Porting and BSP Customization of Linux on ARM Platform

K. Eswar Kumar
M.Tech (ES), Dept. of ECE
Gudlavalleru Engineering College
Andhra Pradesh, India

M. Kamaraju, Ph.D
Professor & HOD, Dept of ECE
Gudlavalleru Engineering College
Andhra Pradesh, India

Ashok Kumar Yadav
Technical Manager
Electronics Corporation of India Ltd
Hyderabad, AP, India

ABSTRACT
Embedded Systems are designed for a specific task based on characterization. But in the modern advancements these are doing several tasks at a time, to achieve this it requires an operating system along with powerful processor. In this paper proposes the Board Support Packages (BSP) customization of Embedded Linux OS, especially ARM9 based Freescale Silicon Vendor platforms with the help of Linux Image Target Builder (LTIB). The successful build the operating system will give the Binary images of custom OS. Finally the images are ported to the target platform.

Keywords
Board Support Packages, Bootloader, Embedded Linux, file system, Kernel, LTIB, Porting.

1. INTRODUCTION
An Embedded system is application oriented special computer system which is accessible on both software and hardware. It can satisfy the strict necessity of functionality, consistency, cost, size, and power consumption of the specific application. With the extremely fast development of IC design and manufacture, CPUs became inexpensive. Lots of consumer electronics have embedded CPU and thus embedded systems became more popular. For example, Tablets, Phablets, point-of-sale devices, industrial control, or even your washing machine can be embedded system. There is a greater extent demand on the embedded system market. According to the present scenario, the demand on embedded CPUs is more times as large as general purpose CPUs. As applications of the embedded systems become more multifaceted, to build the operating system and preparing development environment became crucial.

Figure 1 shows the layered architecture based upon the OS directory structure, and also indicates the how the application in the device accessing the hardware. The main concentration is only the Board Support Packages. It depends on the architecture of that OS. If the architecture is ARM, then the corresponding will be created according to the target platform. The BSP is in detailed as follows.

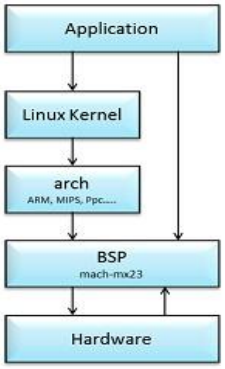

Figure 1: Layered architecture (Based on OS tree)

Board Support Packages *(*BSP) is a collection of the binary, code, and support files that can be used to create a Linux kernel firmware and file system images, for a particular target. In other words a Board Support Package (BSP) is an implementation specific support code for a given board that conforms to a given operating system. It has commonly had a boot loader that contains the minimal device support to load the operating system and device drivers for all the devices in a target system. The detailed BSP layered version is shown in Figure 2.

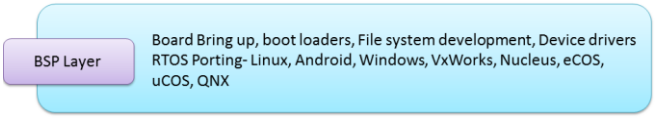

Figure 2: Specific BSP layer

The tasks which are performed by the BSP are to initialize the processor, bus, interrupt controller, clock, RAM settings and configuring the segments. To end with load and run the boot loader from flash or SD card.

2. ARM9 BASED FREESCALE APPLICATION PROCESSOR PLATFORM
The Freescale Silicon Vendor's application processor dependent upon the ARM926EJ-S can run at speed up to 454 MHz and is focused for expense sensitive consumer provisions. The Power administration unit (PMU) integrates a DC-DC switching converter and different direct controllers to give control sequencing for the gadget itself and drive I/o peripherals, for example memories and SD cards and in addition furnishing electric storage device charging capability

for Li-Ion electric cells. The i.mx2xx additionally incorporates blended indicator simple sound with a 1.5w Mono speaker enhancer, a stereo earphone DAC with 99dB Signal-to-Noise ratio and stereo ADC with 85dB Signal-to-Noise ratio with integrated amplifiers.

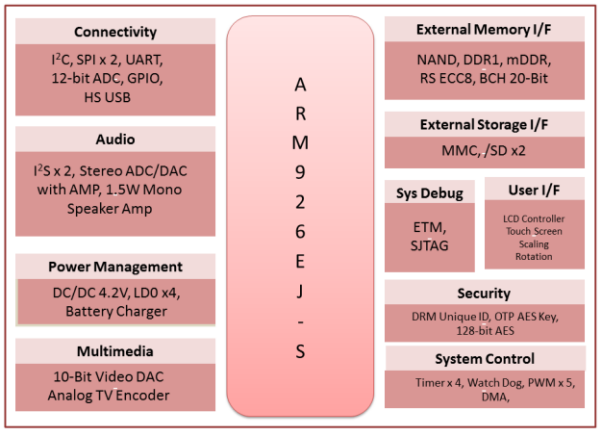

Figure 3: Block Diagram of ARM9 Based Freescale Application Processor Platform

3. THE EMBEDDED OPERATING SYSTEM

The term Operating system is referred to as; it's a special code that acts as an intermediate between the hardware and the user [1]. The main goals of the operating system is to make the system is convenient to use (Hiding the hardware details) and utilizing the resources in efficient manner [2]. The following are the most important factors to choose an Embedded Operating System [3].

Full source availability, Technical support, real-time performance, compatibility, customizable, open source, the processor it supports, purchase price, simplicity/easy to use, availability of the software development tools, small memory footprint, middleware/software/drivers and finally it is also supports the other architectures also. The layered architecture of basic Embedded Linux is shown in the Figure 4. The main variance between the normal Desktop OS and the Embedded OS is the Table. 1 gives an in detailed explanation.

The operating system can be divided into three modules. They are Bootloader, Kernel, Filesystem.

Boot Loader is an initializing code for a particular board, which is executed at the power on or reset the board. Here the proposed boot loader is U-Boot boot loader. To boot the Linux, the boot loader has to load the modules into the memory, one is the Linux kernel and another one is the file system [4].

Kernel is a software layer that interfaces with the hardware. It is responsible for interfacing all peripherals that are currently connected to the system and running in "user mode" down to the physical hardware, and allowing processes, to get information from each other using inter-process communication.

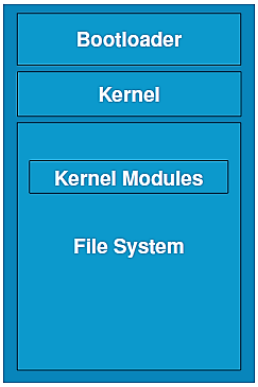

Figure 4: Basic Embedded Linux Structure

File system is the way in which files are named and where they are placed logically for storage and retrieval. The DOS, Windows, Macintosh, and UNIX-based operating systems all have file systems in which files are placed somewhere in a hierarchical (tree) structure. A file is placed in a directory or subdirectory at the preferred place in the tree structure. File systems require conventions for naming files. A file system also includes a format for identifying the path to a file through the structure of directories.

4. BUILDING THE LINUX PLATFORM

This section incorporates to set up the building environment, install and run the LTIB, and finally generate the output binary files to prepare the SD card bootable image compatibles. The main objective of this paper is to make the Embedded Linux OS according to the target platform; in this instance, it is the I.MX2xx application processor platform, which is developed by the Freescale Semiconductors. The required components to build the OS are bootloder, kernel and file system, Of course the development of the Operating System image individually by selecting the bootloder, kernel and file system, but it is very tedious job to do such kind of selection, as per time to market constraint, vendors are developing the target image builders. In this paper proposed building tool is the Linux Target Image Builder (LTIB). The LTIB project is a tool that can be used to develop and deploy the Board Support Packages for various target platforms incorporating with Power PC, ARM and Coldfire architectures.

LTIB has been released under the terms of the GNU General Purpose license (GPL). In general it is a lightweight command line interface, which is used to control the scripts and configuration and also perform the function like building the kernel, bootloader and application packages from the source, preparing the appropriate kernel, RFS images for flash based devices like SD card or other MTD (Memory Technology Devices) devices, capture the source modification patches and auto update the .Spec files and it also manages the changes to a package of the user generated patches. LTIB consists the bundle of packages normally consists of main archives along with Patches and the file formats are the .tar.bz2/.tar/.bz2 + .Patch extension, and these are all located in one of the 3 package pools data flow is shown in Figure 5.

Table 1. Variance in Embedded OS and Desktop OS

S.No	Embedded OS	Desktop OS
1.	Scalability	Less scalable
2.	Real-time performance	Not real-time capable, or limited real-time capability.
3.	Board Architecture support	Board application support
4.	Many Features like Networking, Filesystem, USB, WiFi and GUI support etc. and also implementation is possible by adding the third party libraries	The largest number of developers are available
5.	Separate development host environments required	Same Development and target environments
6.	Specific choice of development tools and languages	It has a wide choice of language tools
7.	Executable footprint is less in size	Large footprint
8.	Examples are Symbiyan OS, I OS, Embedded Linux, Android, Palm OS, Windows CE, etc.	Windows, Unix/Linux, Mac OS, etc.

Private Package Pool: which is located in the inside the Freescale network. The original archives and patches are kept inside it and all these are private accesses but it can be within the Freescale network local machines.

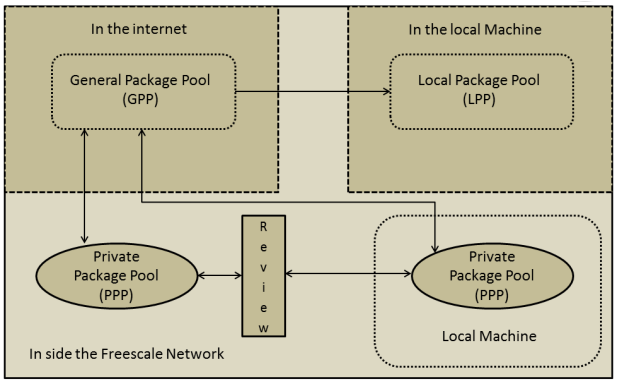

Figure 5: LTIB Package Pool data flow

General Package Pool: This is nothing but the packages are accessed through the internet, it is external to the Freescale network but it is sub-set of the Private Package Pool. All he packages within the GPP are the public content and having a suitable license for copying.

Local Package Pool: This but our local directory on the machine where the packages are cached packages and patches downloaded from the GPP, not only that and also add the additional packages to the LTIB during the development. The downloaded open source packages can share with the local users also.

Steps to be followed:

1. Setting up the building Environment
2. Host Dependencies
3. Installing and Running LTIB
4. Configuration changes
5. Building Manufacture firmware
6. Preparation of Boot stream images

Step1: Setting Up the building environment

This section includes preparing the host machine to build the LTIB; the supported operating systems are the any Linux distributions such as Fedora 4/5, Redhat, or Ubuntu 8.0 on words. To build the LTIB all the resources are directly downloaded from the General Package Pool (GPP). As per earlier discussion all archives are in the format of tar.bz2/.tar/.bz2 compression format. To uncompress this *tar* package must be installed in the host.

root# sudo apt-get install *required packages*

During the development of the image builder lot of packages must be installed according the requirement. Install the corresponding repositories by using the above command.

Step 2: Host Dependencies

To install Linux Target Image builder, the host system having the following packages and Libraries like **Perl**: To run the LTIB script. **sudo:** To run the rpm-install packages with administrative permissions. **Wget:** to download the packages/patches on demand. **rpm-build:** to build the packages. **glibc:** to build or run the host packages. **libstdc++-devel:** having the header files for C++ development, **binutils:** programming tool set for creating and managing the binary programs, obj files, asm files..etc. **gcc:**(GNU C compiler) built- in versions of many of the function in the standard library. **zlib-devel:** it is zlib compression and decompression library**. ncurses:** library provides an API to the programmer to display the text based interface, which is terminal independent. **Bison:** it is parser generator**. Flex:** (Fast lexical analyzer) it is a tool for generating the scanners (is a program which recognizes lexical patterns in the text and generates the executable file) **libtool:** it adds the new generic library building commands to the Makefile. **gettext:** utilities are a set of tools that provides a framework within which other free packages may produce multi-lingual messages. **Textinfo:** it provides a way to easily typeset software manual.

Step 3: Installing and Running LTIB

To install LTIB enters the following commands under the uncompressed source path. Please don't install LTIB under *root user*, because it may corrupt your host file system. At first time the LTIB will build and install the host packages, it will take some time. Then run the LTIB script within the appropriate directory, it will ask End user License agreement (ELUA) simply press *yes*, and specify the proper path where

to install it. When installation is complete, you can find the directory named as *Freescale* in the path */opt/Freescale,* in this directory it installs the all the required packages.

```
host@name#./intall
```

The corresponding next step will be the configurations to the appropriate target platform. The host prompts for you like the fig6, select the platform and exit, to save the changes according to the selection. In the meantime all the platform dependent files will be compiled and configured to the particular platform, the subsequent GUI screen will prompt you, here select the *kernel version, toolchain selection,* and also must set the *boot options, target image type.* Also select the required packages from the list specified in it. Like *a busy box* (containing the basic commands in the Linux which works on the target board), if suppose you write an application in Qt or GTK project, so you must have to select the supporting libraries also. After setting all these things select exit and the save the settings. This building process is not only follow the I.Mx23 but also it supports the different platforms of Freescale architectures.

Step 4: Configuration Changes

To fit the Embedded Linux on the hardware platform, the configuration must be changed according to the type of application, for instance take a Digital Camera [5] after switch on the device, at first initialize the LCD, camera lens and opening the shutter. So that drivers must be included but default kernel configurations having the Ethernet also, it's not required in this case. Like that according to the necessity of the application, the corresponding configurations changes may require. It also effects the boot time of the device [6].

In the kernel some of the configuration changes may required to the target device. it's depends on the application running on that device and it effects the final footprint of the binary image. Here some of the configurations like in the Figure 6, in those some are 'Enable the loadable module support', System type: select imx233, Boot options: This is the one of the significant configuration, for example ***console=ttyS0 115200n8:*** For displaying log screen while booting the board with the baud rate of 115200, ***rootwait:*** for detecting the devices asynchronously like USB or MMC, ***rw:*** mount root device read write on boot, ***initrd:*** to specify the location of the initial ramdisk, ***rootfstype:*** to select the type of root filesystem [10].

It depends on the target boot device, if the device is an SD card, it supports Extended filesystem (ext2) or it may be the NAND flash it supports Journaling flash file system (jffs2). lcd_pannel=lms430: which shows the type of LCD used and its resolution. The configured drivers are the Memory Technology device drivers (MTD), Block Devices, I2C Support, GPIO Support, Multimedia support, USB Support, MMC Card support, Real time clock, Sound card support, Power supply Class support, Watchdog timer support. These are the required drivers for customizing. The selection of options either <*> or <M>, modules as per our requirement [7]. Figure 6 represents the configuration or the kernel.

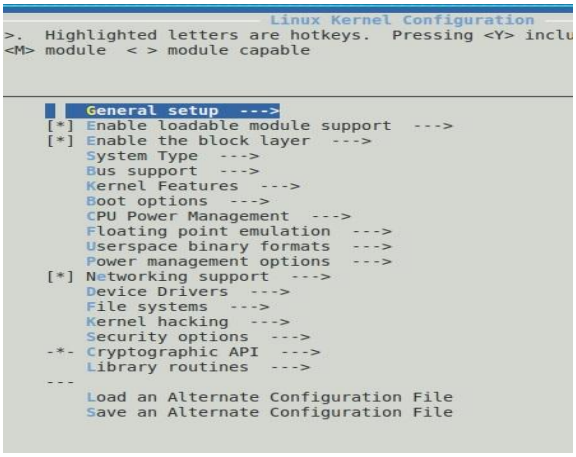

Figure 6: Kernel Configuration Menu

The Figure 7 shows the successful building the LTIB, the terminal displays above message, also it shows the time of the created with date and time elapsed.

Step 5: Building manufactures firmware:

Before starting the building firmware, the original customization is done here, like for instance USB (Universal Serial BUS), it acts as host, at the same time also act as a device. The advanced feature of the USB is OTG (On-The-Go) support means, depending upon the peripheral connected to it acts as either host or device. At last establish the communication path between the main processor and the peripheral's controller.

```
Filesystem stats, including padding:

    Total size           = 13080k
    Total number of files = 1178

Your ramdisk exceeds the old default size of 4096k, you may need to
set the command line argument for ramdisk_size in your bootloader
allowing 10% free this gives 14388k .  For instance, for u-boot:


setenv bootargs root=/dev/ram rw ramdisk_size=14388

creating an ext2 compressed filesystem image: rootfs.ext2.gz
genext2fs: Running in LTIB backwards compatibility mode: -i -> -N

Started: Sat Jul 27 10:47:16 2013
Ended:   Sat Jul 27 10:49:42 2013
Elapsed: 146 seconds

Build Succeeded

npr@npr-desktop:~/Videos/ltib$
```

Figure 7: successful build LTIB Message

So to do that customizing the board support packages like Original Equipment Manufacturer (OEM) file nothing but the ***mach files.*** The location of this files are in the Linux kernel file system tree under the path */arch/arm/mach-mxxx* directory (here platform is imx233 so it is named like this). This directory has the complete set of hardware registers along with the Makefile. For instance it includes one stratup.s file is an asm file format, having the entire address location of the registers, in those some of the registers are irqs, regs.dram, regs.lcdif, regs.gpmi, regs.emi, regs.power, regs.pinctrl,..etc. All these files defined as header files located in the 'include' directory. The Linux kernel supports not only the arm architecture but also it supports the no 'of architectures like

MIPS, SPARC, PowerPC Nexell, AVR, CRIS, Microblaze and so forth.

All these architecture's files and directories are located under the ***arch*** directory. And its sub directory having the all supported files with their default platform dependencies and its corresponding makefiles. The Main advantage of the image builder is that, there is no need to write the entire code for every platform. It's having the similar code, i.e. simply by selecting the similar code which is closer to our target board and by simply re-modifying the code according to our target board. It full fills the one of the most important constraint in the embedded system design i.e. ***Time to market,*** because of there's no requirement to write the entire code again and reusing the existing code, that's why, particular product can deliver into the market in time. This is the advantage of doing the Board Support Package customization.

Step 6: Preparing bootstream images:

The iMXxx application processor's SoC (System On Chip) contains a built-in ROM firmware capable of loading and executing binary images in special format from different locations including MMC/SD card and NAND flash. Such a binary image is called a boot stream and consists of a number of smaller bootable images (bootlets) and instructions for how the ROM firmware should handle these bootlets (e.g. load a bootlet to On-Chip RAM and run it from there). To build the Linux kernel and new boot stream images, the corresponding kernel parameters must pass through the kernel.

Actually u.boot.bin and the kernel image will be combined to gather and generates the imx_linux.sb file. In general this file contains the three preparation file along with the zImage (Linux kernel). The Figure 8 shows the way of calling kernel image.

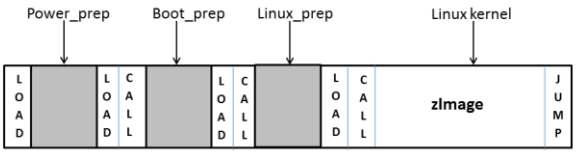

Figure 8: Bootstream images loading kernel

power_prep: this bootlet configure the power supply.

boot_prep: this bootlet configure the clock and sdram.

linux_prep: this bootlet is to prepare the kernel to boot.

5. RESULTS

The successful building the LTIB generates a file, which is an encrypted file that is bootable on i.mx2xx (i.e. ARM9 based development board). It can be found in the following path.

#ltib_directory/rootfs/bin/imx_linux.sb. This file contains the second stage bootloader and the Linux kernel. The size of the kernel will be 2.79MiB. According to the research the size of the kernel will be less than 3MB is accurate. Figure 9 shows the kernel image of custom OS and its version is 2.6.35 and the generated file path.

```
Image Name:   Linux-2.6.35.3-433-g0fae922
Created:      Wed Aug 28 13:30:56 2013
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2402176 Bytes = 2345.88 kB = 2.29 MB
Load Address: 0x40008000
Entry Point:  0x40008000
  Image arch/arm/boot/uImage is ready
```

Figure 9: Kernel Image

The remaining is the root filesystem which can be found in the #ltib_directory/rootfs/rootfs.ext2 directory. Figure 7 shows the filesystem generation and the size of it. The size of the final firmware effects the boot time of the target device. Finally proved that the size of the firmware is very small footprint so the boot time is also improved.

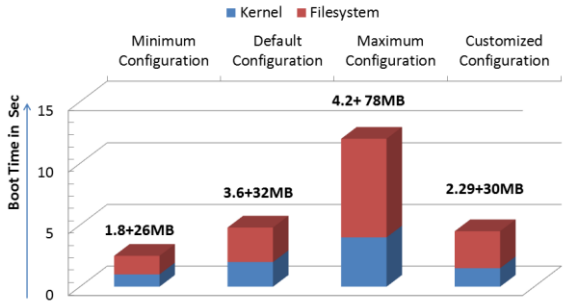

Figure 10. Kernel and Filesystem Footprints Vs Boot time with different configurations

In general the memory footprint of the operating system affects mainly the boot time and also increase the power consumption, cost of the BOM (Bill Of materials) The Figure 10, represents the footprints of the kernel and filesystem sizes with different configurations, here four generalized configurations like minimum configuration means it is having the basic kernel and the filesystem, it requires low memory space, so less time required to boot the target. Default configuration: in this some of the additional components are added to the minimum configuration like USB, NFS, network, etc. so the time taken to boot the target device is more compared to minimum configuration, because of driver modules need some time to load and initialize the target device. Maximum Configurations: it allows the all functionalities ans device drivers are added the OS, so along with the footprint size and boot time is also increased.Finally Customized configuration: Here all the usage of resources are limited i.e the selection for the responsibilities of the kernel is as per the requirement of the target application, sometimes additional drivers re also patched o the kernel depends on the requirement, so this paper proposed Customized OS size is 2.29MB. The final size of the kernel and the file system is 32.29MB, so target device boot time is also very less.

Porting Concept: In order to port the embedded Linux to the target board, the following steps will give an overview [8]-[9].

1. Download the image builder and run it as mention earlier.
2. Patch the Linux kernel with the corresponding platform architecture.
3. Select the appropriate *toolchain* and configured it.

4. Build the Linux target image builder, until it will succeed, with proper configurations.
5. Create the boot stream images like u.boot.sb, uImage.sb, rootfs.ext2.
6. Select the suitable boot mode either SD card or NAND flash (It depends on the target filesystem type in the kernel configurations)
7. For simplicity the selected boot mode is the SD card. Preparing, configuring and flashing the SD card is as follows.

5.1 Preparing the SD card:

Whenever you boot your target device, Manually specify where the binary images are located, and where the filesystem is mounted, it may be read or write, serial console log message and so on. The right place to mention this path should be kernel command line. This concept is discussed earlier in the kernel configuration section. In this section, simply place the firmware on partition 1 and the roofs on partition 2. Assuming you are using the LMS430 touch screen display for the target board, as per the above discussion in the kernel configurations, here set the boot options [10].

To create the SD card bootable images into the SD card using the linux_imx.sb file, then follow the two steps below. (Assuming Linux host having the package with ***DD*** command)

```
#dd if=/dev/zero of=firmware.img bs=512 count=4
```

1. By using the above command, the starting of the raw file, it inserts the 4 blocks and 512 Bytes are filled with zeros. Here call it as firmware.img, however it is name it as whatever you want.

```
# dd if=rootfs/boot/imx233_linux.sb of=mx.img ibs=512
seek=4 conv=sync,notrunc
```

2. It appends imx_linux.sb file to the next section after the zeros.

5.2 Flashing the SD card with Bootable images:

Finally flash these bootable images to the SD card but built alone and flashing is not sufficient to boot Linux. As previous mentioned the SD card will be divided into two partitions, the first partition is flashed with Linux firmware and the second will be the filesystem and also create the third partition (FAT32) also for storage purpose. Before going that insert the card into the card reader in the host. The partitions should be done by using the *fdisk* utility in the host system, in order to boot the Linux on the target; alter the partition table in the card. So the altered partition formats are *OnTrack DM6 Aux3*, which is the first one and another one will be the filesystem in general it ought to be extended file system i.e. *Linux-ext.* Then flash these bootable images to the card. Develop any application in the Qt creator (SDK) according to the target platform. The generated binary executable in the Qt SDK is simply ported along with firmware on to the target.

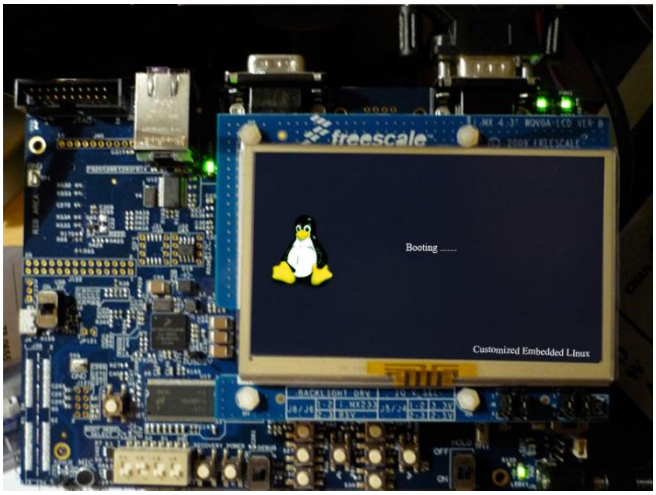

Figure 11: ARM9 Based Application Processor Platform with Customized boot logo.

After porting the custom images to the target platform it displays a custom boot logo. The Figure 11 shows the boot-up the target platform with custom build OS images.

6. APPLICATION DEVELOPMENT

The final task is to develop a Linux application. The Software Development Kit (SDK) is the Qt Creator, which is developed by Nokia [11]. It is a cross compiled platform. Here there is a possibility to develop a Good looking Graphical User Interface (GUI) to any platform. Qt's new user interface technology consists of the QML language, that provide basic building blocks of the language, the Qt Declarative (C++) module that provides a runtime for the QML language, and a scripting language based on JavaScript that allows developers to implement logic in their projects with no C++ coding required. This technology set is supported by the Qt Creator tool that now includes a visual designer and other extensions to support creating, testing, debugging, and optimizing projects. Qt is also having the Add-on's for Visual Studio and Eclipse IDE. The initial release of the Qt is in May 1995, and the updated version 5.1 is released on 3rd July 2013.

7. CONCLUSION

Embedded Linux OS has been successfully applied to the embedded field with its powerful functions, open source and also several advantages like, it needs very little memory, which is very flexible and small. It's also having networking capability, so target devices can be controlled over the network. Features and drivers can be added during the kernel runtime as loaded modules. By doing this the size of the OS can be reduced and the boot time is also being improved. The cross compiled binary images or OS binaries are generated by using the Linux target Image builder (LTIB) as on x86 based system. The main advantage of BSP customization is to create a solid software foundation for specific application, it accelerates the designing of Embedded device development projects, it is very easy to add the additional functionalities like scripts, drivers and other components to the platform according to the end product's application by patching to the LTIB, easy to integrate the modifications in the kernel without affecting the existing functionality of the system. Before going to port the OS binaries, include the GUI application to calibrate the better system performance.

8. REFERENCES

[1] 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.

[2] De Goyeneche, J.-M, De Sousa, E.A.F, "*Loadable Kernel Modules*", Software IEEE, Vol16, Issue1, pp:65-71, Jan/Feb- 1999.

[3] Hu Jie ; Zhang Gen-bao, "Research transplantation method of embedded linux kernel based on ARM platform", *Information Science and Management Engineering* (ISME), 2010 International Conference, Vol2, pp:35-38, 7-8. Aug 2010.

[4] The DENX U-Boot and Linux Guide, available at www.denx.de

[5] Chanju Park, Youngjun Jang, kyungiu Hyum, Kyungiu Kim," *Linux Bootup Time Reduction for Digital Still Camera*", Proceedings of the Linux Symposium,Vol 2, 2006.

[6] Vincent Sanders, "Booting ARM Linux", rev1.1., june,2004.

[7] De Goyeneche, J.-M, De Sousa, E.A.F, "*Loadable Kernel Modules*", Software IEEE, Vol16, Issue1, pp:65-71, Jan/Feb- 1999.

[8] Divya Sharma, Kamal kanth, "Porting the Linux Kernel to Arm System-On-Chip And Implementation of RFID Based Security System Using ARM", *International Journal of Advanced Research in Computer Science and Software Engineering* (IJARCSSC), Vol3, issue5, May-2013.

[9] Jyotsana Thaduru, B. Narasimhachary, "Porting the linux kernel to ARM platform", *International Journal of VLSI and Embedded Systems-IJVES* http://ijves.com ISSN : 2249-6556. Vol3, Sept-oct 2012

[10] Kernel parameters list available in the kernel Documentation and Available at kernel.org

[11] Alan Ezust, Paul Ezust, "Introduction to Design Patterns in C++ with Qt (2nd Edition)", **ISBN :** 0-13-282645-3 / 978-0-13-282645-7.

[12] Pravin, S ; Balakrishnan, R, "Set top box system with android support using Embedded Linux operating system paper", Advances in Engineering, Science and Management (ICAESM), 2012 International Conference, pp: 447-478, 30-31 March 2012

9. AUTHOR'S PROFILE

K. Eswar Kumar obtained his Bachelor's degree from Rao & Naidu Engineering College, Ongole. His areas of interest are Microcontrollers, Microprocessors, Embedded System Design and RTOS. Presently he is doing M.Tech in Embedded Systems at Gudlavalleru Engineering College, JNTU Kakinada, Andhra Pradesh, India.

M.Kamaraju obtained his Bachelor's Degree & Master's degree from Andhra university and Ph.D from JNTUH, Hyderabad in the area of Low Power VLSI Design, Areas of interest are Microprocessors, Microcontrollers, Digital system Design, Embedded System Design, Low Power VLSI Design. He published 51 technical papers in national/ international journals/conferences. He reviewed number of papers for international journal and conferences. He is a Fellow of IETE and IE and member of IEEE. Presently working as Professor& Head of ECE Department, Gudlavalleru Engineering College, Gudlavalleru, India and chairman of IETE, Vijayawada center.

Ashok Kumar Yadav is working as a Technical Manager at Electronics Corporation of India Limited (ECIL), Hyderabad, since July 2000. He has over 11 years of experience in embedded system design and development. His main areas of interest are digital signal processing, FPGA, RTOS, and biometric applications. He has master degree in Digital Signal Processing from Osmania University, Hyderabad, India and Master of Business Administration (MBA) from IGNOU.