S32G Linux Suspend to Ram and Customization

by John Li (nxa08200)

This article explains the details and customization of S32G A53 core STR. The customization part explains how to combine with M7 standby demo to realize the quick start of the whole product.

Please note that this article is a training and auxiliary document. This article is not a substitute for the official document. Please refer to the official document.

History	Comments	Author
V1	● Create the doc	John.Li
V2	● Translate to Eng	John.Li

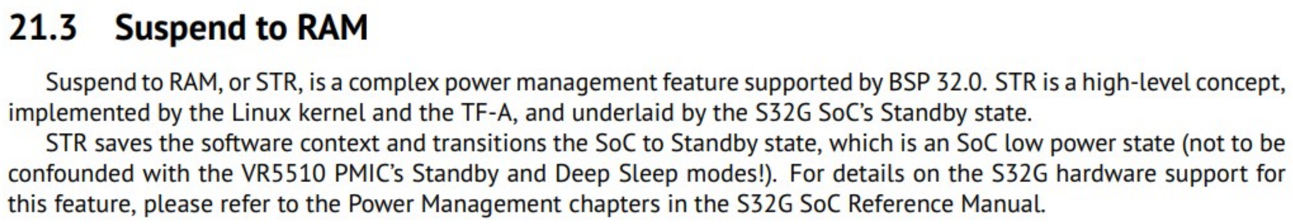
Contents

1 Reference materials

Classification	Name	Commnents	Remark
doc	AN12880-5510-standby.pdf	VR5510 standby doc	Search on www.nxp.com
doc	AN12952-Power Saving Techniques.pdf	S32G2 suspend doc	Search on www.nxp.com. important
doc	S32G2_LinuxBSP_32.0_Quick_Start.pdf	S32G2 Linux development kit, note that its user manual is a detailed description of BSP, which is important	Download from www.nxp.com
doc	S32G2_LinuxBSP_32.0_Release_Notes.pdf		
doc	S32G2_LinuxBSP_32.0_User_Manual.pdf		
doc	S32G2_LinuxBSP32.0_quality_package.zip		
doc	S32G2_PFE_LinuxBSP32.0.0_license.txt		
doc	S32G2RM.pdf	Chipset reference manual	Download from www.nxp.com
doc	ds649820 - VR5510 Datasheet Rev2 (2.0).pdf	Chipset reference manual	Download from www.nxp.com

2 STR Demo

Refer to the document <<S32G_ATF_BSP32_V*.pdf>>, or <<S32G_Uboot_BSP32_V*.pdf>>, or <<S32G_Kernel_BSP32_V*.pdf>>(Johnli), or the official document <<S32G2_LinuxBSP_32.0_User_Manual.pdf>>, to create a Linux environment and image, and run the image reference document <<S32G2_LinuxBSP_32. 0_User_Manual.pdf>> as follows:



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.

to run the Linux STR function.

3 Linux STR call flow

Suspend code entry:

Linux shell execute the following command:

echo mem > /sys/power/state

which will call:

State_store(\kernel\power\main.c)

```
|->pm_autosleep_lock: //Get autosleep lock
|->  if (pm_autosleep_state() > PM_SUSPEND_ON) {
            error = -EBUSY;
            goto out;
   } // Determine the current autosleep state, if > PM_SUSPEND_ON, return
```

|-> decode_state // Parse the current incoming state, if < PM_SUSPEND_MAX, go to the suspend process, call: pm_suspend. If = PM_SUSPEND_MAX, go to hibernate. Because it is passed to mem, it will go pm_suspend

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

| |->enter_state \\ Enter the next step of suspend, if successful suspend_stats.success++, otherwise suspend_stats.fail++

| | |->mutex_trylock \\ Obtain a mutex lock to prevent suspend again during suspend

| | |->ksys_sync_helper\\ sync the file system

| | |->suspend_prepare \\ Suspend preparatory work, prepare the console, freeze the kernel thread.

| | | |->pm_prepare_console； \\ Switch consoles, reassign a console in suspend mode, and then redirect to kmsg

| | | |->pm_notifier_call_chain_robust(PM_SUSPEND_PREPARE, PM_POST_SUSPEND);\\ By calling the PM notification chain and sending the PM_SUSPEND_PREPARE message, the subsystem will process the prepare message at this time only for devices that pass the register_pm_notifier.

| | | |->suspend_freeze_processes

| | |->suspend_devices_and_enter \\Equipment and system-related suspend operations

| | | |->platform_suspend_begin

| | | |->suspend_console \\ Suspends the console, preventing other code from accessing the console

| | | |->suspend_test_start \\ Record the time when the current suspend just started, expressed in jiffies.

| | | |->dpm_suspend_start \\ Call the prepare and suspend callback functions of all devices. If the suspend fails, print fail suspend, and then call the platform_recover function to execute the platform-related recover callback.

| | | | |-> dpm_prepare \\ Execute the prepare callback function of all devices, the order of execution is pm_domain->type->class->bus->driver, if it fails, set the reference count value of failed_prepare

| | | | |->dpm_suspend \\ Execute the suspend callback function of all devices.

| | | |->suspend_enter\\ Put the system into suspend

| | | | |->platform_suspend_prepare \\ Call the platform-specific prepare callback function, if it fails, call the platform-specific finish callback function

| | | | |->dpm_suspend_late\\ This function mainly calls the suspend_later callback function of the devices in dpm_suspend_list, and then adds these devices to the dpm_later_early_list linked list. If there is a failure, it will jump to platform_finish to do the recovery work

| | | | |->dpm_suspend_noirq \\ This function takes a device from the dpm_later_early_list linked list, then calls the suspend_noirq callback of the device, and adds the device to the dpm_noirq_list linked list

```
|   |   |   |   |->platform_suspend_prepare_noirq \\  platform related
|   |   |   |   |->suspend_disable_secondary_cpus \\ disable all non-nonboot CPUs
      for_each_online_cpu(cpu) {
         error = _cpu_down(cpu, 1, CPUHP_OFFLINE);
|   |   |   |   |->arch_suspend_disable_irqs \\ Architecture related
|   |   |   |   |->syscore_suspend \\ Execute the suspend callback function of all system cores
|   |   |   |   |->suspend_ops->enter(state)\\ This function will call the suspend function of the system, which is implemented in ATF. Because in drivers\firmware\psci\psci.c, the enter function of psci is registered
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 \\ do the final recovery work
```

4 ATF Suspend call flow

4.1 Suspend flow

```
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// After this function cleans up the BL32SRAM segment, copy the code segment of bl32sram, and finally the PC pointer will run here to complete the closing action of DDR.
|   |   |   |   |->platform_suspend
|   |   |   |   |   |->gicv3_cpuif_disable，plat_gic_save\\ Close and Save Interrupts
|   |   |   |   |   |->set_warm_entry \\ This function is used to set 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 \\ Configure the PMIC to support standby, which is generally performed on the M core.
|   |   |   |   |   |->s32gen1_wkpu_enable_irqs \\ Enable wake-up interrupt, which can be done in M core.
|   |   |   |   |   |->s32_turn_off_mcores \\ To shut down all M cores, it is generally recommended to be done by the M core responsible for safety.
|   |   |   |   |   |->\\Close other A53 cores except A53_0
/* A53 cores */
    for (i = 0; i < ncores; i++) {
            if (i != current_cpu)
                    s32_turn_off_core(S32_MC_ME_CA53_PART, i);
    }
|   |   |   |   |   |->\\Close related clocks, except for DDR clocks, all are closed
/* 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(); \\ Before shutting down, switch all clocks to the internal firc clock to prepare for shutting down fxosc later, and A53_0 still has clock supply./* 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\\ Since copy_bl31sram_image has prepared the code segment of bl31sram before, it will actually be called to bl31sram_main.
    entry = (void *)BL31SRAM_BASE;
    entry();
|   |   |   |   |   |   |->bl31sram_main \\This function is located in SRAM, so it can do related DDR close action
|   |   |   |   |   |   |   |->disable_mmu_el3
|   |   |   |   |   |   |   |->ddrss_to_io_retention_mode \\Set DDR to self-refresh mode
|   |   |   |   |   |   |   |->disable_ddr_clk \\ Turn off the ddr clock clock
|   |   |   |   |   |   |   |->s32g_disable_fxosc \\ Disable external fxosc, A53_0 uses internal FIRC.
|   |   |   |   |   |   |   |->s32g_set_stby_master_core\\ This function is used to trigger the partition status of the A53_0 core to switch to suspend.
|   |   |   |   |   |   |   |->wfi\\ Execute WFI, the main core A53_0 enters suspend.
```

4.2 Full boot resume flow

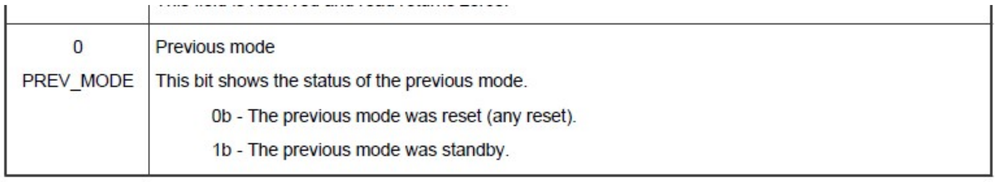
The early stage of the STR full boot resume process is similar to the normal bootup. If the M core is used, it is generally boot from ROM CODE->Bootloader->boot M kernel, meanwhile boot->A kernel ATF.

For the normal bootup process of ATF, please refer to the document <<S32G_ATF_BSP32_V*.pdf>>(Johnli).

But when running to BL2:

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

|-> reset_cause = get_reset_cause();\\The reason for the last reset is obtained here:



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.

If it is CAUSE_WAKEUP_DURING_STANDBY, it indicates that the last time was standby.

|->clear_reset_cause();\\Clear the reset flag

|->struct s32g_ssram_mailbox *ssram_mb = (void *)BL31SSRAM_MAILBOX;\\ Then plan to execute the startup entry in standby ram

#define BL31SSRAM_MAILBOX   (BL31SSRAM_IVT + BL31SSRAM_IVT_SIZE)

|-> If it is the full boot of standby, execute resume_bl31(ssram_mb); if not, execute normal startup, and the program branches here.

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

|->resume_bl31

| |-> Prepare the entry address of the code in the stand ram and the storage address of the ddr train.

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

| |->s32_enable_a53_clock \\ open A53 clock

| |->s32_enable_ddr_clock\\ open DDR clock

| |->ddrss_to_normal_mode(csr_addr); Transition the DDR SubSystem from retention mode to normal mode DDR Exit from self-refresh state to normal state

/* notice as: vi plat/nxp/s32/s32g/bl31_ssram/bl31_ssram.mk

```
BL31SSRAM_SOURCES =  …
                ${DDR_DRV_SRCS} \...
```

So DDR related codes and global arrays are linked in standby sram.

And when ATF boots normally:

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

The training result will be saved in the DDR global variable array in SSRAM, so the above code reads the training result directly from SSRAM and restores it to the DDRC register, thus eliminating the need for re-training.

```
|  |  |->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  \\
```

This resume_entrypoint() actually executes: s32g_resume_entrypoint() function (see static void set_warm_entry(void) function setting), and s32g_resume_entrypoint() function belongs to BL31, located in DDR

```
|  |->s32g_resume_entrypoint
|  |  |->Prepare resume operation: reset_registers_for_lockstep, s32_early_plat_init
|  |  |->pmic_setup \\pmic setup, can be remove if pmic driver in M core.
|  |  |->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 Customization

This article customizes and considers combining the M7 standby demo with the A53 Linux STR function. The main application scenario is that in the case of fast boot, the M core code can be standby full boot, and the A53 Linux can be directly awakened from the STR for execution.

For M core standby demo, please refer to the document: <<S32G_Standby_Demo_V*.pdf>>(JohnLi).

Major modifications include:

- The Linux STR function uses A53_0 as the main core, so it will shut down all M7 cores and other A53 cores in advance. The M7 standby demo uses the M7_0 as the main core. Therefore, the M7 core cannot be turned off in linux ATF, and the last A53 core cannot be turned off by itself, leaving it to the M core to turn off.
- All operations related to PMIC/WKPU on the Linux side are removed, including DTS driver and hard coding.
- There is no need to configure the wakeup source on the Linux side.
- The clock shutdown code on the Linux side is also unnecessary, especially the core clock and UART/I2C peripheral clock required for the operation of the M7_0 core.

5.1 The STR main core is switched to M7 in ATF

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 configuration and driver are all transferred to the M core.
    s32gen1_wkpu_enable_irqs(); \\ The configuration of the wake-up source is performed in the M core

    /* Shutting down cores */
    /* M7 cores */
    s32_turn_off_mcores(); \\ Keep M7_0. As for other M7 cores, it is also recommended to use M7_0 core to shut down. For other non-A53_0 cores, the ATF suspend code will shut down by itself.
#endif

#if 0
\\ The following clock shutdown codes are all turned off by the M7_0 core, so the A core does not turn off the clock
    /* 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 does not turn off the internal clock source
    s32g_disable_fxosc();
#endif
    /* Set standby master core and request the standby transition */ \\ The code here is the place where Linux STR uses A53_0 as the main core, so it is turned off at the end, so remove it, only keep executing WFI, and let M7_0 turn off this 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 remove PMIC and I2C4

In ATF, PMIC hard coding codes include the follows：

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);
```

Whole patch as follows:

```
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 remove wkpu driver

Since the wake-up source configuration is modified to use the M-core configuration, the original driver in Linux ATF is commented out:

Patch as follows:

```
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(); */
```

Since the initialization of the WKPU driver is removed, the gwkpu global variable is not initialized, so some judgments that rely on this variable will fail. Remove the short boot as follows and no longer use it:

```
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 remove PMIC and I2C4

Patch as follows:

```
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 remove I2C4

Patch as follows:

```
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 Release

Please refer to the project and document <<S32G_M7_STBYFULLBOOT_A53STR_V*.pdf>>(John.li) for the Linux release.