Getting Started with the GESBC-9302

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

Introduction

The GESBC-9302 single-board computer from Glomation is an ideal platform for network-based embedded computing. It is a cost-effective ARM SBC that includes Ethernet, USB, serial ports, real-time clock, digital I/O, analog I/O and the Linux operating system. This article describes how to get started using the GESBC-9302 with a Debian Linux development system and cross-compiler.

Setting Up a Development System

The GESBC-9302 is a small, embedded SBC and requires a Linux development platform to create or modify application software. The development platform must support the ARM cross-compiler. This lets you write your C code in a text-based editor on your desktop PC and compile it for the target ARM processor used on the GESBC-9302. You can also modify the contents of the default GESBC-9302 system and re-compile the kernel, if needed.

Selecting the Development System

The Linux used on the GESBC-9302 is a derivative of Debian Linux, so it makes sense to use Debian for your development system. The latest version of Debian can be obtained from www.debian.org and version 3.1, nicknamed "Sarge" is the most current stable release. I found Debian very straightforward to install and easy to modify.

Preparing Your Desktop PC as a GESBC-9302 Development System

I used an old Pentium II desktop PC for my Debian development platform. I have Win98 on it, so I created a new partition to hold the Debian Sarge system. To do this I used the Ranish Partition Manager after first de-fragging the PC hard drive. Ranish (http://www.ranish.com) is fairly easy to use, but as always, use extreme caution when re-partitioning your hard drive. It is like doing brain surgery; one wrong move and you could lose everything on the PC!

You have to calculate the size of the new partition you want to create, and translate that into starting and ending cylinders on the hard drive. Not difficult, but be sure to read up on it if you have never partitioned a hard drive before. I made a 1 Gbyte partition at the end of the hard drive for Debian Linux, and none of my existing Win98 files were disturbed.

Installing Debian

Installing Debian Sarge is really quite easy. You download a stripped-down installation version (about 50 Mbytes) that is burned onto a CD. You boot your PC from this CD and the Debian install program detects all of the peripherals on your PC and locates the available partitions in the hard drive. I simply told Debian to occupy the last partition in the hard drive, and it installed without incident.

The stripped-down Debian install version runs in a text command-line mode, but you can add packages from the Debian website and mirrors. This is really amazing; simply launch the Aptitude package manager, find the package you want to install, Debian goes out on the Internet, finds the mirror site, and automatically installs the package.

Be Sure Your Development System PC Is Connected to the Internet!

That way, packages are easy to find and install. You can browse the list of available stable packages at http://packages.debian.org/stable

Here are the packages I added to my Debian development platform to use with the GESBC-9302:

Wu-ftpd - FTP server. Lets you easily move files to and from the GESBC-9302 over your LAN

Telnetd – Telnet server. Allows you to log into the GESBC or your development system from anywhere.

Bzip2 – Compression utility. You need this to install the GESBC-9302 cross-compiler.

Midnite Commander – A system navigation tool. Makes it easy to find subdirectories and files in the big Debian development system

I also added a web server (thttpd), some text editors (pico) and the GCC compiler so I could test C programs within the development system prior to cross-compiling for the ARM processor on the GESBC-9302. The great thing about Debian is that you can easily add or remove packages any time.

Installing the Cross-Compiler

My Debian development system is on a Pentium PC, so any programs I compile for the GESBC-9302 must be cross-compiled for the ARM processor. Glomation provides a cross-compiler for the ARM processor on the CD that comes with the GESBC-9302. The installation instructions for putting the cross-compiler onto the Debian system are on page 21 of the GESBC-9302 instruction manual.

For some reason my Debian system had a problem finding the cross-compiler file on the CD. I eventually used Midnite Commander to find the file on the CD and I copied it to the Debian hard drive. I decompressed it and untarred, but it came out installed to */arm-linux-gcc-3.3/usr/local/arm/3.3...* which is the wrong sub-directory. So I copied everything under the *…/3.3* subdirectory to */usr/local/arm/* and the cross-compiler worked fine.

Finishing up the Development System

To make it easy to use with the GESBC-9302, I customized my Debian development system as follows:

1. ***Changed default IP address.*** Added two new lines to the /etc/rcS.d/*S55bootmisc.sh* file.

   /sbin/ifconfig eth0 192.168.0.125
   /sbin/route add default gw 192.168.0.1

2. **Added New User**, ***arm***. Do this with the *adduser* command. This creates a */home/arm* subdirectory that can be used for development and ftp movement of files to and from the GESBC-9302.

3. **Create Compile Script –** The cross-compiler command (*arm-linux-gcc file file*) is long and buried pretty deep into the */usr* subdirectory. I made a short script that executes the cross-compiler from the */home/arm* subdirectory. The script looks like this:

   /usr/local/arm/3.3/bin/arm-linux-gcc /home/arm/try1.c

   I named this script *armm* changed the permissions to include execute, and it calls the cross-compiler from the */home/arm* subdirectory. I simply use the same file name for the C program I am compiling, *try1.c*, and the compiler deposits an *a.out* ARM binary executable file to the same subdirectory.

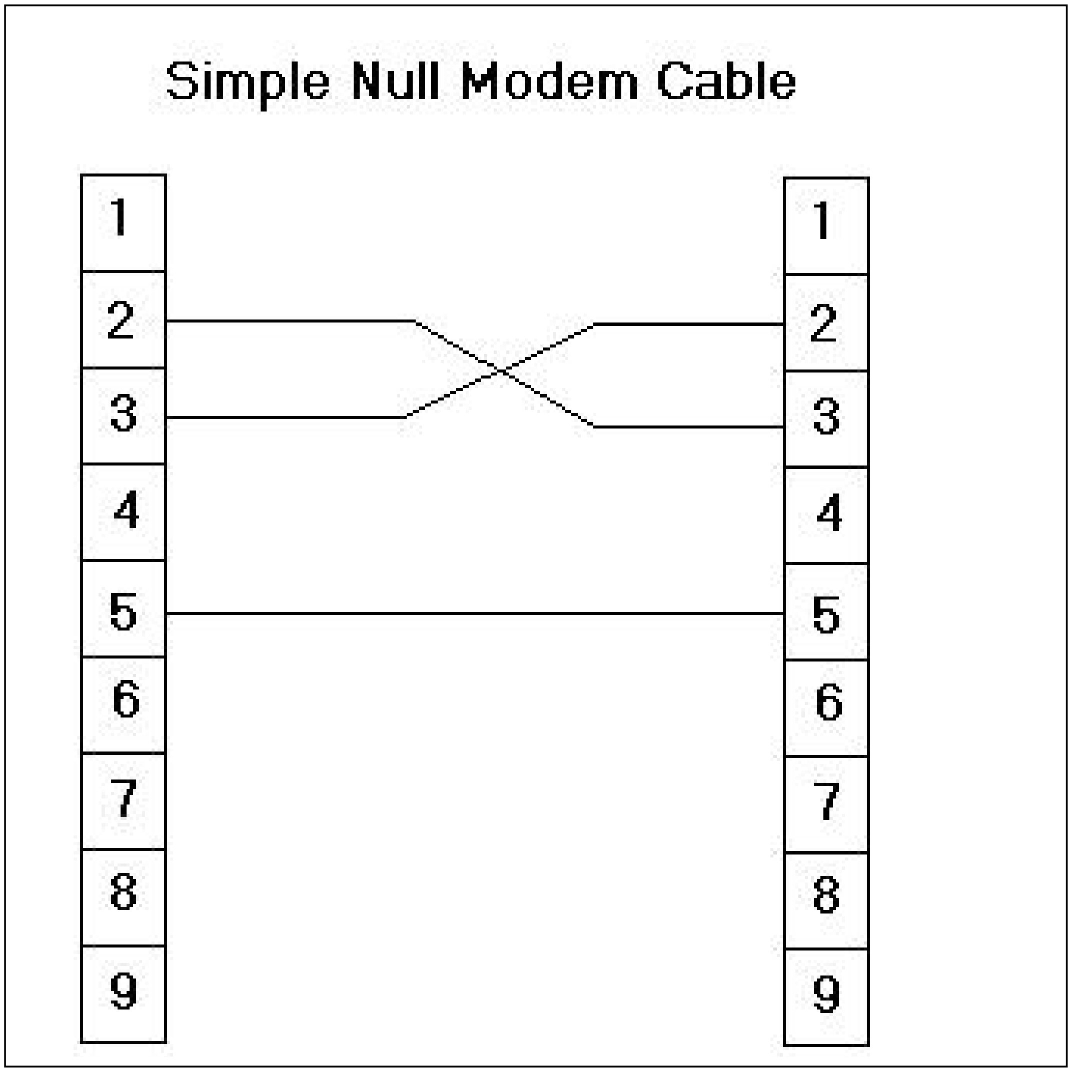
4. **Port Forwarding on Router**. Telenet, browser and ftp ports now can go from my router to 192.168.0.125 (the Debian development system). That way I can log into the development system from anywhere using my static IP Internet address. I can create a program in a text editor, cross-compile it for the GESBC-9302, telnet into the GESBC-9302 and transfer the new files over, and execute them. I can do this from anywhere in the world.

Running the GESBC-9302

With a Debian development system in place, I was ready to use the GESBC-9302. To make sure the GESBC-9302 boots correctly, I followed the instructions on page 17 of the GESBC-9302 instruction manual.

Null Modem Cable

I first made up a null-modem cable from two 9-pin female D connectors and a three-wire cable, as shown below.



This worked exactly as described in the instruction manual, even at a setting of 57,600 baud.

I next booted my development PC into Win98 and started up Windows Hyper-Terminal. I configured the settings for Direct to Com1, 57,600 baud, 8.N,1. I turned on the GESBC-9302 and after a few seconds had the Linux messages scrolling through Hyper-Terminal, just as shown on page 8 of the instruction manual.

I pressed Enter and the console was activated inside the GESBC-9302. I then changed the static IP address of the GESBC-9302 with the following command:

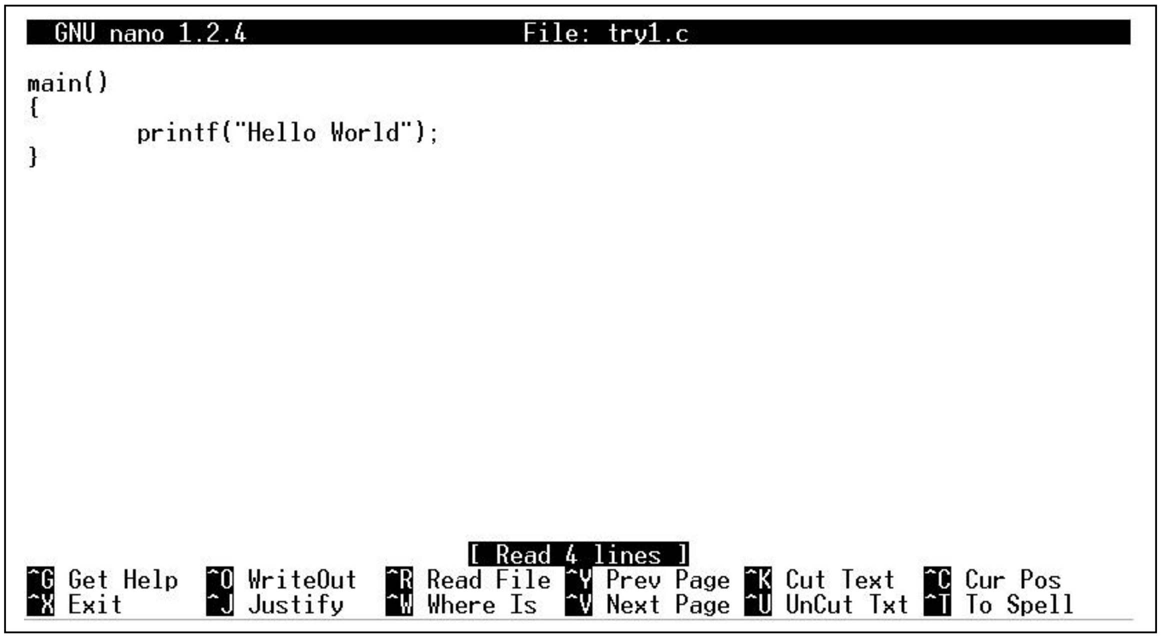
Ifconfig eth0 192.168.0.127

This put the GESBC in the same LAN class C network as the Debian system. I closed down Hyper-Terminal on my desktop PC and re-booted into the Debian development system, logging in as ***arm***. I pinged 192.168.0.127, just to verify that both systems were in communication over the LAN. No problems.

Now I was ready to try some cross-compiling!

Cross-Compiling "Hello World"

From my Debian development system, still logged in as *arm*, I launched the text editor *pico*, and wrote out the classic beginning C program, as shown below. Note the name of the file is *try1.c*. This is so I can use my *armm* script to quickly cross-compile it.



```
  GNU nano 1.2.4                    File: try1.c

main()
{
        printf("Hello World");
}
```

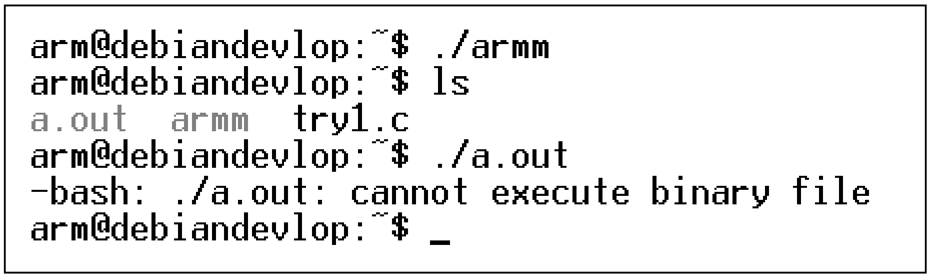
[Read 4 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit   ^J Justify   ^W Where Is ^V Next Page ^U UnCut Txt ^T To Spell

I saved the file using *^O* then exited the pico editor using ^*X*. The "Hello World" source file now existed as a text file named *try1.c*

I now activated the cross-compiler by using the *armm* script:

./armm

The result is that the compiler created an ARM-executable binary named *a.out* in the subdirectory. Just for fun, I tried executing the a.out file, but the Debian development system can't run it; it is not a Pentium compatible binary. So it is apparently ready for the ARM processor in the target GESBC-9302 . This sequence is illustrated below.



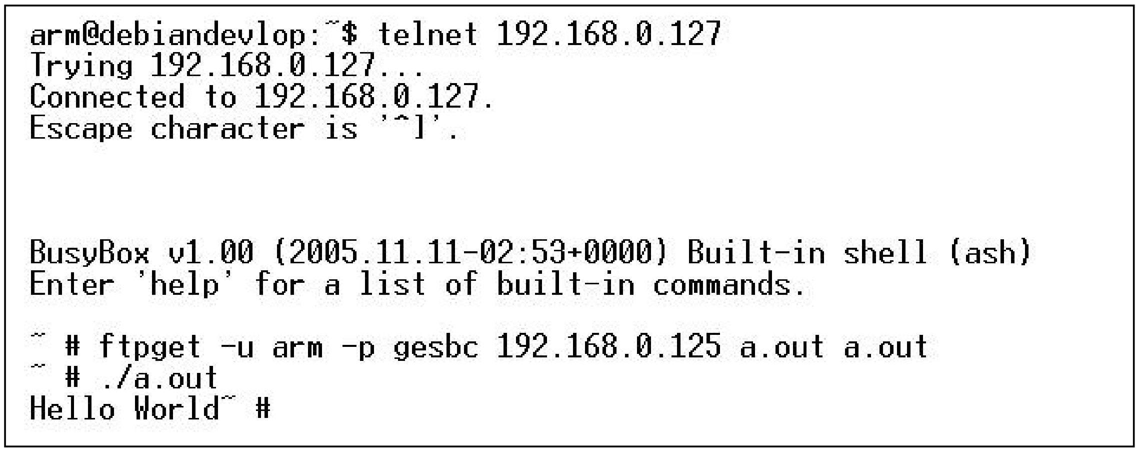
```
arm@debiandevlop:~$ ./armm
arm@debiandevlop:~$ ls
a.out  armm  try1.c
arm@debiandevlop:~$ ./a.out
-bash: ./a.out: cannot execute binary file
arm@debiandevlop:~$ _
```

Moving the File to The GESBC-9302

So a binary file has just been created, but it must be moved to the GESBC-9302 to see if it will run on the target ARM processor. Here is a clever way to do this over the LAN:

1. Telnet into the GESBC-9302 at IP address 192.168.0.127
2. Use the *ftpget* command from the GESBC-9302 to retrieve the newly compiled file. Note the password for user *arm* is *gesbc*.
3. Run the binary on the GESBC-9302.

This sequence is shown in the following:



```
arm@debiandevlop:~$ telnet 192.168.0.127
Trying 192.168.0.127...
Connected to 192.168.0.127.
Escape character is '^]'.



BusyBox v1.00 (2005.11.11-02:53+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # ftpget -u arm -p gesbc 192.168.0.125 a.out a.out
~ # ./a.out
Hello World~ #
```

It worked!

The file was transferred from the Debian development system into the GESBC-9302 top directory. The *./a.out* command executed the file in the ARM processor target system, and it printed out "Hello World".

Saving the Executable Program

Once the GESBC is powered down, the **a.out** file will be lost. In order to save files locally, I installed a USB thumb drive from my Win98 PC into the GESBC-9302 one of the USB ports. You can easily mount this with the following command:

```
mount -t vfat /dev/sda1 /mnt
```

I have set up a subdirectory named */sbc9302* on the USB drive, and keep files there for later use. The command to copy *a.out* and save it as a more convenient name (*try1.run)* is shown below:

```
cp a.out /mnt/sbc9302/try1.run
```

Next time the GESBC-9302 is powered up, you can retrieve the saved files from the thumbdrive just by mounting it and copying what you need back into the system.

Exiting from the GESBC-9302

To exit the GESBC from telnet, type in the escape character *"]"*, then type "*quit*". You will be back in the Debian development system.