KERNEL MASTERS

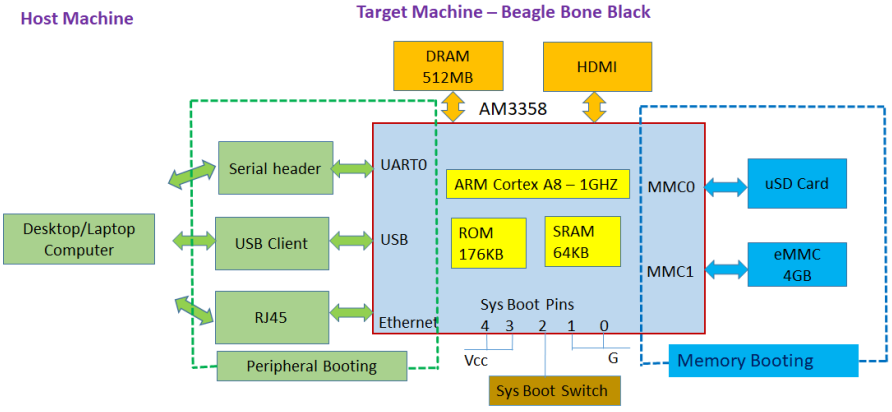
Online Training on

Embedded Linux Porting & BSP

With BeagleBone Black Expansion Board

By Kishore Kumar Boddu

Embedded Linux Boot Sequence



Sysboot SW	Sysboot pins	1st Order	2nd Order	3rd Order	4th Order
OFF	11100	MMC1	MMC0	UART0	USB0
ON	11000	SPI0	MMC0	USB0	UART0

KERNEL MASTERS

Who should attend?

- This session is mainly intended for those looking to start their career in embedded Linux or for those already working in Linux porting.
- Project managers and engineers who are evaluating the use of Linux for their projects
- Project managers and engineers who are considering supporting their own Linux distribution and tools.
- Engineers responsible for build systems and software releases.

All Sessions are highly interactive hands-on-sessions.

Prerequisites:

- Delegates must have attended Developing with Embedded Linux or an equivalent course, or have some experience of using Linux for Embedded Systems.
- We assume that attendees are fully fluent in C, data structures and that the Linux/Unix command line is a familiar environment.
- Embedded Systems development Knowledge/Experience.
- Linux System Programming & Device Drivers.

Hardware used

- The practical exercises will be run on a Beagle Bone Black (BBB) with a Cortex ARM.
- All exercises will be applicable to any other type of board supported by Linux.
- Kernel Masters Designed Beagle bone expansion board. For purchase this board contact Kernel Masters Team.
- Online practical demonstration for Linux porting on BBB. Later on you can buy and practise, support will be provided. Material will be provided with step by step procedure for lab guidance.

End of the course you'll be able to (Session Highlights):

- Build a customized kernel and u-boot for a designated target.
- Configure a bootloader for booting a customized kernel with its root filesystem from a solid-state storage device.
- Choose a GNU toolchain for cross-development.
- Develop a deep understanding of the board support packages with Real Time Examples.
- Add the following skill set to your profile**: Board support packages, boot loaders, Setup Embedded Linux Development Environment, Board Bring up, Boot Time Optimization etc.**
- And more.

Embedded Linux Porting & BSP Syllabus Summary: (Detailed agenda next page)

Session 1: Introduction to Embedded Linux
Session 2: Setup Embedded Linux Development Environment
Session 3: Developing Embedded Linux Device Drivers
Session 4: Project: Falling Edge Interrupt porting on AM335x Controller.
Session 5: Embedded Linux Build System

Course Material:
https://github.com/kernelmasters
Kernel Masters GIT LAB

Authored and Compiled By: Boddu Kishore Kumar
Email: kishore@kernelmasters.org
Reach us online: www.kernelmasters.org
Contact: 9949062828

Embedded Linux Porting & BSP Detailed Agenda

Session 1: Introduction to Embedded Linux (Prebuilt Images)

Objective: To understand Embedded Linux platform and cross compilation toolchain. Understand Beagle Bone Black (BBB) specifications, Block Diagram and Memory Mapping and also AM3358 Boot Sequence. Understand KM-BBB Specifications and Bock Diagram. How to install Prebuilt images on KM-BBB.

1.1. Embedded Hardware and Software.
1.2. C libraries. Building a cross-compiling tool chain.
1.3. BBB Specifications and Block Diagram
1.4. AM3358 Specifications, Block Diagram and Memory Mapping
1.5. X86 vs Embedded Boot Sequence. (AM3358 Boot Sequence)

Hands-On-Session:

Debian 10 Prebuilt Images porting on KM-BBB.

Session 2: Setup Embedded Linux Development Environment (Own Built Images)

Objective: To understand how to create Own built images for u-boot, Kernel and RFS.

Topic 1: Boot Loaders and U-boot

1.1. Board Support Packages
1.2. U-boot commands and Environment variables.
1.3. U-boot Source code flow
1.4. U-boot Customization

Topic 2: Linux Kernel

1.1. KBuild System
1.2. Configuring, (cross) compiling and booting a Linux kernel
1.3. Kernel boot-up flow

Topic 3: Device Tree

2.1. What are Device Trees?
2.2. What Device Trees Do and What They Do Not Do
2.3. Device Tree Syntax
2.4. Device Tree Walk Through
2.5. Device Tree Bindings
2.6. Device Tree support in Boot Loaders
2.7. Using Device Tree Data in Drivers
2.8. Coexistence and Conversion of Old Drivers

Topic 4: Root File System

3.1. Creating a simple, Busybox based root file system from scratch
3.2. Flash file systems – Manipulating flash partitions
3.3. mmc vs emmc

Hands-On-Session:

Walkthrough BBB Device tree source and understand its importance.
Built Own boot loader and Kernel and Root File System.

Session 3: Developing Embedded Linux Device Drivers

Objective: To Learn, GPIO Framework from Bottom to Top. Understand how to develop GPIO API's using system calls and application programming using Shell Program, 'C' and Python GPIO modules.

GPIO Driver Framework

Step 1: Study GPIO Basics and AM335x GPIO controller:

1.1. What is GPIO? GPIO Applications?
1.2. AM335x GPIO Controller

Step 2: GPIO @ U-boot Space:

2.1. Control Mux Configuration from U-Boot Command prompt.
2.2. Control LED and Switches from U-boot Command prompt.
2.3. Customize u-boot source code and verify Mux Configuration and LED/Switches
2.4. Understand GPIO Framework in U-boot

Step 3: GPIO @ Kernel Space [GPIO user space driver]:

3.1. Control GPIO functionalities from sys file system entry.
3.2. Monitoring Mux Configuration and GPIO using debug file system.
3.3. Control Mux Configuration using device tree source
3.4. GPIO Test Modules
3.5. Understand GPIO Framework in Kernel

Step 4: GPIO @ User Space:

GPIO System Programming

4.1. Cross compilation of hello world system program
4.2. Implement GPIO API's using system calls

GPIO Application Programming

4.3. C Program using GPIO API's
4.4. Shell Scripting using Linux commands
4.5. Python Program using python modules

Hands-On-Session:

Two ways possible to Control Mux configuration, one is u-boot and second one is device tree source code. LED and Switches control from u-boot, kernel and user space.

Session 4: Project

Falling Edge Interrupt porting on AM335x Controller [Input Sub system in Kernel]

1.1. Add GPIO Key Mux configuration in u-boot source code.
1.2. Test GPIO Key functionality using gpio command at u-boot command prompt
1.3. Enable GPIO keys driver and interrupt in Device tree source code.
1.4. Enable GPIO Key driver as module and cross compile.
1.5. Test GPIO key from user space using evtest application.
1.6. Prepare a GPIO Key Input sub system framework.

Session 5: Embedded Linux Build System

5.1. Buildroot - Configuring Buildroot for bbb.
5.2. Yocto Project/Open Embedded

Note: For more No. of Drivers (I2C, UART, SPI and Network), attend separate sessions offered by Kernel Masters.