How does a Customer develop a Linux based solution?

Scott Swaringen
FAE Software and Tools

freescale™
semiconductor

Objectives

- Understand the three approaches to Linux development
- Understand the challenges of each approach
- Know what solutions Freescale offers
- Know where and how to obtain the Freescale solutions
- Gain a general understanding of how to build a Linux solution using the Freescale solutions
- Know the Freescale partners, what they offer, and the relative costs of each

What is covered in this class?

- Linux, Open Source, and GPL
- Required software and tools
- Do it yourself - Getting started
- Freescale Linux Tools
- Freescale Linux Target Image Builder
- Boot Loaders
- Applications
- Busybox
- Device Drivers and Modules
- Kernel patches
- Deployment
- Migration
- Commercial Linux Distributions – Who, Why, Value Proposition
- uClinux
- Linux and Real Time

What is not covered in this class?

- Details of the Linux architecture
- Code review
- Linux system calls and shell commands
- Scripting
- Benchmarks
- Comparison to other RTOSes

Linux, Open Source & the GPL

freescale™
semiconductor

What is Linux?

- A Unix-like Operating System
- Open Source Software
- Started as a kernel and released to the public in 1991 for x86
- Kernel provides hardware access, interrupt handling, process management, memory management, task scheduling, inter-process communications, and system calls (application programming interface)
- Linux now includes utilities and libraries from the GNU project (sometime referred to as GNU/Linux)
- Widely distributed (Freescale PowerPC, ColdFire, and i.MX31 and our competition)

freescale™ semiconductor

What is Open Source?

Broad interpretations exist, but let's narrow the definition to this:

Software that is released with source code under a license that ensures that derivative works will also be available to the general public.

For the Linux kernel, device drivers, and many applications, this means that the source is available for use without financial requirements (hence – "free") but all modifications must be made available to the Open Source community.

NOTE: Not all Linux drivers and applications are Open Source!

The GNU General Public License (GPL)

- For the full text of the GPL Version 2, reference www.gnu.org/licenses/gpl-2.0.txt
- For the full text of the GPL Version 3, reference www.gnu.org/licenses/gpl-3.0.txt
- For history, google Richard Stallman

- Scott's interpretation – Definitely not *legal counsel*!

*) You must include the GPL license text in your distributions and display that the software is licensed under GPL if possible.
*) Any work derived from GPL licensed software must be licensed under GPL and made available as open source.
*) GPL software may still be copyrighted.
*) You may use the software freely. You may also charge for your distribution.
*) GPL software is not warranteed.

GNU/Linux is distributed under GPL Version 2. There is an effort to apply Version 3. Linus Torvalds has not supported this transition to date.

Required Software and Tools

freescale™
semiconductor

Linux Host

- A Linux developer must have a Linux host*
- Recommended Linux hosts (proven to support Freescale tools):
  - Fedora Core 3, 4, or 5
  - SuSe 10
  - Red Hat 9*
- Possible alternatives (not proven to support Freescale tools, but some success):
  - Ubuntu
  - Gentoo
  - Mandrake
  - Debian
  - Newer versions of Fedora Core or SuSe

* We will talk about Linux application only development later

Linux Host PC Options

- Dedicated Linux PC
- VMWare running on Windows 2000 or XP and hosting Linux as a "Guest" Operating System.
  - allows the user to host a guest Operating System (Linux) on Windows 2000 and XP.
  - eliminates the need for a dedicated Linux host PC.
  - proven to host Fedora Core 3, 4, 5, SuSe 10, 10.2, and Red Hat 9.
- Dual Boot PC
  - Not recommended except for the brave and those deeply knowledgeable of hard disk partitioning in both Windows and Linux

Linux Host PC System Requirements

- Serial port or USB to Serial converter known to work with Linux
- Null modem cable or converter
- Ethernet port with speed matching the target hardware
- Terminal emulation software (Windows – HyperTerminal or TeraTerm, Linux – Minicom)
- Specific memory and hard disk space is not provided here. Requirements vary according to the choice of the native OS. If a PC can run Windows, it can run Linux. Linux can run on older and smaller machines that will not host Windows.

How to Create a Linux Solution? The “Simple” Answer

freescale™
semiconductor

The “Simple” Answer

- Acquire a bootloader
- Acquire a compatible toolchain
- Build, debug, and flash the bootloader
- Acquire a Linux kernel, device drivers, and packages
- Acquire a compatible toolchain
- Configure the kernel and packages
- Configure the root file system
- Build the target image
- Deploy the target image
- Debug
- Deploy the persistent production image

Do It Yourself - Getting Started

freescale™
semiconductor

Recommended Reading

- Building Embedded Linux Systems by Karim Yaghmour, Copyright 2003 O'Reilly Media, Inc. – This text describes all of the steps of building and deploying a Linux solution without using any configuration or building tools beyond Open Source offerings
- Linux Device Drivers by Alessandro Rubini & Jonathan Corbet, Copyright 2001, 1998 O'Reilly and Associates, Inc. – This is THE Linux device driver reference
- Linux Pocket Guide by Daniel J. Barrett, Copyright 2004 by O'Reilly Media, Inc. – This book presents the Linux system commands and some utilities by function rather than by name. It is a real time saver when you know what you need to do, but don't know the command or utility name
- Beginning Linux Programming by Neil Matthew and Richard Stones, Copyright 1996 & 1999 Wrox Press – This text is a great starter. Many more texts are available.

Acquire a Linux kernel

- Go to www.kernel.org and follow the links to the /pub/linux/kernel index.
- Choose your kernel version (hopefully 2.6)
- Locate and download the specific version and desired compression format “tarball”
- Decompress the tarball
- Build the kernel … But, with what tools and for what processor and how?
- … And what about the boot, daemons, libs, and utils directories? Are those needed?

Perhaps this isn’t as easy as it seems ☺

Acquire a tool chain

- GNU Compiler Collection aka “gcc” includes C, C++, Objective-C, Java, Fortran, and Ada as well as libraries for each language.
- Some versions are available on www.kernel.org. Others are provided by vendors (e.g. ARM)
- Latest versions are available on gcc.gnu.org (and other locations)
- Multiple versions exist for each Freescale processor. Which one should be used?

The choice of gcc version depends on the Linux kernel version and the applications and device drivers to be integrated. Sometimes, the kernel will build but packages will not or vice versa.

Acquire device drivers and applications

- Device drivers are usually acquired from the device vendor. For example, Realtek Wi-Fi card drivers can be downloaded from rtl8180-sa2400.sourceforge.net
- Integrating device drivers or applications may introduce dependencies into the Linux configuration. How is the customer to know?
- Reminder: Some drivers and applications will only compile with specific versions of the gcc tools.

How is the Linux configuration changed to include the device drivers and applications?

Configure the Linux target image

- "make config" is a command line console interface
- "make menuconfig" is a menu driven console interface
- "make xconfig" is a X windows configuration application

For each of these, the customer must know what components are needed in the kernel, what the component names are, and all dependencies for each component.

Examples:

Web Server – httpd or boa

Flash file system – jffs2 and possibly MTD drivers

This doesn't address "make mrclean", "makedep", "makeimg". What are they and when are they used?

Build the file system

- Determine whether the file system will be NFS, cramfs, ramdisk, jffs2, etc.
- Build the file system structure with required directories and files
- “makefs” – How is this used, what are the command line parameters, what file system types can be built?

Recommendation

- Realize that Linux is “free” only if your time has *NO* value!
- “Roll your own” Linux is for the expert or bit banger only.
- Rather than download kernel versions, gcc versions, mixing and matching, integrating device drivers, integrating applications, figuring out how to build the desired file system, ad naseum ….
- Use a Board Support Package or commercial Linux solution that aggregates a working kernel, tool chain(s), device drivers, applications, can create file systems, simplifies the configuration process, and assists deployment.

Freescale Linux Tools

freescale™
semiconductor

What is available?

- Code Warrior
  - Linux Platform Edition
  - Windows Linux Application Edition
- Linux Target Image Builder (aka LTIB)
- Board Support Packages (BSPs)
- Debuggers

Code Warrior

- Linux Platform Edition
  Platform Edition is hosted on Linux. This is required for Linux kernel space development. The kernel space includes the kernel, device drivers, and modules. Platform Edition also supports development of user space applications.

  Platform Edition supports debugging both in kernel and user space. Linux configuration settings are required to enable debugging. “Kernel hacking” is required for kernel space debug. MetroTRK must be included in the target image for user space debug.

- Windows Linux Application Edition – Linux Application Edition allows development of Linux user space applications while running on Windows XP.

LTIB – Linux Target Image Builder

- LTIB is included with all Power, ColdFire, and i.MX BSPS.
- LTIB runs on the Linux host. Some BSPs support LTIB running on the target (Perl support is required).
- LTIB provides Linux kernel, device driver, application, and file system configuration and building.
- LTIB handles all of the path modifications to run the compatible GNU toolchain required for compilation and linking.
- LTIB is Open Source and is licensed under GPL.
- LTIB displaces the Linux Platform Creation Suite (PCS).

Freescale BSPS – Where to get them

- General availability
  - www.freescale.com and click on the Products -> CodeWarrior Development Tools link
  - From the left hand menu, navigate through Downloads -> Linux Board Support Packages to the desired processor architecture
    - BSPs for Power Architecture Technology
    - BSPs for ARM Architectures
    - BSPs for ColdFire Architectures
    - BSPs for Other Architectures
- Alpha / Beta releases
  - Contact your FAE or Sales
  - BSPs can be transferred through the Freescale Compass Extranet or Transweb (user account registration required)

Freescale Board Support Packages – Power Architecture

- PQ2FADS
- HPC-8641DE (HD Ghost Image)
- Media5200 EVB
- Lite5200B
- MPC8313
- MPC832xE-MDS-PB
- MPC8323E-RDB
- MPC8349EA-MDS-PB
- MPC8349E-mITX-GP
- MPC8349E-mITXE
- MPC8360EA-MDS-PB
- MPC8540 EVAL 2.6 Kernel
- MPC8548CDS
- MPC8555/8541 CDS
- MPC8560ADS 2.6 Kernel
- MPC8568
- MPC8641DHPCN
- HPC-8641DE (HD Recovery Image)
- Total5200
- MPC7448HPC2
- MPC8272ADS
- MPC885ADS

freescale™ semiconductor

Freescale Board Support Packages – ColdFire

- MCF5253EVB
- MCF5208 EVB
- MCF5329 EVB
- MCF5474/84 LITE & MCF5475/85 EVB

freescale™ semiconductor

Freescale Board Support Packages – i.MX

- i.MX1/L ADS
- i.MX21ADS
- i.MX27ADS
- i.MX31ADS

Debuggers

- Power Architecture
  - Ethernet PowerTAP
  - USBTAP
  - Abatron BDI-2000
- ColdFire Architecture
  - USBTAP
  - P&E Micro
- i.MX
  - Freescale does not provide any hardware debugger
  - Partner or 3rd party solutions include RealView (ARM) and possibly Abatron BDI-2000

Freescale Bootloaders

freescale™
semiconductor

What does a bootloader do?

- Provides the first executable code when the board comes out of reset
- Begins execution in Flash
- Initializes the processor and memory
- Transfers itself to memory and continues execution
- Initializes additional devices (e.g. Ethernet)
- Provides a command interpreter for configuration
- Loads Linux into memory, passes parameters, and transfers execution to the kernel

- Additional features:
  - Supports transfer of images from host to target
  - Programs flash
  - Executes additional programs or scripts

What bootloaders does Freescale provide?

- Power Architecture
  - u-boot – u-boot is the “Universal bootloader”. Source code for u-boot is provided with all Freescale Power Architecture BSPs. Source can also be obtained from http://sourceforge.net/projects/u-boot. The best location for u-boot information is www.denx.de
- ColdFire
  - dBUG – dBUG is a debugger / monitor. It can be used to download and flash colilo on the MMU-ful ColdFire systems. It is the default bootloader for the MMU-less ColdFire systems.
  - colilo – colilo is the “COldFire LInux LOader. Source code is provided with the 54xx BSPs.
- i.MX
  - blob – blob is the bootloader used with the i.MX21 ADS. Source code is provided in some of the BSPs. Source and additional information can be obtained at http://sourceforge.net/projects/blob/
  - Redboot – Reboot is the bootloader used with the i.MX27 and i.MX31 ADSes. Source code is provided in the BSPs. Source and additional information can be obtained at www.ecoscentric.com.

Working with LTIB

freescale™
semiconductor

Installation and Setup

- LTIB is included with BSPs and requires a 2nd installation step after installing the BSP. Two options for getting to the BSP install:
  1) Burn the .iso image to a CD-ROM. Insert the CD and automount (or mount) the image and change directory to the mount point
  2) Copy the .iso image to the Linux host hard drive and mount it to the filesystem using “mount <BSP.iso> <../mount_directory> -o loop” and then change to the ../mount_directory

- Once the mount point is known and it is the current directory, run “./install” as user. This installs the BSP. The user specifies the LTIB installation directory as part of the BSP installation step.

- At completion of the BSP install, there is text instructing the user to install LTIB by running “./ltib” from the LTIB installation directory.

General Guidance

- The best guide for using LTIB is available on www.bitshrine.org. The LtibFaq link from the main page provides an excellent step by step guide to using LTIB.

- When a BSP .iso is mounted, the user can open a browser and enter the ../mount_directory as the URL. This will open a main page for accessing BSP, LTIB, and processor information. Look for the START_HERE.html link.

- Within START_HERE.html are a BSP User's Manual and a BSP Device or Reference Guide. The User's Manual tells how to build and deploy target images.

Toolchain installation

- The toolchain installation takes place during the BSP and LTIB installation.
- The toolchains of a BSP are not installed in the ../ltib directory
- Toolchains, libraries, and other LTIB content are installed in /opt/mtwk and /opt/freescale

Building a Target Image with LTIB

freescale™
semiconductor

Fundamentals

- Start LTIB from the installation directory by entering “./ltib –c”. The “-c” parameter tells LTIB to configure.
- Accept the default configuration by selecting “Exit” at the bottom of the main LTIB menu.
- Enter “Yes” to “Do you want to save your new configuration”.
- LTIB builds the kernel and all configured packages, builds the configured file system, and places the executables into the ../ltib_install/rootfs directory.

- RESULTS:
  - A complete Linux target image
  - Considerable time savings (no need to gather a kernel, device drivers, packages, compatible toolchains, modify the host PATH, build the file system, etc.)
  - Confidence!

Customizing the Configuration

- If you want to modify the default configuration, start LTIB from the installation directory by entering “./ltib –c”.

- Choose the target C library type
  - The main choices are glibc or uclibc. glibc is the GNU standard C library. uclibc is a smaller footprint C library from uClinux that is useable in MMU-ful Linux.
- Choose the toolchain
  - Having more than one toolchain or using a custom toolchain is an advanced topic
- Choose a kernel
  - Having more than one kernel is an advanced topic

And we will explore the remaining configuration options in more detail

- Configure the kernel
- Choose and configure packages
- Set the target system configuration
- Configure the target image

Building a Linux Kernel with LTIB

freescale™
semiconductor

Configuring the Kernel

- Start LTIB from the installation directory by entering "./ltib –c". The "-c" parameter tells LTIB to configure.
- Key down to the "Configure the kernel" line and press the space bar. An "*" will appear between the brace characters to indicate you have made this selection.
- "Exit" the main menu and enter "Yes" to "Do you want to save your new configuration"
- After some package processing, LTIB will present a new menu for Linux Kernel Configuration.
- From this menu, the user has many options. Common configuration changes are done for networking, device drivers, and Kernel hacking (for debugging)
- Each option requires more advanced kernel knowledge to determine the need and configuration

Building the Kernel as part of a new Target Image

- Once all configuration settings are complete, “Exit” the Linux Kernel Configuration menu.
- Enter “Yes” to “Do you want to save your new configuration?”
- LTIB will build the new Kernel and selected device drivers as part of a full target image build.
- LTIB will put the resulting target image into the ../ltib_install/rootfs/boot directory

- It is possible to modify and build a new kernel and device drivers by using LTIB’s scbuild option.

Modules vs. Statically Linked Device Drivers

freescale™
semiconductor

Statically Linked Device Drivers

- A device driver that is configured with a “*” will be statically linked with the kernel during the build step.
- Statically linked device drivers are started according the system configuration during Linux startup.
- Statically linked device drivers as subject to the terms and condition of the GPL. Remember, GPL requires that all derivate work is made public to the open source community

Modules

- A device driver that is configured with an “M” will be built as a module and not statically linked with the kernel during the build step.
- Modules are introduced into the system by either a command or through system configuration.
- “Insmod” and “modprobe” are commands for starting a module.
- By convention, modules are not subject to the terms and condition of the GPL unless they are a derivative work of some GPL code. Therefore, most all Linux modules are retained as intellectual property and the source code is not published

Building Applications with LTIB

freescale™
semiconductor

LTIB Packages – Application Configuration

- Start with ./ltib –c to configure the target image
- Scroll down to highlight “Package list --->”
- Highlight Select and press Enter. This will display a menu of all of the packages (user space applications) that the BSP contains.

- Press the space bar to toggle between selecting and deselecting the package.
- Selected packages will be built and placed into the target file system. Most packages will be placed in the /usr directory branch

- Some packages have additional configuration options. The prime example of this is busybox

LTIB Packages – Busybox

- Busybox is a single binary that can contain over 200 utilities.
- Each utility has a single executable counterpart but typically the single executables are larger and take more storage and memory.
- www.busybox.net shows the full set of utilities that can be configured into busybox.

- Freescale BSPs provide busybox and a configuration menu to select either the busybox utility or force the file system to contain the single executable.

- Busybox uses symbolic links to direct the Linux utility name to the Busybox equivalent.

Building File Systems with LTIB

freescale™
semiconductor

Configuring File System Types

- LTIB supports multiple file system types. Not all file system types are supported in every BSP.
- To access file system configuration, scroll down to “Target Image Generation ---> Options”.
- Highlight Select and press Enter

- Each file system type, except for NFS, has additional settings that must be configured.

File System Types

- NFS – Network File System – A Network File System exists on a Linux host and is accessed through the NFS Server running over Ethernet.
- ext2 ramdisk – A compressed ext2 filesystem that resides in memory. A ramdisk can be written but changes are lost when the system is shutdown. A ramdisk eliminates the network latency in NFS
- cramfs – Compressed ROM filesystem – cramfs is read-only. Unlike other compressed file systems, cramfs is built so that the image can be used as is without decompression.
- jffs2 – Journaling Flash File System 2 – jffs2 is a compressed log-structured file system for use in Flash. jffs2 supports both NOR and NAND flash, but not all Freescale BSPs may yet support NAND. Some bootloaders (e.g. u-boot) support jffs2.

File System Build Output

- NFS – Network File System – Since the file system resides on the host PC, the full root file system directory structure resides in ../ltib_install/rootfs.
- ext2 ramdisk – LTIB creates rootfs.ext2.gz in the ../ltib_install directory. This compressed image must be deployed to the target separately from the Linux target image.
- cramfs – Compressed ROM filesystem – LTIB creates cramfs.little (little indicates “Little Endian”) in the ../ltib_install directory. This compressed image must be deployed to the target separately from the Linux target image.
- jffs2 – Journaling Flash File System 2 – LTIB creates rootfs.jffs2 in the ../ltib_install directory. This compressed image must be deployed to the target separately from the Linux target image.
- For each file system type, the boot arguments passed to the Linux kernel by the bootloader must specify the file system type

Linux Target Image Deployment

freescale™
semiconductor

Host Services

- Use the documentation available. Remember that the .iso image has a START_HERE.html main page that links to the BSP User's Guide. Each User's Guide has a section on how to deploy a Linux target image and file systems as necessary.
- Linux deployment always requires TFTP.
- Linux deployment requires NFS if and only if the file system is NFS.

TFTP

- Make sure that the TFTP service is installed on the host Linux system. Verification varies depending on the Linux distribution. For Fedora Core 5, the System Administration Services GUI applet shows the Background and On Demand Services. TFTP is listed in the On Demand Services. If TFTP is not installed, it can be added from the installation disks using the RPM command (or through a GUI “Add Software” utility.

- TFTP – Trivial File Transfer Protocol provides a simple file transfer protocol used by the bootloader to transfer an image file into the target system memory.

NFS

- If an NFS file system is configured, make sure that the NFS service is installed on the host Linux system. Verification varies depending on the Linux distribution. For Fedora Core 5, the System Administration Services GUI applet shows the Background and On Demand Services. NFS is listed inn the Background Services. If NFS is not installed, it can be added from the installation disks using the RPM command (or through a GUI “Add Software” utility.

Configure the Bootloader

- Each Freescale Bootloader allows interruption of the Linux booting process. When interrupted, the bootloader display a command prompt and supports command input. Use “help” if you are not familiar with the bootloader commands.
- Configure the target MAC address
- Configure the target and host IP addresses, gateway address, and IP broadcast addresses.
- Either configure a script to automate target image deployment or perform the steps manually

Host Setup to Connect to a Target

- Either turn off the Linux host firewall or configure the firewall to allow TFTP and NFS communications.
- Create a /tftpboot directory.
- Symbolically link the ../ltib_install/rootfs directory to an exportable directory
- Copy the kernel (and bootloader and file system images as needed) to the /tftpboot directory
- Ensure that the /etc/exports file exports the symbolic link for the ../ltib_install/rootfs directory
- Enable tftp by editing the /etc/xinetd.d/tftp file as required
- Restart the NFS and TFTP servers
- Connect the target to the host via a serial connection (usually a null modem cable)
- Start a terminal emulator (minicom on Linux)
- Power on the target and see console messages.

Transferring a Target Image

- Use the bootloader “load” command to transfer an image from the Linux host /tftpboot directory to memory.
- Example using Redboot: “load –r –b 0x100000 /tftpboot/zImage”

- If the host system is configured properly, the user should see Ethernet link activity and scrolling hash or other characters to indicate transfer progress. If the transfer fails, recheck the services, exports, and network addresses.

- Set the kernel command line.
- Example using Redboot for an NFS deployment: “exec –b 0x100000 –l 0x200000 –c “noinitrd console=ttymxc0,115200 root=/dev/nfs nfsroot=192.168.0.90:/tftpboot/ltib/ init=/linuxrc ip=192.168.0.200:192.168.0.90”

- The network addresses must match the host and target IP addresses.
- The /tftpboot/ltib/ reference must match the symbolic link created in the Host Setup.

Integrating Modifications to Existing LTIB Device Drivers and Applications

freescale™
semiconductor

Modifying Existing Packages

- A user may modify an existing kernel, device driver, or application package.
- The package must be extracted using the "./ltib –m prep –p <package .spec file name>" command.
- The source code may be modified as needed in the ../ltib_install/rpm/BUILD/package_directory
- The changes can be tested by using the "./ltib –m scbuild –p <package .spec file name>" and "./ltib –m scdeploy –p <package .spec file name>" commands and then deploying the target image and file system.
- When the changes are fully debugged (is there such a thing? ☺), the changes should be captured as a patch and introduced into the LTIB structure.

LTIB Workflow

- Unpack the sources and apply all current patches:
  - ./ltib -m prep -p <package>
- Edit/add files under **rpm/BUILD/package/**
- Build the package with your changes:
  - ./ltib -m scbuild -p <package>
- Once the package builds successfully, check the install phase:
  - ./ltib -m scinstall -p <package>
- Test your package before committing the changes:
  - ./ltib -m scdeploy -p <package>
- Repeat the above steps until satisfied with the results.
- Generate a patch and update the spec file:
  - ./ltib -m patchmerge -p <package>
- Manually clean up the patch file (as required).
- Build from scratch and install
  - ./ltib -p <package>

Patches

freescale™
semiconductor

What is a Patch?

- A patch is a collection of changes to a source code modules.
- A patch contains the output of the Linux “diff” program when run against the original and the modified source.
- Multiple patches may exist for a single package

Why use Patch?

- Using patches ensures that the user can always retain the original source.
- The importance of this becomes obvious when upgrading Linux or package versions.

LTIB Patches

- All patches known to LTIB reside in the /opt/freescale/pkgs directory.
- The .spec file for a package indicates what patches are to be applied when the package or driver is built.
- .spec files are in the ../ltib_install/dist/lfs-5.1 directory
- To introduce a patch, the user must place the patch into the ../pkgs directory and modify the package .spec file to indicate the new patch is needed to build.

Integrating New Device Drivers and Applications to LTIB

freescale™
semiconductor

Workflow 1/2

- First clean your sources (remove any .o, .a, .so generated files) and then make a 'tarball', for instance:
  - cd <my_new_package>-x.y
  - make clean
  - cd ..
  - tar zcvf <my_new_package>-x.y.tar.gz <my_new_package>-x.y
- Move this tarball to the Local Package Pool so **ltib** can find it.
  - mv <my_new_package>-x.y.tar.gz /opt/freescale/pkgs/
- Create a specfile using the existing template.
  - mkdir dist/lfs-5.1/<my_new_package>
  - cp dist/lfs-5.1/template/template.spec dist/lfs-5.1/<my_new_package>/<my_new_package>.spec
- Edit and fixup the template to reflect your package. The fields that need changing are:

Field	**Description**
Summary	put in a summary of what the package is/does
Name	put in the name of the packge (usually from the tarball name)
Version	put in the version (usually from the tarball/directory
Release	start at 1 and rev each time you change the spec file
License	e.g GPL/LGPL/BSD, look this up in the package's files
Group	If this exists on an rpm based machine, copy from rpm -qi If not, choose something from /usr/share/doc/rpm-/GROUPS
%Build	often you'll need to add **--host=$CFGHOST --build=%{_build}** to the configure clause

Workflow 1/2

- Unpack the new package sources:
  - ./ltib -m prep -p <my_new_package>
- Make any changes you need to the sources to get them to cross compile
- Build the new package with your changes:
  - ./ltib -m scbuild -p <my_new_package>
- Once the new package builds okay, check the install phase:
  - ./ltib -m scinstall -p <my_new_package>
- Install the test package in the NFS root filesystem area (rootfs) and test
  - ./ltib -m scdeploy -p <my_new_package>
- When the package is running correctly, capture the changes.
  - ./ltib -m patchmerge <my_new_package>
- Any changes you've made will be put into a patch file and copied to /opt/freescale/pkgs. In addition, the spec file will be updated to reference the new patch. You should check the patch and eliminate any bogus diffs.

Migrating from a BSP to Custom Hardware

freescale™
semiconductor

An Oversimplification

- Modify the bootloader to match the new hardware configuration.
- Modify the existing Linux kernel device drivers as needed for the on-chip and peripheral devices.
- Add custom device drivers and application packages.
- Modify LTIB to maintain the tool for repeating updates.
- Update the flash with the customized bootloader and Linux target image.

- ….. *Go to market!*

The Truth Hurts – Starting with the Bootloader

- If the custom hardware memory, Ethernet, serial, and other devices that must be initialized at boot time differ from the Freescale reference design, the bootloader must be modified.
- There is little documentation for the source structure of the bootloaders and what to modify. What does exist can be difficult to find and understand.
- The only true guide is the source code itself so the developer is left to browse through the source code to understand where and what to change.
- Bootloaders are common to multiple processor types and reference designs, so there is more to see than is needed for the particular design.
- Porting a bootloader can take considerable time.
- A great debugger is essential

The Truth Hurts – Continuing with Device Drivers

- If the user continues with LTIB, there is a structure for introducing new device drivers.
- Because device drivers are kernel space executables, debugging requires kernel debugging capability (hardware debugger).
- Introducing new device drivers to a stable system may create resource conflicts (interrupts, memory access, port I/O, etc.) which have to be resolved.
- Because device driver code runs in the kernel space, poorly written code can crash the system making debugging more challenging.

- Continuing with LTIB can add tools support to the development burden.

Recommendation

- Carefully evaluate the Linux intelligence within the development team
- Weigh the cost in time (and MONEY!) to do customization of a BSP.
- Consider purchasing a commercial Linux distribution to reduce the challenge and save time (and MONEY!)
- Consider hiring professional services to save time (and MONEY!)

- When evaluating commercial offerings, understand the following:
  - Are there any limits to the number of processors and projects that can be developed using the distribution?
  - What is the lifetime of the distribution?
  - What is the upgrade plan?
  - What support is provided?
  - What tools are provided?
  - What additional purchases (IDE, debugger, support service level) are available and recommended?

Commercial Linux Distribution Partners

freescale™
semiconductor

Commercial Linux Solutions Providers

WIND RIVER

montavista

TimeSys

LYNUXWORKS™

Open. Reliable. Safe. Secure.

freescale™ semiconductor

Wind River Systems

- Long time provider of VxWorks
- Offers Linux Platform Edition for kernel space development
- Offers Application Edition for user space development.
- Has a custom ICE debugger.
- Offers Linux Workbench Eclipse based IDE.
- Offers additional development and debugging tools.
- Offers General Purpose Platform, Platform for Consumer Devices, and Platform for Network Equipment.

- Owners of RTLinux – A real time kernel for Linux Hard Real Time solutions

*Contact WRS for quotes and details

MontaVista

- Only offers Linux solutions
- Significant contributors to kernel.org especially in the area of real time performance improvements to the Linux kernel.
- Offers Platform and Application Developer Kits
- Offers DevRocket Eclipse based IDE
- Offers Professional Edition, Mobilinux, and Carrier Grade distributions

- Platform Developer Kit is sold for one year subscription*.

- Subscription includes support.
- Offers Professional Services for Linux custom development.

*Contact MontaVista for quotes and details

LynuxWorks

- Long time provider of LynxOS – a real time kernel providing POSIX compliance and Linux binary compatibility
- Offers Blue Cat Linux
- Offers a POS focused Linux solution
- Offers Luminosity – an Eclipse based IDE
- Offers additional development and debugging tools

- Pricing model has been a one time purchase. No approximate pricing information is available.

- Sells additional support levels.
- Offers Professional Services for Linux custom development.

freescale semiconductor

TimeSys

- Offers subscriptions to LinuxLink
- Subscription provides access to the LinuxLink repository containing integrated GNU tools, Linux kernels, and packages.
- Subscription provides access to forums, documentation, and on-line support.

- Subscriptions can be purchased for as little as 1 month and scales up to 1 year.*

- Subscription scaled to Pro Developer Seats and Lite Developer Seats.

- Offers TimeStorm Eclipse based development tools
- Offers Professional Services for Linux custom development.

* Contact TimeSys for quotes and details

uClinux

freescale™
semiconductor

uClinux

- uClinux is a Linux derivative for processors that do not have a Memory Management Unit
- uClinux has been mainlined with the Linux kernel since 2.6.
- Still, the best location to get uClinux source and board specific solutions is www.uclinux.org.
- Freescale ColdFire 52xx and 53xx processors and reference designs run uClinux.
- There are some programming and implementation differences between Linux and uClinux (fork, brk, malloc) but porting code is possible. Most device drivers from Linux work well in uClinux
- Arcturus Networks is a good contact. SnapGear were great supporters but they have been acquired. Several of their staff continue to support the uClinux community.

Linux and Real Time

freescale™
semiconductor

Linux and Real Time

- Linux is not suitable for true hard real time systems where responses must occur within tight time boundaries or the system degradation leads to significant loss, injury, or fatalities.
- MontaVista has provided considerable improvements to the Linux kernel real time performance. Their contributions are now mainlined with the kernel source and are available as patches.
- Two real time kernels running Linux as a task exist: RTLinux and RTAI.
- The creator of RTLinux obtained a patent that effectively shut down the use of RTAI in the United States and weakened its presence in other world regions.
- FSMLabs, the corporate owner of RTLinux, was acquired by Wind River Systems in 2006.

Q&A

freescale™
semiconductor

freescale™
semiconductor