GESBC-9302 Development With a USB Drive

Paul H. Muller - Documatrix
www.documatrix-usa.com   ulx@linuxmail.org

***Disclaimer:** The programs and procedures described here have been tested and are thought to be accurate but are offered as is, with no warranty or guarantee.*

Introduction

The GESBC-9302 system includes two USB ports and Linux commands to mount and access them. USB "memory stick" drives are available at low cost and can be used in conjunction with the GESBC-9302 to add non-volatile storage to your embedded system. A USB drive can also be used as an inexpensive development platform that allows the user to make permanent changes and additions to the GESBC-9302 file system rapidly and easily.

This article describes how to use a USB drive as a host for the GESBC-9302 file system and how to modify the Redboot loading sequence to load this file system directly from the USB drive. Cross-compiled programs from a Debian development PC can quickly become part of the GESBC-9302 system on power-up without the need to burn a new file image into GESBC-9302 flash.

USB Drives

USB drives are widely available in a variety of memory denominations. I bought the lowest cost "memory stick" I could find and paid $13 for 128 Mbytes. The drive was not even formatted; I could not mount it on any PC to test it until I plugged it into a Win XP machine, which helpfully offered to format it. So if you have or buy a USB memory drive, be sure to format it to *FAT32* as this is the only Windows format Linux can recognize.

Mounting and Formatting USB Drives

Once your USB drive is formatted for the FAT32 file system it can be mounted and tested on the GESBC-9302. To do this, use the following command:

```
mount –t vfat /dev/sda1 /mnt
```

Now you can copy files to the USB drive just as if it were part of the Linux system.

Because we want to use this with the GESBC-9302, it will be more compatible to re-format the USB drive to a standard Linux ext2 file system. To do this, the USB drive must first be partitioned then re-formatted to ext2.

Partitioning and Formatting the USB Drive

In order to place the ext2 file system on the USB drive, the drive must first be re-partitioned. Do this from the GESBC-9302, using the *fdisk* command as follows:

```
fdisk /dev/sda1
```

The *fdisk* program itself has a series of commands used to view or set partitions on the USB drive. Entering *m* will give a list of *fdisk* commands.

The USB drive as formatted with VFAT will have existing partitions that must first be erased. This is done with *fdisk* command *d*. My USB drive apparently had 4 VFAT partitions, so each one had to be deleted in turn. *fdisk* gave a response each time the *d* command was used.

After deleting the VFAT partitions, I ran the *p* command to verify the state of the USB drive. The report printed by *fdisk* looked like the following:

```
Disk /dev/sda1: 126 MB, 126959616
4 Heads, 61 sectors/track, 1016 cylinders
Units = cylinders of 244 * 512 = 124928
```

The next task is to add a partition with *fdisk* command ***n***:

fdisk requests that you specify a primary or an extended partition *p* or *e*.   Choose ***p*** for primary. *fdisk* next asked for the starting cylinder; I entered **1**. *fdisk* then asked for the last cylinder and I entered **1016**.

A quick check using the *p* command verified that the entire USB drive was to be a single partition.

The last step was to use the *fdisk* ***w*** command to write the new partition to the drive. After a few seconds the process was complete and *fdisk* terminated.

Creating the ext2 File System on the USB Drive Using the GESBC-9302

After the USB drive is partitioned, we must create a Linux ext2 file system on it. To create the ext2 file system, use the following command:

mke2fs /dev/sda1

The screenshot below shows the GESBC-9302 response:

```
# mke2fs /dev/sda1
mke2fs 1.34 (25-Jul-2003)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
31104 inodes, 123984 blocks
6199 blocks (5.00%) reserved for the super user
First data block=1
16 block groups
8192 blocks per group, 8192 fragments per group
1944 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 29 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
~ # mount /dev/sda1 /mnt
~ # cd /mnt
/mnt # ls
lost+found
/mnt #
```

Now that the USB drive is operational, we can put a copy of the GESBC-9302 file system on the USB drive. In a previous article "Getting Started with the GESBC-9302" a Debian development system was described. We can use this system in conjunction with the Glomation CD to capture a copy of the GESBC-9302 file system into the USB drive.

Putting the GESBC-9302 File System on the USB Drive

The first step is to copy the file *ramdisk.gz* from the Glomation CD to the */home/arm* directory on the Debian development PC. You can find *ramdisk.gz* on the CD at */Linux/GESBC-9302/.* To unzip the file, use the following command:

```
gzip –d ramdisk.gz
```

There should now be a file named *ramdisk* in the Debian */home/arm* directory.

The next step is to power up the GESBC-9302 and mount the USB drive:

```
mount /dev/sda1 /mnt
```

Next, we make a special directory on the USB drive to hold the *ramdisk* file. I called this directory *armdisk*, but you can use something better:

```
mkdir /mnt/armdisk
```

From the top directory in the GESBC-9302, use the *ftpget* command to copy the *ramdisk* file from the Debian system at IP 192.168.0.125 (note user name *arm* and password *gesbc* must exist on the Debian system):

```
ftpget –u arm –p gesbc 192.168.0.125 ramdisk /mnt/armdisk/ramdisk
```

Now a copy of the *ramdisk* file should reside on the USB drive at */mnt/armdisk.*

Next, we mount the ramdisk file as a loop device:

```
mount –o loop /mnt/armdisk/ramdisk /mnt/armdisk/
```

Next, *cd* to */mnt/armdisk* and view the contents. You should see a complete copy of the GESBC-9302 file system.

Now (from */mnt/armdisk/*) we simply copy the loop-mounted file system into the USB drive at */mnt*:

```
cp –ra * /mnt/
```

And now we have the complete GESBC-9302 file system permanently copied to the USB drive. You can change or add files and they will remain in the USB drive after power-down.

Note: Be sure to *umount* the USB drive prior to powering down.

Booting from the USB Drive

Now that the entire GESBC-9302 file system exists on the USB drive, we can interrupt the normal Redboot process and direct the GESBC-9302 to load its file system from its USB drive. To do this, we power the board down and re-power with the GESBC-9302 connected to Hyper-Terminal.

To interrupt the Redboot loading sequence, press ^C (Control C). To manually complete the loading process, first load the Linux zimage:

```
fis load zimage
```

Next, load the file system on the USB drive:

```
exec –c “root=/dev/sda1 console=ttyAM0”
```

Don’t forget to include the quotes. If successful, the GESBC-9302 will load normally and you can activate the console as usual.

Modifying the Redboot Script

We can test the file system installed on the USB drive by interrupting the Redboot process with ^C on startup. The Redboot script can be modified, however, so that the GESBC-9302 always comes up using the USB drive file system.

The Redboot loader has an extensive series of commands, documented at www.ecoscentric.com. To view the Redboot default script for the GESBC-9302, capture the text that scrolls by on Hyper-Terminal as the system comes up.

The boot-up script can be seen in the lines starting with *Redboot >,* as shown below:

```
RedBoot> fis load ramdisk
RedBoot> fis load zImage
RedBoot> exec -r 0x800000 -s 0x300000
```

In the default script, Redboot first loads the ramdisk image from the GESBC-9302 flash, then the Linux image. The script finally executes Linux starting at a ram address 0x800000 to begin the Linux booting process.

Below is a screenshot of the normal GESBC-9302 booting process showing the Redboot scripting:

```
--------  Normal GESBC-9302 Boot-up Sequence  -----------------------

+Ethernet eth0: MAC address 00:00:00:00:50:03
IP: 192.168.0.111/255.255.255.0, Gateway: 192.168.90.1
Default server: 0.0.0.0, DNS server IP: 0.0.0.0

RedBoot(tm) bootstrap and debug environment [ROMRAM]
Non-certified release, version v2_0 - built 20:59:09, Nov 10 2005

Platform: Cirrus Logic EDB9302 Board (ARM920T) Rev A
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x00041e48-0x01fdd000 available
FLASH: 0x60000000 - 0x61000000, 128 blocks of 0x00020000 bytes each.
== Executing boot script in 1.000 seconds - enter ^C to abort
RedBoot> fis load ramdisk
RedBoot> fis load zImage
RedBoot> exec -r 0x800000 -s 0x300000
Using base address 0x00080000 and length 0x000b3eb0
Uncompressing Linux.................................................. done,
```

The script can be edited so that the file system is loaded from the USB drive instead of from the GESBC-9302 flash. The Redboot commands will be the same as those used when we interrupted the process with ^C:

fis load zimage
exec –c “root=/dev/sda1 console=ttyAM0”

To edit the Redboot script, first interrupt the boot process with ^C on power-up. To view the existing *fconfig* setup using the following command:

fconfig –l

This shows all of the default Redboot parameters, including the script. We don’t want to change anything but the scripting, so listing and saving the default configuration is a good idea in case something goes wrong.

The –*l* (list) option displays the entire default configuration, as shown in the screenshot below:

```
== Executing boot script in 1.000 seconds - enter ^C to abort
^C
RedBoot> fconfig -l
Run script at boot: true
Boot script:
.. fis load ramdisk
.. fis load zImage
.. exec -r 0x800000 -s 0x300000

Boot script timeout (1000ms resolution): 1
Use BOOTP for network configuration: false
Gateway IP address: 192.168.90.1
Local IP address: 192.168.0.111
Local IP address mask: 0.0.0.0
Default server IP address: 0.0.0.0
DNS server IP address: 0.0.0.0
Set eth0 network hardware address [MAC]: true
eth0 network hardware address [MAC]:
0x00:0x00:0x00:0x00:0x50:0x03
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
```

To change the configuration, enter **fconfig.** You will be asked a series of questions by Redboot covering all of the elements in the configuration. The existing configuration is the default value for any choice, and can be kept by simply pressing the Enter key.

To change the script, enter **true** to the *Run script at boot* query. Enter each line of the new script and then end the script with an empty line. You will be prompted to change the other configuration parameters, but press **Enter** each time to accept the existing default value.

The sequence is shown in the following screenshot:

```
RedBoot> fconfig
Run script at boot: true
Boot script:
.. fis load ramdisk
.. fis load zImage
.. exec -r 0x800000 -s 0x300000
Enter script, terminate with empty line
>> fis load zImage
>> exec -c "root=/dev/sda1 console=ttyAM0"
>>
Boot script timeout (1000ms resolution): 1
Use BOOTP for network configuration: false
Gateway IP address: 192.168.90.1
Local IP address: 192.168.0.111
Local IP address mask:
Default server IP address:
DNS server IP address:
Set eth0 network hardware address [MAC]: true
eth0 network hardware address [MAC]: 0x00:0x00:0x00:0x00:0x50:0x03
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0x60fc0000-0x60fc1000: .
... Program from 0x01fde000-0x01fdf000 at 0x60fc0000: .
```

When *fconfig* is finished you will be asked to update the new configuration in flash. Enter **y**. When Redboot finishes writing to flash, power-down the GESBC-9302 and power-up again, observing the process in Hyper-Terminal.

If successful, Linux will boot and the file system will now be located on the USB drive instead of the ramdisk image. Now changes and additions will stay in the non-volatile memory of the USB drive during the development of your programs and custom GESBC-9302 configuration.

Booting to a File System via NFS

In a previous article “A Simple File Development System for the GESBC-9302” a method of mounting the file system via NFS was described. The GESBC-9302 file system can be captured to the hard drive of a Debian development PC on the LAN so that changes and additions can be tested without having to first copy them to the GESBC-9302.

You can modify the Redboot script so that the GESBC-9302 always looks for the file system on the Debian PC, using the following script:

```
fis load zimage
exec –c “root=/dev/nfs nfsroot=192.168.0.125:/armfiles ip=192.168.0.127 console=ttyAM0”
```

In this example, the Debian PC is at IP 192.168.0.125 and the directory holding the GESBC-9302 file system is */armfiles*. The GESBC-9302 is assigned IP address 192.168.0.127.

Unmounting the File System

When powering down the GESBC-9302, be sure to unmount all file systems!

The USB drive file system could become corrupted from powering the GESBC-9302 Down without first unmounting. Similarly, an NFS mounted file system must be unmounted from the GESBC-9302 prior to power down. The normal GESBC-9302 ramdisk file system does not need to be unmounted; it is simply replaced at the next boot-up. But any Linux system with mounted file systems must unmount them prior to shutdown to avoid file corruption.

To unmount all file systems use the following command:

```
    umount -a
```

Changing Back to a Flash File System Image

To return the GESBC-9302 to the default file image from flash, simply re-edit the script, entering the three original lines, as shown below.

```
.. fis load ramdisk
.. fis load zImage
.. exec -r 0x800000 -s 0x300000
```