Booting ARM Linux

Author: Russell King

Date : 18 May 2002

The following documentation is relevant to 2.4.18-rmk6 and beyond.

In order to boot ARM Linux, you require a boot loader, which is a small program that runs before the main kernel. The boot loader is expected to initialise various devices, and eventually call the Linux kernel, passing information to the kernel.

Essentially, the boot loader should provide (as a minimum) the following:

1. Setup and initialise the RAM.
2. Initialise one serial port.
3. Detect the machine type.
4. Setup the kernel tagged list.
5. Load initramfs.
6. Call the kernel image.

1. Setup and initialise RAM

Existing boot loaders:
    MANDATORY
New boot loaders:
    MANDATORY

The boot loader is expected to find and initialise all RAM that the kernel will use for volatile data storage in the system. It performs this in a machine dependent manner. (It may use internal algorithms to automatically locate and size all RAM, or it may use knowledge of the RAM in the machine, or any other method the boot loader designer sees fit.)

2. Initialise one serial port

Existing boot loaders:
    OPTIONAL, RECOMMENDED
New boot loaders:
    OPTIONAL, RECOMMENDED

The boot loader should initialise and enable one serial port on the target. This allows the kernel serial driver to automatically detect which serial port it should use for the kernel console (generally used for debugging purposes, or communication with the target.)

As an alternative, the boot loader can pass the relevant 'console=' option to the kernel via the tagged lists specifying the port, and serial format options as described in

    Documentation/admin-guide/kernel-parameters.rst.

3. Detect the machine type

Existing boot loaders:
    OPTIONAL
New boot loaders:
    MANDATORY except for DT-only platforms

The boot loader should detect the machine type its running on by some method. Whether this is a hard coded value or some algorithm that looks at the connected hardware is beyond the scope of this document. The boot loader must ultimately be able to provide a MACH_TYPE_xxx value to the kernel. (see linux/arch/arm/tools/mach-types). This should be passed to the kernel in register r1.

For DT-only platforms, the machine type will be determined by device tree. set the machine type to all ones (~0). This is not strictly necessary, but assures that it will not match any existing types.

4. Setup boot data

Existing boot loaders:
    OPTIONAL, HIGHLY RECOMMENDED
New boot loaders:
    MANDATORY

The boot loader must provide either a tagged list or a dtb image for passing configuration data to the kernel. The physical address of

the boot data is passed to the kernel in register r2.

4a. Setup the kernel tagged list

The boot loader must create and initialise the kernel tagged list. A valid tagged list starts with ATAG_CORE and ends with ATAG_NONE. The ATAG_CORE tag may or may not be empty. An empty ATAG_CORE tag has the size field set to '2' (0x00000002). The ATAG_NONE must set the size field to zero.

Any number of tags can be placed in the list. It is undefined whether a repeated tag appends to the information carried by the previous tag, or whether it replaces the information in its entirety; some tags behave as the former, others the latter.

The boot loader must pass at a minimum the size and location of the system memory, and root filesystem location. Therefore, the minimum tagged list should look:

```
                +-----------+
base ->         | ATAG_CORE |   |
                +-----------+   |
                | ATAG_MEM  |   | increasing address
                +-----------+   |
                | ATAG_NONE |   |
                +-----------+   v
```

The tagged list should be stored in system RAM.

The tagged list must be placed in a region of memory where neither the kernel decompressor nor initrd 'bootp' program will overwrite it. The recommended placement is in the first 16KiB of RAM.

4b. Setup the device tree

The boot loader must load a device tree image (dtb) into system ram at a 64bit aligned address and initialize it with the boot data. The dtb format is documented at https://www.devicetree.org/specifications/. The kernel will look for the dtb magic value of 0xd00dfeed at the dtb physical address to determine if a dtb has been passed instead of a tagged list.

The boot loader must pass at a minimum the size and location of the system memory, and the root filesystem location. The dtb must be placed in a region of memory where the kernel decompressor will not overwrite it, while remaining within the region which will be covered by the kernel's low-memory mapping.

A safe location is just above the 128MiB boundary from start of RAM.

5. Load initramfs.

Existing boot loaders:
    OPTIONAL
New boot loaders:
    OPTIONAL

If an initramfs is in use then, as with the dtb, it must be placed in a region of memory where the kernel decompressor will not overwrite it while also with the region which will be covered by the kernel's low-memory mapping.

A safe location is just above the device tree blob which itself will be loaded just above the 128MiB boundary from the start of RAM as recommended above.

6. Calling the kernel image

Existing boot loaders:
    MANDATORY
New boot loaders:
    MANDATORY

There are two options for calling the kernel zImage. If the zImage is stored in flash, and is linked correctly to be run from flash, then it is legal for the boot loader to call the zImage in flash directly.

The zImage may also be placed in system RAM and called there. The kernel should be placed in the first 128MiB of RAM. It is recommended that it is loaded above 32MiB in order to avoid the need to relocate prior to decompression, which will make the boot process slightly faster.

When booting a raw (non-zImage) kernel the constraints are tighter. In this case the kernel must be loaded at an offset into system equal to TEXT_OFFSET - PAGE_OFFSET.

In any case, the following conditions must be met:

- Quiesce all DMA capable devices so that memory does not get corrupted by bogus network packets or disk data. This will save you many hours of debug.

- CPU register settings
  - r0 = 0,
  - r1 = machine type number discovered in (3) above.
  - r2 = physical address of tagged list in system RAM, or physical address of device tree block (dtb) in system RAM
- CPU mode

  All forms of interrupts must be disabled (IRQs and FIQs)

  For CPUs which do not include the ARM virtualization extensions, the CPU must be in SVC mode. (A special exception exists for Angel)

  CPUs which include support for the virtualization extensions can be entered in HYP mode in order to enable the kernel to make full use of these extensions. This is the recommended boot method for such CPUs, unless the virtualisations are already in use by a pre-installed hypervisor.

  If the kernel is not entered in HYP mode for any reason, it must be entered in SVC mode.
- Caches, MMUs

  The MMU must be off.

  Instruction cache may be on or off.

  Data cache must be off.

  If the kernel is entered in HYP mode, the above requirements apply to the HYP mode configuration in addition to the ordinary PL1 (privileged kernel modes) configuration. In addition, all traps into the hypervisor must be disabled, and PL1 access must be granted for all peripherals and CPU resources for which this is architecturally possible. Except for entering in HYP mode, the system configuration should be such that a kernel which does not include support for the virtualization extensions can boot correctly without extra help.
- The boot loader is expected to call the kernel image by jumping directly to the first instruction of the kernel image.

  On CPUs supporting the ARM instruction set, the entry must be made in ARM state, even for a Thumb-2 kernel.

  On CPUs supporting only the Thumb instruction set such as Cortex-M class CPUs, the entry must be made in Thumb state.