# Bootloaders

# What to Expect?

- W's of Bootloaders
- Specifics of a Bootloader
  - With U-Boot in consideration
- U-Boot Hands-On
- U-Boot Source Code
- U-Boot Porting

# What is a Bootloader?

- Simply, a loader (program), which boots up (starts) the system
- A Customized Program started by
  - Controller's Internal Code in Embedded Systems, Or
  - Default Program Counter settings in Desktops

# Tasks of a Bootloader

- Initialization Tasks
  - Memory Setup & Initialization
  - System Peripheral Initialization

  for the kernel
- Actual Task
  - Load the RAM-based File System, like initrd, initramfs, ...
  - Load the Kernel with proper arguments
  - Jump to the start of the Kernel
- Additional Tasks
  - Multiple Kernel Boots
  - Multiple-way Boots

# Design of Bootloaders

- As Bootloader is started by a fixed code
  - It needs to be placed at a hard-coded location
  - Hard-coded locations are not big enough for the complete code (/ logic / tasks) of the bootloader
- Hence, it is typically split into 2 portions
  - Stage 1 – Small enough to load Stage 2 from our desired location
  - Stage 2 – The actual bootloader we want to have

# Bootloader Comparisons

- On Desktops
  - Initialization Tasks are done by BIOS
  - Bootloader is to just Boot the Kernel
- On Embedded Systems
  - All needs to be done by the Bootloader
  - But in an optimized way
- Hence, the 2 bootloaders are
  - Quite different from each other
  - Later being more board dependent & constrained

# Check

- Name the Stage 1 & Stage 2 bootloaders
  - Desktops
  - Embedded Systems

# Stage 2 Bootloader Flavours

- Prevalent Desktop Bootloaders
  - LILO
  - GRUB
  - SYSLINUX
  - loadlin
  - Coreboot (Earlier called LinuxBIOS)
- Popular Embedded System Bootloaders
  - BootLoader Object (BLOB)
  - Redboot
  - U-Boot

# U-Boot Specifics

# U-Boot Initialization Details

- Bootloader starts its execution from flash / *PROM
- Hardware Diagnostics, like POST, ...
- Configuring the CPU speed, MMU setting, etc
- Memory Initialization
  - Determining on-board memory size
  - Turning on the caches
  - Clearing memory
- Optionally, Relocate to RAM, and start execution from there
- Setting up interfacing ports like serial, VGA, ...

# U-Boot's Argument Passing

- Three-Party Communication
- Kernel hard-codes the offset of argument structure
- Developer attaches the load & start addresses
  - In a U-Boot recognizable Header
  - To the kernel image
  - We shall do when building kernel image
- U-Boot
  - Reads the U-Boot Header wrapped kernel image
  - Loads the kernel at the read load address
  - Fills the kernel argument structure at the hard-coded offset
    - Arguments are obtained from its environment
  - Jumps to the kernel start address to start executing

# U-boot Hands-on

- Stopping at the U-Boot
- Help - “?”
- Commands
  - Booting: bootp, bootm, boot, ...
  - NOR Flash: erase, cp, protect, ...
  - NAND Flash: nand
  - Miscellaneous: reset, ...
  - ...
- Environment Variables
  - printenv
  - setenv
  - saveenv

# U-Boot Source Tree

- arch – Architecture dependent Code
- board – Board dependent Code
- common – Environment & Command Line Code
- doc – Documentation
- drivers – Device specific Drivers
- fs – File System support Code
- include – Headers
- lib – Compression, Encryption related Code
- net – Minimal Network Stack
- tools – U-Boot Utilities (mkimage is here)

# U-Boot Compiling

- Preparing the Makefile
  - Setup (ARCH,) CROSS_COMPILE for cross compilation
  - Or, invoke make with these options
- Configuring for a particular board
  - make <board>_config
- Compiling for the configured board
  - make (Output would be u-boot.bin)
- Cleaning up
  - make clean

# U-Boot Porting

- Implies adding a new Board to U-Boot
- That entails
  - Adding <board>_config for make
  - Adding board specific code at the right places

# Adding <board>_config

- Adding an entry in the Makefile with
  - Architecture
  - CPU
  - Board
  - Vendor (May be NULL)
  - SoC (May be NULL)
- Adding the new board directory under board/ with
  - Makefile
  - Initialization Code for the Board
  - Configuration Makefile
- Adding the new board header under include/configs/ with
  - Configuration for the Board

Note: If a new architecture is added, a U-Boot architecture porting would be needed - though, that is uncommon

# Adding Board specific Code

- Highly depends on the Board
  - CPU Architecture
  - Clock Interfaces
  - Console Interfaces
  - LCD Interfaces
  - Network Interfaces
  - Peripherals
  - ...
- Should be added in the appropriate Folders
  - With appropriate file names
  - And following the U-Boot Coding Guidelines
- Existing Code may serve as the example for the same

# What all have learnt?

- W's of Bootloaders
- Specifics of a U-Boot
  - U-Boot Initialization Sequence
  - U-Boot's argument passing to Kernel
- U-Boot Hands-On
- U-Boot Source Code
  - Understanding the Source Structure
  - Configuring for a Board
  - Compiling the u-boot image
- U-Boot Porting for a New Board

# Any Queries?