Pratyusha.Gandham, Ramesh N.V.K / International Journal of Engineering Research and Applications (IJERA)
ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 2,Mar-Apr 2012, pp.1614-1618

PORTING THE LINUX KERNEL TO AN ARM BASED DEVELOPMENT BOARD

Pratyusha.Gandham[1] , Ramesh N.V.K[2]
1Research Scholar, Dept. of E.C.M, K.L. University, A.P,India
2 Associate Professor, Dept. of E.C.M, K.L. University

Abstract
ARM development boards are the ideal platform for accelerating the development and reducing the risk of new SoC designs. The combination of ASIC and FPGA technology in ARM boards delivers an optimal solution in terms of speed, accuracy, flexibility and cost.. The Embedded modules,based on ARM, can become very complex machines since these are meant to support varied tasks such as memory management, process management and peripheral interfaces.For seamless integration of these functional modules an OS has to be ported on these ARM based CPUs .Traditionally this OS porting is often the specialized work of third party vendors having expertise in this domain. For every new CPU architecture, the OS has to be customized, compiled and burnt into the core .With the coming of age of Linux as an embedded OS all this has changed quite significantly. Being in Open Source domain, Linux kernel can be freely downloaded and compiled for any system architecture and this includes ARM based systems also. This enables the developers to port the OS themselves. This paper describes the details of porting of Linux kernel to an ARM board.Building linux kernel,u-boot and x-loader are described in detail. SD card configuration is also described.

Index Terms: *ARM, Operating System, Linux, Kernel.*

---***---

1. INTRODUCTION
Linux has been available for the ARM architecture for many years now. The original 'port' was done by Russell King, and he is still the maintainer through whom all ARM kernel patches generally must pass. GNU/Linux is fast becoming the operating system for embedded devices - mainly due to the efficient and portable design of the Linux kernel. The ARM Linux port effort, headed by Russell M. King, also makes life a bit easier for people who run (or want to run) Linux on their embedded devices.

The work was much harder than than it would be now, since at the time the Linux kernel was still very Intel-centric. In fact, modern Linux kernels come with a handy reference example called asm-generic that shows all of the header files and kernel interfaces that a new architecture port should provide. Once the kernel has been ported to a given architecture, it is necessary to implement support for a specific platform based upon that architecture.

Texas Instruments implements the ARM architecture in its OMAP processors. For example, Panda Board is a particular variant of the OMAP4 platform. Panda Board is based on ARM Cortex-A9 System-on-Chip (SoC) processor.

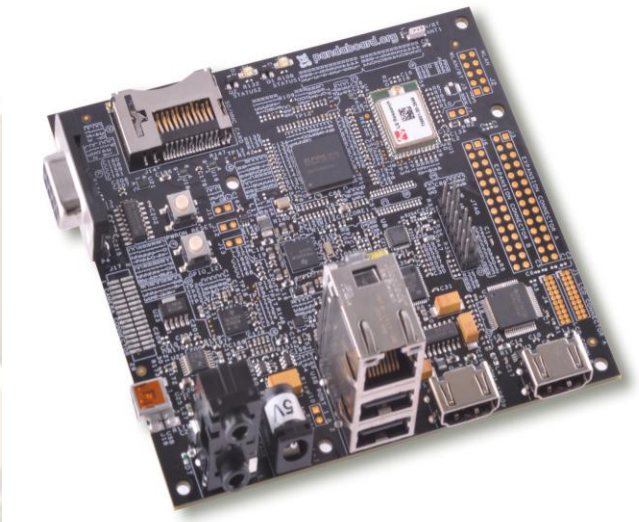

Figure1: Panda Board, an ARM based development board from Texas Instruments.

2. BUILDING LINUX KERNEL
Latest stable version of linux kernel can be obtained from kernel.org website. This is in the form of a source code and we need to compile it. Figure 2 shows the basic architecture of linux2.6.11.6. The Linux kernel supports a lot of different

CPU architectures. The methods to port the Linux kernel to a new board are therefore very architecture dependent. For example, PowerPC and ARM are very different. PowerPC relies on device trees to describe hardware details whereas ARM relies on source code only. In the source tree, each architecture has its own directory arch/arm for the ARM architecture. For building the linux kernel, cross-compiler should be made accessible on your execution path.

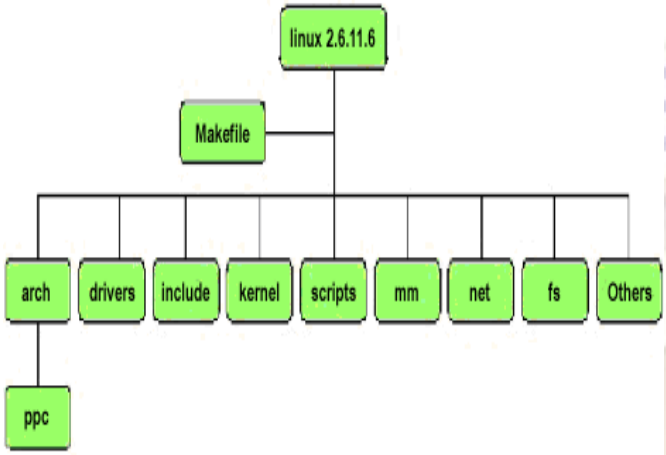

Figure 2:Linux kernel architecture

2.1 Get Latest Linux kernel code

Visit http://kernel.org/ and download the latest source code. File name would be linux-x.y.z.tar.bz2, where x.y.z is actual version number. For example file inux-2.6.25.tar.bz2 represents 2.6.25 kernel version. Use wget command to download kernel source code:

- $wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-x.y.z.tar.bz2

2.2 Extract tar (.tar.bz3) file

The tar program provides the ability to create tar archives, as well as various other kinds of manipulation.Typing the following command will extract it to current directory:

- # tar -xjvf linux-2.6.25.tar.bz2

2.3 Configure kernel

Before you configure kernel make sure you have development tools (gcc compilers and related tools) are installed on your system. If gcc compiler and tools are not installed then use apt-get command under Debian Linux to install development tools.

- # apt-get install gcc

Now you can start kernel configuration by typing any one of the command:

- **$ make menuconfig** - Text based color menus, radiolists & dialogs. This option also useful on remote server if you want to compile kernel remotely.
- **$ make xconfig** - X windows (Qt) based configuration tool, works best under KDE desktop
- **$ make gconfig** - X windows (Gtk) based configuration tool, works best under Gnome Dekstop.

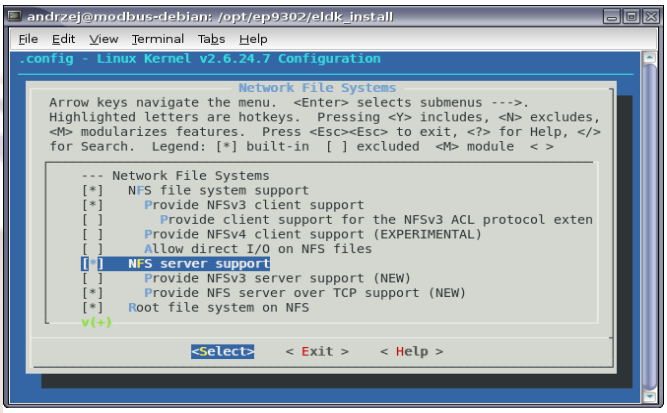

Figure 3: Configuring Linux Kernel using make menuconfig.

2.4 Compile kernel

The Linux kernel supports a lot of different CPU architectures.The methods to port the Linux kernel to a new board are therefore very architecture dependent. In the source tree, each architecture has its own directory arch/arm for the ARM architecture.For building the linux kernel, cross-compiler should be made accessible on your execution path. Entering make command would start compilation and create a compressed kernel image. The resulting kernel image is placed in *arch/arm/boot directory*.

- $export ARCH=arm
- $ export CROSS_COMPILE= arm-none-linux-gnueabi
- $ make

3. X-LOADER AND U-BOOT

X-loader is a small first stage boot loader derived from the u-boot base code to be loaded into the internal static ram. Because the internal static ram is very small (64k-32k), x-loader is stripped down to the essentials and is used to initialize memory and enough of the peripheral devices to access and load the second stage loader (U-Boot) into main memory.U-Boot is designed to be the "Universal Bootloader" and certainly goes a long ways to support multiple processors, boards, and OS's. It's primary purpose is to setup the kernel environment; and load and boot the kernel. The sources of u-boot and x-loader specific to the ARM board can be obtained and built using make utility on terminal. The procedure for building it is same as it is for linux kernel. The output of the build process is u-boot.bin and MLO.

3.1 Building X-loader

In order to compile the X-loader, you need to set the CROSS_COMPILE environment variable and specify the path to tool chain.

- #export PATH=/usr/local/xtools/arm--linux-uclibcgnueabi/bin:$PATH
- #export CROSS_COMPILE= arm-none-linux-gnueabi-

To set the configuration for the target board and build the image, give the command:

- # make BOARDNAME_config
- # make

The resulting file is stored in the x-loader main directory as x--load.bin. This file must be "signed" in order to be executed by the processor. Using the signGP tool in the tools/ subdirectory of the main directory of the lab, sign the x-load.bin file. This produces an x-load.bin.ift file. You can copy it to the MMC card, renaming it as MLO.

3.2 Building U-boot

U-Boot is a typical free software project.It is freely available at http://www.denx.de/wiki/U-Boot.Get the source code from the website, and uncompress it.The include/configs/ directory contains one configuration file for each supported board.It defines the CPU type, the peripherals and their configuration, the memory mapping, the U-Boot features that should be compiled in, etc.Assuming that your board is already supported by U-Boot, there should be one file corresponding to your board, for example include/configs/omap2420h4.h.U--Boot must be configured before being compiled.

- #make BOARDNAME_config

Make sure that the cross-compiler is available in PATH

- #export PATH=/usr/local/uclibc-0.9.29-2/arm/bin/:$PATH

Compile U-Boot, by specifying the cross-compiler prefix. Example, if your cross-compiler executable is arm-linux-gcc:

- #make CROSS_COMPILE=arm-linux

This command would result in u-boot.bin file in working directory

4.SD CARD CONFIGURATION

Some ARM boards do not have any onboard flash, to keep their bootloader. Rather, code onboard the board (presumably in ROM) reads the second-stage bootloaders from the SD card .For the OMAP to find and boot off the SD Card, the first primary partition must contain a FAT32 partition formatted with 255 heads and 63 sectors. It is very specific, but not hard to setup. The SD Card should have 2 partitions : a smaller FAT32 boot partition and a larger ext3 partition for the filesystem or rootfs. Following are the steps to partition the SD card on a linux machine.

4.1 Formatting the SD Card

To determine which device the SD Card Reader is on your system,plug the SD Card into the SD Card Reader and then plug the SD Card Reader into your system. After doing that, do the following to determine which device it is on your system.

$ [dmesg | tail]
...
[6854.215650] sd 7:0:0:0: [sdc] Mode Sense: 0b 00 00 08
[6854.215653] sd 7:0:0:0: [sdc] Assuming drive cache: write through
[6854.215659] sdc: sdc1
[6854.218079] sd 7:0:0:0: [sdc] Attached SCSI removable disk
...
In this case, it shows up as /dev/sdc.

Fdisk utilty is used to partition SD card on a linux machine. The following command starts the fdisk. We must clear the partition table before changing the underlying geome

- Command (m for help): [d]
  Selected partition 1

4.2 Set the Geometry of the SD Card

If the print out above does not show 255 heads, 63 sectors/track, then do the following expert mode steps to redo the SD Card:

- Go into expert mode.

Command (m for help): [x]

- Set the number of heads to 255.

Expert Command (m for help): [h]
Number of heads (1-256, default xxx): [255]

- Set the number of sectors to 63.

Expert Command (m for help): [s]
Number of sectors (1-63, default xxx): [63]

- Now Calculate the number of Cylinders for your SD Card.

#cylinders = FLOOR (the number of Bytes on the SD Card (from above) / 255 / 63 / 512)So for this example: 2021654528 / 255 / 63 / 512 = 245.79. So we use 245 (i.e. truncate, don't round).

- Set the number of cylinders to the number calculated.

Expert Command (m for help): [c]
Number of cylinders (1-256, default xxx): [enter the number you calculated]

- Return to Normal mode.

Expert Command (m for help): [r]

4.3 Partitioning the SD card

4.3.1 Create the FAT32 partition for booting

- Command (m for help): [n]

Command action
e extended
p primary partition (1-4)
[p]
Partition number (1-4): [1]
First cylinder (1-245, default 1): [(press Enter)]
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-245, default 245): [+50]

- Command (m for help): [t]

Selected partition 1
Hex code (type L to list codes): [c]
Changed system type of partition 1 to c (W95 FAT32 (LBA))
Command (m for help): [a]
Partition number (1-4): [1]

4.3.2 Create the Linux partition for the root file system

- Command (m for help): [n]

Command action
e extended
p primary partition (1-4)
[p]
Partition number (1-4): [2]
First cylinder (52-245, default 52): [(press Enter)]
Using default value 52
Last cylinder or +size or +sizeM or +sizeK (52-245, default 245): [(press Enter)]
Using default value 245

- Save the new partition records on the SD Card

This is an important step. All the work up to now has been temporary.

- Command (m for help): [w]

4.4 Formating the partitions

The two partitions are given the volume names LABEL1 and LABEL2 by these commands. You can substitute your own volume labels.
$ [sudo mkfs.msdos -F 32 /dev/sdc1 -n LABEL1]
mkfs.msdos 2.11 (12 Mar 2005)

$ [sudo mkfs.ext3 -L LABEL2 /dev/sdc2]
mke2fs 1.40-WIP (14-Nov-2006)

4.5 Finalizing SD Card

Finalizing SD card involves coping x-loader, u-boot, and kernel images to the boot partition.You can have multiple kernel images and name isn't important.The signed x-loader must be called MLO on the card.The first file you need to copy is the “MLO” from X-Loader. This has to be the first file as it is the primary bootloader and the system just looks on the first few cluster on the SD Card to find it. The second file is the “u-boot.bin” and the third file is the kernel, “uImage”. The root system has now to be untared to the second partition.

5. ARM BOARD SERIAL TERMINAL SET-UP

For a Linux machine, a serial terminal such as Minicom or Kermit can be used. Minicom is a text-based modem control and terminal emulation program for Unix-like operating systems whereas gtkterm is a terminal emulator written with GTK+.It is lightweight and simple that drives serial ports Ubuntu users wanting a graphical terminal program can install gtkterm. Later versions which include many bug fixes can be obtained from the current maintainer's website.On a Windows PC, you could use HyperTerminal or TeraTerm. Insert the SD card into the SD card slot of the target board. Connect the target board to host machine using RS232 serial cable. Apply power supply to the target board.

6. BOOTING UP THE ARM BOARD

The term ‘booting’ is short for ‘bootstrapping’, which refers to the process by which a computer prepares itself to load an operating system. Small, resource-constrained systems such as the BeagleBoard use a slightly different, multi step process because of the constrained memory. The board boots up in four stages . The boot process on the ARM Board works like this:
X-Loader -> U-Boot -> Kernel

Figure4: Booting up the arm board

7.Conclusion

The paper discussed a generic environment build of Linux for ARM core family. This generic build has a limited functionality, as it just enables the kernel to boot and send debug messages through the configured serial port. For a full fledged embedded module his is quit elementary without the support for various peripherals through device drivers which

are paramount to the module. Linux kernel though monolithic has an excellent modular approach which enables the driver modules to be attached and detached at run time itself. This very much suits the embedded environment which are always constrained or system memory.

ACKNOWLEDGEMENT

We thank to our principal, Prof. K. Raja Shekar Rao, for providing necessary facilities towards carrying out this work. We acknowledge the diligent efforts of our Head of the Department Dr.S.Balaji in assisting us towards implementation of this idea.

REFERENCES

[1] HU Jie, ZHANG Gen-bao, "Research transplanting method of embedded linux kernel based on ARM platform", 2010 International Conference of Information Science and Management Engineering, E-ISBN : 978-1-4244-7670-1, 8 Aug. 2010
[2] http://www.kernel.org.
[3] http://www.arm.linux.org.uk/developer/machines
[4] http://sourceforge.net/projects/u-boot
[5] Vincent Sanders, "Booting ARM Linux", rev.1.10,June2004.http://www.simtec.co.uk/products/SWLINUX/files/booting_article.html
[6] Wooking and Tak-Shing, "Porting the Linux kernel to a new ARM Platform", Aleph One, vol. 4, summer 2002.
[7] Chun-yue Bi; Yun-peng Liu; Ren-fang Wang; "Research of key technologies for embedded Linux based on ARM", Computer Application and System Modeling (ICCASM),2010 International Conference, 22-24 Oct. 2010, E-ISBN : 978-1-4244-7237-6.
[8] DongSeok Cho; DooHwan Bae, "Case Study on Installing a Porting Process for Embedded Operating System in a Small Team", Secure Software Integration & Reliability Improvement Companion (SSIRI-C), 2011 5th International Conference, 27-29 June 2011, E-ISBN : 978-0-7695-4454-0.