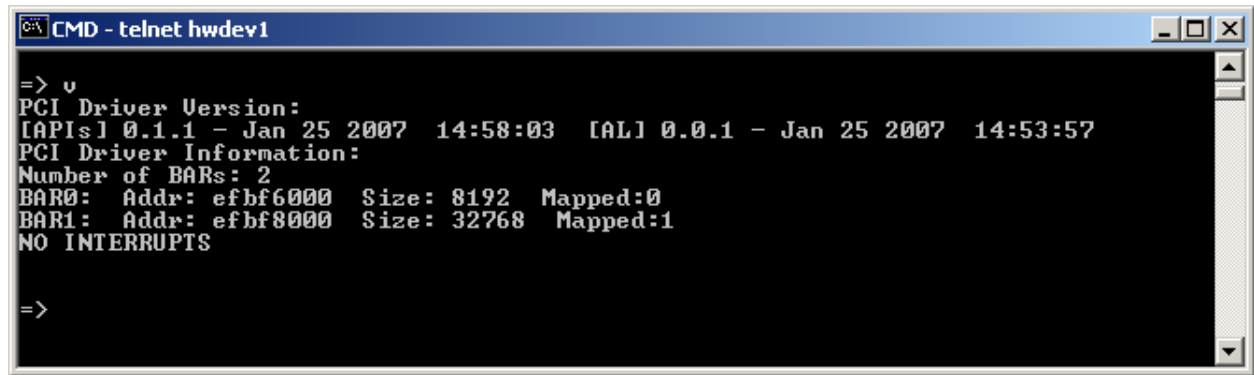
BAR: set upper nibble of address to 0,1,2
=>

```

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 version, and hardware resources used. This indicates that the eval 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.

3.2.1 Version and Information Commands



```
CMD - telnet hwdev1
=> v
PCI Driver Version:
[APIs] 0.1.1 - Jan 25 2007 14:58:03 [AL] 0.0.1 - Jan 25 2007 14:53:57
PCI Driver Information:
Number of BARs: 2
BAR0: Addr: efbf6000 Size: 8192 Mapped:0
BAR1: Addr: efbf8000 Size: 32768 Mapped:1
NO INTERRUPTS

=>
```

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

```

CMD - telnet hwdev1
=> c
PCI Config Registers <00-3f>:
00000000: 04 12 03 53 04 00 10 00 01 00 00 ff 10 00 00 00
00000010: 00 60 bf ef 00 80 bf ef 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 04 12 03 53
00000030: 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00

[00]Vendor ID: 1204
[02]Device ID: 5303
[04]Command: 4
[06]Status: 10
[08]Rev ID: 1
[09]Class Code: ff00
[0c]Cache Line Size: 10
[0d]Latency Timer: 0
[0e]Header Type: 0
[0f]BIOS: 0
[10]BAR0: efbf6000
[14]BAR1: efbf8000
[18]BAR2: 0
[1c]BAR3: 0
[20]BAR4: 0
[24]BAR5: 0
[28]Cardbus CIS Ptr: 0
[2c]Subsystem Vendor ID: 1204
[2e]Subsystem ID: 5303
[30]Exp ROM: 0
[34]Capabilities Ptr: 40
[3c]Interrupt Line: 0
[3d]Interrupt Pin: 0
[3e]Min Grant: 0
[3f]Max Latency: 0

PCI Config Registers <40-ff>:
00000040: 10 68 01 00 00 00 00 00 16 20 0a 00 11 0c 00 00
00000050: 00 00 11 10 00 00 00 00 00 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 01 00 02 00 00 00 00 00
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

=>

```

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

3.2.2 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:

```

CMD - telnet hwdev1

=> r1 10000000 8

10000000: 53030100 00000000 ff00ff00 00000000
10000010: 00000000 00000000 00000000 00000000

=>

```

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 does require typing all 8 digits of the 32bit address. The command “rl 0” would read from BAR0 offset 0. This command sequence shows writing a 32bit word (0x44332211) into the Scratchpad register and then reading it back:

```

CMD - telnet hwdev1

=> w1 10000004 44332211

=> r1 10000000 8

10000000: 53030100 44332211 ff00ff00 00000000
10000010: 00000000 00000000 00000000 00000000

```

The following command sequence will toggle the 16 Segment LEDs on and off:

```

CMD - telnet hwdev1

BAR: set upper nibble of address to 0,1,2

=> ws 10000008 ffff

=> ws 10000008 0000

=> ws 10000008 ffff

```

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.

```

CMD - telnet hwdev1

=> r1 10001000 16

10001000: 03020100 07060504 0b0a0908 0f0e0d0c
10001010: 13121110 17161514 1b1a1918 1f1e1d1c
10001020: 23222120 27262524 2b2a2928 2f2e2d2c
10001030: 33323130 37363534 3b3a3938 3f3e3d3c

=> w1 10001000 0 1 2 3 4 5 6 7 8

=> r1 10001000 16

10001000: 00000000 00000001 00000002 00000003
10001010: 00000004 00000005 00000006 00000007
10001020: 00000008 27262524 2b2a2928 2f2e2d2c
10001030: 33323130 37363534 3b3a3938 3f3e3d3c

=>

```

3.2.3 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 verify every location has the correct value. The command mc clears all contents to 0's. The command mf <hex> fills all memory locations with the same byte value.

```

CMD - telnet hwdev1

=> m
=== Memory Test ===
Writing all 00's...Verifying...PASS
Writing all FF's...Verifying...PASS
Writing all AA's...Verifying...PASS
Writing all 55's...Verifying...PASS
Writing random pattern...Verifying...PASS
Writing increment pattern...Verifying...PASS
PASS

=> mc
EBR Memory cleared to 0x00

=> mf a5
EBR Memory filled with 0xa5

=> rb 10001000 16

10001000: a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5

=> rb 10004fff 16

10004fff: a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5

=>

```

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. Then the letters LATTICE are then displayed, ending with displaying an asterisk (*).

```

CMD - telnet hwdev1
=> l
=== 16 Segment LED Test ===
Segment Test:..0..1..2..3..4..5..6..7..8..9..a..b..c..d..e..f..e..d..c..b..a..
.9..8..7..6..5..4..3..2..1..0
LATTICE
OK
=>

```

3.2.4 Other menu functions

```

CMD - telnet hwdev1
===== PCI Express Menu =====
c - PCI Config Registers
e - PCIe Extended Capabilities Registers
s - PCIe stats
l - LED test
d - DIP switch setting
m[cfl] - EBR memory test
i <file> - Input file to EBR memory
o <file> - Output EBR memory to file
r[bsl] [BAR+addr] [count]
w[bsl] [BAR+addr] <data_1>...<data_n>
v - version and driver info
t[1-5] [num] - run a debug test num times
q,x - exit

BAR: set upper nibble of address to 0,1,2
=>

```

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-E 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. It is not read back. This test could 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 could 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 it reaches the end.
- t5 [n] – the same as t4 except the data size is in bytes, to verify byte lanes are operating properly.

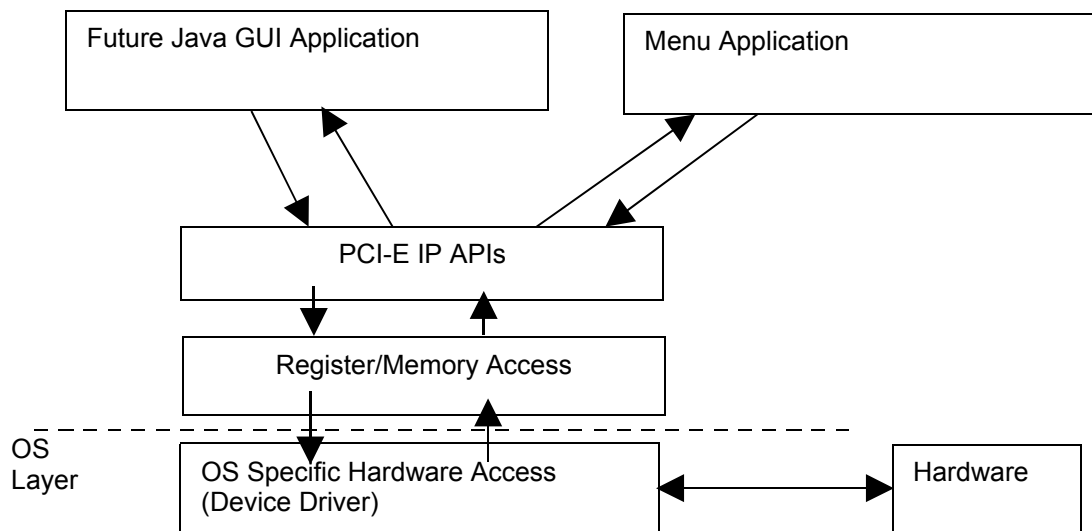
4 Demo Design Details

4.1 Demo Hardware Design

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

4.2 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 following hierarchical layers.



4.2.1 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 Eval 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-E Eval Board. The driver is a Linux 2.4.x kernel module. It is loaded with the insmod command. The driver's main purpose is to provide the mapping between the device node in user space (found in /dev/lscpcie/) and the hardware device (eval board). Once the driver is loaded, the driver searches for known Lattice devices and records the device's resources (how many BARs, sizes, interrupts, etc.) and maps the BARs into user memory space the driver code can access. The driver's main purpose is to provide 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 boards 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.

4.2.2 Register/Memory Access

This portion of the software provides an operating system 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 would be similar to Windows' HAL.

4.2.3 PCI-E IP APIs

These API functions provide simplified access (helper functions) to the standard PCI configuration registers and the Lattice PCI-E 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

4.2.4 Application Software

The application software is the demo or test code that is used to demonstrate PCI-E 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-E driver to:

1. Perform memory read/writes to access the LEDs and switches to show real-time control
2. Stress test the PCI-E interface via high-throughput accesses to the EBR Memory and memory tests to verify 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.

4.3 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-E board. The driver is purposefully designed to be rather generic. The details of the hardware design are deferred to the user space software. The driver merely provides the most basic services to the user space client: read and write memory. This is probably not the optimal design for a production board. For a commercial product, most of the device functionality would be encapsulated in the driver, and the user space would interact at a higher level – send a packet, receive a packet, clear the screen, etc., and the driver would do all this internally.

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

4.3.1 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 new IP designs and/or demo changes do not require the driver to be modified.

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

The Units of access are BARs. Each PCI-E device can have up to 6 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 setup in the applications memory space to directly translate the address into the bus address the BAR resides at.

/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 PCIE demo creates a new directory named `/dev/lscpcie/` and populates it with a number of files that name potential LatticeSCM or LatticeECP2M boards that could be installed in the system. Four boards of each type are created and each board can have up to 6 BARS. The file names are of the format:

SC[n]_[bar]

EC[n]_[bar]

SC = LatticeSCM PCI-E eval board

EC = LatticeECP2M PCI-E eval board

n = board number 1-4

bar = PCI BAR number 0-5

Boards are identified by their type (EC or SC) and enumeration order. SC1 = the first SC board detected in the system. EC2 = the second EC board. If a board does not exist (ie. there is no EC board installed or there are not 2 EC boards, only 1) 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-E bus probing on a motherboard. The numbering of buses can be logical, and not physically tied to a slot. Some experimentation may be needed to determine which of 2 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 physically is SC1.

The EC or SC 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 none) board is installed. This provides for future growth (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 Major device number is dynamically allocated when the **lscpcie.ko** module is installed. This avoids potential collisions with other existing 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/` (there Major number 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 Minor device number is used to indicate to the driver the type of device (EC or SC), the instance of the device (1-4) and the BAR (0-5). Each device entry has the same Major number (they 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 = SC, first one, BAR0
- 122 = ECP2M, second one installed, BAR2
- 019 = global PCI resources of SC board 1 (config regs., IRQ, etc.)
- 000 = not valid
- 210 = not valid – no device type = 2
- 100 = not valid – EC device type, but 0'th one installed is wrong

The eval 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`.

4.3.2 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.

4.3.3 Driver `__init`

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

4.3.4 Driver `__exit`

4.3.5 Driver `open()`

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

4.3.6 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

4.3.7 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 more intended for file or stream type devices. Our implementation is a pure memory mapped register based device. We are twindling bits, not storing MP3 songs.

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-E bus.

4.3.8Driver mmap()

Implements the Unix standard `mmap()` system call for this device (the eval 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.

4.3.9Driver 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.

5Troubleshooting

This section outlines some debug procedures to follow when experiencing trouble installing or running the demo.

5.1Trouble Installing the Demo Software Package

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

5.2Trouble with the Board

Ensure the board is installed in a PCI-E 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-E bitstream loaded in the SPI flash and for ECP2M that the Mode DIP switches are set to program from SPI flash (does not apply to SC eval boards).

Ensure the 4 Status LEDs are on, indicating the board is seen as a PCI-E endpoint. If the 2 yellow LEDs and 2 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-E slot to see if that fixes the link-up problem. You can also try pressing the Eval 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

5.3Trouble with the Driver

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

5.4Trouble Running the Demo

The eval 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.

5.5Debug Tools

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

5.5.1/proc/pci

5.5.2/proc/modules

5.5.3/proc/bus/pci

5.5.4dmesg

6Reference Information

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

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

7Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-408-826-6002 (Outside North America)

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com