Building a Small Embedded Linux Kernel Example

Loc Truong and Brijesh Singh

ABSTRACT

This application note demonstrates NOR kernel building and board setup using the DaVinci DM644x Digital Evaluation Module (DVEVM) package. The goal is to build the smallest possible kernel using the MontaVista® Linux Support Package (LSP) with support for an HTTP server, a TCP/IP stack, and necessary drivers for Ethernet and UART for the serial debug terminal. The kernel resides in NOR flash and uses a RAM disk-based file system, which is also stored on flash.

This setup is found in embedded devices such as routers and print servers, and can be used as a starting point for more sophisticated implementations such as I/O monitors, web cams, and multimedia players.

The application note includes the following sections:

- Overview of the required hardware and software available
- Building the kernel
- Building the RAM disk file system
- Setting up the application
- Storing to flash

Contents

List of Figures

List of Tables

1 Overview

To create a standalone and bootable Embedded Linux System, three main pieces of software must be flashed on the EVM:

- A bootloader, u-Boot in this case
- A Linux kernel with built-in drivers for DaVinci DM644x devices
- An ARM-target Linux file system containing the shell, application and run-time support utilities and stacks

This section quickly reviews the DVEVM, including required hardware and software components.

1.1 DM644x Digital Evaluation Module Package

The DM644x EVM Kit is a collection of hardware and software packages for the embedded Linux developer community.

The hardware components include:

- TMS320DM6446 device-based development board
- NTSC/PAL video camera (region dependent)
- NTSC/PAL LCD display (region dependent)
- Microphone
- IR remote control
- 40GB, 2.5-inch IDE hard disk drive

The development board has multiple accessories and I/O interfaces such as USB, 10/100 Mbps Ethernet, video-in (composite), video-out (analog or digital), audio-in (line or microphone), audio-out (S.PDIF, analog), and UART. The board also includes 4 MB of SRAM memory, 16 MB of NOR memory, 64 MB of NAND memory, a 40 GB HDD, and 256 MB of DDR2 memory.

For a more detailed list of all the available features of the DVEVM, consult the Technical Reference. See Table 1 for the required hardware features for this project.

Table 1. Required DVEVM Hardware Features

Type	Device	Description
CPU	DM6446	Dual-core multimedia processor with video acceleration hardware
Non-Volatile Memory	NOR flash	16 MBytes available, 0x0200 0000 to 0x02FF FFFF
Volatile Memory	DDR2	256 MBytes available, 0x8000 0000 to 0x8FFF FFFF
I/O	LED	8 total, can be used for feature indication and/or user feedback
	Ethernet	10/100 Mbps
	UART0	Serial debug port, set at 115200 one stop bit, no parity, no flow control

1.2 Software Components

Various software components come with the DVEVM package, including multimedia demos such as audio, speech and video encode and decode using various codec formats. However, in this project only the ARM Linux tool chain, the bootloader, and Linux Support Package (LSP) are needed to complete the goal of building the smallest possible flash-based Linux kernel with an HTTP server for the DM644x DVEVM.

Table 2 shows a list of the components that are assumed to be available for use with this project. Although package versions are included in the list, later versions may be available.

Table 2. Required DVEVM Software Packages

Item	Version	Notes
ARM Linux Tool Chain	MVL Pro 4.0.0	Included with DVEVM SW packages
Linux Support Package	MVL-401c	Included with DVEVM version 1.10 release
Bootloader	u-boot-1.1.3	Included with DVEVM version 1.10 release
RAM Disk	MVL Pro 4.0.0	Included with DVEVM SW packages
HTTP Web Server	MVL Pro 4.0.0	Included with DVEVM SW packages

2 Feature Selection and Kernel Build Steps

Building an embedded Linux kernel can be complex if starting from bare silicon. Drivers must be ported or developed, tested, and compatible cross-development tool chain and upper protocol stacks updated or retargeted for the ARM926EJS processor on the DM644x device. The DVEVM package already includes most of the available tools, such as an ARM GNU tool suite, a Linux Support Package with the ARM Linux kernel v2.6, and all the drivers needed for our project.

This section assumes that you have installed the DVEVM software as described in Section 4 of the *DVEVM Getting Started Guide* (SPRUE66). Section 4 of the *DVEVM Getting Started Guide* also documents the general commands for building a Linux kernel.

Thus, building an embedded Linux kernel comprises two simple steps:

- Configure the kernel to select the needed drivers and features
- Compile the kernel to create an appropriate image, uImage, that u-boot can load on DVEVM

2.1 Kernel Configuration

Linux kernel features are collected in the *.config* file at the top level of the kernel directory. This file is used by the GNU make utility in the build process. Although you can edit the *.config* file directly to turn the features on or off, several menu driven methods are available to make this step easier. The oldest one is *make menuconfig*, although graphical methods such as *make xconfig*, which uses the X-windows environment, or *make gconfig*, which uses GTK+ environment, are preferred.

The following sections describe examples of performing the configuration using xconfig. If you are already familiar with the configuration step, use Table 3 to determine the features that must be selected or deselected from the default LSP of the DVEVM.

Table 3. Configuration Summary

Enable	Disable
ARM System Type (TI-Davinci)	Loadable module support
TI DM644x Based system	Built-in firmware loadable support
TI Davinci EVM	MTD support
TI Davinci I2C Expander	Loop back device support
ARM EABI Support	ATA/ATAPI support
High-Resolution Timer	SCSI support
Networking Support	Input device support
Initial RAM disk Support	Video for Linux support
Kernel .config File Support	Ext3/XFS/Minix/Automounter/MSDOS/VFAT/CRAMFS/NFS support
Configure Kernel for Small Devices	Frame buffer device support
POSIX Message Queues	USB Support
System V IPC	Sound card support

Table 3. Configuration Summary (continued)

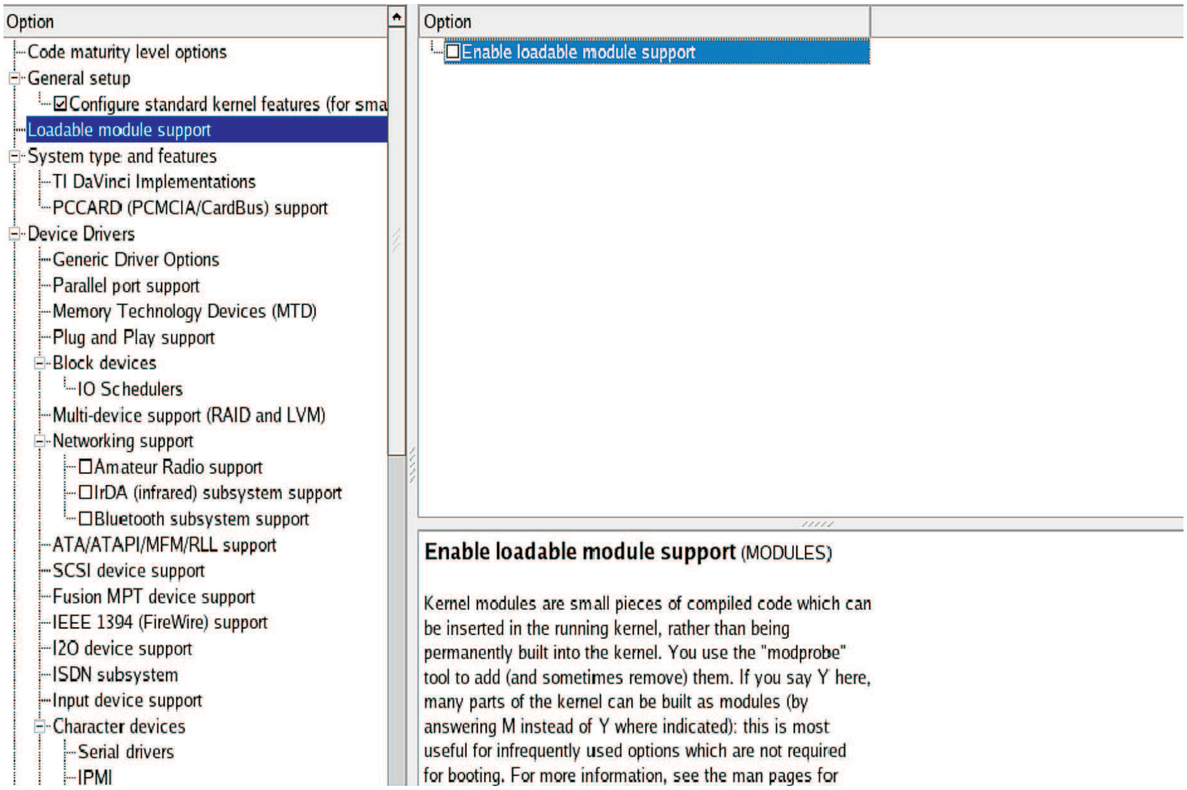
Enable	Disable
ELF Support	MMC Support
8250 Serial Driver Support	

2.1.1 Configuration Steps

The following steps assume that the default installed kernel tree has been copied to a private location at */home/user/workdir/lsp* before compiling. Also note that the directory names of the kernel tree can change from one version of the package to another.

Note: The DVEVM and DVSDK-L or -3L software packages may have slightly different kernel config and build commands. Always check the documentation that comes with the package such as the DVEVM Release Notes, the DVEVM Quick Start Guide or *DVEVM Getting Started Guide* (SPRUE66) for updated information regarding the exact commands for the build steps. The following steps are for the kernel tree from the DVEVM software package.

1. On the host Linux workstation, go to the base directory of the kernel tree:
   ```
   host $ cd /home/user/working/lsp/ti-davinci
   ```
2. Launch the Linux kernel configuration utility:
   ```
   host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-  xconfig
   ```
3. Under *Loadable module support*, uncheck the *Enable loadable module support* to disable the module loading feature. See Figure 1.


Option
Code maturity level options
General setup
☑Configure standard kernel features (for sma
Loadable module support
System type and features
TI DaVinci Implementations
PCCARD (PCMCIA/CardBus) support
Device Drivers
Generic Driver Options
Parallel port support
Memory Technology Devices (MTD)
Plug and Play support
Block devices
IO Schedulers
Multi-device support (RAID and LVM)
Networking support
☐Amateur Radio support
☐IrDA (infrared) subsystem support
☐Bluetooth subsystem support
ATA/ATAPI/MFM/RLL support
SCSI device support
Fusion MPT device support
IEEE 1394 (FireWire) support
I2O device support
ISDN subsystem
Input device support
Character devices
Serial drivers
IPMI
Option
☐Enable loadable module support

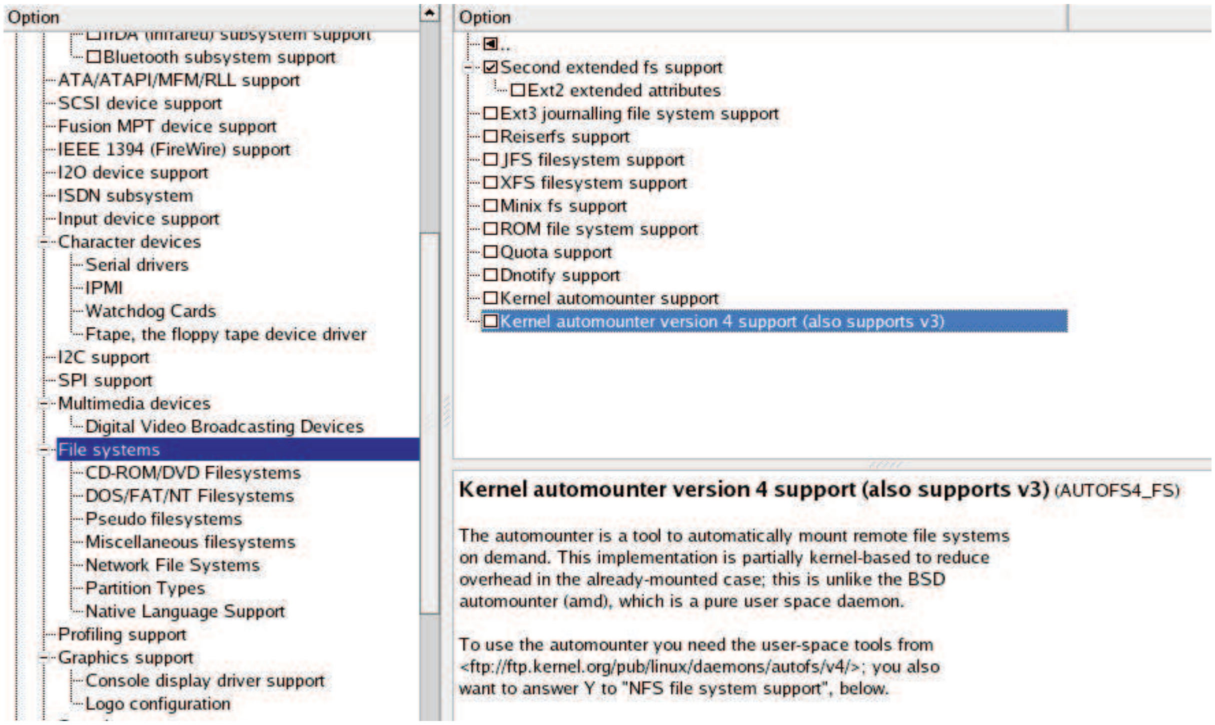
Enable loadable module support (MODULES)

Kernel modules are small pieces of compiled code which can be inserted in the running kernel, rather than being permanently built into the kernel. You use the "modprobe" tool to add (and sometimes remove) them. If you say Y here, many parts of the kernel can be built as modules (by answering M instead of Y where indicated): this is most useful for infrequently used options which are not required for booting. For more information, see the man pages for

Figure 1. Loadable Module Support

4. Under *Device Drivers → Generic Driver Options*, uncheck the *Select only drivers that don't need compile-time external firmware* and *Prevent firmware from being built* boxes to disable firmware loading features.

5. Under *Device Drivers* → *Memory Technology Devices (MTD)*, uncheck the *Memory Technology Devices (MTD) support* box to disable the memory technology driver support.
6. Under *Device Drivers* → *Block devices*, uncheck the *Loopback device support* box to disable the loopback device support used to mount an ISO image.
7. Under *Device Drivers* → *ATA/ATPI/MFM/RRL support*, uncheck the *ATA/ATPI/MFM/RRL support* box to disable the ATA support used to access the EVM hard drive.
8. Under *Device Drivers* → *SCSI device support*, uncheck the *legacy /proc/scsi/ support* and *SCSI disk support* boxes to disable SCSI disk support on the EVM.
9. Under *Device Drivers* → *Input device support*, uncheck the *Mouse interface*, *Event interface*, and *Keyboards* boxes to disable the input device support.
10. Under *Device Drivers* → *Multimedia devices*, uncheck the *Video For Linux* box to disable the v4l2 driver support used to capture video image from the camera.
11. Under *Device Drivers* → *File systems*, uncheck the *Ext3 journalling file system support*, *XFS file system support*, *Minix fs support*, *Dnotify support*, and *Kernel automounter version 4 support* boxes to disable file system supports. Do not uncheck ext2 file system support, as the ext2 file system is used in the initial RAM disk. See Figure 2.



☑Second extended fs support
□Ext2 extended attributes
□Ext3 journalling file system support
□Reiserfs support
□JFS filesystem support
□XFS filesystem support
□Minix fs support
□ROM file system support
□Quota support
□Dnotify support
□Kernel automounter support
□Kernel automounter version 4 support (also supports v3)

Kernel automounter version 4 support (also supports v3) (AUTOFS4_FS)

The automounter is a tool to automatically mount remote file systems on demand. This implementation is partially kernel-based to reduce overhead in the already-mounted case; this is unlike the BSD automounter (amd), which is a pure user space daemon.

To use the automounter you need the user-space tools from <ftp://ftp.kernel.org/pub/linux/daemons/autofs/v4/>; you also want to answer Y to "NFS file system support", below.

Figure 2. Disable File Systems

12. Under *Device Drivers* → *File systems* → *DOS/FAT/NT Filesystems*, uncheck the *MSDOS fs support* and *VFAT (Windows 95) fs support* boxes to disable Windows file system support.
13. Under *Device Drivers* → *File systems*→*Miscellaneous filesystems*, uncheck the *Compressed ROM file system support (cramfs)* box to disable cramfs file system support.
14. Under *Device Drivers* → *File systems* → *Network File Systems*, uncheck the *NFS file system support*, *NFS server support*, and *SMB file system support* boxes to disable network file systems support.
15. Under *Device Drivers* → *File systems*→*Partition Types*, uncheck the *Advanced Partition Selection* box to disable partition support on the hard disk.
16. Under *Device Drivers* → *Graphics Support*, uncheck the *Support for frame buffer devices* box to disable Linux frame buffer support.
17. Under *Device Drivers* → *Sound*, uncheck the *Sound card support* box to disable Linux sound support.
18. Under *Device Drivers* → *USB Support*, uncheck the *Support for Host-side USB* and *Inventra USB Highspeed Dual Role Controller Support* boxes to disable USB driver support.

19. Under *Device Drivers* → *MMC/SD Card Support*, uncheck the *MMC Support* box to disable Multimedia Card support.

2.2 Kernel Compilation

This section describes the kernel compilation steps.

Note: The DVEVM and DVSDK-L or -3L software packages may have different kernel configurations and build commands. Always check the documentation that comes with the package such as the DVEVM Release Notes, the DVEVM Quick Start Guide or *DVEVM Getting Started Guide* (SPRUE66) for the exact commands for the build steps. The following steps are for the kernel tree from the DVEVM software package.

1. If not already logged in as *user*, then log in as *user* prior to building the kernel.
2. Build the Linux kernel with this command:

```
host$ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

Note that the above kernel configuration disables most of the peripheral support, except for the networking stack, Ethernet, and Serial drivers. If additional applications are required beyond the ones used in this application report, other supporting features may need to be enabled.

The generated kernel, the u-boot compatible compressed binary file *uImage*, is located under the arch/arm/boot directory. Copy this file to the /tftpboot directory so that it can be flashed later on the DVEVM.

In the next step, you will build a RAM disk file system to save to flash.

3 Building an Initial RAM Disk File System

Although this section is not dependent on the previous section, it is assumed that you have already installed DVEVM software on the Linux host machine according to the steps outlined in the *DVEVM Getting Started Guide* (SPRUE66).

An initial RAM disk relies on a boot loader (such as u-boot) to load it from non-volatile memory (such as NOR flash) to volatile memory, (such as DDR2) before booting up the kernel. The file system inside the RAM disk is referred to as an initial RAM disk file system, or initrd. This file system can be mounted as a root file system and the application can be executed from it. This is the kernel's local storage. As it is installed on volatile memory, its contents are lost when the system is powered off. For most embedded systems, this is a desired run-time environment. If you must save some parameters generated during run time, you will require a NOR flash file system, which is outside the scope of this project.

With the Davinci EVM platform, you can build a RAM disk file system using either the MontaVista® DevRocket™ IDE (available with the –L or -3L DVSDK packages), or command line scripts (available with DVEVM and all DVSDK software packages).

To execute the web server, the initial RAM disk file system should contain the following GNU packages. They can be found with the DVEVM software MVL Pro install directory under <tool chain install directory>/pro/devkit/arm/v5t_le/packages/pro or pro/optional:

Table 4. Linux Packages for the RAM Disk File System

Item	Version	Description
busybox	1.00r3-5.0.0	Combines small versions of many common Linux utilities.
initscript	2.85-3.0.0	Contains basic system script used to boot the system.
netbase	4.17-1.0.1	Provides necessary infrastructure for TCP/IP networking.
thttpd	2.25b-1.0.0	Contains a small, fast, and secure web server, including CGI support, URL traffic based throttling and basic authentication.

Note: Cross-building these packages provided here or from the GNU source trees is beyond the scope of this application report. Please consult the appropriate document or embedded Linux books on how to perform these tasks.

Several options are available to complete this step, including using an existing RAM disk, or building one for your needs.

3.1 Use an Existing RAM Disk

To save time, a RAM disk is provided with the DVEVM ARM Linux software tool chain. It is located under:

<tool chain install directory>/pro/devkit/arm/v5t_le/images

Note: In later releases of the DVSDK packages the sample RAM disk image is located at:

<dvsdk install dir>/<PSP dir>/bin

In this directory, the RAM disk file is called ramdisk.gz (about 2.1 MB gunzipped). In run time, it occupies about 6.3 MB in DDR. This file system contains some unnecessary utilities for this project, but is appropriate for a typical embedded system.

3.2 Set Up the RAM Disk for Use

1. Copy the existing initial RAM disk to a temporary location:

```
host $ mkdir /mnt/def_cd
host $ cp <tool chain install dir>/pro/devkit/arm/v5t_le/images/ramdisk.gz
/mnt/def_cd
```

2. Unzip the file, creating a file called *ramdisk:*

```
host $  gzip –d /mnt/def_cd/ramdisk.gz
```

3. Create a mount point and mount the RAM disk for use:

```
host $ mkdir –p /mnt/def_cd/ram0
host $ mount –o loop /mnt/def_cd/ramdisk   /mnt/def_cd/ram0
```

   You can browse the RAM disk contents by changing to the mounted directory and listing the contents:

```
host $ cd /mnt/def_cd/ram0
host $ ls
```

   The console output shows the typical Linux directory structure.

   In the next step, you add the application package and the http web server to the RAM disk, as well as some initialization and use scripts before zipping it up again for flashing.

4 Application Support

This section describes how to add a small web server (thttpd) to the initial RAM disk file system and configure it for the DVEVM.

The web server thttpd is a simple, small, portable, fast, and secure HTTP server with the following features:

- Simple: It handles only the minimum information necessary to implement HTTP/1.1.
- Small: It has a small run-time size, because it allocates memory conservatively and does not fork.
- Portable: It compiles cleanly on most Unix-like operating systems, including FreeBSD, SunOS 4, Solaris 2, BSD/OS, Linux, and OSF.
- Fast: In typical use, it is as fast as the best full-featured servers (Apache, NCSA, Netscape). Under extreme loads, it is much faster.
- Secure: It protects the web server machine against attacks and break-ins from other sites.

4.1 Build the http Web Server

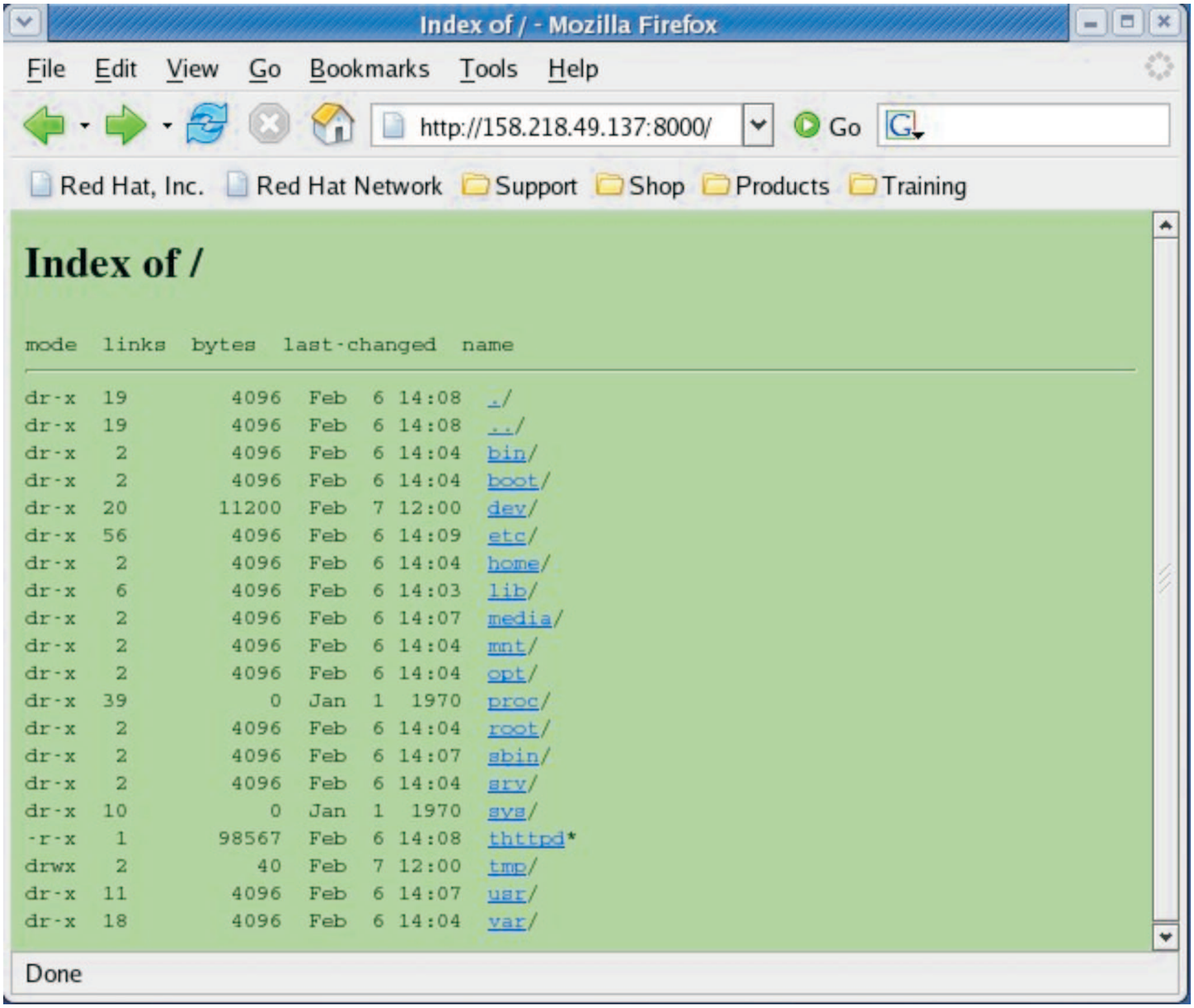
You can build the web server either on the host development PC or natively on the EVM using ARM gcc tool chains. This section describes how to cross-build the web server on the host development PC.

- Download the latest thttpd from the developer's website: http://www.acme.com/software/thttpd/, or use the source provided as part of DVEVM software package. The following instructions are for unpacking and compiling the open source version. The DVEVM software package already has the source files installed under <dvevm install dir>/examples/thttpd-2.25b directory. In later releases the path to the source files may be <dvsdk install dir>/examples/<device>/thttpd-2.25b and the thttpd binary may already exist at this location.
  - `host $ cd ~/workdir`
  - `host $ tar xzf path-to-tar-file/thttpd-2.25b.tar.gz`
  - `host $ cd thttpd-2.25b`
- Verify that the path to the ARM cross-compile tool chain is exported as described in the *DVEVM Getting Started Guide* (SPRUE66).
- Compile the web server as follows:
  - `host $ CC=arm_v5t_le-gcc ./configure`
  - `host $ make`

4.2 Test the Web Server

1. Copy the thttpd executable to the EVM board. The *DVEVM Getting Started Guide* describes the procedure for HDD or NFS configurations.
2. Run thttpd on the EVM using an arbitrary port of 8000.

   ```
   dvevm $./thttpd –p 8000
   ```
3. Connect to the new web server using the PC's browser. The URL is the EVM board's IP address with the addition of the port number: http://evm-ip-address:8000. This URL is set up once you run the thttpd program on the DVEVM. You should see output similar to:


Index of / - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://158.218.49.137:8000/
Go
Red Hat, Inc.
Red Hat Network
Support
Shop
Products
Training
Done

Index of /

```
mode  links  bytes  last-changed  name
dr-x  19      4096  Feb  6 14:08  ./
dr-x  19      4096  Feb  6 14:08  ../
dr-x   2      4096  Feb  6 14:04  bin/
dr-x   2      4096  Feb  6 14:04  boot/
dr-x  20     11200  Feb  7 12:00  dev/
dr-x  56      4096  Feb  6 14:09  etc/
dr-x   2      4096  Feb  6 14:04  home/
dr-x   6      4096  Feb  6 14:03  lib/
dr-x   2      4096  Feb  6 14:07  media/
dr-x   2      4096  Feb  6 14:04  mnt/
dr-x   2      4096  Feb  6 14:04  opt/
dr-x  39         0  Jan  1  1970  proc/
dr-x   2      4096  Feb  6 14:04  root/
dr-x   2      4096  Feb  6 14:07  sbin/
dr-x   2      4096  Feb  6 14:04  srv/
dr-x  10         0  Jan  1  1970  sys/
-r-x   1     98567  Feb  6 14:08  thttpd*
drwx   2        40  Feb  7 12:00  tmp/
dr-x  11      4096  Feb  6 14:07  usr/
dr-x  18      4096  Feb  6 14:04  var/
```

Figure 3. Index of Mozilla Firefox

4.3 Add Web Server to Initial RAM Disk

This section describes how to modify the existing initial RAM disk to include the thttpd web server and cgi scripts. To save time, the web server and TI graphics files used by the sample index.html page are included in the DVEVM/DVSDK packages, which can be downloaded from www.ti.com/dvevmupdates. For more information on creating the index.html file and cgi scripts used in this demo see Appendix A.

1. Create a web directory on the RAM disk to copy the thttpd executable into:

```
host $ mkdir -p /mnt/def_cd/ram0/opt/dvevm/web
host $ cp thttpd /mnt/def_cd/ram0/opt/dvevm/web
```

2. Write the index.html file and copy it to the initial RAM disk file system. For information on the index.html file used in this demo please refer to Appendix A.

   ```
   host $ cp index.html /mnt/def_cd/ram0/opt/dvevm/web
   ```

3. Write the cgi scripts and copy them to the cgi-bin directory of the initial RAM disk file system. For information on the cgi scripts used in this demo please refer to Appendix A.

   ```
   host $ mkdir -p /mnt/def_cd/ram0/opt/dvevm/web/cgi-bin
   host $ cp <cgi files> /mnt/def_cd/ram0/opt/dvevm/web/cgi-bin/
   ```

4. Copy the TI graphics used by the index.html file to the RAM disk file system. These graphics are located in the DVEVM/DVSDK software package, which can be downloaded from www.ti.com/dvevmupdates.

   ```
   host $ cp <dvevm/dvsdk install dir>/examples/web/*.gif
   /mnt/def_cd/ram0/opt/dvevm/web
   ```

   Note: For later releases of the DVEVM/DVSDK software the location of the TI graphics files may look like <dvsdk install dir>/examples/<device>/web

5. Edit a file called startweb.sh and add the following script lines to start the web server:

   ```
   #!/bin/sh
   # script to start web server
   echo "Start web service..."
   /opt/dvevm/web/thttpd -d /opt/dvevm/web -c "/cgi-bin/*"
   ```

6. Copy this script to the /etc/init.d directory of the RAM disk to make it part of the boot-up sequence:

   ```
   host $ cp startweb.sh /mnt/def_cd/ram0/etc/init.d
   host $ chmod +x /mnt/def_cd/ram0/etc/init.d/startweb.sh
   host $ cd /mnt/def_cd/ram0/etc/rc.d/rcS.d
   host $ ln –s ../init.d/startweb.sh S42startweb
   ```

7. Finally, recompress this RAM disk for flashing:

   ```
   host $ cd /mnt/def_cd
   host $ umount /mnt/def_cd/ram0
   host $ gzip ramdisk
   host $ cp ramdisk.gz /tftpboot
   ```

5 Copying Information to NOR Flash

This section requires completion of Section 2, Section 3 and Section 4. In this section, you will copy the kernel image and initial RAM disk to NOR flash.

1. Copy the kernel image into the /tftpboot directory if you have not already done so:

   ```
   host $ cp ~/workdir/lsp/ti-davinci/arch/arm/boot/uImage /tftpboot
   ```

2. Copy the initial RAM disk file system into the /tftpboot directory if you have not already done so:

   ```
   host $ cp /mnt/def_cd/ramdisk.gz /tftpboot/
   ```

3. Download the Linux kernel via TFTP:

   ```
   DVEVM # setenv serverip <tftp server ip address>
   DVEVM # setenv bootfile uImage
   DVEVM # dhcp
   BOOTP broadcast 1
    *** Unhandled DHCP Option in OFFER/ACK: 44
   *** Unhandled DHCP Option in OFFER/ACK: 46
   DHCP client bound to address <dvem ip address>
   TFTP from server <tftp server ip address>; our IP address is <dvevm ip
   address>
   Filename 'uImage'.
   ```

```
Load address: 0x80700000
Loading:
#########################################################
#########################################################
done
Bytes transferred = 823844 (c9224 hex)
```

The dhcp command obtains IP settings and then downloads the Linux kernel image (as specified by the serverip and bootfile environment variables). Note the Load address (0x80700000) and Bytes transferred (0xc9224), as these are needed in the following steps.

4. Download the RAM disk file system via TFTP:

```
DVEVM # tftp 0x85000000 ramdisk.gz
TFTP from server <tftp server ip  address>; our IP address is <dvevm ip
address>
Filename 'ramdisk.gz'.
Load address: 0x85000000
Loading:
#########################################################
#########################################################
done
Bytes transferred = 2304639 (232a7f hex)
```

The tftp command downloads the ramdisk.gz file at 0x85000000. Note the Load address (0x85000000) and Bytes transferred (0x232a7f), as these are needed in the following steps.

5. Determine the location in flash to store image:

```
EVM # flinfo
Bank # 1: MY AMD 29LV256M (256 Mbit)
 Size: 16 MB in 256 Sectors
Sector Start Addresses:

  02000000       02010000       02020000       02030000       02040000 (RO)
  02050000       02060000       02070000       02080000       02090000
  020A0000       020B0000       020C0000       020D0000       020E0000
```

The U-Boot code and data are stored in the first five sectors, starting at 0x2000000. Note that the trailing *(RO)* indicates that the sectors are read-only or protected from erasing and writing. The Linux kernel image should be saved to the 0x2050000 location, the first free sector after U-Boot.

Note: For Intel NOR Flash chips you should use flash address 0x2060000, which is the first start address after U-Boot, rather than 0x2050000. The following commands should be adjusted to use this address.

6. Erase the flash:

```
DVEVM # protect off 0x2050000 +0x2FBCA3
```

Note: 0x2FBCA3 is addition of kernel image and ramdisk image size.

```
DVEVM # erase 0x2050000 +0x2FBCA3
Erasing sector  5 ... done.
Erasing sector  6 ... done.
```

The *protect off* command makes the flash writable (not necessary for this example), while the erase command prepares the flash for writing by erasing the old contents. Note the start address (0x2050000) is derived from the output of flinfo and the length (the size of the Linux kernel image plus the RAM disk file system downloaded via the TFTP server).

7. Copy from RAM into flash:

   ```
   DVEVM # cp.b 0x80700000 0x2050000 0xc9224
   ```

   Copy to Flash.../done

   ```
   DVEVM # cp.b 0x85000000 0x2119224 0x232a7f
   ```

   Note: The destination address 0x2119224 is derived from adding the kernel size 0xc9224 to the start address 0x2050000. This is so that the RAM disk image is written in the flash sectors after the kernel.

   Copy to Flash.../done

   The cp (copy) command is used to copy the Linux kernel image in RAM into the accessible flash memory. The arguments are the source address, the destination address, and the length. The .b extension on the cp command specifies a byte-wise copy.

8. Protect the flash from writing:

   ```
   DVEVM # protect on 0x2050000 +0x2FBCA3
   ```

   The protect command makes the flash sector read-only, to ensure that the kernel image and RAM disk file system are not accidentally overwritten.

9. Set the U-Boot Command and Linux Kernel Command Line:

   Note: For Intel NOR Flash chips use flash address 0x2060000 instead of 0x2050000

   ```
   DVEVM # setenv bootargs console=ttyS0,115200n8 ip=dhcp root=/dev/ram0 rw
   initrd=0x85000000,6M
   DVEVM# setenv bootcmd ‘cp.b 0x2119224 0x85000000  0x232a7f; bootm 0x2050000’
   ```

   The boot command is set to use the kernel image in flash at address 0x2050000. The bootcmd first does a copy of the RAM disk image from the location it was written to in step 7 to a location in RAM. The Linux kernel command line arguments (bootargs) are set to use the RAM disk as the root file system and specify its location in RAM where the RAM disk was copied in the bootcmd operation.

10. Now, the system is ready to boot, so save the u-boot environment variable:

    ```
    DVEVM # saveenv
    DVEVM # boot
    ```

    Linux should now boot from flash and the root file system should be mounted on /dev/ram0.

Table 5 summarizes where u-boot, the Linux kernel, and the compressed RAM disk are loaded on the 16 MB NOR flash memory.

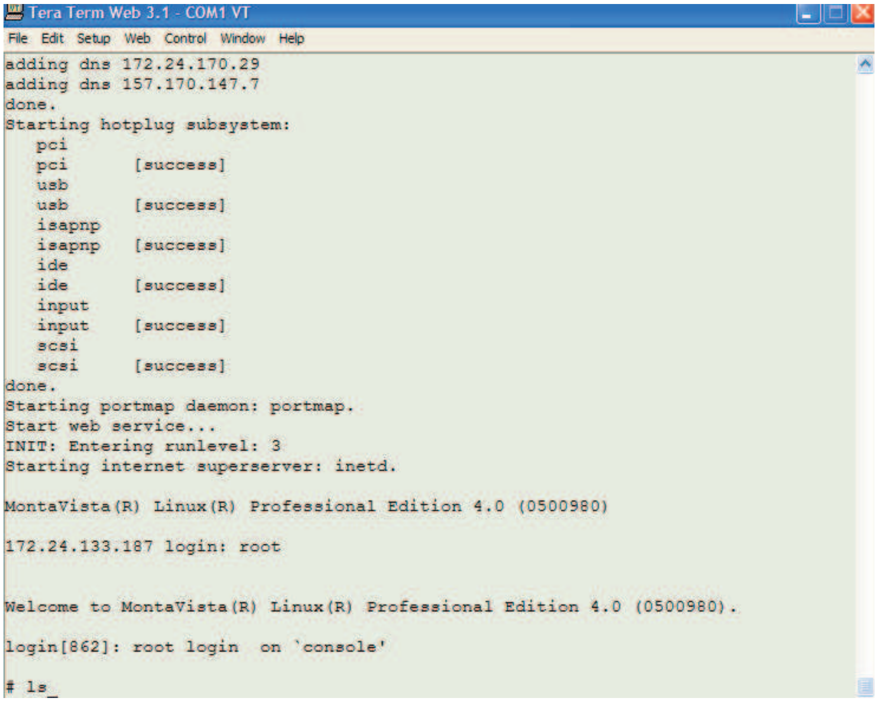
Table 5. Memory Placement of Bootloader, Kernel and RAM Disk in DVEVM NOR Flash

Address	Content
0x0200 0000 – 0x0204 FFFF	u-boot and u-boot parameters (327 KB)
0x0205 0000 – 0x0211 9223	uImage – Linux kernel (823KB)
0x0211 9224 – 0x0234BCA3	Compressed RAM disk (2.1MB)
0x0234BCA4 – 0x02FF FFFF	Unused (12.70MB)

6 Boot Up

This section requires the completion of Section 5. It describes how to access the web server and log in to the EVM.

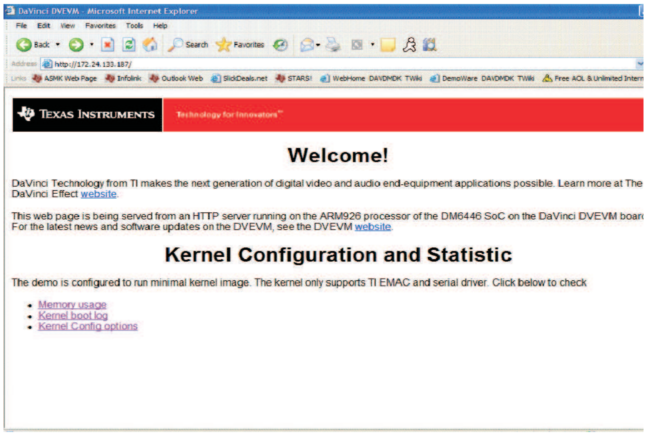
1. Power on the EVM board. On successful boot-up, it prompts for login. See Figure 4.


Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help

```
adding dns 172.24.170.29
adding dns 157.170.147.7
done.
Starting hotplug subsystem:
   pci
   pci      [success]
   usb
   usb      [success]
   isapnp
   isapnp   [success]
   ide
   ide      [success]
   input
   input    [success]
   scsi
   scsi     [success]
done.
Starting portmap daemon: portmap.
Start web service...
INIT: Entering runlevel: 3
Starting internet superserver: inetd.

MontaVista(R) Linux(R) Professional Edition 4.0 (0500980)

172.24.133.187 login: root


Welcome to MontaVista(R) Linux(R) Professional Edition 4.0 (0500980).

login[862]: root login  on `console'

# ls_
```

Figure 4. EVM Boot-Up Screen

2. Log in as *root*.
3. Open the web browser on the host machine and connect to the EVM.
4. Connect to the EVM web server by typing the EVM IP address in the URL address box. See Figure 5.



http://172.24.133.187/

Welcome!

DaVinci Technology from TI makes the next generation of digital video and audio end-equipment applications possible. Learn more at The DaVinci Effect website.

This web page is being served from an HTTP server running on the ARM926 processor of the DM6446 SoC on the DaVinci DVEVM boar For the latest news and software updates on the DVEVM, see the DVEVM website.

Kernel Configuration and Statistic

The demo is configured to run minimal kernel image. The kernel only supports TI EMAC and serial driver. Click below to check

- Memory usage
- Kernel boot log
- Kernel Config options

Figure 5. Web Screen Connected to the DVEVM Web Server

5. Click on the *Memory usage* link. The web page displays the output of the *cat /proc/meminfo* command.
6. Click on the *Kernel Config options* link. The web page displays the *.config* file used to build the kernel image.
7. Click on the *Kernel boot log* link. The web page displays the output of the *dmesg* command.

7 Summary

This application report described kernel configuration and build steps using the DaVinci DVEVM software package. A small kernel feature set was selected and an http server package was added to a RAM disk to use as the root file system. The kernel and RAM disk were subsequently flashed to the NOR flash memory, where u-boot, the boot loader, also resided. You then started this system and demonstrated that a web browser can connect to the DVEVM running this kernel and http server.

As mentioned in the abstract, this type of setup can be used as a starting point for an embedded Linux system development with the DaVinci DM644x EVM. Other features can be added, such as:

- NOR flash file system, like a jffs2 file system, to provide persistent local storage.
- The V4L2 driver, so that video images can be captured and compressed using a encoder running on the DSP of the DM644x device. This can turn the DVEVM into a video server.

8 References

- *Davinci-DM644x Evaluation Module Technical Reference*, Spectrum Digital, 508165-0001. For the latest version visit the Spectrum Digital web site at www.spectrumdigital.com
- *DVEVM Getting Started Guide* (SPRUE66). For the latest version of the software, check the www.ti.com/dvevmupdates site.

Appendix A

A.1 Creating the index.html File

Edit a file called index.html and add the following lines to it:

```
<HTML>
<HEAD>
<TITLE>DaVinci DVEVM</TITLE>

<STYLE TYPE="text/css">
BODY,H1,H2,H3,H4,H5,H6,P,CENTER,TD,TH,UL,DL,DIV {
     font-family: Geneva, Arial, Helvetica, sans-serif;
}
H1 {
     text-align: center;
}
CAPTION { font-weight: bold }
</STYLE>

</HEAD>

<BODY>

<table width=100%>
<tr>
  <td bgcolor="black" width="1"><a href="http://www.ti.com"><img border=0
src="tilogo.gif"></a></td>
  <td bgcolor="red"><img src="titagline.gif"></td>
</tr>
</table>


<H1>Welcome!</H1>
<P>
DaVinci Technology from TI makes the next generation of digital video and
audio end-equipment applications possible. Learn more at The DaVinci Effect
<A HREF="http://www.thedavincieffect.com">website</A>.
</P>

<P>
This web page is being served from an HTTP server running on the
ARM926 processor of the DM6446 SoC on the DaVinci DVEVM board. For the
latest news and software updates on the DVEVM, see the DVEVM
<A HREF="http://www.ti.com/dvevmupdates">website</A>.
</P>

<H1> Kernel Configuration and Statistic</H1>
<P>
The demo is configured to run minimal kernel image. The kernel only supports TI EMAC and serial
driver. Click below to check

<UL>
    <LI> <A HREF="/cgi-bin/memory">Memory usage</A> </LI>
    <LI> <A HREF="/cgi-bin/log">Kernel boot log</A> </LI>
    <LI> <A HREF="/cgi-bin/config">Kernel Config options</A> </LI>
</UL>

</BODY>
</HTML>
```

A.2 Creating the CGI Scripts

1. Edit a file called memory and add the following script lines to create the memory usage script:

   ```
   #!/bin/sh

   cat << EOF
   Content-type: text/plain
   Cache-control: no-cache
   ```

```
EOF

echo "#cat /proc/meminfo"
cat /proc/meminfo

echo
echo "# free -b"
free -b

echo
echo "# ps -el"
ps -el
```

2. Edit a file called memory and add the following script lines to create the kernel boot log script :

```
#!/bin/sh

cat << EOF
Content-type: text/plain
Cache-control: no-cache

EOF

echo
echo "#dmesg"
dmesg
```

3. Edit a file called memory and add the following script lines to create the kernel config options script :

```
#!/bin/sh

cat << EOF
Content-type: text/plain
Cache-control: no-cache

EOF

rm -rf /tmp/config*
cp /proc/config.gz /tmp
gzip -d /tmp/config.gz
cat /tmp/config
```

4. Make the scripts executable:

```
chmod +x memory
chmod +x log
chmod +x config
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265