Booting AArch64 Linux

Author: Will Deacon <will.deacon@arm.com>

Date : 07 September 2012

This document is based on the ARM booting document by Russell King and is relevant to all public releases of the AArch64 Linux kernel.

The AArch64 exception model is made up of a number of exception levels (EL0 - EL3), with EL0, EL1 and EL2 having a secure and a non-secure counterpart. EL2 is the hypervisor level, EL3 is the highest priority level and exists only in secure mode. Both are architecturally optional.

For the purposes of this document, we will use the term *boot loader* simply to define all software that executes on the CPU(s) before control is passed to the Linux kernel. This may include secure monitor and hypervisor code, or it may just be a handful of instructions for preparing a minimal boot environment.

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

1. Setup and initialise the RAM
2. Setup the device tree
3. Decompress the kernel image
4. Call the kernel image

1. Setup and initialise RAM

Requirement: 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. Setup the device tree

Requirement: MANDATORY

The device tree blob (dtb) must be placed on an 8-byte boundary and must not exceed 2 megabytes in size. Since the dtb will be mapped cacheable using blocks of up to 2 megabytes in size, it must not be placed within any 2M region which must be mapped with any specific attributes.

NOTE: versions prior to v4.2 also require that the DTB be placed within the 512 MB region starting at text_offset bytes below the kernel Image.

3. Decompress the kernel image

Requirement: OPTIONAL

The AArch64 kernel does not currently provide a decompressor and therefore requires decompression (gzip etc.) to be performed by the boot loader if a compressed Image target (e.g. Image.gz) is used. For bootloaders that do not implement this requirement, the uncompressed Image target is available instead.

4. Call the kernel image

Requirement: MANDATORY

The decompressed kernel image contains a 64-byte header as follows:

```
u32 code0;                  /* Executable code */
u32 code1;                  /* Executable code */
u64 text_offset;            /* Image load offset, little endian */
u64 image_size;             /* Effective Image size, little endian */
u64 flags;                  /* kernel flags, little endian */
u64 res2      = 0;          /* reserved */
u64 res3      = 0;          /* reserved */
u64 res4      = 0;          /* reserved */
u32 magic     = 0x644d5241; /* Magic number, little endian, "ARM\x64" */
u32 res5;                   /* reserved (used for PE COFF offset) */
```

Header notes:

- As of v3.17, all fields are little endian unless stated otherwise.
- code0/code1 are responsible for branching to stext.

- when booting through EFI, code0/code1 are initially skipped. res5 is an offset to the PE header and the PE header has the EFI entry point (efi_stub_entry). When the stub has done its work, it jumps to code0 to resume the normal boot process.
- Prior to v3.17, the endianness of text_offset was not specified. In these cases image_size is zero and text_offset is 0x80000 in the endianness of the kernel. Where image_size is non-zero image_size is little-endian and must be respected. Where image_size is zero, text_offset can be assumed to be 0x80000.
- The flags field (introduced in v3.17) is a little-endian 64-bit field composed as follows:

Bit 0	Kernel endianness. 1 if BE, 0 if LE.
Bit 1-2	Kernel Page size. ◦ 0 - Unspecified. ◦ 1 - 4K ◦ 2 - 16K ◦ 3 - 64K
Bit 3	Kernel physical placement 0 2MB aligned base should be as close as possible to the base of DRAM, since memory below it is not accessible via the linear mapping 1 2MB aligned base may be anywhere in physical memory
Bits 4-63	Reserved.

- When image_size is zero, a bootloader should attempt to keep as much memory as possible free for use by the kernel immediately after the end of the kernel image. The amount of space required will vary depending on selected features, and is effectively unbound.

The Image must be placed text_offset bytes from a 2MB aligned base address anywhere in usable system RAM and called there. The region between the 2 MB aligned base address and the start of the image has no special significance to the kernel, and may be used for other purposes. At least image_size bytes from the start of the image must be free for use by the kernel. NOTE: versions prior to v4.6 cannot make use of memory below the physical offset of the Image so it is recommended that the Image be placed as close as possible to the start of system RAM.

If an initrd/initramfs is passed to the kernel at boot, it must reside entirely within a 1 GB aligned physical memory window of up to 32 GB in size that fully covers the kernel Image as well.

Any memory described to the kernel (even that below the start of the image) which is not marked as reserved from the kernel (e.g., with a memreserve region in the device tree) will be considered as available to the kernel.

Before jumping into the kernel, 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.
- Primary CPU general-purpose register settings:
  - x0 = physical address of device tree blob (dtb) in system RAM.
  - x1 = 0 (reserved for future use)
  - x2 = 0 (reserved for future use)
  - x3 = 0 (reserved for future use)
- CPU mode

  All forms of interrupts must be masked in PSTATE.DAIF (Debug, SError, IRQ and FIQ). The CPU must be in non-secure state, either in EL2 (RECOMMENDED in order to have access to the virtualisation extensions), or in EL1.
- Caches, MMUs

  The MMU must be off.

  The instruction cache may be on or off, and must not hold any stale entries corresponding to the loaded kernel image.

  The address range corresponding to the loaded kernel image must be cleaned to the PoC. In the presence of a system cache or other coherent masters with caches enabled, this will typically require cache maintenance by VA rather than set/way operations. System caches which respect the architected cache maintenance by VA operations must be configured and may be enabled. System caches which do not respect architected cache maintenance by VA operations (not recommended) must be configured and disabled.
- Architected timers

  CNTFRQ must be programmed with the timer frequency and CNTVOFF must be programmed with a consistent value on all CPUs. If entering the kernel at EL1, CNTHCTL_EL2 must have EL1PCTEN (bit 0) set where available.

- Coherency

  All CPUs to be booted by the kernel must be part of the same coherency domain on entry to the kernel. This may require IMPLEMENTATION DEFINED initialisation to enable the receiving of maintenance operations on each CPU.

- System registers

  All writable architected system registers at or below the exception level where the kernel image will be entered must be initialised by software at a higher exception level to prevent execution in an UNKNOWN state.

  For all systems: - If EL3 is present:

    - SCR_EL3.FIQ must have the same value across all CPUs the kernel is executing on.
    - The value of SCR_EL3.FIQ must be the same as the one present at boot time whenever the kernel is executing.

  - If EL3 is present and the kernel is entered at EL2:
    - SCR_EL3.HCE (bit 8) must be initialised to 0b1.

  For systems with a GICv3 interrupt controller to be used in v3 mode: - If EL3 is present:

    - ICC_SRE_EL3.Enable (bit 3) must be initialiased to 0b1.
    - ICC_SRE_EL3.SRE (bit 0) must be initialised to 0b1.
    - ICC_CTLR_EL3.PMHE (bit 6) must be set to the same value across all CPUs the kernel is executing on, and must stay constant for the lifetime of the kernel.

  - If the kernel is entered at EL1:

    - ICC.SRE_EL2.Enable (bit 3) must be initialised to 0b1
    - ICC_SRE_EL2.SRE (bit 0) must be initialised to 0b1.

  - The DT or ACPI tables must describe a GICv3 interrupt controller.

  For systems with a GICv3 interrupt controller to be used in compatibility (v2) mode:

  - If EL3 is present:

    ICC_SRE_EL3.SRE (bit 0) must be initialised to 0b0.

  - If the kernel is entered at EL1:

    ICC_SRE_EL2.SRE (bit 0) must be initialised to 0b0.

  - The DT or ACPI tables must describe a GICv2 interrupt controller.

  For CPUs with pointer authentication functionality:

  - If EL3 is present:
    - SCR_EL3.APK (bit 16) must be initialised to 0b1
    - SCR_EL3.API (bit 17) must be initialised to 0b1
  - If the kernel is entered at EL1:
    - HCR_EL2.APK (bit 40) must be initialised to 0b1
    - HCR_EL2.API (bit 41) must be initialised to 0b1

  For CPUs with Activity Monitors Unit v1 (AMUv1) extension present:

  - If EL3 is present:
    - CPTR_EL3.TAM (bit 30) must be initialised to 0b0
    - CPTR_EL2.TAM (bit 30) must be initialised to 0b0
    - AMCNTENSET0_EL0 must be initialised to 0b1111
    - AMCNTENSET1_EL0 must be initialised to a platform specific value having 0b1 set for the corresponding bit for each of the auxiliary counters present.
  - If the kernel is entered at EL1:
    - AMCNTENSET0_EL0 must be initialised to 0b1111
    - AMCNTENSET1_EL0 must be initialised to a platform specific value having 0b1 set for the corresponding bit for each of the auxiliary counters present.

  For CPUs with the Fine Grained Traps (FEAT_FGT) extension present:

  - If EL3 is present and the kernel is entered at EL2:
    - SCR_EL3.FGTEn (bit 27) must be initialised to 0b1.

  For CPUs with support for HCRX_EL2 (FEAT_HCX) present:

  - If EL3 is present and the kernel is entered at EL2:
    - SCR_EL3.HXEn (bit 38) must be initialised to 0b1.

For CPUs with Advanced SIMD and floating point support:

- If EL3 is present:
  - CPTR_EL3.TFP (bit 10) must be initialised to 0b0.
- If EL2 is present and the kernel is entered at EL1:
  - CPTR_EL2.TFP (bit 10) must be initialised to 0b0.

For CPUs with the Scalable Vector Extension (FEAT_SVE) present:

- if EL3 is present:
  - CPTR_EL3.EZ (bit 8) must be initialised to 0b1.
  - ZCR_EL3.LEN must be initialised to the same value for all CPUs the kernel is executed on.
- If the kernel is entered at EL1 and EL2 is present:
  - CPTR_EL2.TZ (bit 8) must be initialised to 0b0.
  - CPTR_EL2.ZEN (bits 17:16) must be initialised to 0b11.
  - ZCR_EL2.LEN must be initialised to the same value for all CPUs the kernel will execute on.

For CPUs with the Scalable Matrix Extension (FEAT_SME):

- If EL3 is present:
  - CPTR_EL3.ESM (bit 12) must be initialised to 0b1.
  - SCR_EL3.EnTP2 (bit 41) must be initialised to 0b1.
  - SMCR_EL3.LEN must be initialised to the same value for all CPUs the kernel will execute on.

- If the kernel is entered at EL1 and EL2 is present:

  - CPTR_EL2.TSM (bit 12) must be initialised to 0b0.
  - CPTR_EL2.SMEN (bits 25:24) must be initialised to 0b11.
  - SCTLR_EL2.EnTP2 (bit 60) must be initialised to 0b1.
  - SMCR_EL2.LEN must be initialised to the same value for all CPUs the kernel will execute on.

For CPUs with the Scalable Matrix Extension FA64 feature (FEAT_SME_FA64)

- If EL3 is present:
  - SMCR_EL3.FA64 (bit 31) must be initialised to 0b1.

- If the kernel is entered at EL1 and EL2 is present:

  - SMCR_EL2.FA64 (bit 31) must be initialised to 0b1.

The requirements described above for CPU mode, caches, MMUs, architected timers, coherency and system registers apply to all CPUs. All CPUs must enter the kernel in the same exception level. Where the values documented disable traps it is permissible for these traps to be enabled so long as those traps are handled transparently by higher exception levels as though the values documented were set.

The boot loader is expected to enter the kernel on each CPU in the following manner:

- The primary CPU must jump directly to the first instruction of the kernel image. The device tree blob passed by this CPU must contain an 'enable-method' property for each cpu node. The supported enable-methods are described below.

  It is expected that the bootloader will generate these device tree properties and insert them into the blob prior to kernel entry.

- CPUs with a "spin-table" enable-method must have a 'cpu-release-addr' property in their cpu node. This property identifies a naturally-aligned 64-bit zero-initalised memory location.

  These CPUs should spin outside of the kernel in a reserved area of memory (communicated to the kernel by a /memreserve/ region in the device tree) polling their cpu-release-addr location, which must be contained in the reserved region. A wfe instruction may be inserted to reduce the overhead of the busy-loop and a sev will be issued by the primary CPU. When a read of the location pointed to by the cpu-release-addr returns a non-zero value, the CPU must jump to this value. The value will be written as a single 64-bit little-endian value, so CPUs must convert the read value to their native endianness before jumping to it.

- CPUs with a "psci" enable method should remain outside of the kernel (i.e. outside of the regions of memory described to the kernel in the memory node, or in a reserved area of memory described to the kernel by a /memreserve/ region in the device tree). The kernel will issue CPU_ON calls as described in ARM document number ARM DEN 0022A ("Power State Coordination Interface System Software on ARM processors") to bring CPUs into the kernel.

  The device tree should contain a 'psci' node, as described in Documentation/devicetree/bindings/arm/psci.yaml.

- Secondary CPU general-purpose register settings
  - x0 = 0 (reserved for future use)
  - x1 = 0 (reserved for future use)
  - x2 = 0 (reserved for future use)
  - x3 = 0 (reserved for future use)