U-Boot as PSCI provider on ARM64

Marek Vasut

February 4th, 2023

PSCI – What

- Power State Coordination Interface
- ARM DEN 0022 [LINK]
- Software interface used on ARM systems to control:
  - CPU core power off and on
  - CPU idle entry/exit
  - System suspend/resume
  - System reset and power off
- Optional on ARMv7a, mandatory on ARMv8a and newer
- SCMI – another interface designed toward DVFS and power management on ARM

PSCI – Why

- Convenience
  - Repeated integration into all OSes
- Complexity of contemporary system power management
  - Increasingly complex code
  - Abundance of quirks and hardware bug workarounds
- Security
  - Better privilege separation between kernel and firmware
  - Broken kernel cannot easily damage hardware or cause harm
- Virtualization
  - CPU core state management in virtualization

PSCI – How

- SMCCC – SMC Call Convention
- ARM DEN 0028 [LINK]
- Define call convention around SMC/HVC instructions
- Call parameters and return values passed via registers
- SMC – Trigger synchronous exception in EL3 (Supervisor)
- HVC – Trigger synchronous exception in EL2 (Hypervisor)
- EL – Exception Level
  - EL3 – highest privilege, basically access to all
  - EL2 – provides virtualization for EL0 and EL1
  - EL1 – virtualized OS
  - EL0 – unprivileged exection
  - EL3, EL2 are optional – EL1, EL0 mandatory
  - Exception can be taken from higher EL to lower EL
  - ARM DDI 0487 D1.1 (aarch64) and G1.2 (aarch32) [LINK]

PSCI – SMC CC

- Software running in low exception level (EL) SMCCC call:
  - Set up CPU register `x0` containing function ID
  - Set up CPU registers `x1..x6` containing parameters
  - Issue `smc` or `hvc` instruction
  - Exception occurs on CPU core
  - Exception handler installed by firmware
  - Exception handler runs in higher exception level:
    - SMC – Secure monitor – EL3
    - HVC – Hypervisor – EL2
  - Handler validates request, optionally performs request
  - Handler returns
  - Software execution continues after `smc` or `hvc` in original EL

PSCI – Function IDs

- Defined in ARM DEN 0022 [LINK]
- SMC CC Function IDs reserved for PSCI
- 32bit and 64bit Function IDs and parameters
- SiP (Silicon Provider) extensions – custom Function IDs
- Function IDs also e.g. in U-Boot `arch/arm/include/asm/psci.h`:

```
1  /* PSCI 0.2 interface */
2  #define ARM_PSCI_0_2_FN_BASE                    0x84000000
3  #define ARM_PSCI_0_2_FN(n)                      (ARM_PSCI_0_2_FN_BASE + (n))
4
5  #define ARM_PSCI_0_2_FN64_BASE                  0xC4000000
6  #define ARM_PSCI_0_2_FN64(n)                    (ARM_PSCI_0_2_FN64_BASE + (n))
7
8  #define ARM_PSCI_0_2_FN_PSCI_VERSION            ARM_PSCI_0_2_FN(0)
9  #define ARM_PSCI_0_2_FN_CPU_SUSPEND             ARM_PSCI_0_2_FN(1)
10 #define ARM_PSCI_0_2_FN_CPU_OFF                 ARM_PSCI_0_2_FN(2)
11 #define ARM_PSCI_0_2_FN_CPU_ON                  ARM_PSCI_0_2_FN(3)
12 #define ARM_PSCI_0_2_FN_AFFINITY_INFO           ARM_PSCI_0_2_FN(4)
13 #define ARM_PSCI_0_2_FN_MIGRATE                 ARM_PSCI_0_2_FN(5)
14 #define ARM_PSCI_0_2_FN_MIGRATE_INFO_TYPE       ARM_PSCI_0_2_FN(6)
15 #define ARM_PSCI_0_2_FN_MIGRATE_INFO_UP_CPU     ARM_PSCI_0_2_FN(7)
16 #define ARM_PSCI_0_2_FN_SYSTEM_OFF              ARM_PSCI_0_2_FN(8)
17 #define ARM_PSCI_0_2_FN_SYSTEM_RESET            ARM_PSCI_0_2_FN(9)
```

PSCI – Where

- Callers
  - U-Boot 2023.04-rc1 `arch/arm/cpu/armv8/fwcall.c` `smc_call()`
  - Linux 6.2.0-rc5 `drivers/firmware/psci/psci.c` `invoke_psci_fn()` and `arm_smccc_smc()`
- Handlers
  - ATF BL31 sources `bl31/aarch64/runtime_exceptions.S`
  - U-Boot sources `arch/arm/cpu/armv8/psci.S`

PSCI – SMC CC call code

U-Boot 2023.04-rc1 arch/arm/cpu/armv8/fwcall.c

```
1  /*
2   * void smc_call(arg0, arg1...arg7)
3   *
4   * issue the secure monitor call
5   *
6   * x0~x7: input arguments
7   * x0~x3: output arguments
8   */
9  void smc_call(struct pt_regs *args)
10 {
11         asm volatile(
12                 "ldr x0, %0\n" /* <--- Function ID */
13                 "ldr x1, %1\n" /* <--- Input parameters */
14                 "ldr x2, %2\n"
15                 "ldr x3, %3\n"
16                 "ldr x4, %4\n"
17                 "ldr x5, %5\n"
18                 "ldr x6, %6\n"
19                 "smc     #0\n" /* <--- SMC call (or hvc for HVC) */
20                 "str x0, %0\n" /* <--- Results */
21                 "str x1, %1\n"
22                 "str x2, %2\n"
23                 "str x3, %3\n"
24                 : "+m" (args->regs[0]), "+m" (args->regs[1]),
25                   "+m" (args->regs[2]), "+m" (args->regs[3])
26                 : "m" (args->regs[4]), "m" (args->regs[5]),
27                   "m" (args->regs[6])
28                 : "x0", "x1", "x2", "x3", "x4", "x5", "x6", "x7",
29                   "x8", "x9", "x10", "x11", "x12", "x13", "x14", "x15",
30                   "x16", "x17");
31 }
```

PSCI – U-Boot shell wrapper

U-Boot 2023.04-rc1

```
u-boot=> smc
smc - Issue a Secure Monitor Call

Usage:
smc <fid> [arg1 ... arg6] [id]
  - fid Function ID
  - arg SMC arguments, passed to X1-X6 (default to zero)
  - id  Secure OS ID / Session ID, passed to W7 (defaults to zero)

# arch/arm/include/asm/psci.h
# ARM_PSCI_0_2_FN_PSCI_VERSION  0x84000000
# ARM_PSCI_VER_1_1              0x00010001
u-boot=> smc 0x84000000 0 0 0 0 0 0
Res:  65537 0 0 0
# Note that 65537 = 0x10001 = ARM_PSCI_VER_1_1
```

PSCI – Caller – Linux

Linux 6.2.0-rc5 drivers/firmware/psci/psci.c

```
1  /* Calls 'smc' instruction in arch/arm64/kernel/smccc-call.S */
2  static unsigned long __invoke_psci_fn_smc(unsigned long function_id,
3                          unsigned long arg0, unsigned long arg1,
4                          unsigned long arg2)
5  {
6          struct arm_smccc_res res;
7
8          arm_smccc_smc(function_id, arg0, arg1, arg2, 0, 0, 0, 0, &res);
9          return res.a0;
10 }
11 ...
12 static u32 psci_0_2_get_version(void)
13 {
14         return invoke_psci_fn(PSCI_0_2_FN_PSCI_VERSION, 0, 0, 0);
15 }
16 ...
17 static void set_conduit(enum arm_smccc_conduit conduit)
18 {
19         switch (conduit) {
20         case SMCCC_CONDUIT_HVC:
21                 invoke_psci_fn = __invoke_psci_fn_hvc;
22                 break;
23         case SMCCC_CONDUIT_SMC:
24                 invoke_psci_fn = __invoke_psci_fn_smc;
25 ...
26 }
```

PSCI – Handler requirements

- Ability to receive exception on CPU core
- SMP
  - Correctly configured GIC
  - CPU start code for secondary cores
  - Ability to receive IPI

U-Boot

- PSCI implementation for ARM32
  - SMC handler – `arch/arm/cpu/armv7/psci.S`
  - CPU entry code – `arch/arm/cpu/armv7/start.S`
  - C Example – `arch/arm/mach-stm32mp/psci.c`
- PSCI implementation for Aarch64
  - SMC handler – `arch/arm/cpu/armv8/psci.S`
  - CPU entry code – `arch/arm/cpu/armv8/start.S`
  - C Example – `arch/arm/mach-rmobile/psci-r8a779a0.c`

U-Boot PSCI on UP

- Entire U-Boot runs in EL3, OS still runs in EL2
- EL3 to EL3 SMC not possible, EL2 to EL3 SMC possible
- PSCI setup on U-Boot start up:
  - `armv8_setup_psci()` called on U-Boot start up
    - Relocate all code marked `__secure` to secure RAM
    - Setup secure MMU tables to protect secure RAM
    - Entire PSCI is in `__secure` section of RAM or SRAM
  - `psci_setup_vectors()` installs SMC handler vectors
- PSCI entry on `smc` exception:
  - `arch/arm/cpu/armv8/psci.c el3_exception_vectors handle_sync`
  - `handle_sync` calls either `handle_smc32` or `handle_smc64` (or fails on unhandled sync exception)
  - `handle_smc64` calls `handle_psci`
  - `handle_psci` saves params and triggers PSCI function ID hook
  - PSCI function ID hook is a piece of C code

PSCI – Entry point

arch/arm/cpu/armv8/psci.S

```
1  handle_sync:
2          mov       x15, x30
3          mov       x14, x0
4
5          bl        psci_get_cpu_id
6          bl        psci_get_cpu_stack_top
7  ...
8          cmp       x9, #0x13
9  ***     b.eq      handle_smc32
10         cmp       x9, #0x17
11 ***     b.eq      handle_smc64
12
13         b         unhandled_exception
14 ...
15 el3_exception_vectors:
16         b         unhandled_exception        /* Sync, Current EL using SP0 */
17         .align    7
18         b         unhandled_exception        /* IRQ, Current EL using SP0 */
19 ...
20 ***     b         handle_sync                /* Sync, Lower EL using AArch64 */
21         .align    7
22         b         unhandled_exception        /* IRQ, Lower EL using AArch64 */
23 ...
24         b         unhandled_exception        /* SError, Lower EL using AArch32 */
25 ...
26 ENTRY(psci_setup_vectors)
27         adr       x0, el3_exception_vectors
28         msr       vbar_el3, x0
29         ret
30 ENDPROC(psci_setup_vectors)
```

PSCI – Entry point

arch/arm/cpu/armv8/psci.S

```
1  handle_psci:
2          psci_enter
3  1:      ldr             x10, [x9]       /* Load PSCI function table */
4          cbz             x10, 3f         /* If reach the end, bail out */
5          cmp             x10, x0
6          b.eq            2f              /* PSCI function found */
7          add             x9, x9, #16     /* If not match, try next entry */
8          b               1b
9
10 2:      ldr             x11, [x9, #8]   /* Load PSCI function */
11 ***     blr             x11             /* Call PSCI function */
12         psci_return
13
14 3:      mov             x0, #ARM_PSCI_RET_NI
15         psci_return
16 ...
17 handle_smc64:
18         /* check SMC32 or SMC64 calls */
19         ubfx        x9, x0, #30, #1
20 ***     cbz         x9, handle_smc32
21
22         /* SMC function ID 0xC4000000-0xC400001F: 64 bits PSCI */
23         ldr         x9, =0xC400001F
24         cmp         x0, x9
25 ***     b.gt        handle_svc
26         ldr         x9, =0xC4000000
27         cmp         x0, x9
28 ***     b.lt        handle_svc
29
30         adr         x9, _psci_64_table
31 ***     b           handle_psci
```

PSCI – Function IDs hook

arch/arm/mach-imx/imx8m/psci.c ([patches posted])

```
1  __secure void psci_system_off(void)
2  {
3          writel(SNVS_LPCR_TOP | SNVS_LPCR_DP_EN | SNVS_LPCR_SRTC_ENV,
4                 SNVS_BASE_ADDR + SNVS_LPCR);
5
6          while (1)
7                  wfi();
8  }
9
10 ...
11
12 __secure s32 psci_features(u32 __always_unused function_id, u32 psci_fid)
13 {
14         switch (psci_fid) {
15         case ARM_PSCI_0_2_FN_PSCI_VERSION:
16         case ARM_PSCI_0_2_FN_CPU_OFF:
17         case ARM_PSCI_0_2_FN_CPU_ON:
18 ...
19         case ARM_PSCI_0_2_FN_SYSTEM_OFF:
20         case ARM_PSCI_0_2_FN_SYSTEM_RESET:
21 ...
22                 return 0x0;
23 ...
```

U-Boot PSCI entry on SMP

- CPU entry code – `arch/arm/cpu/armv8/start.S`
- Secondary cores must be configured after power on
- Multi-entry U-Boot configuration
- Each core starts in EL3
- Each core enters U-Boot via U-Boot entry point
- GIC is configured so core can receive IPI
- In case of secondary cores:
  - Core placed in EL2 in which it enters OS
  - Core waiting for IPI from core which triggered PSCI core ON
  - Once IPI received, CPU core is released in EL2
  - Code reads jump target provided by PSCI call from `CPU_RELEASE_ADDR`
  - Core in EL2 jumps to address written to `CPU_RELEASE_ADDR`
  - EL2 is used to prevent EL2 OS from elevated EL3 accesses

U-Boot PSCI implementation quick start

- Define `GICD_BASE` (distributor) and `GICR_BASE` (redistributor)
- Make sure `struct mm_region` is configured correctly
  - e.g. RAM is NonSecure and SRAM is Secure
- Potentially configure any other security related registers in EL3
- Fill in desired SoC specific PSCI function hooks
  - e.g. `arch/arm/mach-imx/imx8m/psci.c`
- Remove BL31 blob from the boot process altogether
- Define:

```
1  # CONFIG_PSCI_RESET is not set ... make sure U-Boot does no SMC calls in EL3
2  CONFIG_ARMV8_MULTIENTRY=y
3  CONFIG_ARMV8_SET_SMPEN=y
4  CONFIG_ARMV8_SPL_EXCEPTION_VECTORS=y
5  CONFIG_ARMV8_EA_EL3_FIRST=y ............ Handle EL3 exceptions
6  CONFIG_ARMV8_PSCI=y
7  CONFIG_ARMV8_PSCI_CPUS_PER_CLUSTER=4 ... 4 CPU cores
8  CONFIG_ARMV8_SECURE_BASE=0x970000 ...... i.MX8M Plus SRAM
9  CONFIG_ARM_SMCCC=y
10 CONFIG_SYS_HAS_ARMV8_SECURE_BASE=y
```

- Compile and enjoy

U-Boot PSCI implementation debugging

- Use U-Boot debug UART and separate UART on board to print debug messages
- When PSCI functions are called, print to debug UART works
- Define in board config:

```
CONFIG_DEBUG_UART_BASE=0x30880000
CONFIG_DEBUG_UART_CLOCK=24000000
CONFIG_DEBUG_UART=y
CONFIG_DEBUG_UART_MXC=y
CONFIG_SPL_DEBUG_UART_BASE=0x30880000
CONFIG_DEBUG_UART_SHIFT=0
```

- Usage in `arch/arm/mach-imx/imx8m/psci.c`:

```
#define psci_debug() ({ \
  printascii(__func__); \
  printascii("["); printascii(__stringify(__LINE__)); printascii("]\n");\
})
...
__secure s32 psci_cpu_on_64(u32 __always_unused function_id, u64 mpidr,
 {
        u32 cpu = 0;
        int ret;
psci_debug();
...
```

U-Boot example

```
U-Boot SPL 2023.01-rc4-00056-g93d1c33e08f (Jan 30 2023 - 20:33:39 +0100)
DDR:    4096 MiB [0x5]
WDT:    Started watchdog@30280000 with servicing every 1000ms (60s timeout)
Trying to boot from BOOTROM
image offset 0x1000, pagesize 0x1, ivt offset 0x0


U-Boot 2023.01-rc4-00056-g93d1c33e08f (Jan 30 2023 - 20:33:39 +0100)

CPU:    Freescale i.MX8MP[8] rev1.1 1600 MHz (running at 1200 MHz)
CPU:    Industrial temperature grade (-40C to 105C) at 43C
...
u-boot=> boot
...
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[    0.000000] Linux version 6.2.0-rc4-next-20230119-00157...
...
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.0 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: Trusted OS migration not required
[    0.000000] psci: SMC Calling Convention v1.0
...
[    0.004965] smp: Bringing up secondary CPUs ...
[    0.017788] Detected VIPT I-cache on CPU1
[    0.017797] CPU features: SANITY CHECK: Unexpected variation in SYS_CNTFRQ_EL0. Boot CPU: 0x000000007a
[    0.017820] CPU features: Unsupported CPU feature variation detected.
[    0.017855] GICv3: CPU1: found redistributor 1 region 0:0x00000000388a0000
[    0.017888] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
...
```

Linux example

```
...
[    0.004965] smp: Bringing up secondary CPUs ...
[    0.017788] Detected VIPT I-cache on CPU1
[    0.017797] CPU features: SANITY CHECK: Unexpected variation in SYS_CNTFRQ_EL0. Boot CPU: 0x000000007a
[    0.017820] CPU features: Unsupported CPU feature variation detected.
[    0.017855] GICv3: CPU1: found redistributor 1 region 0:0x00000000388a0000
[    0.017888] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
...
~ # cat /proc/cpuinfo
processor       : 0
BogoMIPS        : 16.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 1
BogoMIPS        : 16.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4
...
```

Debug UART example

```
psci_version[129]
psci_migrate_info_type[219]
psci_features[261]
psci_features[262]
    [pscii_fid=80000000]
psci_features[261]
psci_features[262]
    [pscii_fid=c4000001]
psci_features[261]
psci_features[262]
    [pscii_fid=c400000e]
psci_features[261]
psci_features[262]
    [pscii_fid=c4000012]
psci_cpu_on_64[141]
psci_cpu_on_64[142]
    [mpidr=00000001]
psci_cpu_on_64[147]
    [cpu=1]
psci_cpu_on_write_entry_point[83]
psci_set_state[55]
psci_cpu_on_power_on[99]
    [cpu=1]
psci_cpu_on_64[154]
psci_cpu_on_64[141]
psci_cpu_on_64[142]
    [mpidr=00000002]
```

End

Thank you for your attention

Marek Vasut <marek.vasut+fosdem23@mailbox.org>