Embbeded RTLinux
Using DiskOnChip 2000 Tsop
With M-Systems Driver
Version 1.0

Reference Configuration

The configuration used in this procedure is the following:

- Base install of Red Hat Linux 6.2 and 7. 3 performed on an RTD CPU PC/104 CMX27686HX300R-128D with 128MB SDRAM cpuModule (more details – www,rtd.com).
- Prepatched.2.4.29-rtl-3.1 kernel built from source code downloaded from www.fsmlabs.com and with the CPU type set to 386 Pentium MMX.
- 256 MB M-Systems DiskOnChip 2000 Tsop
- HDD Seagete 20 Gb
- Floppy Disk 1.44
- CDROM ATAPI 42x
- M-Systems TrueFFS Linux driver version 5.1.4
- LILO version 22.3.2 (distributed with driver)
- DiskOnChip DOS utilities and BIOS driver version 5.1.4
- Creating a single partition on the device
- Using ext2 as the DiskOnChip file system type
- Setting the DiskOnChip as the first available disk drive when booting from it

Introduction

The reasons for the design of RTLinux can be understood by examining the work of the standard Linux kernel. The Linux kernel separates the hardware from the user-level tasks. The kernel uses scheduling algorithms to assign priorities to each task to provide good average performances or throughput. Thus, the kernel has the ability to suspend any user-level task, once that task has outrun the time-slice allotted to it by the CPU. This scheduling algorithms, along with the device drivers, uninterruptible system calls, the use of interrupt disabling and virtual memory operations, are sources of unpredictability. That is to say, these sources cause a hindrance to the realtime performance of a task. You might already be familiar with the non-realtime performance, say, when you are listening to the music played using 'mpg123' or any other player. After executing this process for a pre-determined time-slice, the standard Linux kernel could pre-empt the task and give the CPU to another one (e.g. one that boots up the X server or Netscape). Consequently, the continuity of the music is lost. Thus, when trying to ensure an adequate distribution of CPU time among all the processes, the kernel can prevent other events from occurring.
A realtime kernel should be able to guarantee the timing requirements of the processes running below it. The RTLinux kernel accomplishes realtime performances by removing such sources of unpredictability as discussed above. We can consider the RTLinux kernel as placed between the standard Linux kernel and the hardware. The Linux kernel sees the realtime layer as the actual hardware. Now, the user can both introduce and set priorities to every task. The user can achieve correct timing for the processes by deciding on the scheduling algorithms, priorities, frequency of execution etc. The RTLinux kernel assigns the lowest priority to the standard Linux kernel. Thus the user-task will be executed in realtime.
The actual realtime performance is obtained by intercepting all the hardware interrupts. Only for those interrupts that are related to the RTLinux, the appropriate interrupt service routine is run. All other interrupts are held and passed to the Linux kernel as software interrupts when the RTLinux kernel is idle and then the standard Linux kernel runs. The RTLinux executive is itself nonpreemptible.
Realtime tasks are privileged (that is, they have direct access to hardware), and they do not use virtual memory. Realtime tasks are written as special Linux modules that can be dynamically loaded into memory. The initialization code for a realtime task initializes the realtime task structure and informs RTLinux kernel of its deadline, period, and release-time constraints.
RTLinux co-exists along with the Linux kernel since it leaves the Linux kernel untouched. Via a set of relatively simple modifications, it manages to convert the existing Linux kernel into a hard realtime environment without hindering future Linux development.

To use a DiskOnChip under Linux, a device driver is needed. M-Systems provides the TrueFFS driver, consisting of a binary kernel module and some source code. According to M-Systems, "TrueFFS for Linux is GPL compliant". However, restrictions exist on its distribution. See the M-Systems web site (http://www.m-sys.com) for more information. M-Systems recommends that you use their driver instead of the Linux Memory Technology Device (MTD) driver.

For More Information

The Linux Documentation Project contains a HOWTO for making Linux boot disks. This document gives some ideas on selecting files for inclusion in a minimal, self-contained boot disk; when such a disk is booted, it provides a bare-bones Linux environment. This information can be generalized to create a Linux boot environment on a DiskOnChip. The HOWTO also includes some troubleshooting tips to manage at the situation when the boot process succeeds partway but then fails.

M-Systems produced a document called "Using the DiskOnChip with Linux OS" (document number IM-DOC-021). Unfortunately, it is no longer available from M-Systems, but you should be able to find

it by doing a web search for the document number. The last revision appears to be 3.2. This document is out of date (referring to prior versions of the Linux driver and building process) but does provide some guidance such as selecting files for inclusion in the minimal file system and how to create a partition table on the DiskOnChip device. It should be used as a general reference only, and not as an example of how to set up your system.

Kernel source code contains instructions on how to build the kernel. These directions can be found in the text file linux-<kernel version>/README, where linux-<kernel version> is the directory created when the kernel source is extracted.

O'Reilly (www.oreilly.com) published a book called "Building Embedded Linux Systems" written by Karim Yaghmour. This manuscript contains much information pertinent to those seeking to run Linux from Solid State Devices.

Preliminaries

You will need several files. Download the following sources:

From rediris (cedes.upv.es):

- Red Hat 6.2 or Red Hat 7.3

From M-Systems web site (www.m-sys.com):

- TrueFFS Linux driver (current version 5.1.4; gzip'd tar file)
- DiskOnChip DOS utilities and BIOS driver (current version 5.1.4; zip file)

From Fsmlabs (www.fsmlab.com):

- prepatched_kernel_linux_2.4.29-rtl.tgz
- rtlinux-3.2-rc1.tgz

Some red hat rpm's (www.google.com):

- nasm-0.98-2.i386.rpm

All the downloaded files should be stored on a cdrom.

First, we will install Red hat 6.2 or Red Hat 7.3 in the embedded system. We will use a bootable cdrom and follow the installation process. With a red hat linux distribution running properly, we will unzip the DOS utilities to a directory. Create a DOS boot floppy disk. From the directory where you unzipped the utilities, copy the dformat.exe and doc514.exb files onto the floppy disk. This boot disk will be used whenever you need to format the DiskOnChip.

All the projects' sources are stored on a data CDROM, the next list shows all the necessary software:

Configuring Linux for DiskOnChip Usage

1. Before you begin

If you decide to create a Linux boot disk before starting, use the native Linux shell script *(/sbin/mkbootdisk* on Red Hat) instead.

2. Installing DiskOnChip driver

It seems that M-Systems does not support compiling the TrueFFS driver into the kernel. Therefore, it must be created as a loadable module.

2.1. Format the DiskOnChip

Instead of using the Linux **./dformat** supplied with the driver, use the **dformat** command on the DOS boot floppy you created earlier. This means you will need to boot from this floppy each time that formatting the device is required.

If your DiskOnChip currently contains firmware, once you enable the extension window in the computer's BIOS you will need to remove this firmware. The extension window needs to be enabled whenever the device accessed for any reason, for example writing files to it or booting from it. If you want to temporarily disable the device, you can simply turn off the BIOS extension window; you will not be able to access it again until you enable the window.

The command you use to format the DiskOnChip depends upon whether you want to use it as a storage device or as a boot device. To it use as a storage device :

dformat /win:xxxx /s:doc514.exb! /y /nodos

If the DiskOnChip is to be used as a boot device:

dformat /win:xxxx /s:doc514.exb /y /nodos /empty

If you ever need to remove the firmware (yet keep space reserved for it) without disturbing the rest of the data on a device:

dformat /win:xxxx /s:doc514.exb /y /noformat /empty

The xxxx in the *"/win:xxxx*" argument refers to the extension window you set in the BIOS. See the M-Systems DOS utilities documentation for more information about **dformat** and its options.

In this example, the goal is to use the DiskOnDisk as a boot device, therefore the correct command is:

dformat /WIN:D000 /S:doc514.exb /Y /NODOS /EMPTY (on DOS shell).

Once you format the device, you can reboot into Linux to continue with the process.

2.2. Install driver sources

The patch file *linux-2_4_7-patch* is known to work with recent kernels. It was used for the purpose of creating this document.

The steps to conduct patching are as follows:

1. Mount the CDROM where the sources are

mount /dev/cdrom/

2. Put all the sources in /usr/src/ directory

   cd /usr/src/

3. Untar the prepatched kernel and the rtlinux sources files from www.fmslabs.com

   tar -xvzf /mnt/cdrom/prepatched_linux_kernel-2.4.29-rtl.tgz

   tar -xvzf /mnt/cdrom/rtlinux-3.2-rc1.tgz

   This results in the creation of two directories: rtlinux-3.2-rc1 and linux-2.4.29-rtl-3.1

4. Untar the M-systems TrueFFS Linux driver version 5.14.

   tar -xvzf /mnt/cdrom/linux_binary.5_1_4.tgz

   This results in one directory: linux_binary.5_1_4.

5. Make a symbolic link to the new kernel source directory.

   rm –f linux

   ln –s /usr/src/linux-2.4.29-rtl-3.1 linux

6. The TrueFFS Linux driver package contains three different folders:

   - Documentation: it contains a PDF document describing the various functions of TrueFFS.
   - dformat_5_1_4_37: it contains a utility **dformat**, which is used to update the firmware on the DiskOnChip (DOC) and to create low level partitions on the DOC.
   - doc-linux-5_1_4_20: it contains patches, initrd scripts and other utilities.

7. Now apply the patch to the kernel. We will use the linux-2_4_7-patch file that is present in linux_binary.5_1_4/doc-linux-5_1_4_20/driver. The following commands are used for this purpose:

   cd linux_binary.5_1_4/doc-linux-5_1_4_20/driver

   patch -p1 -d /usr/src/linux < linux-2_4_7-patch

This will create a directory named *doc* in the *linux/drivers/block* directory.

8. The patch created the doc directory, but did not copy the source files of the M-Systems driver, which are necessary in order to build the driver, into this directory. So execute the following command:

   cp linux_binary.5_1_4/doc-linux-5_1_4_20/driver/doc/* /usr/src/linux/drivers/block/doc

2.3. Create nodes

Now we will create block devices, which are required to access the DOC. These block devices will use the M-Systems driver to access the DOC. The script ./**mknod_fl** in *linux_binary.5_1_4/doc-linux-5_1_4_20/driver* is used for this purpose.

./mknod_fl

This will create the following devices in /dev/msys with major number 100:

```
fla...fla4
flb...flb4
flc...flc4
fld...fld4
```

2.4. Change kernel configuration

Complete the following tasks for compiling the kernel:

1. **cd /usr/src/linux**
2. **make menuconfig**

Check for the following options:

- In the "Block devices menu", select:
  - M-Systems driver as module i.e. (M)
  - Loopback device support as built-in i.e. (*)
  - RAM disk support as built-in i.e. (*)
  - Initial RAM disk (initrd) support as built .in i.e. (*)
- In the "Processor type and features menu", select "Disable Symmetric Multiprocessor Support". The system tested use Pentium MMX processor (*).
- In the "filesystem menu", select:
  - Ext3 journaling file system support as built-in
  - DOS FAT fs support as built-in[a]

- MSDOS fs support as built-in[b]
- VFAT (Windows-95) fs support as built-in[c]

File System Menu

a,b,c options should be activated if you want to mount your MS Windows partition, otherwise they can be left out. It is, however, generally recommended to use them.

An excellent resource on kernel compilation is the Kernel Rebuild Guide.

3. **make dep**
4. **make bzImage**
5. **make modules**
6. **make modules_install**

7. Copy the newly created *bzImage* to the */boot* directory and name it *vmlinuz-2.4.29-rtl-3.1*, using this command:

   cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.29-rtl-3.1

   cp System.map /boot/System.map-2.4.29-rtl-3.1

8. Modify the /etc/lilo.conf file, using a text editor, and add the following lines to it:

```
image=/boot/vmlinuz-2.4.29-rtl-3.1
label=rtlinux
root=/dev/hda8
read-only
```

9. Execute */sbin/./lilo* to apply the changes in the *lilo.conf* file

   /sbin/./lilo

10. Reboot the system

Check for *lib/modules/2.4.18/kernel/drivers/block/doc.o*. This is the M-Systems driver that we need to access DiskOnChip.

2.5. Installing RTLinux

1. Access to */usr/src/rtlinux-3.2-rc1* and configure the sources

cd /usr/src/rtlinux-3.2-rc1

make menuconfig

2. Compile RTLinux

   make

   make devices

   make install

3. Reboot the System

The step 2 will create the directory */usr/rtlinux-3.2-rc1***,** which contains the default installation directory for RTLinux which is needed to create and compile user programs (that is, it contains the include files, utilities, and documentation). It will also create a symbolic link */usr/rtlinux*, which points to */usr/rtlinux-32-rc1*. In order to maintain future compatibility, please make sure that all of your own RTLinux programs use */usr/rtlinux* as its default path.

NOTE : If you change any Linux kernel options, please don't forget to do:

1. **cd /usr/src/rtlinux**
2. **make clean**
3. **make**
4. **make install**

2.6. Insert the DOC Module

The M-Systems driver by default gets loaded with major number 100.

1. Inserting the module using **insmod**:

   /sbin/./insmod doc

2. Then use modprobe doc to insert the modules.

   /sbin/./modprobe doc

Check for the correct loading of the module using the **lsmod** command without options.

2.7. Initialize file system on DiskOnChip

Before we can start using DiskOnChip we need to create a file system on it. We will create an ext2 file system since it is small in size.
This involves a hidden step of making partitions on the DOC using **fdisk**. The actual steps are as follows:

1. **fdisk /dev/msys/fla**

   This command will ask to create partitions. Create a primary partition number 1 with start cylinder as 1 and final cylinder as 1002.

   Check the partition table, which should look like this:

```
Device          Boot  Start     End      Blocks   ID    System
/dev/msys/fla1           1       1002      255984  83    Linux
```

2. **mke2fs -c /dev/msys/fla1**
   Make the filesystem on /dev/msys/fla1 with this command. Where fla1 is the first primary partition on the DOC. (We have created only one partition in order to avoid unnecessary complexity.)

2.8. Mount DiskOnChip partitions

1. Create a new mount point for the DiskOnChip in the */mnt* directory:

   mkdir –p /mnt/fla1

2. Mount the DOC partition on the newly created directory:

   mount -t auto /dev/msys/fla1 /mnt/fla1

2.9. Install the correct boot loader

Unfortunately, you will end up having to build LILO from source code. Neither the executable in *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/* nor the one in *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2*. has DiskOnChip support. Using either of these will result in an error message similar to "Fatal: Sorry, don't know how to handle device" when you attempt to write the boot sector. Building LILO requires that the Netwide Assembler *nasm* be installed on your system.

The *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/boot.b* file supplied in the tar file will work.

We need to compile the lilo-22-3.2 source code to get the executable file for LILO.
We will use the source code from *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2.*
Before starting the building process we need to do the following:

1. Patch file: *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/lilo.h*:

   Modify the following line in the *lilo.h* :

```
#define MAJOR_FL 62

by

#define MAJOR_FL 100
```

2. Patch file: *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/common.h*:

The lilo-22.3.2 source code that comes with the M-Systems *linux_binary.5_1_4.tgz* is buggy as one of the variables PAGE_SIZE is not defined. We need to patch the LILO source code as follows:
Add the following lines in the *common.h* after the line *"#include .lilo.h."*:

```
+ #ifndef PAGE_SIZE
+ #define PAGE_SIZE 4096U
+ #endif
#define 0_NACCESS 3
```

Where "+" indicates the lines to be added.

3. Start the build process.

   make *clean* && make

This will create a new LILO executable, *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/lilo*. Copy this LILO executable into */sbin/lilo* :

cp linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/lilo /sbin/lilo

2.10. Modify LILO Configure

Now modify the */etc/lilo.conf*, using a text editor, and add the following lines to it:

```
boot=/dev/msys/fla
compact
install=/boot/boot.b
map=/boot/System.map-2.4.29-rtl-3.1
disk=/dev/msys/fla
bios=0x80

prompt
delay=50
timeout=50
default=rtlinuxdoc

image=/boot/vmlinuz-2.4.29-rtl-3.1
label=rtlinuxdoc
root=/dev/msys/fla1
initrd=/boot/initrd-2.4.29-rtl-3.1.img
read-only

image=/boot/vmlinuz-2.4.29-rtl-3.1
label=rtlinuxhda
root=/dev/hda8
read-only
```

2.11. Create initrd file

An initrd file is an initial ramdisk image. Such a file is needed because the TrueFFS driver is built as a module. This module is needed before the DiskOnChip file system is mounted, which creates a problem because that's where the module is. The initial ramdisk image contains kernel modules which must be

available to get the kernel to boot to the point where the file system can be mounted. The boot process loads the file into memory and the kernel loads modules from this memory area.

Run the **./mkinitrd_doc** script from *linux_binary.5_1_4/doc-linux-5_1_4_20/driver/:*

./mkinitrd_doc

This may give warning messages similar to the following, which can be safely ignored:

```
cp: cannot stat ./sbin/insmod.static.: No such file or directory
cp: cannot stat ./dev/systty.: No such file or directory
```

Check for the newly created initrd image, *initrd-2.4.29-rtl-3.1.img*, in the */boot* directory.
Running the ./**mkinitrd_doc** script produces this image. The reason for making an initrd image is that the provided M-Systems driver cannot be added as a built-in support in the kernel, which leaves no other option than adding it as a loadable module. If we want to boot from DOC, the kernel should know how to access the DOC at the time of booting to search for */sbin/init* in the root filesystem on the DOC (the root filesystem is necessary to get the Linux system up). In the booting sequence of the Linux, */sbin/init* is the file (a command actually) that the kernel looks for in order to start various services and, finally, give the login shell to the user.

2.12. Copy Linux system files to DiskOnChip

This part of the procedure is very distribution specific. A detailed discussion of which files need to be copied to the device is beyond the scope of this application note. Contact your distribution vendor for more information.

The M-System ./**mkdocimg** shell script can be used as a starting point for copying files to the DiskOnChip. Even though Red Hat 7.3 Linux was used, the file redhat-7.1.files provided as well a decent beginning. Even so, the */bin*, */dev*, */etc* and */lib* directories needed to be copied entirely to the bootable file system. One can quickly run out of space on the target device.

If you do use the **./mkdocimg** script, here are some things you'll need to keep in mind:

- It does not copy the initial ramdisk image from */boot* to the DiskOnChip file system.
- */boot/boot.b* is not copied to the DiskOnChip file system.
- */lib* library files were not duplicated on the device.

Because of these issues, pay particular attention to what gets copied over to the bootable file system. Double check everything.

A missing critical file will manifest itself during a boot from the DiskOnChip. The boot will proceed past kernel decompression in some extension and then fail. Error messages may vary greatly, as will the location of the failure. Please see the Linux boot disk HOWTO for more information.

Before starting with this step make sure that you have not mounted */dev/msys/fla1* on any of the mount points, as this step will involve reformatting the DiskOnChip. Also, in order to understand the details of Root File system refer to The Linux Bootdisk How To available at http://www.tldp.org. We will use the ./**mkdocimg** script located in *linux_binary.5_1_4/doc-linux-5_1_4_20/build*. We will also use the redhat-7.1.files directory, located in the same directory (i.e. *build*), which contains the list of the files that will be copied in the root filesystem that will be created on the DOC.

./mkdocimg redhat-7.1.files

This step will take a few minutes to complete.
Now mount the */dev/msys/fla1* partition on the mount point */mnt/*doc and check the files that have been created:

mount -t *auto* /dev/msys/fla1 /mnt/fla1

cd /mnt/fla1

Copy the recently created */etc/lilo.conf* to *mnt/fla1/etc/lilo.conf*.

cp /etc/lilo.conf /mnt/fla1/etc/lilo.conf

The following directories are created on the DOC as a result of running the script:

- */bin*
- */dev*
- */sbin*
- */etc*
- */lib*
- */usr*
- */home*
- */mnt*
- */tmp*
- */var*
- */boot*

The most important is the boot directory. This contains the vmlinuz-2.4.29-rtl-3.1 and initrd-2.4.29-rtl-3.1.img which gets copied from the /boot directory. This directory is required when booting from DiskOnChip.

2.13. Store new lilo configuration

Earlier use of the M-Systems **./mkdocimg** shell script causes problems to pop up at this point. When it runs, **./mkdocimg** creates a LILO configuration file in the bootable file system from a template (making some substitutions to arrive at the final file).

To be safe, make sure that:

1. */etc/lilo.conf* has appropriate path and file names
2. all files referenced in */etc/lilo.conf* exist exactly where they're expected to be in the bootable file system
3. copy */etc/lilo.conf* to the DiskOnChip. Then, rerun LILO as indicated in this section of the installation manual.

This step will configure LILO in the MBR of the DiskOnChip and hence make the DiskOnChip bootable. Ensure that */dev/msys/fla1* (root filesystem partition for the DOC) is mounted on the mount point */mnt/doc*.

Execute the following commands to store the LILO configuration on the DOC:

lilo-v -C /etc/lilo.conf -r /mnt/fla1

cp linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/lilo /mnt/fla1/sbin/lilo

We need to copy the file *boot.b* from *linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/* to . The */mnt/fla1/boot*./mnt/fla1 denotes the location where the BootLoader will be installed, so it is installed on the DiskOnChip, as */mnt/fla1* is the mounting point for the primary partition of DOC where LILO was configured. It will create the following two files in the boot directory of DOC (i.e. */mnt/fla1/boot*):

- **System.map-2.4.29-rtl-3.1**
- **boot.3E00**

Now you should make a backup of the entire DiskOnChip to allow for an easy restore of the files damaged by possible fatal errors:

cd /home

tar -cvzf docimg.tgz /mnt/fla1

This will create a compressed copy of all the contents of DiskOnChip with the name *docimg.tgz* in */home*.

2.14. Update firmware

You must reinstall the firmware to be able to boot from the DiskOnChip. Instead of using the Linux **./dformat** supplied with the driver, use the **dformat** command on the DOS boot floppy you created earlier. Therefore, at this point you will have to shutdown and reboot from the DOS floppy. To prepare the device, issue the command *"dformat /win:xxxx /s:doc514.exb /y /noformat /first"*. Recall that the xxxx in the *"/win:xxxx"* argument refers to the extension window you set in the BIOS.

dformat /WIN:D000 /S:doc514.exb /Y /NOFORMAT /FIRST (in DOS shell).

2.13. Reboot and enjoy

After reinstalling the firmware, remove the DOS floppy and reboot again. You should see a TrueFFS message and an M-Systems copyright; these indicate that the firmware is loaded. Next, the LILO boot menu should appear. If you do not see the LILO boot menu, then LILO probably was not installed in the DiskOnChip boot sector.

Once the LILO menu appears, wait five seconds and Linux will begin booting. First, the kernel uncompresses itself. If the system hangs at this point, potential causes are an incorrect */etc/lilo.conf* file, forgetting to rerun LILO after making changes, and using a LILO which is not DiskOnChip aware. Other causes of failure during boot include a missing configured kernel, a bad initial ramdisk image, missing kernel modules, or missing files in the boot file system.

If the kernel fails to boot, the firmware will need to be removed so that you can troubleshoot and then make repairs. Using the DOS floppy, issue the *"dformat /win:xxxx /s:doc514.exb /y /noformat /empty"* command; this leaves the file system intact so that it can be mounted once Linux is booted.