Journal of Computer Science and Applications.
ISSN 2231-1270 Volume 7, Number 1 (2015), pp. 11-19

http://www.irphouse.com

Customization of U-Boot for TFTP and Flashloader

Siddharth Narayanan and Sameer Kadam

Research Center Imarat, DRDO Hyderabad, India 500069
Scientist-E, Research Center Imarat, DRDO Hyderabad, India 500069

Abstract

This paper proposes a booting scheme for U-Boot using TFTP and Flashloader for booting in cases where repeated recompilation of the kernel is required and loading from an external memory card becomes tedious. This is especially helpful with a network connection available, where U-Boot can load files quickly and easily via TFTP. TFTP has been cross compiled for the target board based on Power Pc 7410, and inserted into board kernel. The mknod functionhas been used to create Device interface files and application for scheduling message for booting target Board.The study uses an embedded Linux development kitthat includes the GNU cross development toolsand libraries necessary to provide some functionality on the target system. The kit is available in two versions, which use *Glibc*and *uClibc*as the main C library for target packages.Packaging and installation is based on the RPM package manager. The result of the study is observed in the form of successful scheduling on the Bus through an Extra Monitor Terminal.

Keywords- U-Boot, TFTP, Flashloader,Kermit, Embedded Linux Development, ELDK Configuration

I. INTRODUCTION

A. U-boot

The U-Boot is an open-source, cross-platform boot loader that provides out-of-box support for hundreds of embedded boards and many CPUs, including PowerPC, ARM, XScale, MIPS, Cold fire, NIOS, Micro blaze, and x86. The U-Boot project is hosted by DENX. The current version of the U-Boot source code can be retrieved from the DENX Git repository.The trees can be accessed through the Git, HTTP, and rsync protocols. Obtaining a copy of the U-Boot sources from the Git repository provides an unpacked directory tree, while the DENX FTP server contains a compressed tarball [1].As also adopted by the Linux kernel, Device tree is intended to

ameliorate the situation in the embedded industry, where a vast number of product specific forks (of Das U-Boot and Linux) exist.Device tree is a data structure for describing hardware layout. U-boot provide the basic infrastructures to bring up a board to a point where it can load a Linux kernel and start booting your operating system.

B. Trivial File Transfer Protocol

The Trivial File Transfer Protocol (TFTP) is a simple, lock-step, file transfer protocol that allows a client to get from or put a file onto a remote host. One of its primary uses is in the early stages of nodes booting from a Local Area Network. There are a few noteworthy points for a terminal connected to the board and sitting at a U-Boot prompt. The file will transfer in binary mode and the location of the file is relative to the root of the TFTP server.The IP address of the board and TFTP server IP address are controlled by the U-Boot environment variables. Any file can be loaded to any address that is accessible to embedded processor.TFTP iseasily implemented by small footprint code.

It is therefore the protocol of choice for the initial stages of any network booting strategy like BOOTP, PXE, BSDP, etc., when targeting from highly resourced computers to very low resourced Single-board computers (SBC) and System on a Chip (SoC). It is also used to transfer firmware images and configuration files to network appliances like routers, firewalls, IP phones, etc. Today TFTP is virtually unused for Internet transfers.

C. Kermit

Kermit is the name of a file-transfer and -management protocol and a suite of computer programs for many types of computers that implements that protocol as well as other communication functions ranging from terminal emulation to automation of communications tasks through a high-level cross-platform scripting language. The software is transport-independent, operating over TCP/IP connections in traditional clear-text mode or secured by SSH, SSL/TLS, or Kerberos IV or V, as well as over serial-port connections, modems, and other communication methods. Kermit is used for SSH sessions from the desktop to CUNIX, and by the technical staff for system and network administration tasks [2]; for example, configuring racks full of HP blade servers as they arrive, management of the University's telephone system, CGI scripting, alpha paging of on-call staff, and so on. Plus, of course, by old-timers who just plain prefer the safety and efficiency of text-mode shell sessions for email and to get their work done; for example, software development and website management. All popular Kermit programs can make TCP/IP network connections (clear-text or secured), direct serial-port connections, and dialed modem connections, and can accept incoming connections of all these types. They can also conduct interactive terminal sessions.

The Kermit protocol has developed into a sophisticated, powerful, and extensible transport-independent tool for file transfer and management, incorporating, among other things:

1. Transmission of multiple files in a single operation and File attribute transmission (size, date, permissions). File name, record-format, and character-set conversion, and File collision options, including an "update" feature.
2. File transfer recovery (resumption of an interrupted binary-mode transfer from the point of failure) and Auto upload and download along with Client/Server file transfer management
3. Automatic per-file text/binary mode switching during file-group transmission and Recursive directory-tree transfer, even between unlike platforms
4. Uniform services on serial and network connections and Internet Kermit Service Daemon

This paper is organized as follows. In Section 2 a brief overview of the embedded Linux development kit is given. Section 3 discusses the use of proposed tools for a more precise and automated means of handling boot loading operations. Section 4 describes the implementation and the results are presented in Section 5. Section 6 gives an insight into its application. Section 7 provides the conclusion and in Section 8 references follow

II. RESEARCH ELABORATIONS

A. Embedded Linux Development

Stable versions of the ELDK are distributed in the form of ISO images. Linux device driver (Linux host only).For the Power Architecture® target, the ELDK distribution was split into three independent ISO images: one targeting the 4xx family of processors (AMCC), one targeting the ppc64 family of processors and another one for the 8xx, 6xx, 74xx and 85xx families (Free scale). This makes the ISO images fit on standard DVDROM media. Development versions of the ELDK are available as directory trees so it is easy to update individual packages.

The ELDK contains an installation utility and a number of RPM packages [3], which are installed onto the hard disk of the cross development host by the installation procedure. The RPM packages can be logically divided into embedded Linux development tools and target components. The first part contains the cross development tools that are executed on the host system. Most notably, these are the GNU cross compiler, binutils, and gdb. The target components are pre-built tools and libraries which are executed on the target system. The ELDK includes necessary target components to provide a minimal working NFS-based environment for the target system.

The ELDK contains several independent sets of the target packages, one for each supported target architecture CPU family. Each set has been built using compiler code generation and optimization options specific to the respective target CPU family [4].If one or more CPU family parameters are specified, only the target components specific to the respective CPU families are installed onto the host. If omitted, the target components for all supported target architecture CPU families are installed.

The components should be installed in a directory with write and execute permissions. The installation process does not require super user privileges. Depending on the parameters the install utility is invoked with, it installs one or more sets of target components. If the user intends to usethe installation as a root file system exported over NFS, then the packages must be rebuilt.The ELDK has an RPM-based structure. After installation, the ELDK maintains its own database which contains information about installed packages. The RPM database is kept local to the specific ELDK installation, which allows multiple independent ELDK installations on the host system.Also, this provides for easy installation and management of individual ELDK packages. It is crucial that the correct rpm binary gets invoked. In case of multiple ELDK installations, there may well be several rpm tools installed on the host system. The rpm utility is located in the bin subdirectory relative to the ELDK root installation directory.

The target components of the ELDK can be mounted via NFS as the root file system for the target machine.Some of the target utilities included in the ELDK, such as *mount* and *su*, have the SUID bit set. When run, they will have privileges of the file owner of these utilities. The distribution image contains a script, which can be used to change file owners of all the appropriate files of the ELDK installation to root. Super user privileges are requiredto run this script.

B. Flash Loader Port

The Flash Loader is a targetbasedapplication that programs a user application into the external Flash devicepresent on the development platform when using serial communicationprotocols and a terminal application such as a HyperTerminal. The target-basedis designed for thefollowing customer-driven scenarios:

1. When manufacturing a product requires download of an executable image into product
2. When updating an executable image after the product is released to their customer base (field upgrades)
3. When supporting other types of Flash devices is required. This support is made possible by merely replacing the External Flash Loader’s Flash driver library with the customer’s own version

III. PROPOSED CONCEPT

Many different tools are needed to install and configure U-Boot and Linux on the target system. Also, especially during development, interactionwith the target system is crucial. The host system must be configured appropriately for this purpose.To make full use of all capabilities of U-Boot and Linux as development systems, a serial console port is required on the target system. U-Boot and Linux can be configured to allow for automatic execution without any user interaction [5].There are several ways to access the serial console port on the target system, such as using a terminal server,but the most common way is to attach it to a serial port on the host. Additionally, a terminal emulation program is needed on the host system, which in this case is Kermit.

The name Kermit stands for a whole family of communications software for serial and network connections.Kermit executes commands in its initialization file,*.kermrc*, in its home directory before it executes any other commands, so this can be easily used to customize its behavior using appropriate initialization commands.

IV. IMPLEMENTATION

It is important to check that there are no build results from any previous configurations left in the source code directory. By default the build is performed locally and methods can be used to change this behavior. Flash memory is used as the storage device for the firmware on the board.A fast and simple way to write new data to flash memory is via the use of a debugger or flash programmer with a BDM or JTAG interface. In cases where there is no running firmware at all (for instance on new hardware), this is the only way to install any software.

A. Installation using U-Boot

If U-Boot is already installed and running on the board, it will be replaced by the newly downloaded image. However, the image must be erased first. Any error during this stage could lead to a dead board and it is therefore highly recommended to have a backup of the old, working U-Boot image.U-Boot uses a special image format when loading the Linux kernel or ramdisk or other images. This image contains (among other things) information about the time of creation, operating system, compression type,image type, image name and CRC32 checksums [6].The tool mkimage is used to create such images or to display the information they contain. When using the ELDK, the mkimage command is already included with the other ELDK tools.

To initialize the U-Boot firmware running on the board, a terminal is connected to the board's serial console port.The default configuration of the console port uses a baudrate of 115200/8N1 (115200 bps, 8 Bit per character, no parity, 1 stop bit, no handshake). This study uses Kermit as a terminal emulation program running on a Linux host system.In the default configuration, U-Boot operates in an interactive mode which provides a simple command line-oriented user interface using a serial console on port UART1. In the simplest case, this means that U-Boot is ready to receive user input. When a command is entered, U-Boot will try to run the required action(s), and the prompt for another command.

B. FLASH memory information

The command *flinfo* can be used to get information about the available flash memory. The number of flash banks is printed with information about the size and organization into flash "sectors" or erase-units. For all sectors the start addresses are printed; write-protected sectors are marked as read-only (RO). Some configurations of U-Boot also mark empty sectors with an (E). A bank is an area of memory implemented by one or more memory chips that are connected to the same chip select signal of the CPU, and a flash sector or erase unit is the smallest area that can be erased in one operation. With flash memory a bank is something that eventually may be erased as a whole in a single operation. This may be more efficient (faster) than erasing the same area sector

by sector.In U-Boot, flash banks are numbered starting with 1, while flash sectors start with 0.

The boot application image is stored in memory by passing arguments. During booting a Linux kernel, *arg* can be the address of an *initrd* image. When booting a Linux kernel which requires a flat device-tree, a third argument is required which is the address of the device-tree blob. The *bootm* command is used to start operating system images. From the image header it gets information about the type of the operating system, the file compression method used (if any), the load and entry point addresses, etc. The command will then load the image to the required memory address, uncompressing it on the fly if necessary. Depending on the OS it will pass the required boot arguments and start the OS at its entry point.The first argument to *bootm* is the memory address (in RAM, ROM or flash memory) where the image is stored, followed by optional arguments that will depend on the OS [7].Linux requires the flattened device tree blob to be passed at boot time, and *bootm* expects its third argument to be the address of the blob in memory. The second argument to *bootm* depends on whether an *initrd* initial ramdisk image is to be used. If the kernel should be booted with the initial ramdisk, the second argument is interpreted as the start address of *initrd* (in RAM, ROM or flash memory).When booting images that have been loaded to RAM (for instance using TFTP download) the user must be careful that the locations where the (compressed) images are stored do not overlap with the memory needed to load the uncompressed kernel. For instance, if a ramdisk image is loaded at a location in low memory, it may be overwritten when the Linux kernel gets loaded. This will cause undefined system crashes. The variables used are set to correct addresses for a kernel, fdt blob and an initrd ramdisk image.

Parameter name	Type	Allowed value	Default value
fname	String	-	[empty string]
fnameWrite	String	-	[empty string]

Figure 1*:*Provides a description of the configuration parameters for the Flash Loader component

C. U-Boot Environment Variables

The U-Boot environment is a block of memory that is kept on persistent storage and copied to RAM when U-Boot starts. It is used to store environment variables which can be used to configure the system. The environment is protected by a CRC32 checksum. This section lists the most important environment variables, some of which have a special meaning to U-Boot. These variables can be used to configure the behavior of U-Boot to the user's liking.

V. RESULTS

The designated serial port of the host is connected the port labeled UART1 on the target board. The terminal program is then started and power supply of the board is connected.

A. Power On

Any error messages at this stage would be harmless as the system has not been initialized yet, and will go away as soon as the environment variables have been initialized and saved.At first, the serial number and Ethernet address of the board must be entered. Special attention must be given here since these parameters are write protected and cannot be changed once saved (this isusually done by the manufacturer of the board). If there is something wrong with the parameters, the board must be reset and the activity restarted from the beginning. A simple way to store parameters permanently is through the *saveenv* command.

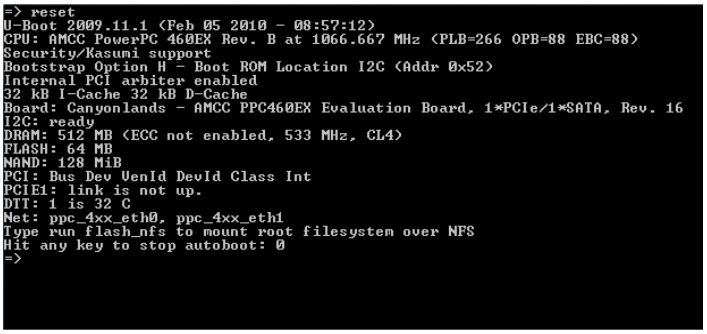

Figure 2*:*Message prompt should be displayed during the first power-on activity.

B. Setting Variables

To enter the data you have to use the U-Boot Use the variable name serial# for the board ID and/or serial number, and *ethaddr* for the Ethernet address.

Figure 3*:* Screenshot of the message prompt for setting environment variables. The command *setenv*, followed by the variable name and the data, all separated by white space (blank and/or TAB characters) is the required format.

C. Kermit script

A Kermit script is used to automate settings during the reset procedure and the following settings are recommended for use with U-Boot and Linux.

Figure 4*:*This example assumes that the first serial port of the host system (/dev/ttyS0) at a baudrate of 115200 is used to connect to the target machine's serial console port.

D. Sample Application

A sample Hello World demo application is also executed to demonstrate the functionality. It's configured to run at address 0x00040004 and is automatically compiled when U-Boot is built.TFTP can also be used to download the image over the network. In this case the binary image (*hello_world.bin*) is used.Note that the entry point of the program is at offset 0x0004 from the start of file, i.e. the download address and the entry point address differ by four bytes.

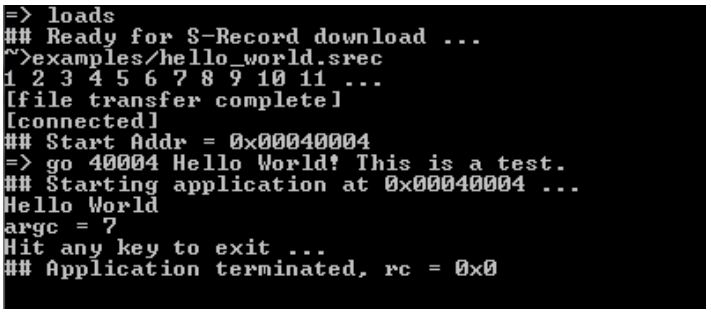

Figure 5*:*Screenshot of sample Hello World application that is configured to run at address 0x00040004

VI. CONCLUSION

TFTP was successfullyCross Compiled for Power Pc 7410 based board and inserted into kernel 3.0.8 on the target Board. The application was successfully driven into Target Board and results of scheduling on the Bus were successfully observed on Extra Monitor Terminal.The ELDK 4.1 was installed in Linux Kernel without errors and the U-Boot program for user space was compiled for Kernel Space in the form of a *u-boot.bin* file. The Kermit file transfer and management protocol was used for terminal emulation, automation of communication tasks and other communication functions through high-level cross-platform language scripting.

VII. REFERENCES

[1] Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, PhilippeGerum: "Building Embedded Linux Systems 2nd edition", Paperback: 462 pages, O'Reilly & Associates; (August 2008); ISBN 10: 0-596-52968-6; ISBN 13: 9780596529680 ISBN 059600222X Wachs, J.P., Kölsch, M., Stern, H., Edan, Y., Vision-Based Hand-Gesture Applications. Communications of the acm, 54 (2) (2011) 60-71.

[2] Greg Kroah-Hartman: "Linux Kernel in a Nutshell", 198 pages, O'Reilly ("In Nutshell" series), (December 2006), ISBN 10: 0-596-10079-5; ISBN 13: 9780596100797 Biswas, K.K., Basu, S., Gesture Recognition using Microsoft Kinect, Proceedings of the IEEE International Conference on Automation, Robotics and Applications (ICARA), Delhi, India, 6–8 December 2011.

[3] Craig Hollabaugh: "Embedded Linux: Hardware, Software, and Interfacing", Paperback: 432 pages; Addison Wesley Professional; (March 7, 2002); ISBN 0672322269

[4] Christopher Hallinan: "Embedded Linux Primer: A Practical Real-World Approach", 576 pages, Prentice Hall, September 2006, ISBN-10: 0-13-167984-8; ISBN-13: 978-0-13-167984-9

[5] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman: "Linux Device Drivers", 3rd Edition ; Paperback: 636 pages; O'Reilly & Associates; 3rd edition (February 2005); ISBN: 0-596-00590-31 Massa, B., Roccella S., Carrozza C., Dario P. (2002) Design and Development of an Underactuated Prosthetic Hand. Proceedings of the 2002 IEEE International Conference on Robotics and Automation, May 2002, 3374-3359.

[6] Jürgen Quade, Eva-Katharina Kunst: "Linux-Treiberentwickeln"; Broschur: 436 pages; dpunkt.verlag, Juni 2004; ISBN 3898642380 Smagt P. Grebenstein M. Urbanek M. Fligge N. Strohmayr M Stillfried G. Parrish J. and Gustus A. (2009) Robotics of human movements. Journal of Physiology – Paris 103, 119-132. doi: 10.1016/j.jphysparis.2009.07.009

[7] SreekrishnanVenkateswaran: "Essential Linux Device Drivers", 744 pages, Prentice Hall, March 2008, ISBN-10: 0-13-239655-6; ISBN-13: 978-0-13-239655-4