Booting Linux kernel on the Ajit Processor

Neha Karanjkar
May 2018

Running bare-metal programs

- Upon hardware reset:
  - PC=0x0, nPC=0x4
  - Supervisor mode is set
  - MMU is disabled (VA=PA)
  - Device registers can be accessed directly through their physical addresses.
- To run bare-metal programs, just load the program bytes in memory starting from addr 0x0.

Contributors

- **Renju Boben** : Kernel modifications at the entry (romvec and device tree), Kernel compilation flow using Buildroot
- **Harshal Kalyane** : MMU validation, UART driver
- **Aniket Deshmukh** : Low level-timer interfacing, low-level UART interfacing, UART driver, Ajit_bootloader
- **Yeshwanth** : Low-level timer interfacing

Requirements for running Linux on the Ajit processor

(A) Processor-side (hardware) requirements

- Compatible MMU hardware
- Devices: Atleast a timer, UART and Interrupt controller
- Correct **memory layout** and device addressing

(B) A Bootloader

(C) Modifications to the Linux kernel:

- Kernel entry
- MMU setup and routines
- Device interfacing

(D) Flow for generating the kernel image and final memory map

(A) Memory Layout

The kernel's memory layout

- If each user process has its own virtual address space, how can a user process access kernel routines ?

Convention followed by the Linux kernel (architecture independant)

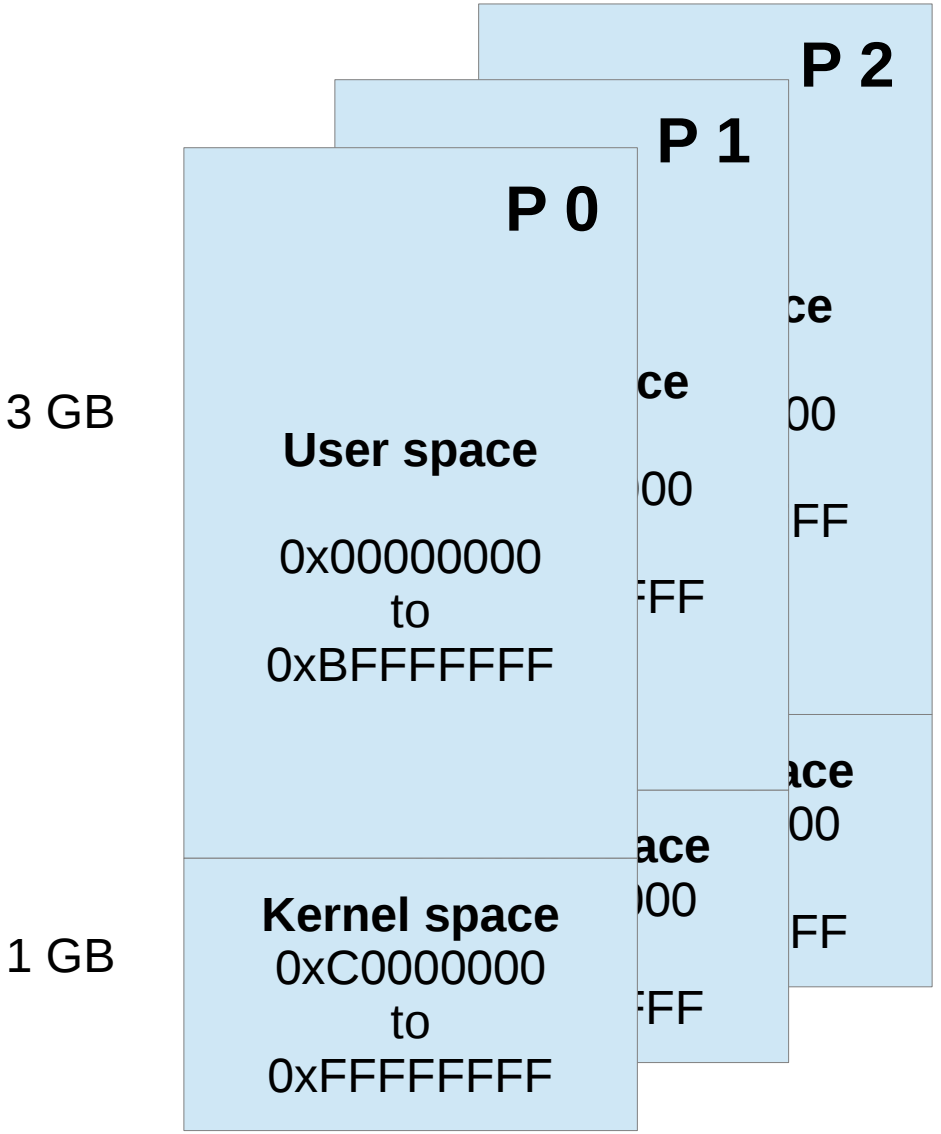
- A part of each user process' virtual address space is reserved for the kernel. The set of virtual addresses reserved for the kernel is fixed (0xC0000000 to 0xFFFFFFFF, total 1GB)
- Further, the kernel is always present (at a fixed location) in physical memory and never swapped out. The physical address of the kernel in memory is an architecture-specific parameter.

Convention followed by the Linux kernel (Sparc-v8 specific, fixed) :

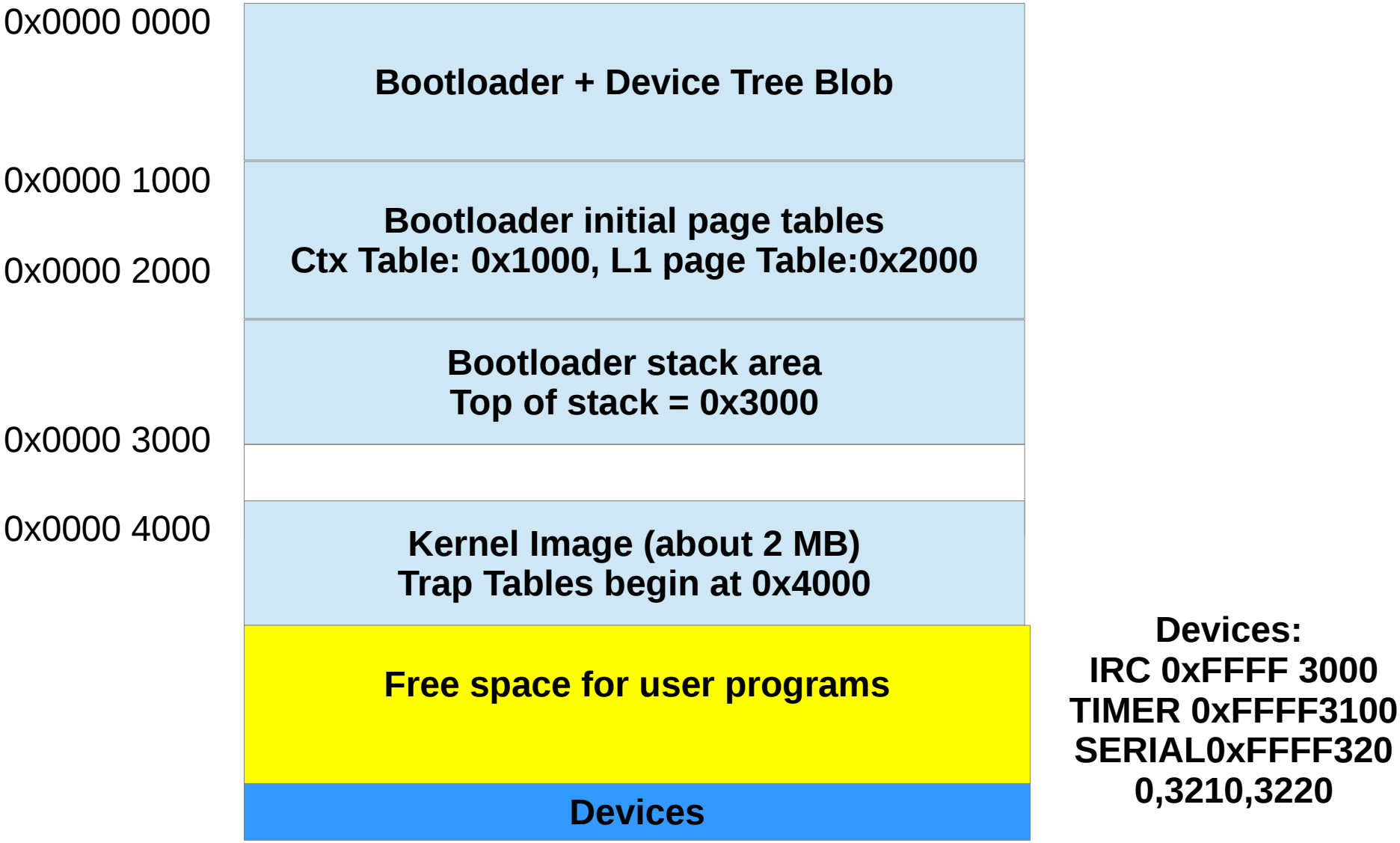
- The kernel symbols begin at virtual address 0xF0004000. The kernel assumes that this virtual address is mapped to physical address 0x00004000.

Virtual memory layout

(for 32b architectures)



Physical Memory Layout for the Ajit Processor



(B) Bootloader

- Kernel is already placed in memory at the correct physical address (0x00004000)
- The Bootloader needs to:
  - Initialize registers (zero out reg file, initialize stack ptr)
  - Setup initial page tables and initialize the MMU
  - Populate a “romvec” structure through which hardware information is passed to the kernel
  - Enable the MMU and
  - Jump to start of kernel (address 0xF0004000)

(B) Bootloader

- Initial VA to PA mapping:
  - A very corase mapping is used by the Ajit Bootloader.

    (could be optimized if needed)
  - Bootloader area :

    VA 0x0000 0000 to 0x 00FF FFFF is mapped to the same PA (VA=PA)
  - Kernel image area:

    VA 0xF000 0000 to 0xF0FF FFFF is mapped to 0x0000 0000 to 0x00FF FFFFF
- No trap tables etc. Interrupts are not enabled yet.

The Romvec Structure

- Important fields:
  - Pointer to a putchar() routine. The kernel printk's in-turn uses this putchar routine.
  - Information about available memory.
  - The nodeops. The nodeops consist of:
    - A device tree data-structure containing info about hardware (also called a device tree blob)
    - Routines to traverse this data structre.
  - Ajit-specific: The Ajit bootloader only passes a pointer to the device tree blob. The routines to traverse this blob are added to the kernel. The routines are implemented as wrappers arount the standard flattened devices tree routines (in libfdt.h)

The Ajit_linux Kernel

- Derived from Linux 3.16.1
- Modifications limited to arch/sparc/
  - At kernel entry (head_32.s, setup_32.c)
  - Supplying machine-info to the kernel using device tree routines (arch/sparc/Ajit/ folder)
  - Low-level timer and interrupt-controller routines (Ajit_irq.c)
  - MMU routines (Ajit_srmmu.c)
- Serial device driver (Ajit_serial.c) added to /drivers/tty/serial

The Ajit_linux Kernel

- **Kernel entry:**
- The kernel's trap tables are located at the start of kernel address space. (/arch/sparc/kernel/ttable_32.S)
- The reset trap handler (located at 0xF0004000) jumps to the symbol “gokernel” (in /arch/sparc/kernel/head_32.S)

Generating the kernel memory map

- Detailed documentation at :

  /os/Ajit_linux/Documentation/README
- Script for the overall flow:

  /os/Ajit_linux/GeneratememMapForAjit.sh
- **Required:**
  - Buildroot

    See /os/Ajit_linux/Documentation/README
  - Sparc-v8 cross compiler toolchain (installed)

Generating the kernel memory map

The bootloader passes hardware info to the kernel through a device tree blob. The first step is to compile this device tree into assembly.

Device tree *(/arch/sparc/Ajit/device_tree/Ajit_device_tree.dts)*

compile_device_tree.sh

uses device tree compiler (dtc) + patch

Ajit_device_tree.S

(The file Ajit_device_tree.S is then linked together with the bootloader code to generate a single bootloader executable)

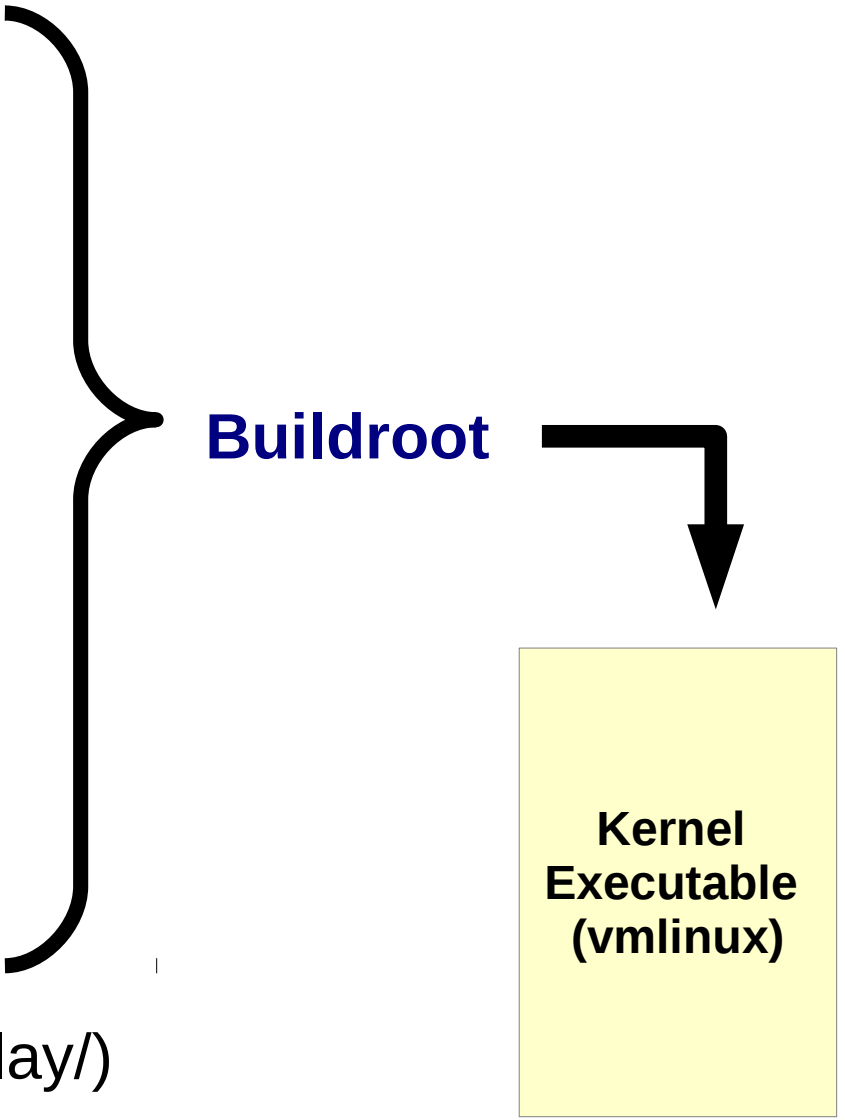
Generating the kernel memory map

Kernel source
(Ajit_linux_3.16.1)

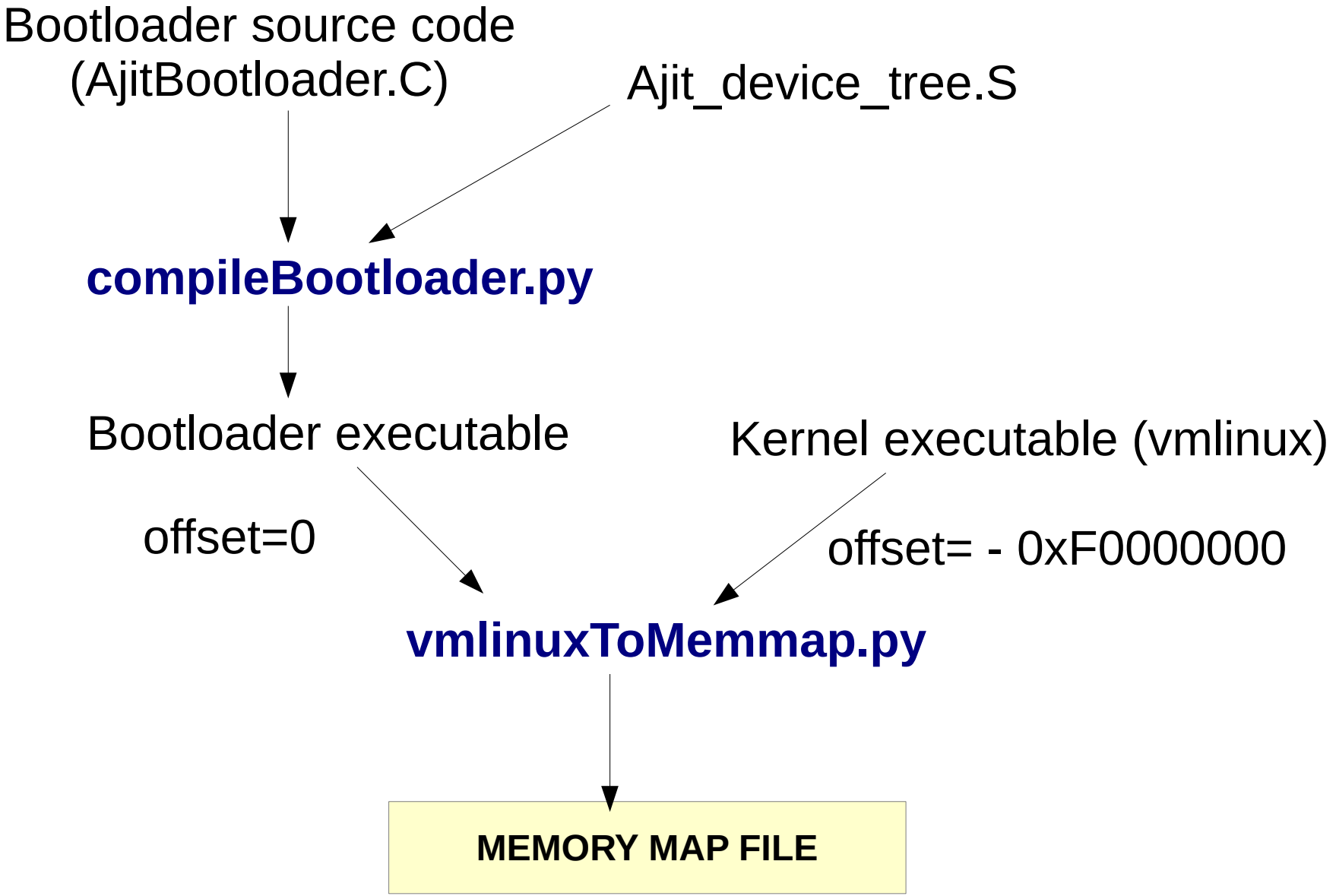
Buildroot configuration (Ajit_defconfig)

Linux kernel configuration
(Ajit_minimal_linux_kernel.config)

Initramfs contents
rootfs.cpio (generated by buildroot)
+
Rootfs Overlay
(Ajit_buildroot_configs/board/rootfs_overlay/)



Generating the kernel memory map



Contributors

- **Renju Boben** : Kernel modifications at the entry (romvec and device tree), Kernel compilation flow using Buildroot
- **Harshal Kalyane** : MMU validation, UART driver
- **Aniket Deshmukh** : Low level-timer interfacing, low-level UART interfacing, UART driver, Ajit_bootloader
- **Yeshwanth** : Low-level timer interfacing