S32G Linux Suspend to Ram 说明

by John Li (nxa08200)

本文说明S32G A53核STR详细情况及定制，定制部分说明如何与M7 standby demo结合，来实现整个产品的快速启动。

请注意本文为培训和辅助文档，本文不是官方文档的替代，请一切以官方文档为准。

历史	说明	作者
V1	● 创建本文	John.Li

目录

1 参考资料说明

分类	名称	说明	备注
文档	AN12880-5510-standby.pdf	VR5510 standby 说明文档	Search on www.nxp.com
文档	AN12952-Power Saving Techniques.pdf	S32G2 suspend 说明文档	Search on www.nxp.com 重要
文档	S32G2_LinuxBSP_32.0_Quick_Start.pdf	S32G2 Linux 开发包，注意 User Manual 为 BSP 详细说明，重要。	Download from www.nxp.com
文档	S32G2_LinuxBSP_32.0_Release_Notes.pdf		
文档	S32G2_LinuxBSP_32.0_User_Manual.pdf		
文档	S32G2_LinuxBSP32.0_quality_package.zip		
文档	S32G2_PFE_LinuxBSP32.0.0_license.txt		
文档	S32G2RM.pdf	芯片手册	Download from www.nxp.com
文档	ds649820 - VR5510 Datasheet Rev2 (2.0).pdf	芯片手册	Download from www.nxp.com

2 Demo 创建运行过程

参考文档《S32G_ATF_BSP32_V*.pdf》，或《S32G_Uboot_BSP32_V*.pdf》，或《S32G_Kernel_BSP32_V*.pdf》，或官方文档《S32G2_LinuxBSP_32.0_User_Manual.pdf》，来创建 Linux 环境与镜像，运行镜像参考文档《S32G2_LinuxBSP_32.0_User_Manual.pdf》如下章节：

21.3 Suspend to RAM

Suspend to RAM, or STR, is a complex power management feature supported by BSP 32.0. STR is a high-level concept, implemented by the Linux kernel and the TF-A, and underlaid by the S32G SoC's Standby state.

STR saves the software context and transitions the SoC to Standby state, which is an SoC low power state (not to be confounded with the VR5510 PMIC's Standby and Deep Sleep modes!). For details on the S32G hardware support for this feature, please refer to the Power Management chapters in the S32G SoC Reference Manual.

来运行 Linux STR 功能。

3 Linux STR 流程

Suspend 代码入口：

S32G Linux STR

Linux shell 中执行以下命令：

echo mem > /sys/power/state

会调用到：

State_store(\kernel\power\main.c)

|->pm_autosleep_lock: 获得 autosleep 锁

|-> if (pm_autosleep_state() > PM_SUSPEND_ON) {

error = -EBUSY;

goto out;

} //判断当前 autosleep 状态，如果> PM_SUSPEND_ON，则返回

|-> decode_state //解析当前传入的 state，如果< PM_SUSPEND_MAX，则走 suspend 流程，调用：pm_suspend.如果= PM_SUSPEND_MAX，则走 hibernate。因为是传入 mem，所以会走 pm_suspend

|->pm_suspend(\kernel\power\suspend.c)

| |->enter_state \\进入 suspend 的下一步，如果成功 suspend_stats.success++，否则 suspend_stats.fail++

| | |->mutex_trylock \\获得 mutex 锁，防止在 suspend 时，再次 suspend

| | |->ksys_sync_helper\\ 同步文件系统

| | |->suspend_prepare \\suspend 前期准备工作，准备控制台，冻结内核线程。

| | | |->pm_prepare_console；\\切换控制台，重新分配一个 suspend 模式下的控制台，然后得定向到 kmsg

| | | |->pm_notifier_call_chain_robust(PM_SUSPEND_PREPARE, PM_POST_SUSPEND);\\通过调用 PM 通知链，发送 PM_SUSPEND_PREPARE 消息，只有通过 register_pm_notifier 的设备，子系统就会在这个时候处理 prepare 消息。

| | | |->suspend_freeze_processes

| | |->suspend_devices_and_enter \\设备及系统相关 suspend 操作

| | | |->platform_suspend_begin

| | | |->suspend_console \\挂起控制台，防止其它代码访问该控制台

| | | |->suspend_test_start \\记录当前 suspend 刚开始的时候的时间，使用 jiffies 表示。

| | | |->dpm_suspend_start \\ 调用所有设备的 prepare 和 suspend 的回调函数，如果 suspend 失败，打印 fail suspend，然后调用 platform_recover 函数执行平台相关的 recover 回调。

| | | | |-> dpm_prepare \\执行所有设备的 prepare 回调函数，执行的顺序是 pm_domain->type->class->bus->driver，如果失败设置 failed_prepare 的引用计数值

| | | | |->dpm_suspend \\执行所有设备的 suspend 回调函数。

| | | |->suspend_enter\\使系统进入到 suspend 中

| | | | |->platform_suspend_prepare \\调用平台相关的 prepare 回调 函数，如果失败，则调用平台相关 finish 回调函数

| | | | |->dpm_suspend_late\\，此函数 主要调用 dpm_suspend_list 中的设备的 suspend_later 回调函数，然后又将这些设备加入到 dpm_later_early_list 链表中，如果出现失败，则跳到 platform_finish 做恢复工作

| | | | |->dpm_suspend_noirq \\此函数从 dpm_later_early_list 链表中取一个设备，然后调用 该设备的 suspend_noirq 回调，同时将该设备加入到 dpm_noirq_list 链表中

| | | | |->platform_suspend_prepare_noirq \\平台相关

| | | | |->suspend_disable_secondary_cpus \\ disable所有非 nonboot 的 CPU

```
        for_each_online_cpu(cpu) {
        error = _cpu_down(cpu, 1, CPUHP_OFFLINE);
|  |  |  |  |->arch_suspend_disable_irqs \\架构相关
|  |  |  |  |->syscore_suspend \\执行所有 system core 的 suspend 回调函数
|  |  |  |  |->suspend_ops->enter(state)\\此函数会调用系统的 suspend 函数，在 ATF 里实现。
因为在 drivers\firmware\psci\psci.c 中，注册了 psci 的 enter 函数
Psci_probe->psci_init_system_suspend->suspend_set_ops(&psci_suspend_ops);
.enter          = psci_system_suspend_enter,
|  |  |  |  |  |->psci_system_suspend
return invoke_psci_fn(PSCI_FN_NATIVE(1_0, SYSTEM_SUSPEND),
                                    __pa_symbol(cpu_resume), 0, 0);
|  |  |->suspend_finish \\做最后的恢复工作
```

4 ATF Suspend 流程

4.1 Suspend 流程

psci_smc_handler(atf\lib\psci\psci_main.c)

case PSCI_SYSTEM_SUSPEND_AARCH64:

|->ret = (u_register_t)psci_system_suspend(x1, x2);

| |-> psci_cpu_suspend_start(&ep,

PLAT_MAX_PWR_LVL,

&state_info,

PSTATE_TYPE_POWERDOWN);//参数为 PSTATE_TYPE_POWERDOWN

| | |-> psci_plat_pm_ops->pwr_domain_pwr_down_wfi(state_info)= s32_pwr_domain_pwr_down_wfi

| | | |->ncore_caiu_offline(A53_CLUSTER0/1_CAIU)

| | | |->s32_plat_suspend

| | | | |->copy_bl31sram_image//此函数清理掉 BL32SRAM 段后，将 bl32sram 的代码段拷贝过来，最终 PC 指针会运行到这儿来完成对 DDR 的关闭动作。

| | | | |->platform_suspend

| | | | | |->gicv3_cpuif_disable，plat_gic_save\\关闭与保存中断

| | | | | |->set_warm_entry \\ 本函数用于设置 bl31_warm_entrypoint =s32g_resume_entrypoint

uintptr_t warm_entry, short_boot;

warm_entry = BL31SSRAM_MAILBOX + offsetof(struct s32g_ssram_mailbox,

bl31_warm_entrypoint);

short_boot = BL31SSRAM_MAILBOX + offsetof(struct s32g_ssram_mailbox,

short_boot);

mmio_write_64(warm_entry, (uintptr_t)s32g_resume_entrypoint);

mmio_write_8(short_boot, (uint8_t)s32gen1_is_wkp_short_boot());

| | | | | |->pmic_prepare_for_suspend \\配置 PMIC 支持 standby，一般会在 M 核进行。

| | | | | |->s32gen1_wkpu_enable_irqs \\使能唤醒中断，可在 M 核进行。

| | | | | |->s32_turn_off_mcores \\关闭所有 M 核，一般建议由负责安全的 M 核来进行。

| | | | | |->\\ 关闭除了A53_0之外的其它A53核

/* A53 cores */

for (i = 0; i < ncores; i++) {

if (i != current_cpu)

s32_turn_off_core(S32_MC_ME_CA53_PART, i);

}

| | | | | |->\\关闭相关时钟，除了 DDR 时钟外，都关闭

/* PFE blocks */

s32_disable_cofb_clk(S32G_MC_ME_PFE_PART, 0);

/* Keep the DDR clock */

s32_disable_cofb_clk(S32_MC_ME_USDHC_PART,

S32_MC_ME_PRTN_N_REQ(S32_MC_ME_DDR_0_REQ));

/* Switching all MC_CGM muxes to FIRC */

s32g_sw_clks2firc(); \\再关闭前先将所有时钟切换到内部 firc 时钟，为之后关闭 fxosc 做准备，A53_0 仍然有时钟供给。

/* Turn off DFS */

s32g_disable_dfs(S32_PERIPH_DFS);

s32g_disable_dfs(S32_CORE_DFS);

/* Turn off PLL */

s32g_disable_pll(S32_ACCEL_PLL, 2);

s32g_disable_pll(S32_PERIPH_PLL, 8);

s32g_disable_pll(S32_CORE_PLL, 2);

| | | | | |->bl31sram_entry\\由于之前 copy_bl31sram_image 已经准备好 bl31sram 的时钟， 所以实际上会调用到 bl31sram_main 去。

entry = (void *)BL31SRAM_BASE;

entry();

| | | | | | |->bl31sram_main \\此函数位于 SRAM 中，所以可以做相关的 DDR 关闭动作

| | | | | | | |->disable_mmu_el3

| | | | | | | |->ddrss_to_io_retention_mode \\将 DDR 设置为自刷新模式

| | | | | | | |->disable_ddr_clk \\关闭 ddr clock 时钟

| | | | | | | |->s32g_disable_fxosc \\关闭外部 fxosc，A53_0 使用内部 FIRC。

| | | | | | | |->s32g_set_stby_master_core\\此函数用于触发 A53_0 核所在 partition 状态切换为 suspend.

| | | | | | | |->wfi\\执行 WFI，主核 A53_0 进入 suspend.

4.2 Full boot resume 流程

STR full boot resume 的过程前期与正式启动类似。如果是用 M 核的话，一般也是 ROM CODE->Bootloader->boot M kernel, meantime boot-> A kernel ATF。

ATF 的正式启动过程请参考文档《S32G_ATF_BSP32_V*.pdf》。

但到当运行到 BL2 时：

bl2_el3_early_platform_setup(Atf\plat\nxp\s32\s32g\s32g_bl2_el3.c)

|-> reset_cause = get_reset_cause();\\此处获得上次 reset 的原因：

0 PREV_MODE	Previous mode This bit shows the status of the previous mode. 0b - The previous mode was reset (any reset). 1b - The previous mode was standby.

如果是 CAUSE_WAKEUP_DURING_STANDBY，表明上一次是 standby。

|->clear_reset_cause();\\ 清除掉复位标志

|->struct s32g_ssram_mailbox *ssram_mb = (void *)BL31SSRAM_MAILBOX;\\则计划执行 standby ram 中的启动入口

#define BL31SSRAM_MAILBOX (BL31SSRAM_IVT + BL31SSRAM_IVT_SIZE)

|->如果是 standby 的 full boot，则执行 resume_bl31(ssram_mb);，如果不是，执行正常启动，程序在此处分支。

```
    if ((reset_cause == CAUSE_WAKEUP_DURING_STANDBY) &&
        !ssram_mb->short_boot) {
            /* Trampoline to bl31_warm_entrypoint */
            resume_bl31(ssram_mb);
            panic();
    }
```

|->resume_bl31

| |->准备好 stand ram 中代码的入口地址，及 ddr 的 traing 保存地址。

```
   resume_entrypoint = ssram_mb->bl31_warm_entrypoint;
    csr_addr = (uintptr_t)&ssram_mb->csr_settings[0];
```

| |->s32_enable_a53_clock \\打开A53 clock

| |->s32_enable_ddr_clock\\ 打开 DDR clock

| |->ddrss_to_normal_mode(csr_addr); Transition the DDR SubSystem from retention mode to normal mode DDR 从自刷新状态退出到正式状态

/* 注意如vi plat/nxp/s32/s32g/bl31_ssram/bl31_ssram.mk

BL31SSRAM_SOURCES = …

${DDR_DRV_SRCS} \...

所以 DDR 相关代码和全局数组是链接在 standby sram 之中的。

而在 ATF 正常 Boot 时：

Ddr_init(driver\nxp\s32\ddr\ddr_init.c)-> post_train_setup((uint8_t)(STORE_CSR_MASK |

INIT_MEM_MASK |

ADJUST_DDRC_MASK));

会将 training 结果保存在 SSRAM 中的 DDR 全局变量数组中，所以以上代码就是直接从 SSRAM 中将 training 结果读出来恢复到 DDRC 寄存器中，从而免去了重新 training 的过程。*/

| | |->load_register_cfg(ddrc_cfg_size, ddrc_cfg);

| | |->load_ddrc_regs(csr_array);

| | |-> post_train_setup(STORE_CSR_DISABLED | INIT_MEM_DISABLED |

ADJUST_DDRC_DISABLED);

| |->resume_entrypoint \\这个 resume_entrypoint()其实就是执行：s32g_resume_entrypoint()函数（参见 static void set_warm_entry(void)函数的设置）， 而 s32g_resume_entrypoint()函数 属于 BL31，位于 DDR

| |->s32g_resume_entrypoint

| | |->Prepare resume operation: reset_registers_for_lockstep, s32_early_plat_init

| | |->pmic_setup \\pmic相关设置，可以去掉

| | |->reset_rtc

| | |->s32gen1_wkpu_reset

| | |->console_s32_register

| | |->plat_gic_restore

| | | |->bl31_warm_entrypoint

| | | | |->psci_warmboot_entrypoint

/*

* This CPU could be resuming from suspend or it could have just been

* turned on. To distinguish between these 2 cases, we examine the

* affinity state of the CPU:

* - If the affinity state is ON_PENDING then it has just been

* turned on.

* - Else it is resuming from suspend.

*

* Depending on the type of warm reset identified, choose the right set

```
        * of power management handler and perform the generic, architecture
        * and platform specific handling.
        */
       if (psci_get_aff_info_state() == AFF_STATE_ON_PENDING)
                psci_cpu_on_finish(cpu_idx, &state_info);
       else
                psci_cpu_suspend_finish(cpu_idx, &state_info);
```

5 定制修改

本文定制考虑将 M7 standby demo 与 A53 Linux STR 功能结合，主要应用场景是快速启动的情况下， M 核代码可以 standby full boot，而 A53 Linux 则从 STR 直接唤醒执行。

M 核 standby demo 的情况请参考文档：《S32G_Standby_Demo_V2-20221028.pdf》。

主要的修改包括：

- Linux STR 功能是将 A53_0 作为主核的，所以他事先会关闭掉所有 M7 核及其它 A53 核。而 M7 standby demo 是将 M7_0 当成主核。所以 linux ATF 中不能关闭 M7 核，并且最后一个 A53 核不能自行关闭，留给 M 核去关闭。
- Linux 端关于 PMIC/WKPU 的所有操作全部去掉，包括 DTS 驱动和 hard coding。
- Linux 端关于唤醒源配置也不需要。
- Linux 端的时钟关闭代码也不需要，特别是指 M7_0 核运行需要的核心时钟和 UART/I2C 外设时钟。

5.1 ATF 中实现主核切换为 M7

Atf\plat\nxp\s32\s32g\s32g_plat_funcs.c

```
static void __dead2 platform_suspend(unsigned int current_cpu)
{
#if 0
    pmic_prepare_for_suspend(); \\PMIC的配置与驱动全部移值到 M 核进行。
    s32gen1_wkpu_enable_irqs(); \\ 唤醒源的配置在 M 核进行

    /* Shutting down cores */
    /* M7 cores */
```

```
    s32_turn_off_mcores(); \\保留住 M7_0，至于其它 M7 核，也建议使用 M7_0 核来关闭，其它非 A53_0 的核，ATF suspend 代码会自行关闭。
#endif

#if 0
\\ 以下的时钟关闭代码，全部由 M7_0 核来关闭，所以 A 核这边不关闭时钟
    /* Switching all MC_CGM muxes to FIRC */
    s32g_sw_clks2firc(); /

    /* Turn off DFS */
    s32g_disable_dfs(S32_PERIPH_DFS);
    s32g_disable_dfs(S32_CORE_DFS);

    /* Turn off PLL */
    s32g_disable_pll(S32_ACCEL_PLL, 2);
    s32g_disable_pll(S32_PERIPH_PLL, 8);
    s32g_disable_pll(S32_CORE_PLL, 2);
#endif
}
```

ATF\plat\nxp\s32\s32g\bl31_sram\bl31sram_main.c

```
void bl31sram_main(void)
{
    disable_mmu_el3();
    ddrss_to_io_retention_mode();
    disable_ddr_clk();

#if 0 //A53_0 不关闭内部时钟源
    s32g_disable_fxosc();
#endif
    /* Set standby master core and request the standby transition */ \\ 此处代码是 Linux STR 装 A53_0 作为主核，所以最后将其关闭的地方，所以去掉，只保留执行 WFI，由 M7_0 来关闭此 A53_0
    //s32g_set_stby_master_core(S32G_STBY_MASTER_PART, plat_my_core_pos());

    /*
```

```
    * A torn-apart variant of psci_power_down_wfi()
    */
    dsb();
    wfi();

    plat_panic_handler();
}
```

5.2 ATF 中去掉 PMIC 与 I2C4

在 ATF 中 PMIC hard coding 的代码还有如下：

Atf\plat/nxp/s32/s32g/s32g_bl31.c

```
bl31_platform_setup-> dt_init_pmic();
```

Atf\plat\nxp\s32\s32g\s32g_bl12_el3.c

```
bl2_el3_plat_arch_setup->ret = pmic_setup()
```

Atf\plat\nxp\s32\s32g\s32g_resume.c

```
s32g_resume_entrypoint->ret = pmic_setup()
```

Atf\plat\nxp\s32\s32_psci.c

```
s32_system_off-> pmic_system_off
```

Atf\plat\nxp\s32\s32g\s32g_pinctrl.c\

```
void i2c_config_pinctrl(void)
{…
    /* PMIC - I2C4 */
    /* I2C4 Serial Data Input */
    mmio_write_32(SIUL2_0_MSCRn(SIUL2_MSCR_S32G_PC_01),
                  SIUL2_MSCR_S32G_PAD_CTRL_I2C4_SDA);
    mmio_write_32(SIUL2_1_IMCRn(SIUL2_PC_01_IMCR_S32G_I2C4_SDA),
                  SIUL2_IMCR_S32G_PAD_CTRL_I2C4_SDA);
/* I2C4 Serial Clock Input */
    mmio_write_32(SIUL2_0_MSCRn(SIUL2_MSCR_S32G_PC_02),
                  SIUL2_MSCR_S32G_PAD_CTRL_I2C4_SCLK);
    mmio_write_32(SIUL2_1_IMCRn(SIUL2_PC_02_IMCR_S32G_I2C4_SCLK),
                  SIUL2_IMCR_S32G_PAD_CTRL_I2C4_SCLK);
```

总共 patch 如下：

```
diff --git a/fdts/fsl-s32g-rdb.dtsi b/fdts/fsl-s32g-rdb.dtsi
index 7f03da6b6..630f3c5aa 100644
--- a/fdts/fsl-s32g-rdb.dtsi
+++ b/fdts/fsl-s32g-rdb.dtsi
@@ -6,19 +6,19 @@

 /* PMIC */
 &i2c4 {
-	status = "okay";
+	status = "disabled";
 	clock-frequency=<100000>;

 	vr5510@20 {
 		compatible = "fsl,vr5510";
 		reg = <0x20>;
-		status = "okay";
+		status = "disabled";
 	};

 	vr5510_fsu@21 {
 		compatible = "fsl,vr5510";
 		reg = <0x21>;
-		status = "okay";
+		status = "disabled";
 	};

 };
diff --git a/plat/nxp/s32/s32_common.mk b/plat/nxp/s32/s32_common.mk
index 4bc42e7f3..c23935a45 100644
--- a/plat/nxp/s32/s32_common.mk
+++ b/plat/nxp/s32/s32_common.mk
@@ -399,7 +399,7 @@ else
 	echo "DATA_FILE SIZE $$T_SIZE" >> $@
 endif
```

```
-FIP_ALIGN := 16
+FIP_ALIGN := 64
 all: add_to_fip
 add_to_fip: fip ${BL2_W_DTB}
     $(eval FIP_MAXIMUM_SIZE_10 = $(shell printf "%d\n" ${FIP_MAXIMUM_SIZE}))
diff --git a/plat/nxp/s32/s32_psci.c b/plat/nxp/s32/s32_psci.c
index 6c66f83f1..918dda96f 100644
--- a/plat/nxp/s32/s32_psci.c
+++ b/plat/nxp/s32/s32_psci.c
@@ -176,7 +176,9 @@ static void __dead2 s32_system_reset(void)
 static void __dead2 s32_system_off(void)
 {
 #if defined(PLAT_s32g2) || defined(PLAT_s32g3)
+#if 0
     pmic_system_off();
+#endif
 #endif
     plat_panic_handler();
 }
diff --git a/plat/nxp/s32/s32g/bl31_sram/bl31sram_main.c b/plat/nxp/s32/s32g/bl31_sram/bl31sram_main.c
index c3c1d73c1..68768ccb6 100644
--- a/plat/nxp/s32/s32g/bl31_sram/bl31sram_main.c
+++ b/plat/nxp/s32/s32g/bl31_sram/bl31sram_main.c
@@ -23,12 +23,12 @@ void bl31sram_main(void)
     disable_mmu_el3();
     ddrss_to_io_retention_mode();
     disable_ddr_clk();
-
+#if 0
     s32g_disable_fxosc();

     /* Set standby master core and request the standby transition */
     s32g_set_stby_master_core(S32G_STBY_MASTER_PART, plat_my_core_pos());
```

```
-
+#endif
	/*
	 * A torn-apart variant of psci_power_down_wfi()
	 */
diff --git a/plat/nxp/s32/s32g/s32g_bl2_el3.c b/plat/nxp/s32/s32g/s32g_bl2_el3.c
index dc74b2ce9..58fdb5371 100644
--- a/plat/nxp/s32/s32g/s32g_bl2_el3.c
+++ b/plat/nxp/s32/s32g/s32g_bl2_el3.c
@@ -147,11 +147,11 @@ void bl2_el3_plat_arch_setup(void)

	dt_init_ocotp();
	dt_init_pmic();
-
+#if 0
	ret = pmic_setup();
	if (ret)
		ERROR("Failed to disable VR5510 watchdog\n");
-
+#endif
	s32_sram_clear(S32_BL33_IMAGE_BASE, get_bl2_dtb_base());
	/* Clear only the necessary part for the FIP header. The rest will
	 * be cleared in bl2_plat_handle_post_image_load, before loading
diff --git a/plat/nxp/s32/s32g/s32g_pinctrl.c b/plat/nxp/s32/s32g/s32g_pinctrl.c
index e6a823448..4fb1473bd 100644
--- a/plat/nxp/s32/s32g/s32g_pinctrl.c
+++ b/plat/nxp/s32/s32g/s32g_pinctrl.c
@@ -92,7 +92,7 @@ void i2c_config_pinctrl(void)
				SIUL2_MSCR_S32G_PAD_CTRL_I2C0_SCLK);
	mmio_write_32(SIUL2_0_IMCRn(SIUL2_PB_01_IMCR_S32G_I2C0_SCLK),
				SIUL2_IMCR_S32G_PAD_CTRL_I2C0_SCLK);
-
+#if 0
	/* PMIC - I2C4 */
```

```
     /* I2C4 Serial Data Input */
     mmio_write_32(SIUL2_0_MSCRn(SIUL2_MSCR_S32G_PC_01),
@@ -105,4 +105,5 @@ void i2c_config_pinctrl(void)
                           SIUL2_MSCR_S32G_PAD_CTRL_I2C4_SCLK);
     mmio_write_32(SIUL2_1_IMCRn(SIUL2_PC_02_IMCR_S32G_I2C4_SCLK),
                           SIUL2_IMCR_S32G_PAD_CTRL_I2C4_SCLK);
+#endif
 }
diff --git a/plat/nxp/s32/s32g/s32g_plat_funcs.c b/plat/nxp/s32/s32g/s32g_plat_funcs.c
index 4df1fe092..19876f554 100644
--- a/plat/nxp/s32/s32g/s32g_plat_funcs.c
+++ b/plat/nxp/s32/s32g/s32g_plat_funcs.c
@@ -50,13 +50,14 @@ static void __dead2 platform_suspend(unsigned int current_cpu)

     plat_gic_save();
     set_warm_entry();
+#if 0
     pmic_prepare_for_suspend();
     s32gen1_wkpu_enable_irqs();

     /* Shutting down cores */
     /* M7 cores */
     s32_turn_off_mcores();
-
+#endif
     if (is_lockstep_enabled())
                 ncores /= 2;

@@ -65,7 +66,7 @@ static void __dead2 platform_suspend(unsigned int current_cpu)
                 if (i != current_cpu)
                           s32_turn_off_core(S32_MC_ME_CA53_PART, i);
     }
-
+#if 0
```

```
     /* PFE blocks */
     s32_disable_cofb_clk(S32G_MC_ME_PFE_PART, 0);
     /* Keep the DDR clock */
@@ -83,7 +84,7 @@ static void __dead2 platform_suspend(unsigned int current_cpu)
     s32g_disable_pll(S32_ACCEL_PLL, 2);
     s32g_disable_pll(S32_PERIPH_PLL, 8);
     s32g_disable_pll(S32_CORE_PLL, 2);
-
+#endif
     bl31sram_entry();
     plat_panic_handler();
 }
diff --git a/plat/nxp/s32/s32g/s32g_resume.c b/plat/nxp/s32/s32g/s32g_resume.c
index 6110d36c1..800402de9 100644
--- a/plat/nxp/s32/s32g/s32g_resume.c
+++ b/plat/nxp/s32/s32g/s32g_resume.c
@@ -32,17 +32,18 @@ static void reset_rtc(void)

 void s32g_resume_entrypoint(void)
 {
+#if 0
     int ret;
-
+#endif
     /* Prepare resume operation */
     reset_registers_for_lockstep();
     s32_ncore_isol_cluster0();
     s32_early_plat_init(true);
-
+#if 0
     ret = pmic_setup();
     if (ret)
             ERROR("Failed to disable VR5510 watchdog\n");
-
```

```
+#endif
     reset_rtc();
     s32gen1_wkpu_reset();

diff --git a/plat/nxp/s32/s32g/s32g_bl31.c b/plat/nxp/s32/s32g/s32g_bl31.c
index 896e4eaae..b4cd5f581 100644
--- a/plat/nxp/s32/s32g/s32g_bl31.c
+++ b/plat/nxp/s32/s32g/s32g_bl31.c

-       dt_init_pmic();
-       dt_init_wkpu();
+/*     dt_init_pmic();
+       dt_init_wkpu(); */
```

5.3 ATF 中去掉 wkpu 驱动

由于唤醒源配置修改为使用 M 核配置，所以 Linux ATF 中的原有驱动注释掉

Patch 如下：

```
diff --git a/fdts/fsl-s32g-rdb.dtsi b/fdts/fsl-s32g-rdb.dtsi
index 7f03da6b6..01fac39b0 100644
--- a/fdts/fsl-s32g-rdb.dtsi
+++ b/fdts/fsl-s32g-rdb.dtsi
…
&wkpu {
-       status = "okay";
+       status = "disabled";
 };
diff --git a/plat/nxp/s32/s32g/s32g_bl31.c b/plat/nxp/s32/s32g/s32g_bl31.c
index 896e4eaae..b4cd5f581 100644
--- a/plat/nxp/s32/s32g/s32g_bl31.c
+++ b/plat/nxp/s32/s32g/s32g_bl31.c
+/*
 static void dt_init_wkpu(void)
 {
…
```

```
 }
+*/
-       dt_init_pmic();
-       dt_init_wkpu();
+/*     dt_init_pmic();
+       dt_init_wkpu(); */
```

由于去掉了 WKPU 驱动的初始化，导致 gwkpu 全局变量没有初始化，所以一些依赖此变量的判断会失败，如下去掉 short boot，不再使用：

```
diff --git a/plat/nxp/s32/s32g/s32g_bl2_el3.c b/plat/nxp/s32/s32g/s32g_bl2_el3.c
index dc74b2ce9..b6f818161 100644
--- a/plat/nxp/s32/s32g/s32g_bl2_el3.c
+++ b/plat/nxp/s32/s32g/s32g_bl2_el3.c
@@ -99,9 +99,12 @@ void bl2_el3_early_platform_setup(u_register_t arg0, u_register_t arg1,
        /* No resume on emulator */
 #if S32G_EMU == 0
        struct s32g_ssram_mailbox *ssram_mb = (void *)BL31SSRAM_MAILBOX;
-
+       /*
        if ((reset_cause == CAUSE_WAKEUP_DURING_STANDBY) &&
-           !ssram_mb->short_boot) {
+            !ssram_mb->short_boot)
+       */
+if (reset_cause == CAUSE_WAKEUP_DURING_STANDBY)
+       {
```

5.4 Uboot 中去掉 PMIC 与 I2C4

Patch 如下：

```
diff --git a/arch/arm/dts/fsl-s32gxxxardb.dtsi b/arch/arm/dts/fsl-s32gxxxardb.dtsi
index f661f168b5..d1d1ed288b 100644
--- a/arch/arm/dts/fsl-s32gxxxardb.dtsi
+++ b/arch/arm/dts/fsl-s32gxxxardb.dtsi
@@ -71,18 +71,18 @@
     pinctrl-1 = <&pinctrl0_i2c4_gpio &pinctrl1_i2c4_gpio>;
     scl-gpios = <&gpio0 34 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
```

```
    sda-gpios = <&gpio0 33 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
-   status = "okay";
+   status = "disabled";

    vr5510 {
            compatible = "fsl,vr5510";
            reg = <0x20>;
-           status = "okay";
+           status = "disabled";
    };

    vr5510_fsu {
            compatible = "fsl,vr5510";
            reg = <0x21>;
-           status = "okay";
+           status = "disabled";
    };

    pf5020_a {
```

5.5 Kernel 中去掉 I2C4

Patch 如下：

```
diff --git a/arch/arm64/boot/dts/freescale/fsl-s32gxxxa-rdb.dtsi
b/arch/arm64/boot/dts/freescale/fsl-s32gxxxa-rdb.dtsi
index a6438dffc2d9..82d31312e4f0 100644
--- a/arch/arm64/boot/dts/freescale/fsl-s32gxxxa-rdb.dtsi
+++ b/arch/arm64/boot/dts/freescale/fsl-s32gxxxa-rdb.dtsi
@@ -119,7 +119,7 @@
    pinctrl-1 = <&pinctrl0_i2c4_gpio &pinctrl1_i2c4_gpio>;
    scl-gpios = <&gpio0 34 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
    sda-gpios = <&gpio0 33 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
-   status = "okay";
+   status = "disabled";
```

```
};

&gpio0 {
```

6 发布

Linux 端发布请参考工程与文档《S32G_M7_STBYFULLBOOT_A53STR_V*.pdf》。