S32G How to Put A53 to WFI

by John Li (nxa08200)

S32G的partition off流程要求核稳定的进入到WFI状态，本文说明如何修改Linux内核，在A53 Linux关机或kernel panic时，如何让所有A53 Core进入WFI。

History	说明	作者
V1	● 创建本文	● John.Li

目录

1 背景说明与参考资料

1.1 背景说明

根据 S32G 芯片手册说明：

28.12.5 Software reset partition turn-off flowchart

For the Cortex-A53 reset domain, clocks for the initial frequencies can be set up by configuring MC_CGM_1 to the FIRC clock. See the clocking chapter for details.

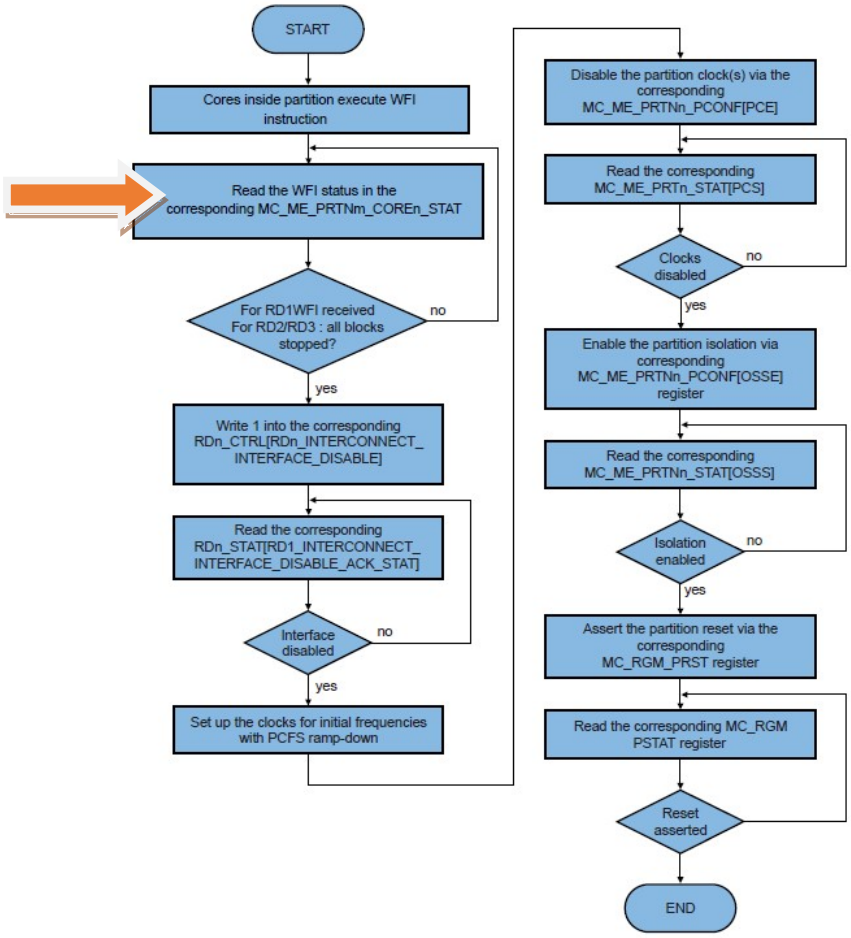


Figure 147. Turn-off flowchart for software reset partition

所以在 partition off 之前，是需要所有的 A53 稳定的处于 WFI 状态。

一般来讲，电源管理作为功能安全的一部分，是在 M7 核实现的，而 M7 核的 MCAL MCU PM partition off 代码如下：

Mcu_ts_*\src\power_ip_MC_ME.c

```
static void Power_Ip_MC_ME_ConfigureCore(const Power_Ip_MC_ME_CoreConfigType * CoreConfigPtr, uint8 PartitionIndex)
{…


  if (MC_ME_PRTNX_COREX_PCONF_CCE_DIS_U32 == (CoreConfigPtr->CorePconfRegValue & MC_ME_PRTN1_CORE0_PCONF_CCE_MASK))
  {
    /* Wait until WFI is set */
    Power_Ip_StartTimeout(&StartTime, &ElapsedTime, &TimeoutTicks, POWER_IP_TIMEOUT_VALUE_US);
    do
    {
      TimeoutOccurred = Power_Ip_TimeoutExpired(&StartTime, &ElapsedTime, TimeoutTicks);


      CoreStatus = Power_Ip_pxMC_ME->McMePrtnArray[PartitionIndex].McMePrtnCoreArray[CoreIndex].PRTN_CORE_STAT;
    } while ( (MC_ME_PRTNX_COREX_STAT_WFI_EXECUTED_U32 != (CoreStatus & MC_ME_PRTN1_CORE0_STAT_WFI_MASK)) && (!TimeoutOccurred) );
    /* timeout notification */
```

所以 M7 代码会去轮询 A53 的 WFI 状态，一旦获得其 WFI 状态标志位，就继续执行之后的电源管理操作代码(也可以参考功能安全的 SPD 包里的代码)。

但是在 Linux 的 idle 进程中(以 BSP30 为例)：

Drivers\cpuidle\cpuidle-arm.c

```
device_initcall(arm_idle_init);
|-> arm_idle_init_cpu
|   |-> ret = dt_init_idle_driver(drv, arm_idle_state_match, 1);
static const struct of_device_id arm_idle_state_match[] __initconst = {
    { .compatible = "arm,idle-state",
      .data = arm_enter_idle_state },
    { },
};
static struct cpuidle_driver arm_idle_driver __initdata = {
    .name = "arm_idle",
    .owner = THIS_MODULE,
    /*
     * State at index 0 is standby wfi and considered standard
```

```
     * on all ARM platforms. If in some platforms simple wfi
     * can't be used as "state 0", DT bindings must be implemented
     * to work around this issue and allow installing a special
     * handler for idle state index 0.
     */
    .states[0] = {
            .enter                  = arm_enter_idle_state,
            .exit_latency           = 1,
            .target_residency       = 1,
            .power_usage                    = UINT_MAX,
            .name                   = "WFI",
            .desc                   = "ARM WFI",
    }
};
|   |   |->CPU_PM_CPU_IDLE_ENTER(arm_cpuidle_suspend, idx);
#define __CPU_PM_CPU_IDLE_ENTER(low_level_idle_enter,                   \
                                idx,                                    \
                                state,                                  \
                                is_retention)                           \
({                                                                      \
    int __ret = 0;                                                      \
                                                                        \
    if (!idx) {                                                         \
            cpu_do_idle();                                              \
            return idx;                                                 \
    }                                                                   \
                                                                        \
    if (!is_retention)                                                  \
            __ret =  cpu_pm_enter();                                    \
    if (!__ret) {                                                       \
            __ret = low_level_idle_enter(state);                        \
            if (!is_retention)                                          \
                    cpu_pm_exit();                                      \
    }                                                                   \
                                                                        \
```

```
    __ret ? -1 : idx;                                                   \
})

#define CPU_PM_CPU_IDLE_ENTER(low_level_idle_enter, idx)        \
    __CPU_PM_CPU_IDLE_ENTER(low_level_idle_enter, idx, idx, 0)
| | | |->cpu_do_idle
| | | | |->__cpu_do_idle();
static void noinstr __cpu_do_idle(void)
{
    dsb(sy);
    wfi();
}
```

也就是说 Linux idle 进程会随时让 A53 进入 WFI 状态，在有中断来时，又退出而运行，这就会导致一个问题，就是就算 Linux 正在工作的情况下，M 核的代码也会轮询到 WFI 状态，从而进行 partition off 操作，而实际上 A53 并没有稳定在 WFI 状态。

在实际产品设计中有以下两种典型应用工况，分别是 Linux 主动关机，和被动重启的情况：

1. M 核升级完 Linux 镜像后，执行 A53 的 partiton off/on 重启操作，这个时候可以通知 Linux 主动先关机，所以实际上 Linux 是执行了 poweroff 命令的。
2. M 核监控 A 核的运行状态，(一般是使用 IPCF/ENET/底层 SRAM Flag 核间同步)的方法，如果发现了 A 核 panic 了，则需要 M 核将 A 核重启。

所以结论就是需要在 poweroff 和 panic 代码中做修改，来保证所有 A53 稳定在 WFI 状态，(这个时候因为所有应用进程已经被杀死，所以大部分时间 A53 处于 idle 状态，并且偶尔会被中断唤醒回来)。

1.2 参考资料

分类	文档编号	名称	如何获得
文档	S32G2.pdf S32G3.pdf	S32G 芯片手册	从www.nxp.com/s32g下载
文档	IHI0069H_gic_architecture_specification.pdf	ARM GIC spec	从 ARM 网站下载
文档	DDI0516E_gic5000_r1p1_trm.pdf	ARM GIC 芯片手册	从 ARM 网站下载

Bsp30 文档包	包括 S32G_BSP30.0_User_Manual.pdf	BSP 手册	从www.nxp.com个人帐号下载
Bsp36 文档包	包括 S32G2/3_LinuxBSP_36.0_User_Manual.pdf	BSP 手册	从www.nxp.com个人帐号下载

1.3 测试工具

考虑如果单纯 Linux 的测试环境，不能使用 Lauterbach 连接 A 核，这样会导致 WFI 退出，所以要使用 Lauterbach 连接 M 核，则需要一个脚本来初始化 M 核及连接。

我们使用~\pfe\Driver\1.0.0\PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\run_main.cmm 脚本，并将其中与此 PFE 相关的部分删除掉，做为 M 核的连接脚本：

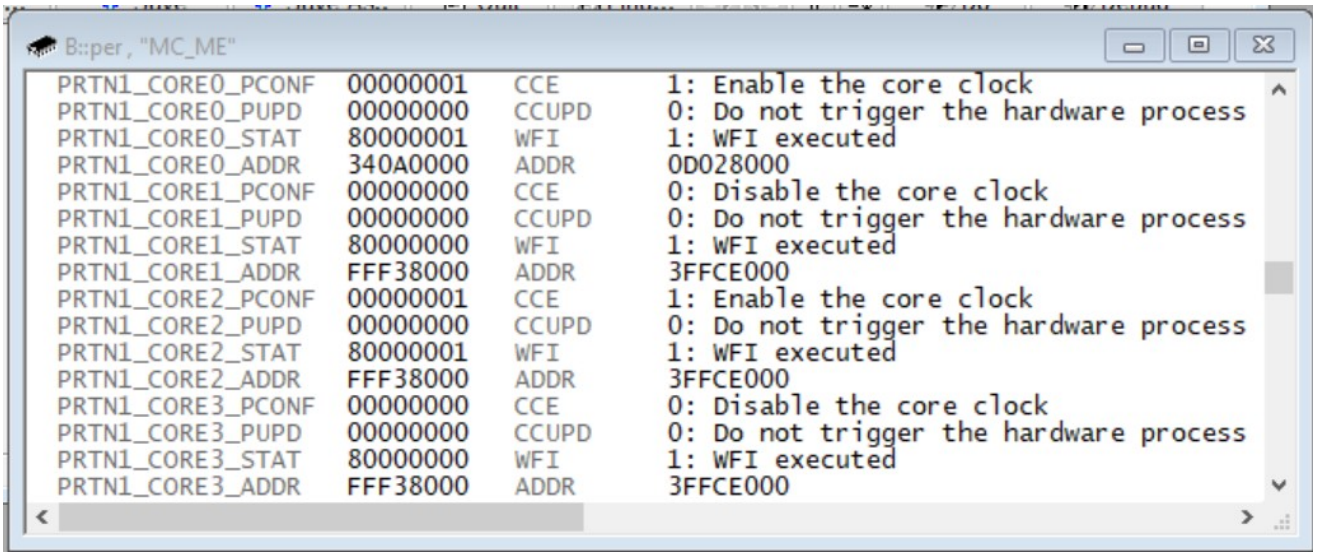
```
; in case of linux slave, uncomment the following line to give some time to linux to boot
; wait 5s
```

;以下两句代码之前的内容全部删除掉：

```
;data.load.elf &elf_file /GLOBTYPES
;…
;MENU.ReProgram ~~/mens32g2.men
```

然后运行脚本，A53 会重启，之后就可以通过 M 核的 Lauterbach 来查看相关寄存器了：

t32marm.exe ->run script->S32G(menu item)->MC_ME：



可以看到默认 Linux BSP 镜像的 idle 进程让四个 A53 处于 WFI 状态，而且如果 A53 忙会偶尔退出 WFI 状态。

2 Panic

2.1 Panic 代码流程分析

以下对比 BSP36(有 ATF 支持版本)和 BSP30(无 ATF 支持版本)：

\kernel\panic.c

	Bsp36	Bsp30
Panic()	… **local_irq_disable();** … /* * It's possible to come here directly from a panic-assertion and * not have preempt disabled. Some functions called from here want * preempt to be disabled. No point enabling it later though... * * Only one CPU is allowed to execute the panic code from here. For * multiple parallel invocations of panic, all other CPUs either * stop themself or will wait until they are stopped by the 1st CPU * with smp_send_stop(). * * `old_cpu == PANIC_CPU_INVALID' means this is the 1st CPU which * comes here, so go ahead. * `old_cpu == this_cpu' means we came from nmi_panic() which sets * panic_cpu to this CPU. In this case, this is also the 1st CPU. */ **panic_smp_self_stop();** |-> local_cpu_stop | |-> cpu_park_loop | | |-> for (;;) { **wfe();** **wfi();** }	
	If set /proc/kernel/panic_on_oops: oops will lead panic **If set /proc/kernel/panic > 0, will restart kernel** if (panic_timeout != 0) { /* * This will not be a clean reboot, with everything * shutting down. But if there is a chance of * rebooting the system it will be rebooted. */	

	if (panic_reboot_mode != REBOOT_UNDEFINED)
	reboot_mode = panic_reboot_mode;
	emergency_restart();
	}
	emergency_restart();
	|-> machine_emergency_restart
	| |-> machine_restart
	/* Disable interrupts first */
	local_irq_disable();
	smp_send_stop();
	/* Now call the architecture specific reboot code. */
	| | |-> do_kernel_restart
	| | | |->atomic_notifier_call_chain(&restart_handler_list, reboot_mode, cmd);	

Reboot.c	int register_restart_handler(struct notifier_block *nb)
	{
	return atomic_notifier_chain_register(&restart_handler_list, nb);
	}	

	\arch\arm64\kernel\psci.c	\drivers\power\reset S32gen1_reboot.c

	register_restart_handler(&psci_sys_reset_nb);
	static struct notifier_block psci_sys_reset_nb
	= {
	.notifier_call = psci_sys_reset,
	.priority = 129,
	};
	psci_sys_reset,
	|->invoke_psci_fn
	(PSCI_0_2_FN_SYSTEM_RESET, 0, 0, 0);
	#define
	PSCI_0_2_FN_SYSTEM_RESET PSCI_0_2_FN(9)
	#define PSCI_0_2_FN_BASE 0x84000000
	#define PSCI_0_2_FN(n) (PSCI_0_2_FN_BASE + (n))
		priv->s32gen1_reboot_nb.notifier_call
		= s32gen1_reboot;
		err = register_restart_handler
		(&priv->s32gen1_reboot_nb);
		s32gen1_reboot
		static int **s32gen1_reboot**
		(struct notifier_block
		*this, unsigned long mode, void *cmd)
		{
		unsigned long timeout;
		struct s32gen1_reboot_priv *priv =
		container_of
		(this, struct s32gen1_reboot_priv,
		if (!priv->mc_rgm || !priv->mc_me)
		return 0;
		regmap_write(priv->mc_rgm,
		MC_RGM_FRET,
		MC_RGM_FRET_VALUE);
		…
		return 0;
		}

	Atf/lib/psci/psci_main.c	
	psci_smc_handler() case PSCI_SYSTEM_RESET: psci_system_reset(); /* We should never return from psci_system_reset() */ #define PSCI_SYSTEM_RESET U(0x84000009) |-> psci_plat_pm_ops->system_reset(); =s32_system_reset | |-> static void __dead2 s32_system_reset(void)	

	{
	NOTICE("S32 TF-A: %s\n", __func__);
	if (is_scp_used()) {
	scp_reset_platform();
	} else {
	s32_destructive_reset();
	}
	plat_panic_handler();
	}
	| | |->
	
	func plat_panic_handler
	wfi
	b plat_panic_handler
	endfunc plat_panic_handler	

对于是否执行到 WFI 总结如下：

- panic_smp_self_stop 之前执行了关 irq 中断，然后非主核会运行 WFI 语句。
- 以下操作将 oops 导入到 panic，并且 panic 会运行 reset，细节后述。

If set /proc/kernel/panic_on_oops: oops will lead panic
If set /proc/kernel/panic > 0, will restart kernel

- machine_restart 中先执行关 irq 中断，然后调用 do_kernel_restart 触发 restart_handler_list 通知链。
- BSP36 由函数 psci_sys_reset 接收通知链事件，并通知 ATF，ATF 代码执行 s32_destructive_reset();直接重启，主核没有调用 plat_panic_handler();执行 WFI。
- BSP30 由函数 s32gen1_reboot 函数接收通知链事件，直接执行 partition off/on 重启，没有执行 WFI。

从上分析可知，默认的 BSP 及内核代码应该让非主核进入 WFI 状态，而主核不能进入 WFI 状态。

实际测试结果是所有 A53 都无法进入 WFI。(当执行 panic 代码时，所有的 A53 会先退出 idle 执行代码，所以会先退出 WFI 状态，只是 panic 代码无法让 A53 进入 WFI)。所以需要修改内核及驱动代码来实现。

在修改 panic 之前需要配置内核来增加测试工具，及将 panic 行为修改为重启：

~/BSP30/standalone/linux-bsp30$

Vi .config

CONFIG_PANIC_ON_OOPS_VALUE=1 #让 oops 也执行 panic。

CONFIG_PANIC_TIMEOUT=5 # panic 之后 5S 进入重启流程。

CONFIG_MAGIC_SYSRQ=y #增加 SYSRQ 功能，用于制造 panic。

也可以通过 make menuconfig 修改。

2.2 BSP30 修改说明(Non-ATF)

修改包括：

1. 主核不重启，修改为调用 WFI：

vi drivers/power/reset/s32gen1-reboot.c

```
static int s32gen1_reboot(struct notifier_block *this, unsigned long mode,
                          void *cmd)
{
   for(;;)
   {
          dsb(sy);
          wfi();
   }
  #if 0
      //将重启的函数体去掉。
  #endif
}
```

2. 考虑 WFI 之前，关掉 IRQ 不够，还需要将 GIC 从 CPU 上断掉，如下代码分析：

Kernel\cpu_pm.c

```
cpu_pm_enter
|->cpu_pm_notify_robust(CPU_PM_ENTER, CPU_PM_ENTER_FAILED); //触发 cpu_pm_notifier_chain 通知链，参数为 CPU_PM_ENTER
```

Drivers\irqchip\irq-gic-v3.c

```
static void gic_cpu_pm_init(void)
{
    cpu_pm_register_notifier(&gic_cpu_pm_notifier_block); //注册上 cpu_pm_notifier_chain 通知链
}
static struct notifier_block gic_cpu_pm_notifier_block = {
    .notifier_call = gic_cpu_pm_notifier,
};
static int gic_cpu_pm_notifier(struct notifier_block *self,
                               unsigned long cmd, void *v)
{
    if (cmd == CPU_PM_EXIT) {…
    } else if (cmd == CPU_PM_ENTER && gic_dist_security_disabled()) {
```

```
            gic_write_grpen1(0);
            gic_enable_redist(false);
        }
    return NOTIFY_OK;
}
```

所以理论上在非主核及主核执行 WFI 与关 IRQ 之前，调用 cpu_pm_enter()，就可以通知 A53 CPU 断开 GIC CPU interface，如下：

vi arch/arm64/kernel/smp.c

```
static void local_cpu_stop(void)
{
    set_cpu_online(smp_processor_id(), false);
    cpu_pm_enter();//johnli add 在非主核 cpu_park_loop 调用 WFI 前先关掉 GIC 接口
    local_daif_mask();
    sdei_mask_local_cpu();
    cpu_park_loop();
}
```

vi arch/arm64/kernel/process.c

```
void machine_restart(char *cmd)
{
    /* Disable interrupts first */
    cpu_pm_enter(); //johnli 主核关 irq 中断前，先关掉 GIC 接口
    local_irq_disable();
    smp_send_stop();
```

3. 修改非主核 WFI 执行代码：

vi arch/arm64/include/asm/smp.h

```
static inline void cpu_park_loop(void)
{
    for (;;) {
//johnli test        wfe();
        dsb(sy); //johnli add
        wfi();
    }
}
```

4. 修改 gic 内核驱动代码：

GIC 驱动中：

vi drivers/irqchip/irq-gic-v3.c

```
static int gic_cpu_pm_notifier(struct notifier_block *self,
                               unsigned long cmd, void *v)
{… else if (cmd == CPU_PM_ENTER && gic_dist_security_disabled()) {
                gic_write_grpen1(0);
                gic_enable_redist(false);
    }
static inline bool gic_dist_security_disabled(void)
{
    return readl_relaxed(gic_data.dist_base + GICD_CTLR) & GICD_CTLR_DS;
}
```

参考 GIC 文档：

DS, bit [6]

Disable Security.

0b0 Non-secure accesses are not permitted to access and modify registers that control Group 0 interrupts.

0b1 Non-secure accesses are permitted to access and modify registers that control Group 0 interrupts.

If DS is written from 0 to 1 when GICD_CTLR.ARE_S == 1, then GICD_CTLR.ARE for the single Security state is RAO/WI.

If the Distributor only supports a single Security state, this bit is RAO/WI.

If the Distributor supports two Security states, it IMPLEMENTATION DEFINED whether this bit is programmable or implemented as RAZ/WI.

When this field is set to 1, all accesses to GICD_CTLR access the single Security state view, and all bits are accessible.

When set to 1, this field can only be cleared by a hardware reset.

Writing this bit from 0 to 1 is UNPREDICTABLE if any of the following is true:

- GICD_CTLR.EnableGrp0==1.
- GICD_CTLR.EnableGrp1S==1.
- GICD_CTLR.EnableGrp1NS==1.
- One or more INTID is in the Active or Active and Pending state.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

调试时发现函数 gic_dist_security_disabled()返回 0，所以 GIC 关接口代码实际上不会执行。

将其修改为：

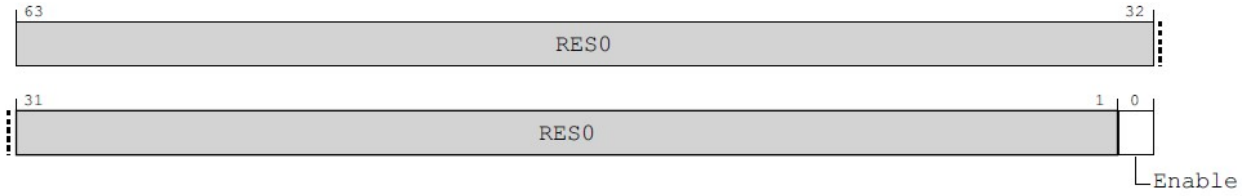
```
//else if (cmd == CPU_PM_ENTER && gic_dist_security_disabled()) {
else if (cmd == CPU_PM_ENTER) {
```

解决。

另外再分析在 DS=0 时的执行代码是否正确：

```
static inline void gic_write_grpen1(u32 val)
{
    write_sysreg_s(val, SYS_ICC_IGRPEN1_EL1);
    isb();
}
```

参考 GIC 文档：



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 1 interrupts for the current Security state.

0b0 Group 1 interrupts are disabled for the current Security state.

0b1 Group 1 interrupts are enabled for the current Security state.

Virtual accesses to this register update ICH_VMCR_EL2.VENG1.

If EL3 is present:

- The Secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1S bit.
- The Non-secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

所以根据文档说明，就算是 BSP30 没有 ATF，在内核处于 EL1 的情况，也是可以配置到 EL3 的，(是否有理解错误?)。

以下代码将当前核的 GIC 接口置为 Sleep 状态，从而断掉 GIC CPU 接口：

```
gic_enable_redist
```

```
|-> gic_data_rdist_rd_base
if (enable)
        /* Wake up this CPU redistributor */
        val &= ~GICR_WAKER_ProcessorSleep;
    else
        val |= GICR_WAKER_ProcessorSleep;
    writel_relaxed(val, rbase + GICR_WAKER);
    if (!enable) {          /* Check that GICR_WAKER is writeable */
        val = readl_relaxed(rbase + GICR_WAKER);
        if (!(val & GICR_WAKER_ProcessorSleep))
            return;   /* No PM support in this redistributor */
    }

    while (--count) {
        val = readl_relaxed(rbase + GICR_WAKER);
        if (enable ^ (bool)(val & GICR_WAKER_ChildrenAsleep))
            break;
        cpu_relax();
        udelay(1);
    }
```

测试：S32G2 RDB2+Linux BSP30+Lauterbach:

1. 将 RDB2 设置为 SDcard 启动，运行 Lauterbach+脚本，然后打开 MC_ME 窗口，可以看到 274 的四个 A53 在 WFI 状态，偶尔退出。
2. A53 Linux 会重启，然后运行：

```
root@s32g274ardb2:~# cd /proc/
root@s32g274ardb2:/proc# echo c > sysrq-trigger
[   28.471915] sysrq: Trigger a crash
[   28.471935] Kernel panic - not syncing:
[   28.471938] sysrq triggered crash
[   28.579564] SMP: stopping secondary CPUs
[   28.583059] johnli debug;local_cpu_stop,cpu_id=279011944
[   28.583060] johnli debug;local_cpu_stop,cpu_id=279011944
[   28.583060] johnli debug;local_cpu_stop,cpu_id=279011944
[   28.588351] GICv3: johnli debug: pm_enter
[   28.598940] Kernel Offset: disabled
```

```
[   28.602932] GICv3: johnli debug: pm_notifier
[   28.606402] CPU features: 0x0040022,2000200c
[   28.610661] GICv3: johnli debug: pm_enter
[   28.614909] Memory Limit: none
[   28.618903] GICv3: johnli debug: pm_notifier
[   28.621944] Rebooting in 5 seconds..
[   28.626196] GICv3: johnli debug: pm_enter
[   28.633745] GICv3: johnli debug: pm_notifier
[   33.629925] GICv3: johnli debug: pm_enter
[   33.633497] GICv3: johnli debug: pm_notifier
[   33.637752] enter wfi
```

3. 然后在 Lauterbach 看 MC_ME 窗口，可以看到 274 的四个 A53 可以长期稳定在 WFI 状态。

2.3 BSP36 修改说明(ATF)

根据之前 BSP36 Panic 调用分析：

- 非主核调用了 irq 关闭代码，但是没有调用 gic cpuif 断开代码，导致非主核无法进入 WFI，在执行 wfi 语句处死循环。
- 主核调用了 irq 关闭代码后切换至 ATF 中，最后调用：

vi plat/nxp/s32/s32_psci.c

```
static void __dead2 s32_system_reset(void)
{
    NOTICE("S32 TF-A: %s\n", __func__);

    if (is_scp_used()) {
            scp_reset_platform();
    } else {
            s32_destructive_reset(); //此处 reset CPU
    }
    plat_panic_handler();//WFI 语句未调用
}
```

以上调用没有执行 GIC CPU interface 断开操作，

- 所以如果在 ATF 中增加所有 A53 的断开 GIC CPU 接口操作，去掉 s32_destructive_reset()，并且执行 plat_panic_handler()进入 WFI，则非主核与主核都可以进入 WFI。

修改如下：

vi atf/plat/nxp/s32/s32_psci.c

```
static void __dead2 s32_system_die_in_wfi(void)
{
   NOTICE("S32 TF-A: %s\n", __func__);
  for (int i = 0; i < PLATFORM_CORE_COUNT; i++) { //对于不同版本 S32G，需要修改 A53 数量定义
PLATFORM_CORE_COUNT
      gicv3_cpuif_disable(i);
  }
   plat_panic_handler();
}
```

```
#ifdef CONFIG_ENABLE_WFI_FOR_SYSTEM_RESET
.system_reset = s32_system_die_in_wfi,
#else
.system_reset = s32_system_reset,
#endif
#ifdef CONFIG_ENABLE_WFI_FOR_SYSTEM_OFF
.system_off = s32_system_die_in_wfi,
#else
.system_off = s32_system_off,
#endif
```

gicv3_cpuif_disable 的调用如下：与 Linux GIC 驱动中的通知链事件处理函数相似，而 ELx 级别不同。

```
/*******************************************************************************
 * This function disables the GIC CPU interface of the calling CPU using
 * only system register accesses.
 ******************************************************************************/
void gicv3_cpuif_disable(unsigned int proc_num)
{ …
    assert(IS_IN_EL3()); //ATF 工作在 EL3，而 Linux 是 EL1
    /* Disable legacy interrupt bypass */
    write_icc_sre_el3(read_icc_sre_el3() |
                      (ICC_SRE_DIB_BIT | ICC_SRE_DFB_BIT));
    /* Disable Group0 interrupts */ //Linux 不使用 FIQ，所以没有操作 group0，而 ATF 使用了
    write_icc_igrpen0_el1(read_icc_igrpen0_el1() &
                          ~IGRPEN1_EL1_ENABLE_G0_BIT);
    /* Disable Group1 Secure and Non-Secure interrupts */ // Linux 使用 EL1 寄存器，ATF 使用 EL3 寄存器
    write_icc_igrpen1_el3(read_icc_igrpen1_el3() &
```

```
                                ~(IGRPEN1_EL3_ENABLE_G1NS_BIT |
                                IGRPEN1_EL3_ENABLE_G1S_BIT));

    /* Synchronise accesses to group enable registers */
    isb();

    /* Mark the connected core as asleep */
    gicr_base = gicv3_driver_data->rdistif_base_addrs[proc_num];
    assert(gicr_base != 0U);
    gicv3_rdistif_mark_core_asleep(gicr_base); //以下代码设置 WAKER to sleep。
}
```

编译增加编译宏#define CONFIG_ENABLE_WFI_FOR_SYSTEM_RESET。测试结果如下：

```
root@s32g274ardb2:/proc# echo c > sysrq-trigger
[   52.380011] sysrq: Trigger a crash
[   52.380032] Kernel panic - not syncing:
[   52.380036] sysrq triggered crash
Message from syslogd@  at Thu Apr 28 17:59:07 2022 ...
: Kernel panic - not syncing:
[   38.279533] SMP: stopping secondary CPUs
[   38.283046] Kernel Offset: disabled
[   38.286506] CPU features: 0x1,20002001,20000842
[   38.291020] Memory Limit: none
[   38.294063] Rebooting in 5 seconds..
```

Lauterbach 连接查看 274 四个 A53 稳定在 WFI 状态。

3 Poweoff

3.1 Poweroff 代码流程分析

如下分析 poweroff 命令及执行代码：

	Bsp36	Bsp30
Shell command	root@s32g274ardb2:~# which poweroff /sbin/poweroff root@s32g274ardb2:~# cd /sbin/	root@s32g274ardb2:/usr/bin# which reboot

	root@s32g274ardb2:/sbin# ls -l poweroff lrwxrwxrwx 1 root root 14 Mar 9 2018 poweroff -> **/bin/systemctl** root@s32g274ardb2:/sbin# which reboot /sbin/reboot root@s32g274ardb2:/sbin# ls -l /sbin/reboot lrwxrwxrwx 1 root root 14 Mar 9 2018 /sbin/reboot -> **/bin/systemctl**	/sbin/reboot root@s32g274ardb2:/usr/bin# which poweroff /sbin/poweroff root@s32g274ardb2:/sbin# ls -l lrwxrwxrwx 1 root root 23 Mar 9 12:34 poweroff -> /sbin/poweroff.sysvinit lrwxrwxrwx 1 root root 13 Mar 9 12:34 poweroff.sysvinit -> **halt.sysvinit** lrwxrwxrwx 1 root root 21 Mar 9 12:34 reboot -> /sbin/reboot.sysvinit lrwxrwxrwx 1 root root 13 Mar 9 12:34 reboot.sysvinit -> halt.sysvinit

/kernel/reboot.c	SYSCALL_DEFINE4()
	{
	case LINUX_REBOOT_CMD_RESTART:
	kernel_restart(NULL);
	break;
	case LINUX_REBOOT_CMD_POWER_OFF:
	kernel_power_off();
	do_exit
	}
		SYSCALL_DEFINE4()
		{…
		case LINUX_REBOOT_CMD_RESTART:
		kernel_restart(NULL);
		break
		case LINUX_REBOOT_CMD_HALT:
		kernel_halt();
		do_exit(0);
		panic("cannot halt");
		}

	kernel_power_off
	|-> machine_power_off();
	{
	local_irq_disable();
	smp_send_stop();
	if (pm_power_off)
	pm_power_off();
	}
	| |->pm_power_off= psci_sys_poweroff
		kernel_halt
		|-> machine_halt
		void machine_halt(void)
		{
		local_irq_disable();
		smp_send_stop();
		while (1);
		}

Drivers/firmware/psci/psci.c	psci_sys_poweroff invoke_psci_fn(PSCI_0_2_FN_SYSTEM_OFF, 0, 0, 0); #define PSCI_0_2_FN_SYSTEM_OFF PSCI_0_2_FN(8) #define PSCI_0_2_FN_BASE	

	0x84000000 #define PSCI_0_2_FN(n) (PSCI_0_2_FN_BASE + (n))	

Atf/lib/psci/
Psci_main.c
	#define PSCI_SYSTEM_OFF
	U(0x84000008)
	case PSCI_SYSTEM_OFF:
	psci_system_off();
	|->psci_plat_pm_ops->system_off();
	static void __dead2 s32_system_off(void)
	{
	if (is_scp_used()) {
	scp_shutdown_platform();
	} else {
	pmic_system_off();
	}
	plat_panic_handler();
	}	

3.2 BSP30 修改说明(Non-ATF)

所以修改方法为：

vi arch/arm64/kernel/process.c

```
void machine_halt(void)
{
    cpu_pm_enter(); //johnli add
    local_irq_disable();
    smp_send_stop();
    for(;;)//johnli add
    {
      dsb(sy);
      wfi();
    }
    while (1);
}
```

测试过程与这前相似，只是不需要触发 panic，直接执行 poweroff 命令：

```
root@s32g274ardb2:~# poweroff
…
[   32.911489] reboot: System halted
```

[32.911497] printk: enabled sync mode

[32.913494] printk: console [ttyLF0]: printing thread stopped

[33.036385] GICv3: johnli debug: pm_enter

[33.039960] GICv3: johnli debug: pm_notifier

[33.044221] johnli debug;local_cpu_stop,cpu_id=279011944

[33.044221] johnli debug;local_cpu_stop,cpu_id=279011944

[33.044221] johnli debug;local_cpu_stop,cpu_id=279011944

[33.049508] GICv3: johnli debug: pm_enter

[33.064089] GICv3: johnli debug: pm_notifier

[33.068343] GICv3: johnli debug: pm_enter

[33.072335] GICv3: johnli debug: pm_notifier

[33.076588] GICv3: johnli debug: pm_enter

[33.080581] GICv3: johnli debug: pm_notifier

然后如之前一样用 Lauterbach 检查 MC_ME 寄存器。

3.3 BSP36 修改说明(ATF)

ATF 中的修改参考前章，测试结果如下：

root@s32g274ardb2:/sbin# poweroff

…

[980.221368] reboot: Power down

[980.221377] printk: enabled sync mode

[980.223976] printk: console [ttyLF0]: printing thread stopped

Lauterbach 连接可以看到 274 四个 A53 稳定在 WFI 状态。

4 Reboot 情况说明

需要说明的是 reboot 命令在实际应用中，是使用 M 核来重启的，Linux 这边一般不需要执行 partition off/on 操作。

```
case LINUX_REBOOT_CMD_RESTART:
                        kernel_restart(NULL);
kernel_restart
|->machine_restart
```

所以与之前 panic 分析代码相同。

5 STR 情况说明

STR 需要 ATF 支持，目前已经支持 WFI操作，详情参考文档《S32G_Linux_STR_V*.pdf》JohnLi。https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-Linux-STR/ta-p/1652680下载。