Design and Implimentation of Embedded Linux on ARM Platform

Mr. Amol R.Wagh, Prof.Vijay S.Patil

Abstract— In this paper we explains the Embedded Linux, generic architecture of Embedded Linux and different types of host/target development set up. In this paper we also explain design procedure of Embedded Linux for target architecture ARM. Building of x-loader; u-boot and root file system is required for porting of designed Embedded Linux on ARM processor. To port designed Embedded Linux on ARM platform required SD-card partitioning also described. Kernel source code is modified according to requirement and modified source code ported on OMAP3530 processor based on ARM Cortex-A8 as its core.

Index Terms— Embedded Linux, Beagleboard-xM, Toolchain, porting, root file system, SD-card partitioning, ARM.

1. INTRODUCTION

In all the fields there is an increasing demand of portable and embedded system. It includes embedded applications such as cellular telephones, MP3 players and a host of digital home entertainment devices, also less obvious examples such as bank ATMs, printers, cars, traffic signals, medical equipment, technical diagnostic equipment, aerospace and many more. Essentially anything with a microprocessor that is not considered a computer but performs some kind of function using computing is a form of embedded system.

1.1 WHAT IS LINUX –

Linux is an operating system kernel written by Linus Torvalds. The Linux kernel provides a variety of core system facilities for any system based upon Linux to operate correctly. In everyday conversation Linux is termed as Linux distribution. There are number of Linux distributions from MontaVista, Wind River, Timesys, Fedora etc. This Linux distributions are vary in purpose size and price with each other but they share a common goal: to provide a user with a prepackaged, shrink-wrapped set of files and an installation procedure to get the kernel and various overlaid software installed on a certain type of hardware for a certain purpose.

1.2 WHAT IS EMBEDDED LINUX –

Embedded Linux typically refers to a complete system for Linux distribution targeted at embedded devices. The same Linux kernel source code used for building the wide range of devices such as workstations, servers etc. It is possible to configure a variety of optional features according to the intended use of the kernel.

To devolop a new embedded system product operating system modified according to fit the target embedded board and the modified operating system according to our requirement is ported on to specific embedded system board. Accuracy, speed, less power consumption, networkability and stringent time consntraint are important chararacteristics of an embedded system.

1.3 TYPICAL EMBEDDED LINUX SETUP

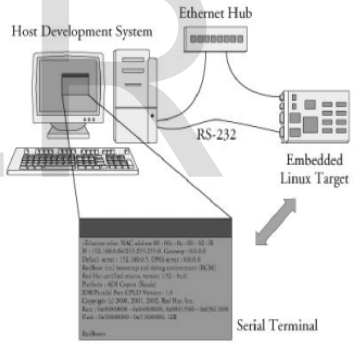
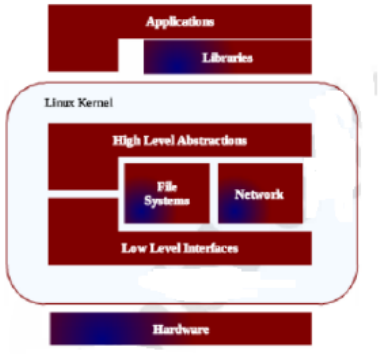


Fig.1 Typical Embedded Linux Set Up

2. GENERIC ARCHITECTURE OF EMBEDDED LINUX SYSTEM



- *Mr.Amol R.Wagh is currently pursuing masters degree program in E&TC engineering from R.C.Patel Institute of Technology Shirpur, India, PH-09579518209. E-mail: amolrwagh@gmail.com*
- *Prof.Vijay S.Patil is currently working as associate Professor in E&TC Dept. R.C.Patel Institute of Technology Shirpur, India, PH-07588814275. E-mail: vijayshri12@gmail.com*

Fig.2 Generic architecture of embedded linux system

HARDWARE MUST MEET SOME BROAD CHARACTERISTICS TO RUN LINUX SYSTEM

i) Linux normally requires at least a 32-bit CPU containing a MMU.
ii) A sufficient amount of RAM must be available to accommodate the system.
iii) Minimal I/O capabilities are required.
iv) The kernel must be able to load a root file system through some form of permanent storage, or access it over a network.

Kernel is core core component of the operating system. Its purpose to manage hardware and provide high level abstractions for user level softwares. Kernel acts as a interface between hardware and user level applications. Low level interfaces of kernel provide the direct control of hardware on which kernel runs. High level components include files, sockets, processes and signals. Between this two levels kernel use some interpretation components for the purpose to understand and interact with structured data coming from or going to some perticuler devices for example some networking protocols.

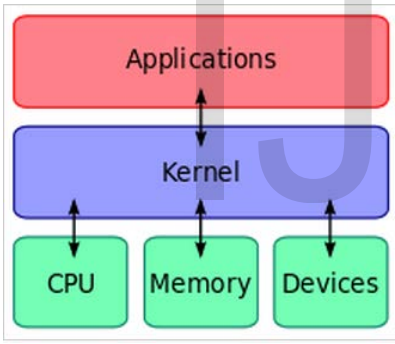


Fig.3 Kernel interface

Applications rely on libraries and special system daemons to provide familiar APIs and abstract services that interact with the kernel on the applications behalf to obtain the desired functionality. The main library used by most Linux applications is the GNU C library, glibc. Libraries are typically linked dynamically with applications

3. TYPES OF HOST/TARGET DEVOLOPMENT SETUP

There are three host/target architectures are available for development of Embedded Linux system.

3.1. LINKED SETUP –

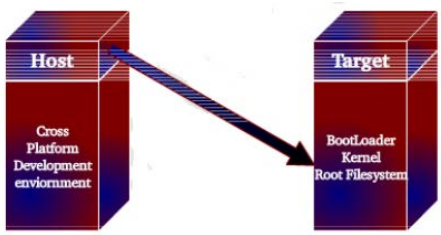


Fig .4.1 Linked Setup

In this set up the target and host are permanently linked together using a physical cable. This link is typically a serial cable or an Ethernet link. The main property of this setup is that no physical hardware storage device is being transferred between the target and the host. As illustrated in above fig. 3.1 the host contains the cross platform development environment, while the target contains an appropriate boot-loader, a functional kernel, and a minimal root filesystem.

3.2 REMOVABLE STORAGE SETUP –

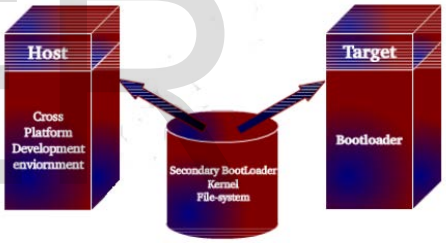


Fig. 4.2 Removable Storage Setup

In this set up, there are no direct physical links between the host and the target. A storage device is written by the host which is then transferred into the target and is used to boot the device. As shown in above fig. 3.2 the host contains the cross platform development environment. The target contains only a minimal boot-loader. The rest of the components are stored on a removable storage media such as Compact-Flash IDE device, MMC Card, which is programmed on the host and loaded by the targets minimal bootloader upon startup.

3.3 STANDALONE SETUP-

In this set up the target is self-contained development system and includes all the required software to boot, operate and develop additional software. In such a type of method does not require any cross-platform development environment, since all development tools run in their native environments. There is no need to transfer between target and host.

Fig.4.3 Standalone Setup

4. THE SYSTEM DEVOLOPING PLATFORM

Linux runs on a large and ever-growing number of machine architectures, but not all these architectures are actually used in configrations. Linux kernel supports 24 different architectures and ARM is one of them.

- **ARM –**

ARM, which stands for Advanced RISC Machine, is a family of processors maintained and promoted by ARM Holdings Ltd. According to Wikipedia around 70% of all 32 bit embedded CPU's are based on ARM architecture. Instead it designs complete CPU cores for its customers. Based on the ARM core, charges customers licensing fees on the design, and lets them manufacture the chip wherever they say fit. There is one important characteristics all ARM processors share the same ARM instruction set. This does noy mean that all ARM CPUs and boards can be programmed and set up in the same way, only that the assembly language and resulting binary codes are identical for all ARM processors meeting a certain revision of the architecture. The ARM architecture is very populer in many fields of application from cellphones and PDAs to networking equipment, and there are hundreds of vendors providing products and services around it.

5. CREATING A TARGET EMBEDDED LINUX –

A target Embedded Linux is created by configuring and bundling together the approprite system components. There are following main steps to creating a target Linux system.

i. Determine system components
ii. Configure and build the kernel and creation of toolchain
iii. Building the x-loader, u-boot and root file system
iv. SD-Card partitioning
v. Set up boot software and configuration then bringing up the OMAP-3530 ARM processor board.

5.1 Determine system components

At first decide the target architecture. In our work we use ARM as target architecture. Following tools are needed for creation of toolchain -

- Binutils-2.20.1a
- Kernel-2.6.22
- gcc-4.5.1
- glibc-2.13
- glibc-port-2.13
- gmp-5.0.1
- mpfr-3.1.0
- mpc-0.8.2

This, all sytem components are compatible with each other.

5.2 A Toolchain –

Following are the steps for creation of toolchain by using all the tools.

- Firstly downloads all the packages required to build the toolchain from respective sources.
- Make the environment of toolchain by defining environment variables.
- Firstly compile and install the binutils, which assembles the object files generated by GCC in an executable image.
- Compiles and install the header files of the kernel
- Compile the temporary version of GCC
- Compile the c libraries glibc or uclibc according to our need for target architecture.
- Compile and install GMP and MPFR which are the libraries required for mathematical operations.
- Install the final verson of GCC with previous libraries.

5.2.1 CONIFGURE THE KERNEL

To extract the kernel in current directory use the command:
tar -xjvf linux-2.6.22.tar.bz2
Before configure kernel makes sure development tools (gcc compilers and related tools) are installed on our system. Now start kernel configuration by typing any one of the command:

make config: The most traditional text-based configuration interface, it is hard to use and is very Cumbersome enter the command line, we can configure the line by line.

make menuconfig: Text-based configuration menu interface, is a common way under the character terminal.

make xconfig: Window mode configuration based on a graphical interface, under the Xwindow recommended.

5.2.2 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. 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

5.3 BUILDING THE X-LOADER, U-BOOT AND ROOT FILE SYSTEM

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 (32k-64k), 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 (UBoot) into main memory. U-Boot is designed to be the "Universal Bootloader" and goes a long ways to support multiple processors, boards, and OS's. u-boot is a second stage boot loader that is loaded by the x-loader into DDR. The u-boot can perform CPU dependent and board dependent initialization and configuration not done in the x-loader. 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 output of the build process is u-boot.bin and MLO.

5.4 SD-Card partitioning

Some ARM boards do not have any onboard flash, to keep their bootloader. Rather, code onboard the board 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

Following are the steps to partition the SD card on a linux machine

5.4.1 FORMATTING THE SD CARD

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

$ [dmesg | tail

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

5.4.2 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 setup 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 of the 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, not 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]

5.4.3 PARTITIONING THE SD CARD -

5.4.3.1 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]

5.4.3.2 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)]

- Now, Save the new partition records on the SD Card
- Command (m for help): [w]

5.4.4 FORMATING THE PARTITIONS

The two partitions are given the volume names LABEL1 and LABEL2 by these commands.

$ [sudo mkfs.msdos -F 32 /dev/sdc1 -n LABEL1] mkfs.msdos 2.11

$ [sudo mkfs.ext3 -L LABEL2 /dev/sdc2] mke2fs 1.40-WIP

5.4.5 FINALISING SD CARD

Finalizing SD card involves coping x-loader, u-boot, and kernel images to the boot partition. The signed x-loader must be called MLO on the card. The first file gets 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.

5.5 BRINGING UP OMAP-3530 ARM PROCESSOR BOARD -

5.5.1 OMAP-3530 ARM PROCESSOR BOARD (BEAGLEBOARD-XM) –

The main features of BeagleBoard-xM are listed below:

- OMAP3530 Microprocessor – 1200 DMIPS, based on ARM Cortex-A8 running at 600Mhz
- TMS320C64x+ DSP for versatile signal processing at up to 430MHz

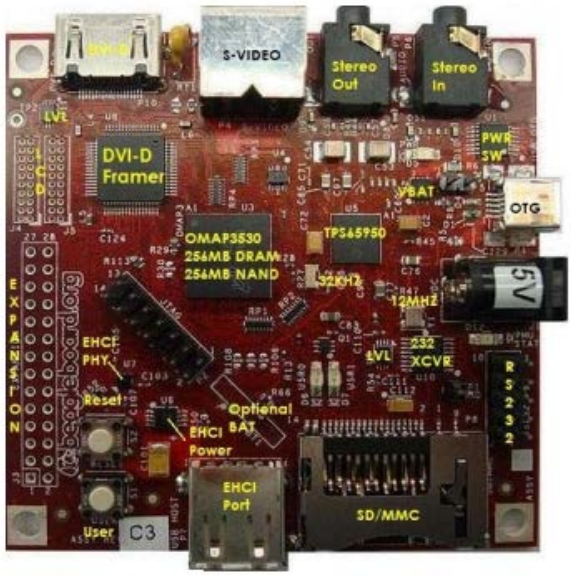


Fig. 5 BeagleBoard-xM

- Memories: 256MB of NAND Flash, 256MB of SDRAM
- SD/MMC Card slot, can be used as OS file system, file storage and more
- USB Host port which gives the option to connect a full set of peripherals using a hub (ethernet adaptor, key-board, mouse...).
- DVI-D digital output and S-VIDEO (NTSC/PAL) out-put Stereo audio output and microphone input
- Multiplexable expansion header : I2C, SPI, UART, GPIO, SD/MMC
- Typical power consumption : 2W (at 5V)

5.5.2 ARM BOARD SERIAL TERMINAL SET-UP –

For a Linux machine, a serial terminal such as Minicom is used. Minicom is a text-based modem control and terminal emulation program for Unix-like operating systems.

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 OMAP-3530 ARM processor board.

5.5.3 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.

The boot process on the ARM Board works like this:
X-Loader -> U-Boot -> Kernel

Thus, the designed Embedded Linux which we store in SD-card gets ported on OMAP-3530 processor.

6. CONCLUSION

In this way, we describe Linux internels along with design procedure of Embedded Linux for target architecture ARM. Also give details about toolchain, building of root file system, SD-card partitioning and porting of Embedded Linux on OMAP-3530 ARM processor.

7. REFERENCES

[1] Hu Jie, "Research transplanting method of embedded linux kernel based on ARM platform", International conference on Information Science and Management Engineering vol 2, pp. 35-38, 2010

[2] Pratyusha.Gandham1, Ramesh N.V.K2, "Porting the linux kernel to an arm based development board.", International Journal of Engineering Research and Applications (IJERA), Vol 2, pp.1614-1618, 2012

[3]] Chun-yue Bi, Yun-peng Liu, Ren-fang Wang, "Research of Key Technologies for Embedded Linux Based on ARM", International Conference on ICCAAM,vol B, pp.V8-373 - V8-378, 2010

[4] Peng Zhou Ligang Hou, "Implementation of CAN bus device driver design Based on Embedded System", International Conference on Communication, Networking & Broadcasting, pp.1252 – 1255, 2010

[5] H.Lyytinen, K.Haataja & P.Toivanen, "Designing and Implementing an Embedded Linux for Limited Resource Devices", Eighth International Conference on Networks, pp:18-23, 2009

[6] A book "Building Embedded Linux System" by Karim Yaghmour and John Masters.

[7]] 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, 2011 5th International Conference, 27-29 June 2011.

[8] Wooking and Tak-Shing, "Porting the Linux kernel to new ARM Platform", Aleph One, vol. 4, summer 2002