

# *Embedded Linux System & Applications*

**Ruth Gu**

**[ruth4jobs@gmail.com](mailto:ruth4jobs@gmail.com)**

**UUASC OC Chapter**

**May 12, 2008**

# OUTLINE

---

- Introduction
  - Some basic concepts
- Build Embedded Linux System
  - System Start Process
  - Embedded Target Board
  - Software on Embedded Target
  - Host Machine Setup
  - Make Your Own System
- Recommendations
- References

# I. INTRODUCTION

---

- **Some Basic Concepts**

- Linux OS
- Embedded System
- RTOS
- Embedded Linux System
- Linux and Real-Time (not included within this presentation)

# Linux OS

---

- Linux Operating System
  - Everyone knows ???!!!!!!
    - Yes or no?

# Embedded System

---

## ● Characteristics

- Designed for a specific application or purpose
  - Cell phone, Router, MP player, Set-Top-Box, Microwave ...
- Contain a processing engine
  - A general-purpose microprocessor, as CPU
  - Multi cores possible (dual-core, quad-core)
- Include a simple or no user interface (GUI)
  - GUI could become complicated for current high-tech applications
- Carry limited resource
  - A small memory footprint, no hard drive (generally)
  - Possible with hard drive, like large printer, network appliance, etc.

# Embedded System (cont.)

---

- **Characteristics**

- **Power limitations**

- Might be required to operate from batteries

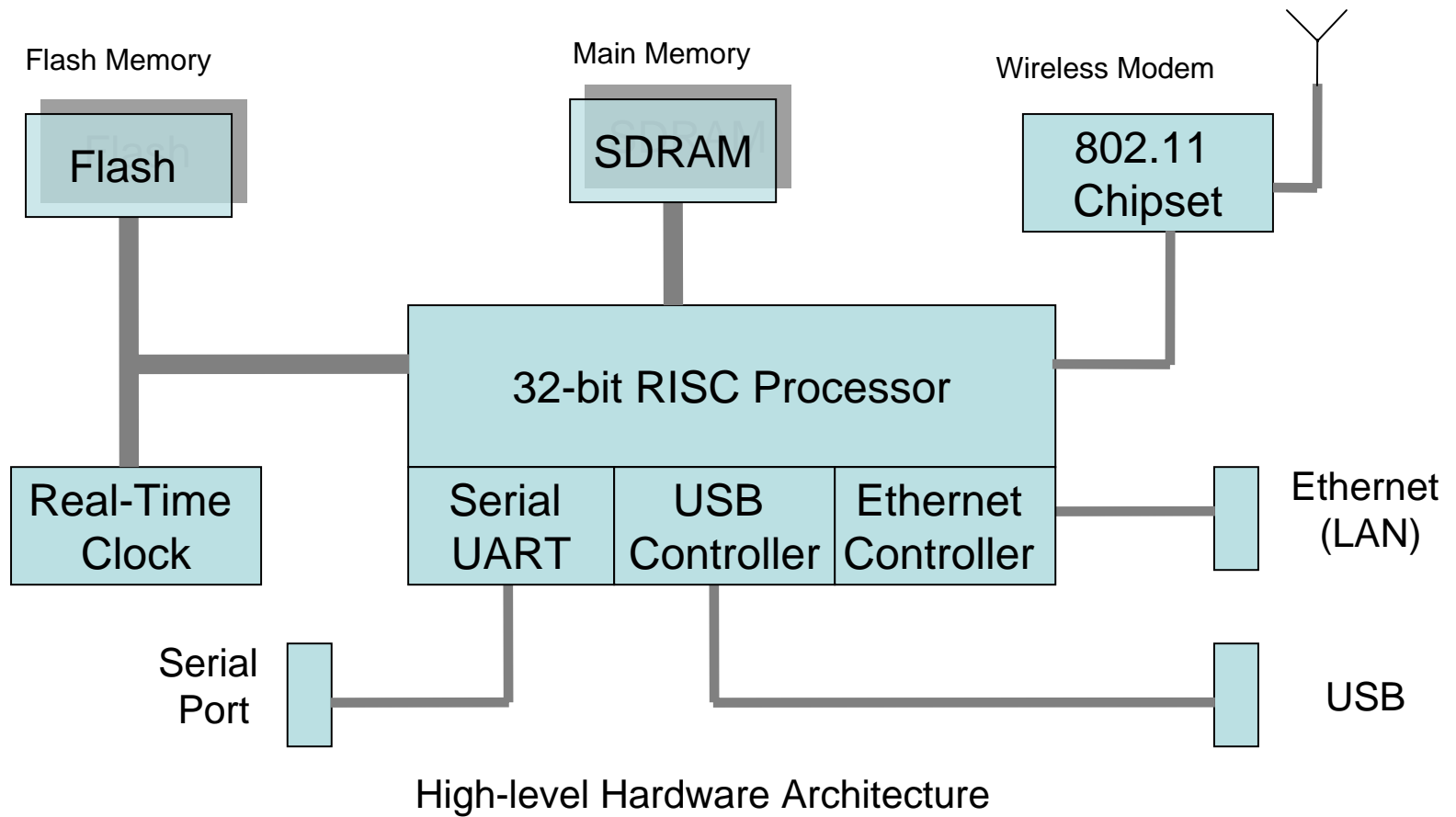
- **Shipped with all intended applications**

- Hardware & software pre-integrated
- Applications software build-in (not user selected)
- Not used as a general-purpose computing platform

- **Intended for a applications without human intervention**

- Could have HMI interface, depends on applications
- Some fancy GUI could provide certain interactive operations

# Typical Embedded System



# RTOS

---

- **Soft / Hard Real-time**
  - Soft RT (deadline pass)
  - Hard RT (deterministically)
- **Multi-task OS for RT Applications**
  - Embedded systems
    - Household appliance controllers, mobile phones, ...
  - Industrial Process & Applications
    - Robots, spacecraft, industrial process control system, ...
  - Scientific research equipment
    - Target tracking system, radar / image measurements, ...
- **Commercial Products**
  - VxWorks, QNX, Nucleus, etc., ...
  - Microkernel, more drivers loaded, ...

# RTOS (cont.)

---

- **Scheduling Algorithms**

- Even-driven: to switch tasks only when an event of higher priority needs service, it is “pre-emptive priority”
- Time-sharing: to switch tasks on a clock interrupt and on events, it is “round robin”.
- Interrupt Handlers & Scheduler

- **Inter-task Communication & Resource Sharing**

- Temporarily masking/disabling interrupt
- Binary semaphores
- Message passing

- **Memory Allocation**

- More critical: speed of allocation, fragmentation

# Embedded Linux System

---

- Differences

- Open Source vs. Commercial Products

- Windows CE, RTOS

- Embedded vs. Desktop Environments

- Memory: small foot print (4K – 256MB)
- Flash Memory: instead of Hard Drive, 2\*16MB
- Special Purpose Device ~ Generic OS Platform
- Limited GUI Feature ~ Rich GUI Applications
- Hardware Platform Dependent ~ unified (x86 Intel, AMD)
- BIOS ~ Bootloader

# Embedded Linux System (cont.)

---

- **Challenges**

- **Money Wise**

- Costs, Budgets, ROI (Return-On-Investment)

- **Limited Hardware Resource**

- Memory, Hard drive, Flash memory, etc.

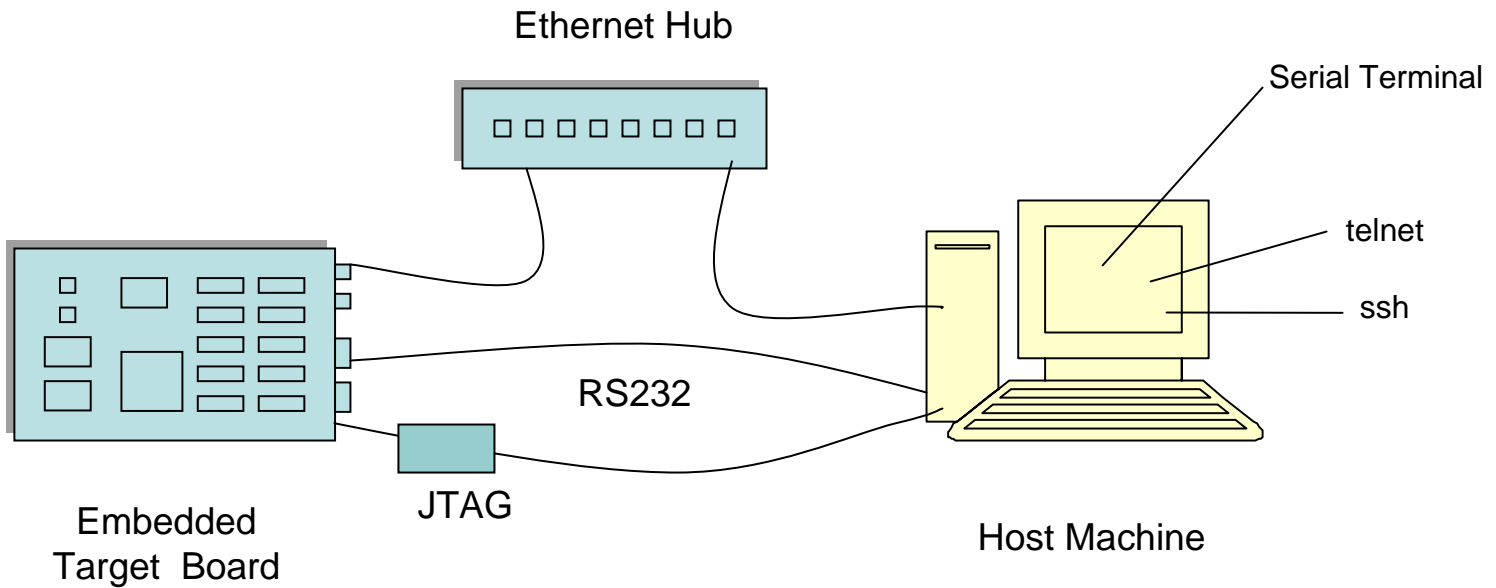
- **Maintenance**

- Firmware updates, technical supports, sustaining

- **Hardware Architecture Dependency**

- Select different SoC solutions
- Cross development environment

# Embedded Linux System (cont.)



Development Environment Setup

## II. BUILD EMBEDDED LINUX SYSTEM

---

- System Start Process
- Embedded Target Board
- Software on Embedded Target
- Host Machine Setup
- Make Your Own System

# System Start Process

---

- Power on to start target board
- Hardware initialization
- Booting kernel
- Kernel initialization
- First user space process (init)
- Application processes

# Embedded Target Board

---

- **Embedded Processor & Architecture**
  - PowerPC, ARM, MIPS, etc.
- **Target Board**
  - Flash Memory
  - DRAM
  - RS232 Serial Port
  - Ethernet Interface
  - USB Interface
  - Other special-purpose or fancy interfaces

# Embedded Processors

---

- **Stand-alone Processors**

- IBM 970FX
- Intel Pentium M (IA32, IA64)
- Freescale MPC7448, etc.

- **System on Chip (SoC)**

- PowerPC
- MIPS
- ARM
- Others

# SoC Processors - PowerPC

---

- **PowerPC**

- RISC Design
- Apple, IBM, Motorola's semiconductor
- One of most popular architecture for embedded applications
  - Automotive, consumer, network telecom switches

- **AMCC PowerPC**

- 405xx (EP, GP, EP), 440xx (EPx, GX, SP)

- **Freescale PowerPC**

- Host Processors: MPCxxxx
- PowerQUICC I, PowerQUICC II, PowerQUICC II Pro, PowerQUICC III

# SoC Processors - ARM

---

- ARM Core

- Very large market share in consumer electronics, specially mobile device, majority of digital cellular phones.
- Well-known examples:
  - Sony PalyStation Portable (PSP)
  - Apple iPod Nano
  - Nintendo Game Boy Micro and DS
  - TomTom GO 300 GPS
  - Motorola E680i Mobile Phone
- **Products:**
  - 8-bit, 16-bit, 32-bit
  - ARM7, ARM9, ARM11 cores, etc.

# SoC Processors – ARM (cont.)

---

- Atmel ARM
  - ARM AT91RM9200, AT91SAM9XXX
- TI ARM
  - TI ARM OMAP
- Freescale ARM
  - i.MX21, i.MX31
- Intel ARM XScale
  - Well-known Examples:
    - GPS iQue M5 (Garmin), iPAQ (HP), Treo smart phone (Palm), A760 smart phone (Motorola)
- Other ARM
  - Altera, PMC-Sierra, Samsung Electronics, Philips Semiconductor

# SoC Processors – MIPS (cont.)

---

- **MIPS**

- RISC Design (with 32-bit and 64-bit)
- MIPS Technologies
  - 1<sup>st</sup> Architecture, 1981, Stanford Univ., Dr. John Hennessey
  - MIPS Computer System Inc.
  - Licensing IP of MIPS architecture and cores
- Powerhouse in embedded processor market, high-end to consumer products
  - Sony HD TV sets, Linksys wireless access points, Sony PlayStation 2 game console
- 73 licensees
  - Sony, Texas Instruments, Cisco's Scientific Atlanta (cable TV STC), Broadcom, etc.

# SoC Processors - MIPS (cont.)

---

- **Broadcom MIPS**

- Cable TV set-top boxes, cable modem, HDTV, wireless switches, Gigabit Ethernet, VoIP, etc.
- SiByte multi-core processor (single, dual, quad-core)
- BCM1125H, BCM1250, BCM1280, BCM1480

- **AMD MIPS**

- Alchemy MIPS
- Au1xxx (1000, 1100, 1200, 1500, 1550)

- **Other MIPS**

- ATI technologies, Cavium Network's Octeon, Integrated Devices Technology, PMC-Sierra, NEC, Toshiba, etc.

# Integrated Peripherals Interfaces (Examples)

---

- Graphic Accelerators (2D, 3D)
- LCD (screen & backlight) and Keypad controller
- MPEG-4 encoder
- Audio multiplexer
- IrDA infrared I/O
- S-Video output
- DACs for direct TV (PAL/NTSC) video output
- MMC/SD card controller
- Battery-management hardware
- Camera Interface
- Integrated DSPs for Audio / Video
- Integrated security accelerator
- USB client/host interfaces
- External I/O ...
- More and more .....

# Embedded Hardware Platforms

---

- Common Hardware Reference Platform
  - PC/104
  - VMEbus
- Commercial Off-the-shelf (COTS) Solutions
  - CompactPCI (cPCI)
  - ATCA-based (Advanced Telecommunications Computer Architecture)

# Software on Embedded Target

---

- Bootloader
- Kernel
- Device Driver
- Root File System
- Packages (Useful Programs)
- Your Applications

# Bootloader

---

- **Role & Lifetime**

- Performs very low-level hardware initialization
  - To setup processor and memory
  - To initialize UART controlling serial port
  - To initialize Ethernet controller
- Loads kernel image
- Executes operating system image

- **DAS U-boot**

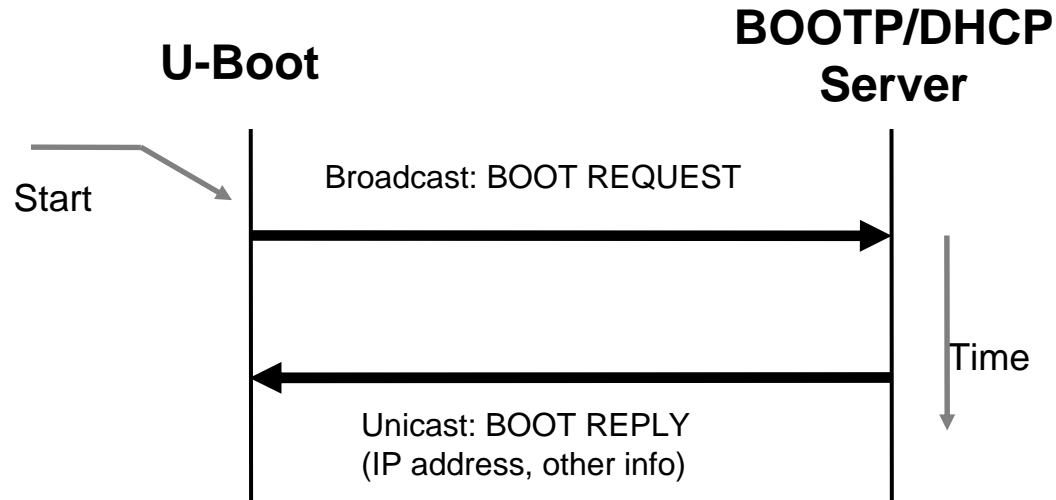
- A Universal Bootloader, popular for many processor architectures
- Obtain U-boot source applied to your target architecture
- Make images using system configuration
  - `$ make <platform>_config`

- **Network Operations**

- Support for BOOTP, DHCP, and TFTP protocols

# BOOTP/DHCP Server

---



BOOTP Client / Server Handshake

# BOOTP/DHCP Server (cont.)

---

- DHCP Target Specification (example)

```
host at91rm9200 {  
    hardware ethernet 04:0e:0d:ef:93:7f  
    netmask 225.255.255.0  
    fixed-address 192.168.5.3  
    server-name 192.168.5.200  
    filename "some-zImage"  
    option root-path "/home/rgu/test/arm-target"  
}
```

# Other Bootloaders

---

- Lilo (Linux Loader)
  - Intel x86/IA32, lilo.conf
- GRUB (Grand Unified Bootloader)
  - GNU project, grub.conf
- Redboot
  - Open source for Intel and XScale family processors
- YAMON
  - Popular at MIPS architecture and circles
- LinuxBIOS
  - X86 environment
- “Cutting-edge” or “Bleeding-edge”?

# Kernel

---

- Source & Versions
  - Obtain source from “[www.kernel.org](http://www.kernel.org)”
  - Linux 2.4.x (production), 2.5.x (exper.), 2.6.x (produ.)
  - Latest Linux 2.6.25?
  - Support 25 different architecture and sub-architectures
- Compile the Kernel
  - Using cross compiler to generate target image
    - “`$make ARCH=arm CROSS_COMPILER=xscale_be-zImage`”
    - Large ELF build target called “vmLinux”, a common target
    - Loadable image used by bootboader:
      - “zImage”, “ulImage”, “bzImage”
  - Kernel configuration
    - Using configuration editor, such as config, menuconfig, etc.

# Kernel – Device Driver

---

- Purpose

- To isolate the user space programs from ready access to critical kernel data structure and hardware devices
- To hide the complexity and variability of the hardware device from the user
- To provide a consistent user interface to a large variety of hardware devices

- Types

- Character Devices
  - Serial port, keyboard
- Block Devices
  - Hard drives, floppy disk

# Kernel – Device Driver (cont.)

---

- **Loadable Module**

- Loadable kernel module (LKM)
- Modules can be installed after the kernel has booted, using start-up scripts or service “demand loaded”
- Be added and removed from kernel component at running time, using utilities:
  - modprobe (-r), insmod, lsmod, rmmod, modinfo
- It can also be statically compiled into kernel

# Kernel - Root File System

---

- How to select Linux file system to deploy
  - Optimized for performance
  - Optimized for size
  - Optimized for data recovery
  - Optimized for storage
  - Designed for use on Flash memory devices

# Kernel - Root File System

---

- **Create File System**

- Make partitions on physical medium (hard drive, Flash memory):
  - “fdisk /dev/sdb”
- Partition includes:
  - file system metadata, file system data
  - 90 different partition types
- Make a file system type of ext2:
  - “mke2fs2 /dev/sdb1 -L CFlash\_Boot\_Vol”
- Check file system integrity:
  - “e2fsck -y /dev/sdb1”
- Mount a file system:
  - “mount /dev/sdb1 /mnt/flash”

# Desktop File System

---

- **Ext2**
  - Very small block size is always compromise for best performance, overhead of metadata (block-to-file mapping)
- **Ext3**
  - Journaling capability
  - Redhat, Fedora Core
- **ReiserFS4**
  - SuSE, Gentoo
- **ReiserFS4**
  - Implements high performance “atomic” FS operations designed to protect both the state of the FS (its consistency) and data involved in a FS operation
  - Provides a user-level API to guarantee the atomicity of a FS transaction

# Embedded File System

---

- **Challenges to Flash File System**

- Flash memory can be erased only a block at a time (128KB)
- Linux contains small files (10-100KB, most 10KB, 40KB)
- Time consuming process
  - To invalidate entire 128KB block and re-write every file
  - Much slow than hard disk write
- Data corruption
  - Window of slow writing process
  - Suddenly loss power
- Flash memory has limited life time
  - Minimum 100,000 write cycles, more recently, 1,000,000-cycle

- **Select Embedded File System**

- JFFS2 (2n Generation Journaling Flash File System)
- Original JFFS was designed by Axis Communication AB of Sweden

# JFFS2

---

- JFFS2 can do
  - Having knowledge of the flash architecture, and more important, architectural limitations imposed by devices
  - Using technique “wear leveling”, to spread the writes evenly across the blocks of a Flash memory device
- Building a JFFS2 image
  - Make sure kernel be able to support for JFFS2
  - Use mkfs.jffs2 utility:
    - “mkfs.jffs2 -d ./jffs2-image-dir -o jffs2.bin”
  - Enable system logger (syslogd and klogd) is not good ideal
- JFFS2 Root File System for Target
  - Convert RootFS to JFFS2 (rootfs.arm.jffs2)
- MTD Subsystem (Memory Technology Devices)

# Other File Systems

---

- Cramfs
  - A read-only FS perfect for small-system boot ROMs and other read-only programs and data
- Network File System (NFS)
  - One of most powerful development tools for embedded developers
- Pseudo File System
  - Proc FS (/proc)
    - “mount -t proc /proc /proc”
- Others
  - ramfs, tmpfs, etc.

# Packages

---

- Some useful programs
  - Includes: libraries, tools, utilities, docs, etc.
  - Typical Examples
    - Init scripts, glibc, busybox, ...
    - telnet, apache, snmp, ...
    - DHCP server, TFTP client, NFS client, ...
- Busybox
  - The Swiss Army Knife of Embedded Linux
  - A small and efficient replacement for a large collection of standard Linux command line utilities
  - All symbolic links

# Your Applications

---

- It's your world
  - Life is wonderful if you can do whatever you like to, isn't it ?!!!

# Host Machine Setup

---

- **Cross Platform Toolchain**
  - Cross compiler (gcc)
  - Binary utils (binutils) (assembler, linker)
  - C standard library (GNU glibc, uClibc, dietlibc)
- **A serial terminal connected to RS232 serial port**
  - Install minicom, ymodem, sx-at91 utility, and/or other utilities (depending on target architecture)
- **A terminal session for Telnet and/or SSH**
  - Active Ethernet Interface
- **Install TFTP Server**
- **DHCP Client (or Server)**
- **Install NFS**
- **Debugger Tools**

# TFTP Server

---

- TFTP (Trivial File Transfer Protocol)
  - Use to transfer kernel image
  - “mkdir /tftpboot/rootfs.arm.jffs2”
- Install TFTP Server
- Configuration File
  - “vim /etc/xinetd.d/tftp” → disable=no
- Start TFTP Server
  - “/etc/rc.d/init.d/xinetd restart”
  - “/usr/sbin/in.tftpd” (on demand service)
- Bypass Firewall for tftp port

# NFS Server

---

- **Configuration File**
  - “vim /etc/exports”
  - “/tftpboot \*(rw, sync, no\_root\_squash)”
- **Start NFS Server on Host**
  - “/etc/init.d/nfs start” (or restart)”
  - or “/sbin/service portmap status”
  - “/sbin/service nfs start”
- **Target NFS client mount to Host**
  - “mount -t nfs 192.168.5.200:/tftpboot /mnt/nfs\_remote”
  - Or using “/etc/fstab”

# Methods to Boot System

---

- Re-boot via TFTP
  - Load kernel via TFTP server and boot
- Re-boot via Flash Memory
  - Kernel bootable image pre-loaded into Flash
- Re-boot via NFS Server
  - Loading kernel via TFTP server, booting with NFS root mount
- Re-boot via Hard Drive
  - Kernel bootable image pre-loaded into HD

# Build Your Own System

---

- You really need one more thing, no kidding!
- What is Buildroot?
- Buildroot is a set of Makefiles and patches
  - It allows to easily generate a cross-compilation toolchain
  - It allows to build root filesystem for your target
  - The cross-compilation toolchain uses uClibc, a tiny C standard library
- Obtain Buildroot
  - <http://buildroot.uclibc.org/downloads/snapshots/>
- Run buildroot configuration
  - “\$ make menuconfig”

# Build Your Own System (cont.)

---

- **Make cross compiler toolchain**

- “\$ make” (/build\_arm, /dl, /toolchain\_build\_arm)
- /build\_arm/busybox (../mtd\_org, ../staging\_dir, ../dhcp)
- Setup path (export PATH=\$PATH"/home../buildroot/build\_arm/bin”)
- To find compiler stuffs arm-linux-gcc, -objdump, -ld, etc.

- **Make U-boot**

- “\$ make ARCH=arm”
- “cp u-boot-rom.bin /ul”, “cp u-boot-ram.bin /ul”
- “/buildroot/build\_arm/linux-2.6.20/archarm/boot/ulmage”
- “cp ../u-boot/tools/mkimage /usr/local/bin/mkimage”

# Build Your Own System (cont.)

---

- Make Kernel Image
  - “\$ “make ARCH=arm”
  - “\$ cp ../buildroot/rootfs.arm.jffs2 /tftpboot/rootfs.arm.jsff2”
- Upload u-boot images into target
  - via serial at91loader.comet.in protocol
- Upload kernel image into target
  - via TFTP
- Setup method to boot system
- Re-boot system

# Some Commercial Players

---

- BlueCat® embedded Linux® from LynuxWorks™
- MontaVista Linux
- Timesys Embedded Linux
- Wind River Linux 1.3

# III. Recommendations

---

- **Define System Requirements**
  - Budgets
  - Resources
  - Timeframe
- **Select Available Hardware**
  - SoC Processors & Target architecture
  - Services & Prices
- **Select Available System Software**
  - Commercial products or Open source

## IV. References

---

- <http://www.buildroot.uclibc.org> (Buildroot)
- <http://www.busybox.net> (Busybox)
- <http://www.uclibc.org> (uClibc)
- <http://www.denx.de/wiki/UBoot> (U-Boot)
- <http://www.kernel.org> (Kernel)
- <http://www.linuxdevices.com>
- <http://www.sourceforge.net>
- <http://freshmeat.net>
- “Linux Journal”
- “Linux Magazine”

# References (cont.)

---

- <http://www.slashdot.org>
- <http://www.linuxtoday.com>
- <http://www.tldp.org/> (Linux Documentation Project)
- <http://www.tldp.org/docs.html#howto> (Linux How To)
  
- "Build Embedded Linux Systems"
  - Karim Yaghmour, O'Reilly, 2003
- "Embedded Linux Primer"
  - Christopher Hallinan, Prentice Hall, 2006
- Other Valuable Books & Magazines