Create a Mini PC or Server with Olimex's Olinuxino A13/A13Micro

STEP-BY-STEP INSTRUCTIONS ON HOW TO CREATE A PERSONAL COMPUTER OR EVEN A SMALL SERVER WITH THIS GREAT EMBEDDED SYSTEM FOR LESS THAN $80 US!

RONALD KURNIAWAN

Olimex is a Bulgarian company known for its innovative hobbyist products. It has a wide array of microcontroller-based products, ranging from the small Arduino clones to the very able system that has the Allwinner A13 microcontroller as its brain. In this article, I describe how you can create a working Linux system for the Olinuxino A13 and Olinuxino A13Micro from scratch.

Let's begin by obtaining and compiling the kernel, creating the U-Boot system, preparing the root filesystem and getting the necessary packages to create a comfortable minimal computing environment. At the end of this article, I also explain how to install a compact desktop environment.

I am using Ubuntu 12.04 (Precise Pangolin) for my build system. Any Debian-based system users should be able to follow the instructions in this article with relative ease. Before you begin, you should create a directory under your home directory to contain all your work. I am going to call mine "A13System".

What Are Arm and eabihf?

As you progress further into the article, you will encounter the terms Arm and eabihf more than once. Let me clarify those terms in order to avoid confusion with other terms that you might encounter if you decide to go further into the world of cross-compilation.

Arm is a general name for a family of microcontroller architectures designed by ARM Holdings, a British company. You can find Arm microcontrollers inside most portable modern gadgets, ranging from mobile phones, Nintendo DS portable game consoles to Apple iPhone and Apple TV. ARM Holdings does not manufacture these microcontrollers; rather, it licenses the designs to other companies. These companies then add their own "secret recipes" into the designs and then manufacture and sell the finished microcontrollers. This is why there are so many variants of Arm architecture and so many companies that produce Arm microcontrollers.

EABI stands for Embedded Application Binary Interface. It specifies the low-level conventions for embedded software application. When it comes to Arm microcontrollers, they come in many sizes, ranging from very small to large. The smaller variants don't have the necessary memory or power to process floating-point computation on the hardware itself, thus making it

necessary to do it by software. These variants are called Arm soft float. There are other variants that can process the floating-point calculation by hardware, like vector floating point (vfp). These two EABIs (soft float and vfp) are what is usually known as *armel*. A newer EABI that targets the higher end of Arm microcontrollers with more efficient floating-point instructions than vfp is called hard float, thus *armhf*.

Olimex's A13WiFI, A13 and A13Micro boards are powered by the Allwinner A13 Arm microcontroller, which are based on ARMv7 design. ARMv7, as with the newer ARMv8, fully supports armhf.

Prerequisites

You can find all the necessary URLs for file downloads and other information in the Resources section of this article. You need to install the following programs before you can commence the build process:

- build-essential.
- gcc-4.6-arm-linux-gnueabihf (the version might vary from one distro to another; the latest I came across at the time of this writing is version 4.7).
- ncurses-dev.
- uboot-mkimage.
- git.
- debootstrap.
- debian-archive-keyring (if you decide later that you want to use Debian rootfs).
- qemu-user-static.

Once you finish installing the prerequisites, you then need to create several softlinks in the same directory where gcc-4.6-arm-linux-gnueabihf is installed (in my case, it is located in /usr/bin). Use the `which` command to find the installation directory:

```
$ which arm-linux-gnueabi-gcc-4.6
```

Next, create softlinks for arm-linux-gnueabihf-gcc-4.6, arm-linux-gnueabihf-gcov-4.6 and arm-linux-gnueabihf-cpp-4.6:

```
$ sudo ln -s /usr/bin/arm-linux-gnueabihf-gcc-4.6 \
 /usr/bin/arm-linux-gnueabihf-gcc
$ sudo ln -s \
 /usr/bin/arm-linux-gnueabihf-gcov-4.6 \
 /usr/bin/arm-linux-gnueabihf-gcov
$ sudo ln -s /usr/bin/arm-linux-gnueabihf-cpp-4.6 \
 /usr/bin/arm-linux-gnueabihf-cpp
```

Preparing the Kernel and U-Boot

The good people at Linux Sunxi are kind enough to share the kernel and U-Boot code tailored to run on Allwinner chips. You have the option of getting and compiling version 3.0 or 3.4 of the Linux kernel. The compilation procedures are similar. For the purpose of this article, I am using kernel version 3.4. Get the kernel and U-Boot source from Linux Sunxi's GitHub repository:

```
$ git clone -b sunxi-3.4
 https://github.com/linux-sunxi/linux-sunxi.git
$ git clone -b sunxi
 https://github.com/linux-sunxi/u-boot-sunxi.git
```

Let's compile U-Boot first. Depending on the target system (A13 or A13Micro), go to the U-Boot directory and issue the following command:

```
$ make a13-olinuxino \
 CROSS_COMPILE=arm-linux-gnueabihf-
```

or:

```
$ make a13-olinuxinom \
 CROSS_COMPILE=arm-linux-gnueabihf-
```

Note: the dash (-) at the end of the commands are not typos. After the make process finishes, if everything goes correctly, you should end up with u-boot.bin and spl/sunxi-spl.bin.

Go to the kernel source directory. Check the configuration directory ($KERNEL_DIR/arch/arm/configs) for the A13 configuration file (a13_defconfig) or A13Micro (a13om_defconfig). If you do not have the configuration file for A13Micro (which is usually the case), you can find the download URL in the Resources section.

Now you need to check the configuration file for a specific line. I learned the hard way that without this line, the compilation will fail. Add the following line to your configuration file if it does not exist or uncomment it:

```
CONFIG_GPIOLIB=y
```

Once again, depending on the target system, issue one of these sets of commands to compile the kernel source:

```
$ make ARCH=arm a13_defconfig
$ make menuconfig
```

or:

```
$ make ARCH=arm a13om_defconfig
$ make menuconfig
```

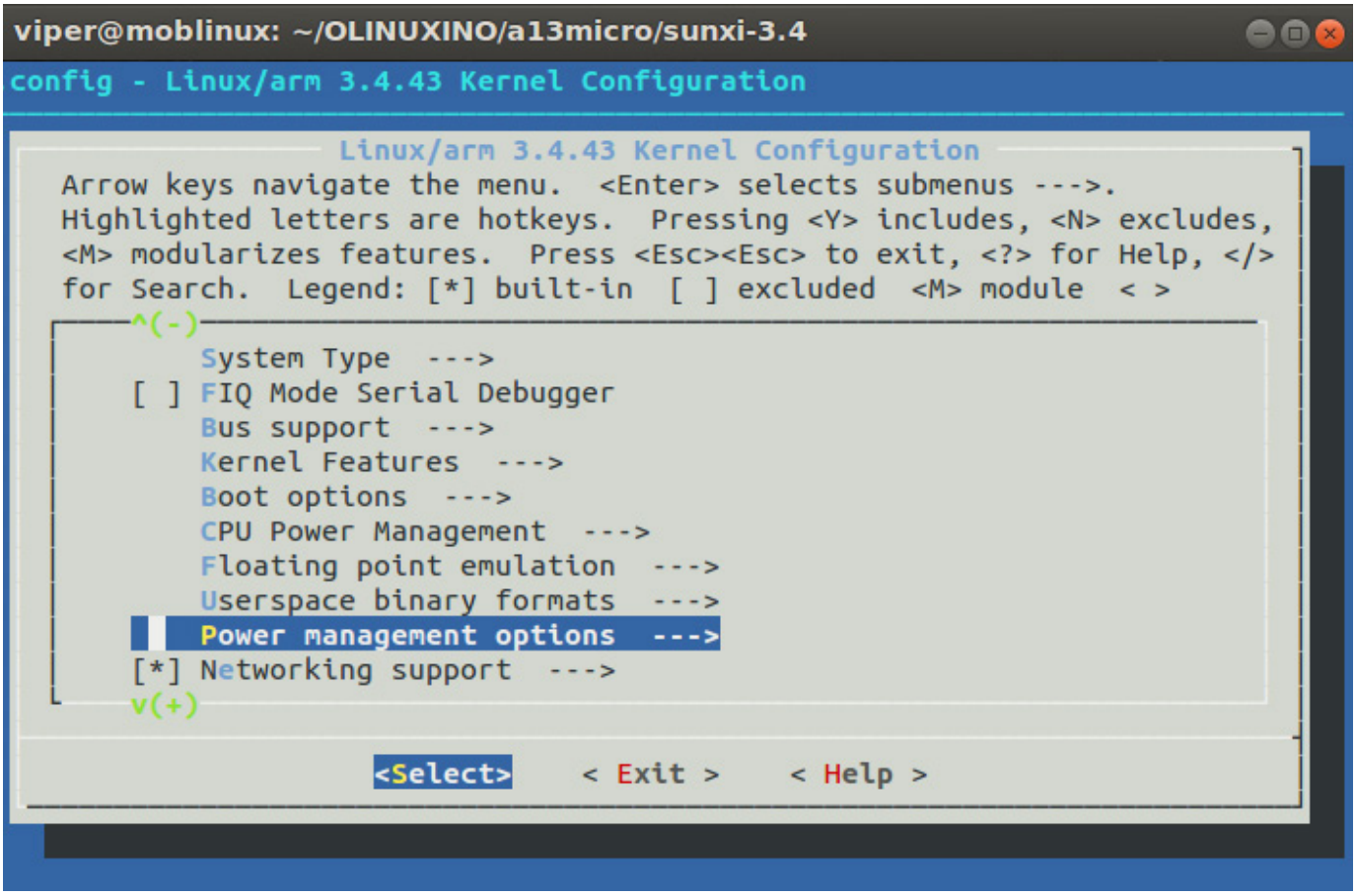

Figure 1. Selecting Power Management Options

The last step allows you to customize your kernel. In order to avoid a long and painful debugging process, always make sure you are able to compile the minimal kernel (that means compiling without any options added) successfully first. Once you succeed, you can add more options to your kernel. Note any options you add to the kernel configuration, as it will aid you in figuring out which feature(s) does not work.

There is a special step you should adhere to if you are compiling the kernel for A13Micro boards. You need to remove the option to "Suspend to RAM and standby", which is located under "Power Management options". A13Micro boards do not support this option.

If you are planning to use any of the board's GPIO pins, make sure that you select "An ugly sun4i gpio driver" option under "Device Drivers" and "Misc devices".

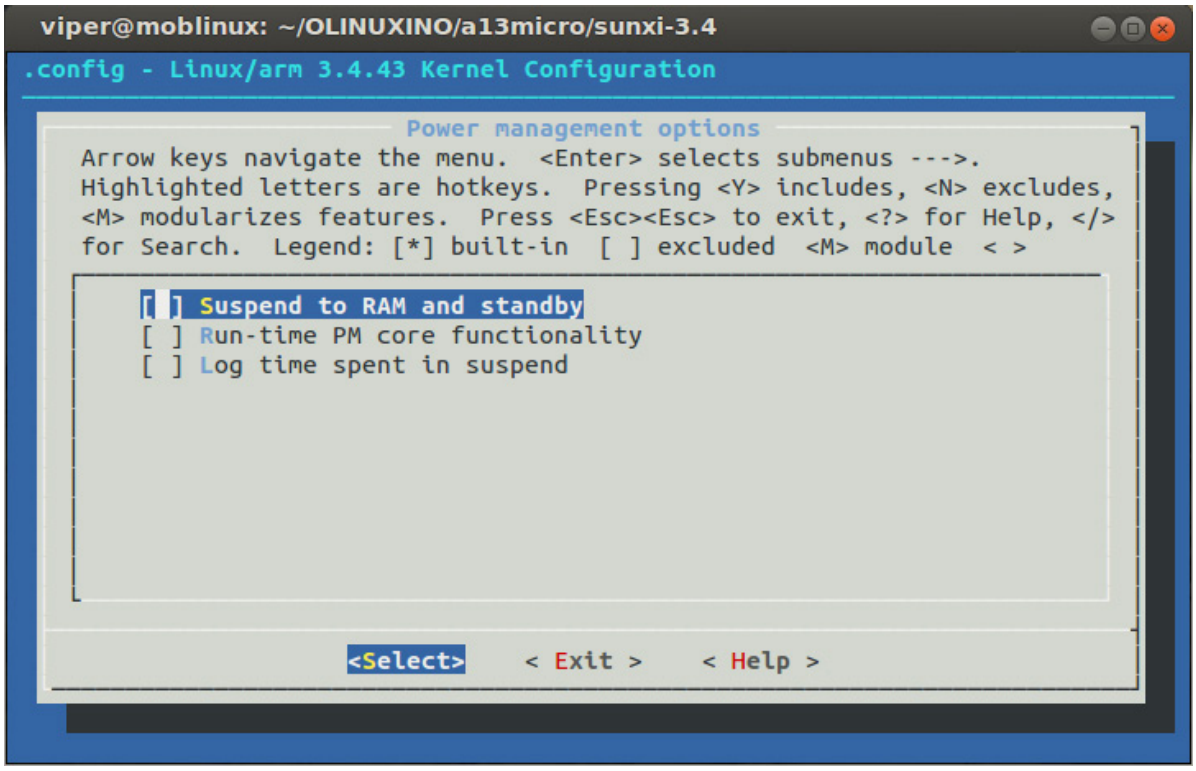

Figure 2. Uncheck Suspend to RAM and Standby

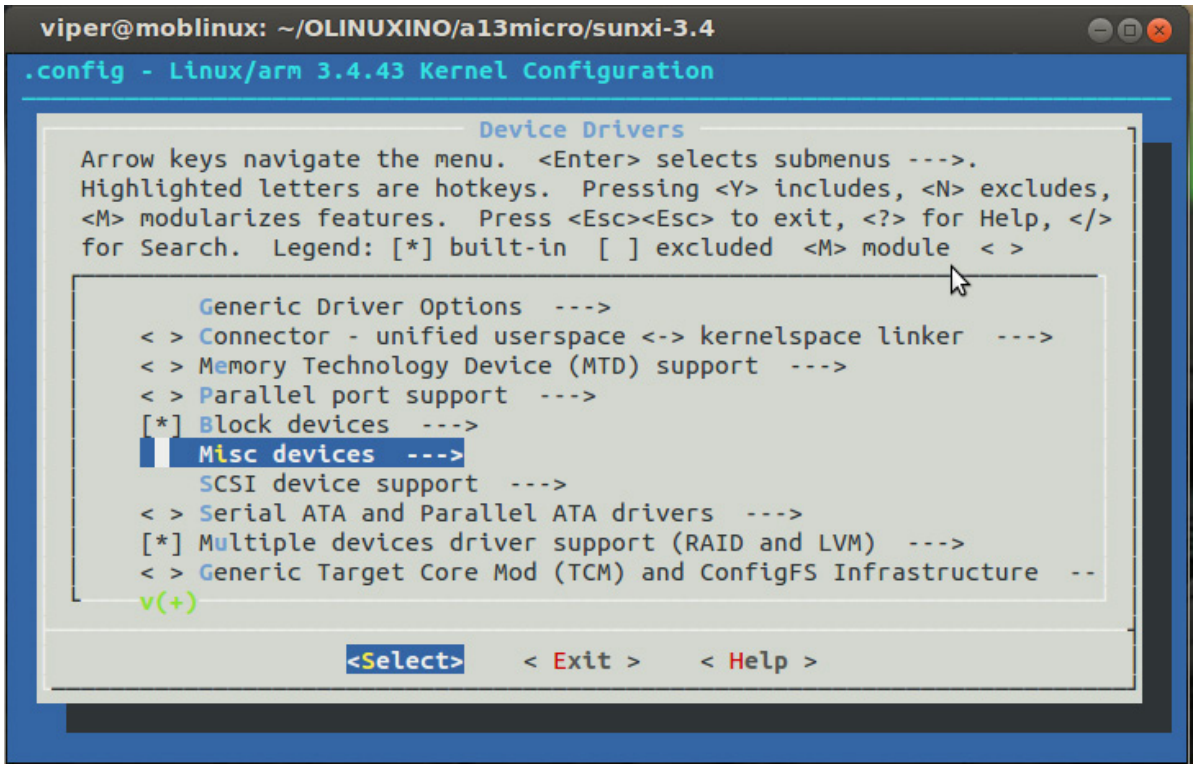

Figure 3. Selecting Misc Devices under Device Drivers

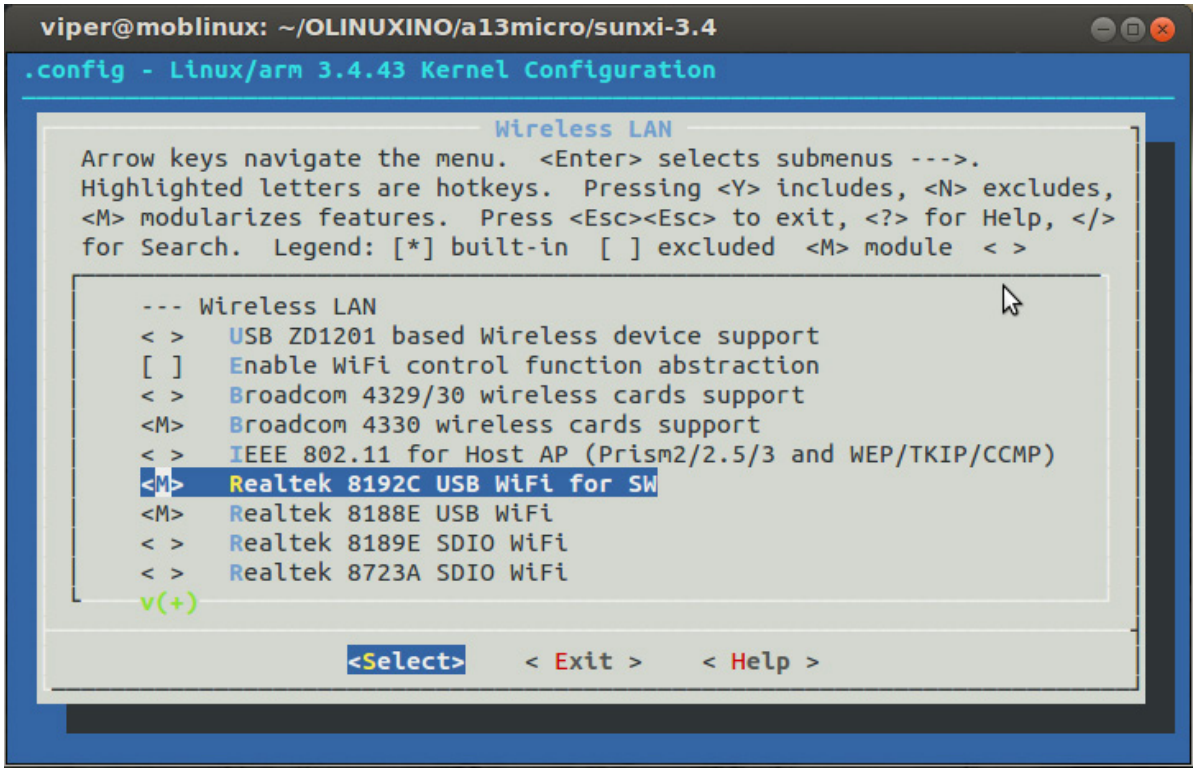

Figure 4. Realtek 8192C Driver as a Module

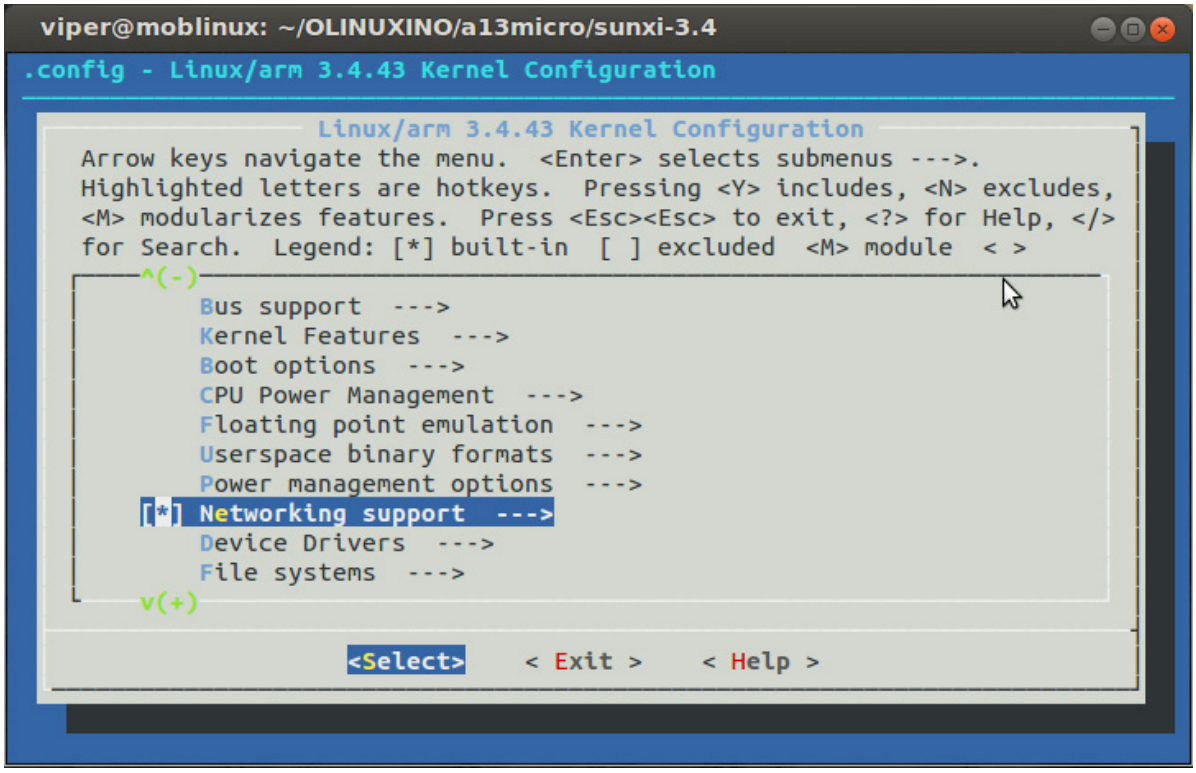

Figure 5. Networking Support Option

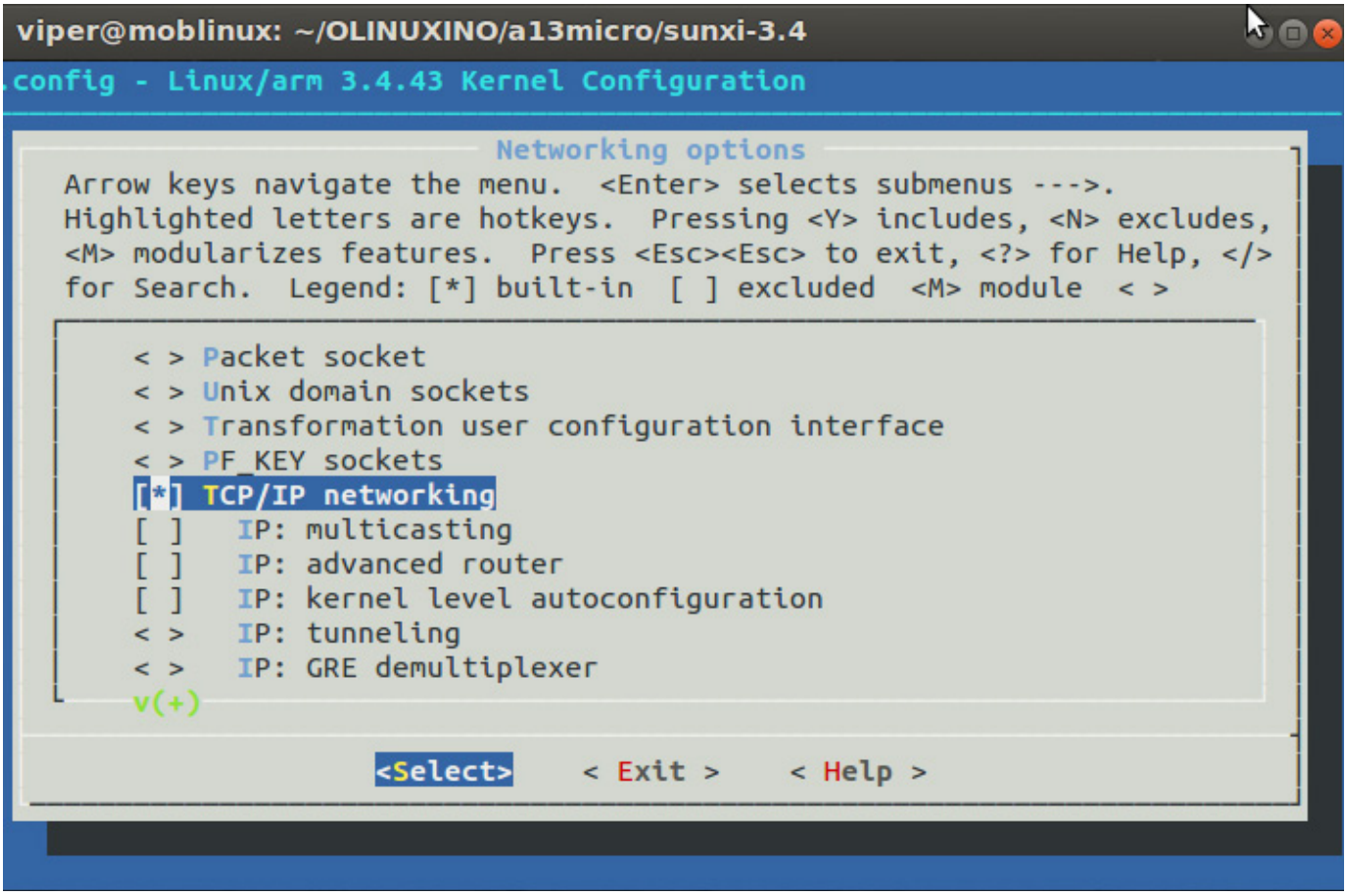

Figure 6. Make sure that TCP/IP networking is selected.

I also plan to use Olimex's MOD-WIFI USB Stick Module[2] that adds the Wi-Fi capabilities to the board. To do that, you need to include the driver for the module, which is called "Realtek 8192C USB Wifi for SW" and is located under "Device Drivers/Network device support/Wireless LAN". You are welcome to experiment with other Wi-Fi devices. I can vouch that I successfully run the A13 board with the Netgear WG111v2 USB Wifi Stick module. The driver I used for this Wi-Fi device was "Realtek 8187 and 8187B USB support".

You also have to make sure that TCP/IP is selected and included in the kernel. Tick Networking Support and press Enter to select it.

Go inside Networking options and make sure that TCP/IP networking is selected.

Once you are satisfied with your configuration, save it and go back to the command prompt. Issue the

following commands to compile the kernel and build the drivers:

```
$ make ARCH=arm \
 CROSS_COMPILE=arm-linux-gnueabihf- uImage
$ make ARCH=arm \
 CROSS_COMPILE=arm-linux-gnueabihf- \
 INSTALL_MOD_PATH=out modules
$ make ARCH=arm \
 CROSS_COMPILE=arm-linux-gnueabihf- \
 INSTALL_MOD_PATH=out modules_install
```

When the compilation finishes, you will end up with the kernel image in $KERNEL_DIR/arch/arm/boot/uImage and the modules and drivers in $KERNEL_DIR/out/lib/modules/$KERNEL_VERSION.

The next step is to prepare a minimal filesystem for your board. The easiest option I've found so far is by using the root filesystem from the Debian project or Ubuntu, as both distributions provide armhf binaries for the essential applications. I explain how to prepare both options next.

Preparing the Filesystem: Debian Wheezy

Start by creating a new directory for your root filesystem. For the sake of clarity, I call mine debian-rootfs. You'll use an application called debootstrap to pull the basic filesystem structure from a Debian repository. You are free to use a repository that is closer to you, rather than the same one I use in this example. Enter the following as root or using sudo, inside your newly created directory:

```
# debootstrap --foreign --arch armhf wheezy \
 /home/user/A13System/debian-rootfs \
 http://ftp.debian.org/debian
```

Note that the resulting structure is still not a complete filesystem. The next step is to create a chroot system within your new directory. For those of you who are not familiar with chroot, this command effectively creates an isolated system within your "host" system:

```
# cp $(which qemu-arm-static) \
 /home/user/A13System/debian-rootfs/usr/bin
# mount -t proc proc \
 /home/user/A13System/debian-rootfs/proc
# chroot /home/user/debian-rootfs /bin/bash

I have no name!# ./debootstrap/debootstrap \
 --second-stage
```

Copy the qemu-arm-static binary into your root filesystem's /usr/bin directory. The qemu-arm-static binary helps run the armhf binaries from your x86/64-bit systems. You

also need to mount the host's proc filesystem into your chroot system. When you first get inside the chroot system, you might find a strange prompt greeting you ("I have no name!"). This is not a cause for concern, and you can safely disregard it. Once you are inside your chroot system, execute another call to debootstrap to complete the base system (with `--second-stage`).

If you are curious whether you really are running an armhf system within your chroot system, issue the `uname` command to check. If you see something like "armv7l" somewhere in the output, it is an indication that your chroot is running the armhf system.

The next step is to update your apt source list file. Within your chroot system, or using the build host's editor, go and edit the file /etc/apt/sources.list that resides inside your root filesystem directory. Add the following lines to this file (remember, you can use other Debian repositories as well):

```
deb http://ftp.debian.org/debian wheezy main \
 contrib non-free
deb-src http://ftp.debian.org/debian wheezy main \
 contrib non-free

deb http://ftp.debian.org/debian wheezy-updates \
 main contrib non-free
deb-src http://ftp.debian.org/debian \
 wheezy-updates main contrib non-free

deb http://security.debian.org/ wheezy/updates \
 main contrib non-free
deb-src http://security.debian.org/ \
 wheezy/updates main contrib non-free
```

Preparing the Filesystem: Ubuntu

If you are feeling adventurous, you always can try to debootstrap your Ubuntu root filesystem, just like I described in the previous section. (You also can find instructions on the Internet for that.) Here, let's opt for an easier way and just download a ready-made minimal root filesystem provided by Ubuntu. Several packages are available, including Ubuntu 12.04, 12.10 and 13.04. Make sure that you are downloading the armhf version of the root filesystem package.

Create a directory for your Ubuntu root filesystem and extract the contents of the file you just downloaded into it. Your next move should be to edit /etc/resolv.conf and add your nameserver in there. Also, take a look at your sources.list file in /etc/apt/. You might want to add `universe` and `multiverse` at the end of each deb and deb-src line.

You should check the version number of your qemu-arm-static. Version 1.0.50 that comes with standard install of Ubuntu 12.04 generates errors when running the following steps on my build system for the Ubuntu root filesystem. To solve the problem, I had to compile my own qemu-arm-static. I used version 1.0.91 (see Resources for the download URL of the source package). Do the following steps to configure and compile the binary, and copy the resulting qemu-arm to qemu-arm-static inside your Ubuntu root filesystem's /usr/bin directory:

```
$ ./configure \
 --prefix=/home/user/A13System/qemu-arm-static \
 --static --disable-kvm \
 --target-list=arm-linux-user

$ make
$ make install
```

Finishing Touches for the Root Filesystem

What you have so far is just a very basic filesystem. Now let's improve it so that you have the tools required for a comfortable basic computing environment. Change the locales generated according to your own locale. All the processes described next are done inside the chroot system:

```
root@host:/# apt-get update
root@host:/# apt-get install apt-utils ncurses-dev
root@host:/# apt-get install dialog locales tzdata
root@host:/# locale-gen en_AU en_AU.UTF-8
root@host:/# dpkg-reconfigure locales
root@host:/# dpkg-reconfigure tzdata
root@host:/# apt-get install iputils-ping \
 wpasupplicant dhcpcd5 sudo openssh-server ntp \
 openssh-client
root@host:/# apt-get install nano vim gettext \
 bison automake autoconf
root@host:/# apt-get install python rsyslog \
 network-manager alsa-utils
```

Now let's configure Wi-Fi connectivity. I'm assuming that you're using a Wi-Fi USB adapter for your connectivity and that your wireless network connection configuration is using WPA for security. Change the steps accordingly for your configuration. Edit your /etc/network/interfaces and add the following lines, changing the values as needed:

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid YOUR_ESSID
wpa-psk YOUR_PASSPHRASE
```

Next, if you want your bash shell to have autocompletion, edit /etc/bash.bashrc and uncomment some

of the lines to be something like the following:

```
# Commented out, don't overwrite xterm -T
# "title" -n "icontitle" by default.
# If this is an xterm set the title to
# user@host:dir
case "$TERM" in
xterm*|rxvt*)
 PROMPT_COMMAND='echo -ne \
 "\033]0;${USER}@${HOSTNAME}: ${PWD}\007"'
 ;;
*)
 ;;
esac

# enable bash completion in interactive shells
if [ -f /etc/bash_completion ] && ! shopt -oq \
posix; then
    . /etc/bash_completion
fi
```

Check /etc/shadow for the following:

```
root:*:15629:0:99999:7:::
daemon:*:15629:0:99999:7:::
```

If you see an asterisk (*) after the first colon on the line for root, you should remove it. This will allow you to set the root password yourself on the first run.

You have completed the process of building the root filesystem for your board. Next, let's compress the entire root filesystem so you can deploy it easily to your MicroSD card later. Exit the chroot environment and do the following inside your root filesystem directory:

```
# umount proc
# rm ./usr/bin/qemu-arm-static
# tar -zcvf /home/user/my-rootfs.tar.gz *
```

Preparing the MicroSD Card

I am using a 4GB MicroSD card for my board. I am sure that a 2GB MicroSD card would be sufficient to contain all your files, but it is nice to have some room for additional applications. You need to create two partitions on your empty MicroSD card. The first one is a VFAT partition of around 17MB for U-Boot and the kernel image. The rest will be used to store your root filesystem.

Mount the MicroSD card. Take note of the device name your computer gives the MicroSD card. Some computers recognize the card as /dev/sdX, while others call it /dev/mmcblkX (for this example, I assuming that your card is recognized as /dev/sdb):

```
# fdisk -u=sectors /dev/sdb
```

Type "p" to list the partitions inside the card. If you have any

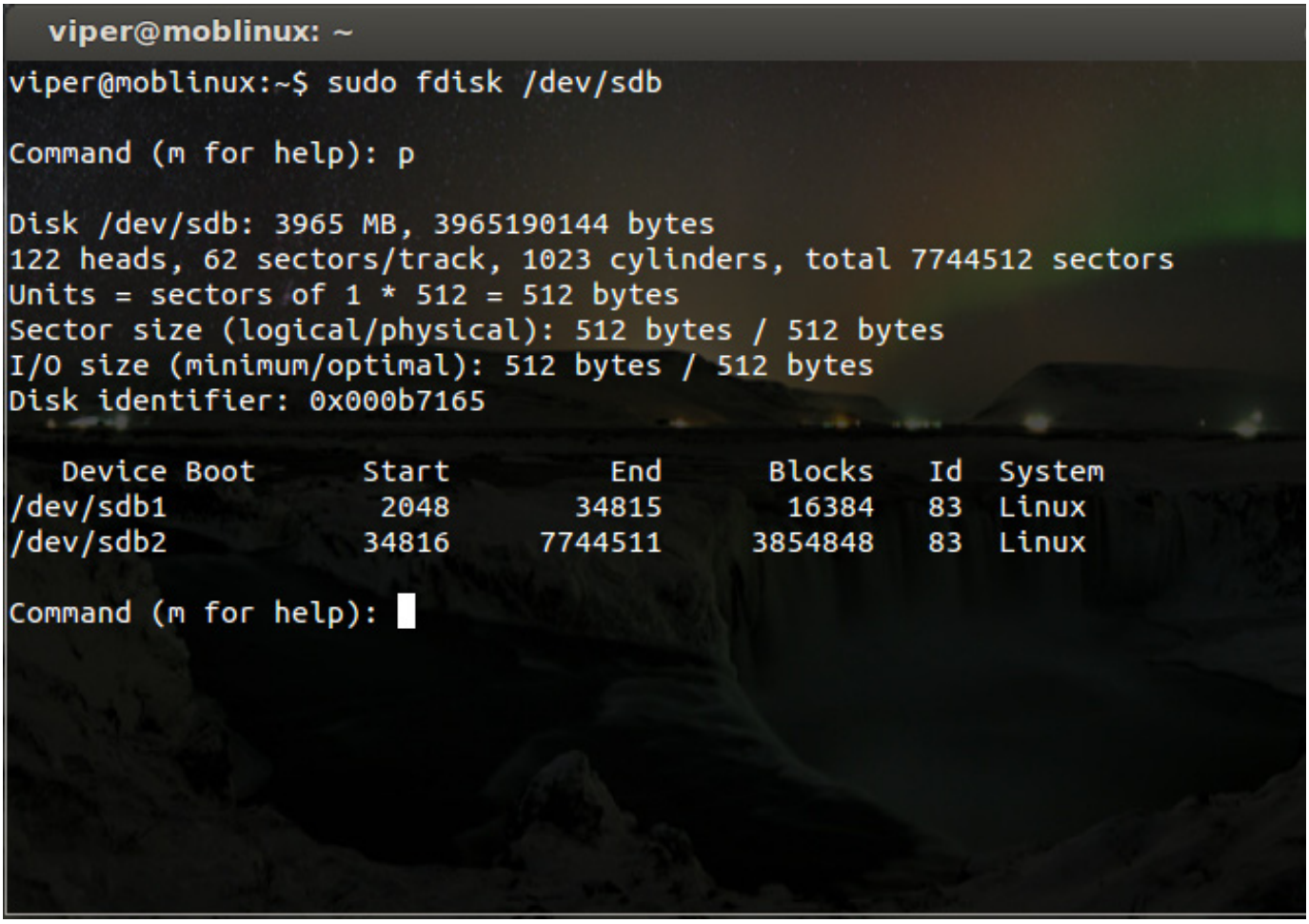

Figure 7. List of Partitions on a 4GB MicroSD Card

partitions at all listed, delete them by pressing "d". Once the card is empty, create a new partition by pressing "n". Make this the first primary partition. fdisk is going to ask you for starting and ending sector numbers. Type "2048" and "34815", respectively. Repeat the process for the second partition. This time, just press Enter when asked for starting and ending sector numbers; fdisk will use the default values, which will fill the remainder of the card.

Type "p" again to list the partitions. You should see something like what is shown in Figure 7.

Type "w" to write the changes permanently onto the card. Now, create the two filesystem types on the partitions:

```
# mkfs.vfat /dev/sdb1
# mkfs.ext3 /dev/sdb2
```

Don't forget to do the sync after every command you type for the MicroSD card. Sync ensures that the changes are flushed and keeps the card in the correct state. Next, mount the partitions. I am assuming the mountpoints are /media/card1 for /dev/sdb1 and /media/card2 for /dev/sdb2. First, populate the root filesystem and copy the kernel modules onto the first partition:

```
# cd /media/card2
# tar -xzvf /home/user/A13System/my-rootfs.tar.gz
# sync
...[THIS WILL TAKE SOME TIME]
# cp -a $KERNEL_DIR/out/lib/modules/3.4.43+/ \
 ./lib/modules/.
# sync
```

Copy the uImage file from your kernel directory (in arch/arm/boot) to the first partition, along with a file called script.bin. script.bin stores the system configuration settings necessary for Allwinner chips. If you want to edit these settings, convert this .bin file into a .fex file using a tool called bin2fex. You can edit the resulting file with any text editor.

For the last step, you need to write U-Boot onto the card itself. Pay extra attention to what you type here, as you are not going to write to /dev/sdb1 or /dev/sdb2 but to /dev/sdb:

```
# cd /home/user/A13System/u-boot
# dd if=spl/sunxi-spl.bin of=/dev/sdb bs=1024 \
 seek=8
# dd if=u-boot.bin of=/dev/sdb bs=1024 seek=32
# sync
```

Now your MicroSD card is ready to use.

First Run

Plug in the card in the slot on the board. Also plug in the Wi-Fi USB Stick, a keyboard and the VGA monitor (use a USB hub if you have to). Plug in the power cord and wait for the login prompt.

Log in with the root account. You shouldn't need a password for the first run. After you get in, set a secure password for your root account and create another account for your daily use. Put this new user into the sudoers file. Check whether you have network connectivity. Test the board remotely by connecting to it via SSH. If you can do all that successfully, congratulations! You have a great minimalist PC/server at your disposal.

IF YOU ARE INTERESTED IN USING THIS BOARD WITH A GRAPHICAL USER INTERFACE, YOU NEED TO USE A LIGHTWEIGHT GUI ENVIRONMENT, BECAUSE THE BOARD DOES NOT HAVE MUCH RAM TO SPARE.

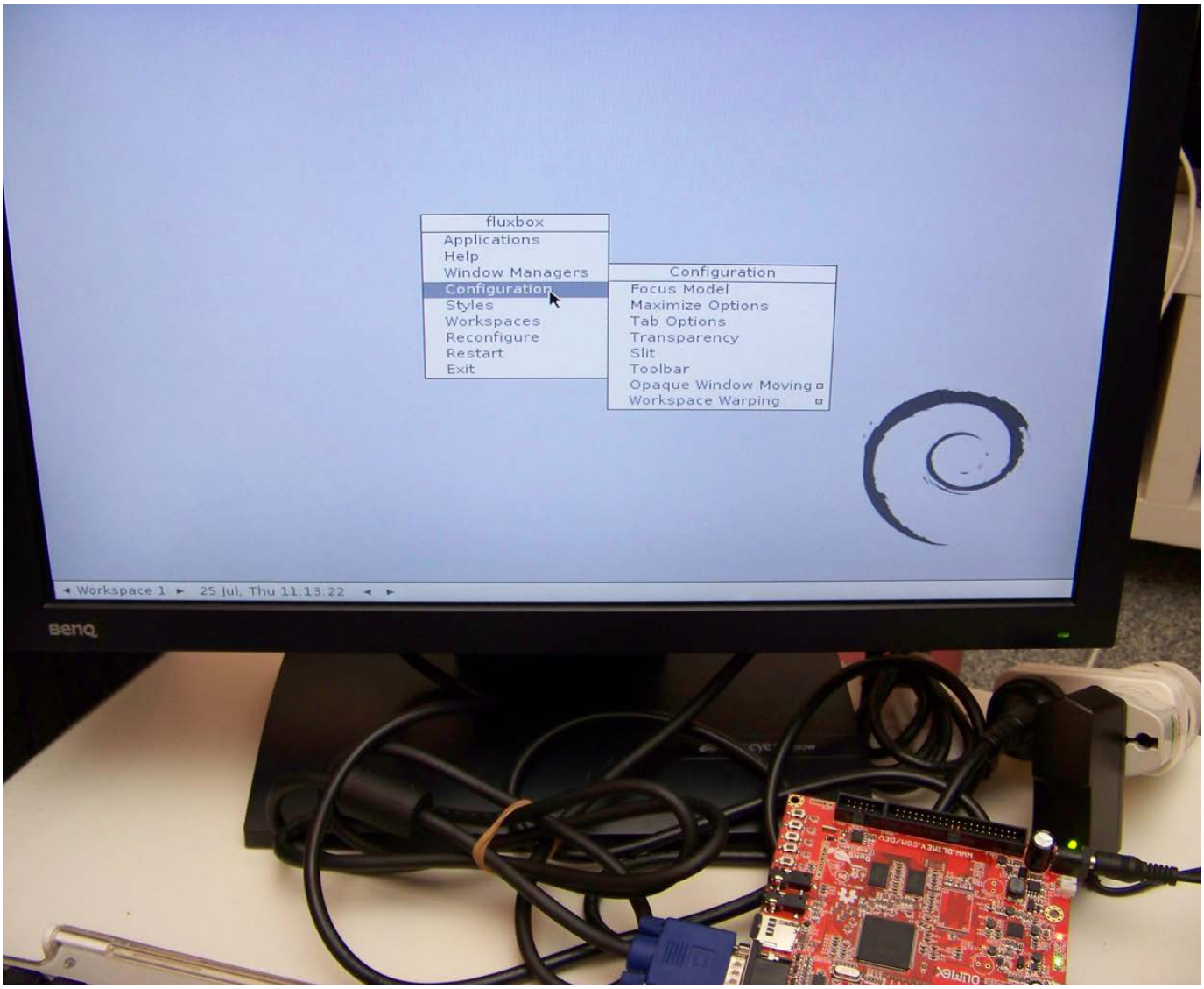

Figure 8. A13Micro Running Fluxbox

Desktop Environment

If you are interested in using this board with a graphical user interface, you need to use a lightweight GUI environment, because the board does not have much RAM to spare. There are several options from which to choose, such as LXDE and XFCE4; however, I use a different package here called Fluxbox. You also need to install a light graphical login manager. Using the package manager, install lightdm and fluxbox. Yes, it is really that easy. These commands will install the desktop environment, graphical login manager and their required servers and libraries:

```
root@a13board:/# apt-get install lightdm fluxbox
```

Restart the board. When the board restarts, you will be greeted with your new login manager. Make sure you select Fluxbox from the session menu on Lightdm screen when you are logging in. Enjoy your new mini-personal computer/server!

Ronald Kurniawan is a software developer living in Brisbane, Australia. Ronald is interested in embedded systems, Linux, Java development and trying to come up with interesting and wacky ways to combine them. Ronald can be reached at r.kurniawan@fluxodesign.net.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

Resources

Debian Repositories List: **http://www.debian.org/mirror/list**

A13Micro's Kernel Configuration File: **http://goo.gl/YnZ1s**

Script.bin for A13 and A13Micro Boards: **http://goo.gl/7QZuoU**

Ubuntu 12.04 Core Root Filesystem: **http://goo.gl/eoALA**

Ubuntu 12.10 Core Root Filesystem: **http://goo.gl/iLcV8**

Ubuntu 13.04 Core Root Filesystem: **http://goo.gl/cytEY**

Qemu Source Code Download: **https://launchpad.net/qemu-linaro/+download**