

Extending the swsusp Hibernation Framework to ARM

Russell Dill

Introduction

- Russ Dill of Texas Instruments
- swsusp/hibernation on ARM
 - Overview
 - Challenges
 - Implementation
 - Remaining work
 - Debugging
- swsusp restore from U-Boot
- Code at:
 - <https://github.com/russdill/linux/commits/arm-hibernation-am33xx>
 - <https://github.com/russdill/commits/hibernation>
- eLinux.org page: http://elinux.org/ARM_Hibernation

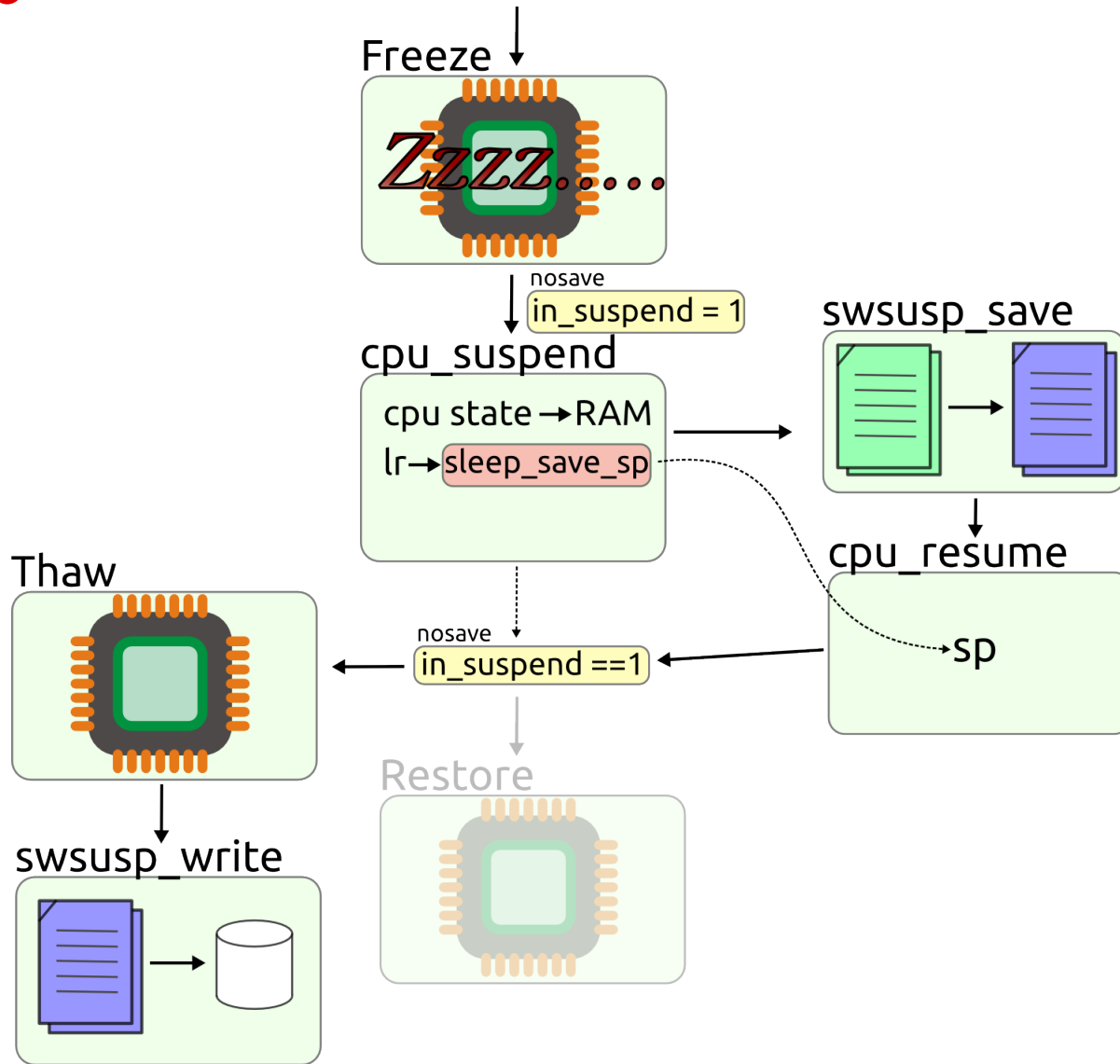
Motivation

- Hibernation provides zero power consumption sleep
- Allows for snapshot boot
- Shares requirements with self-refresh only sleep modes
 - RTC-Only+DDR self-refresh

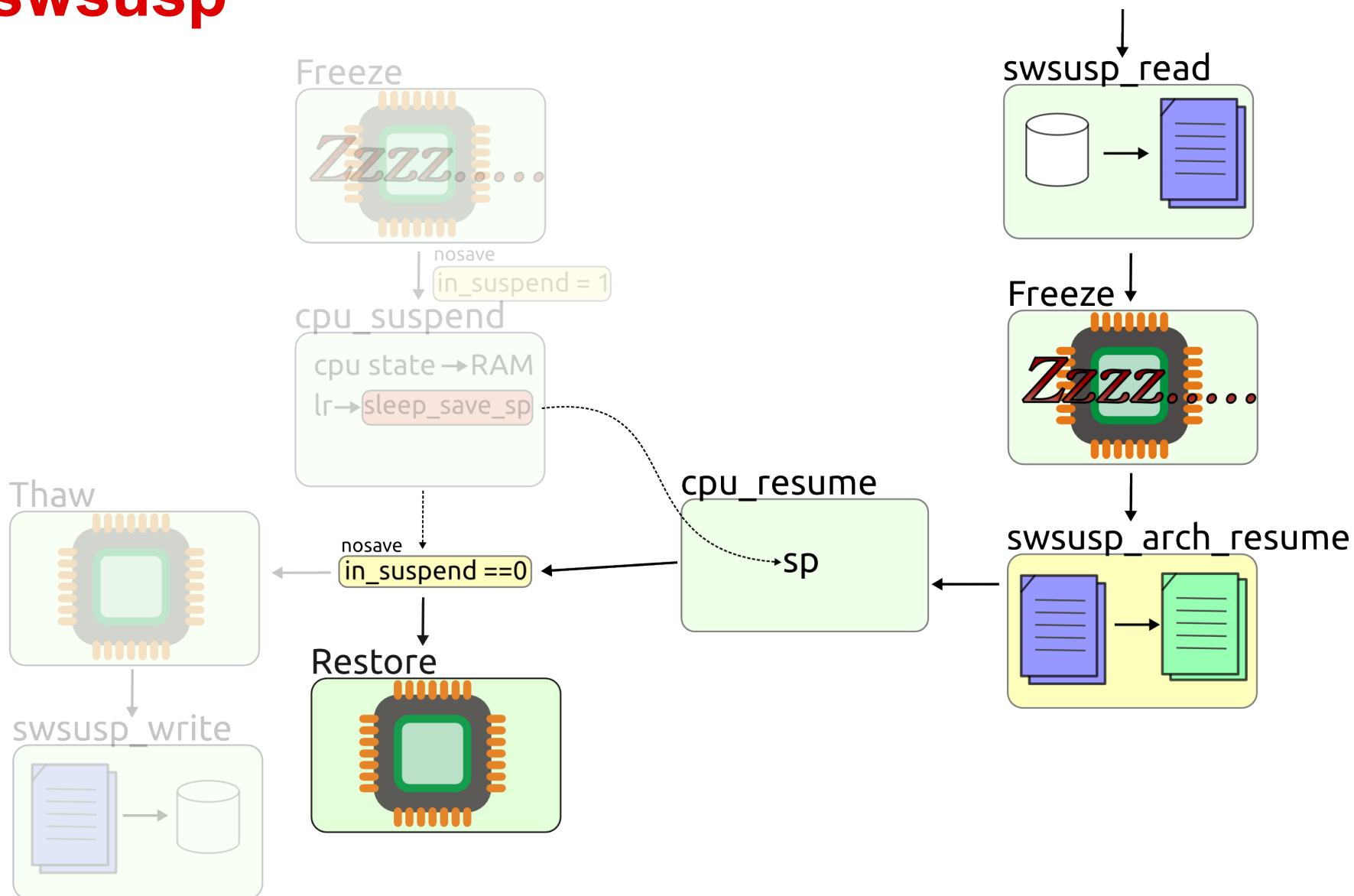
swsusp

- Mainline hibernation implementation since 2.6.0
 - TuxOnIce (Suspend2)
- Uses swap device to store image
- Requires 1/2 of system RAM to be free
- Can be used with uswsusp to support additional features
 - Encryption
 - Limitless storage options
 - Graphical progress

swsusp



swsusp



OMAP PM

- Clocks
 - Clock gating
 - Clock domains
 - Clock scaling
- Power
 - Power domains
 - Logic
 - Retention
 - Voltage scaling
- PRCM Controls these features

Figure 8-1. Functional and Interface Clocks

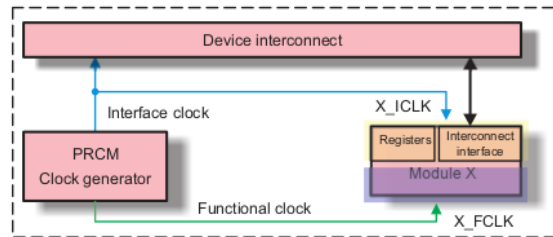


Figure 8-2. Generic Clock Domain

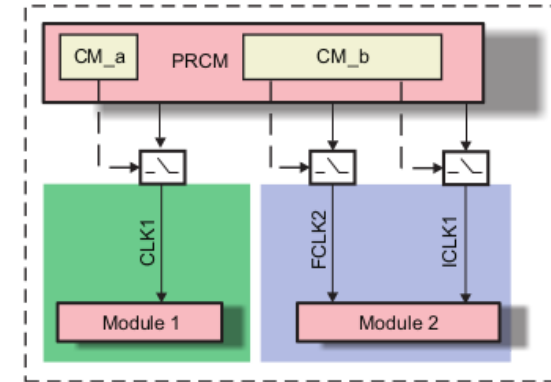
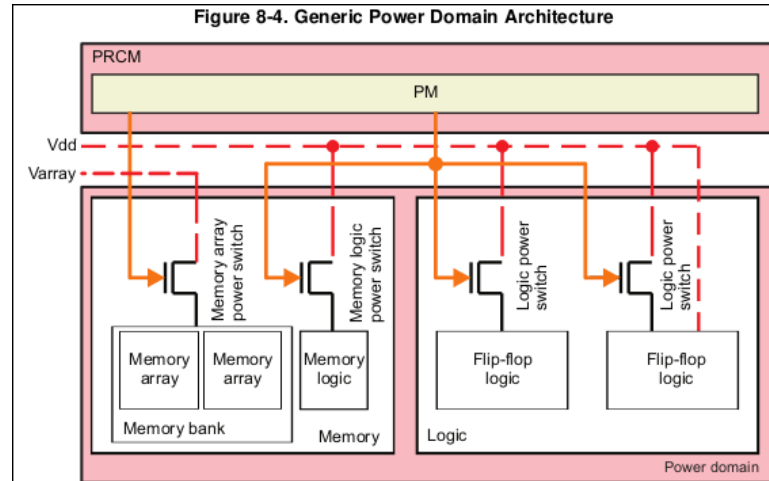
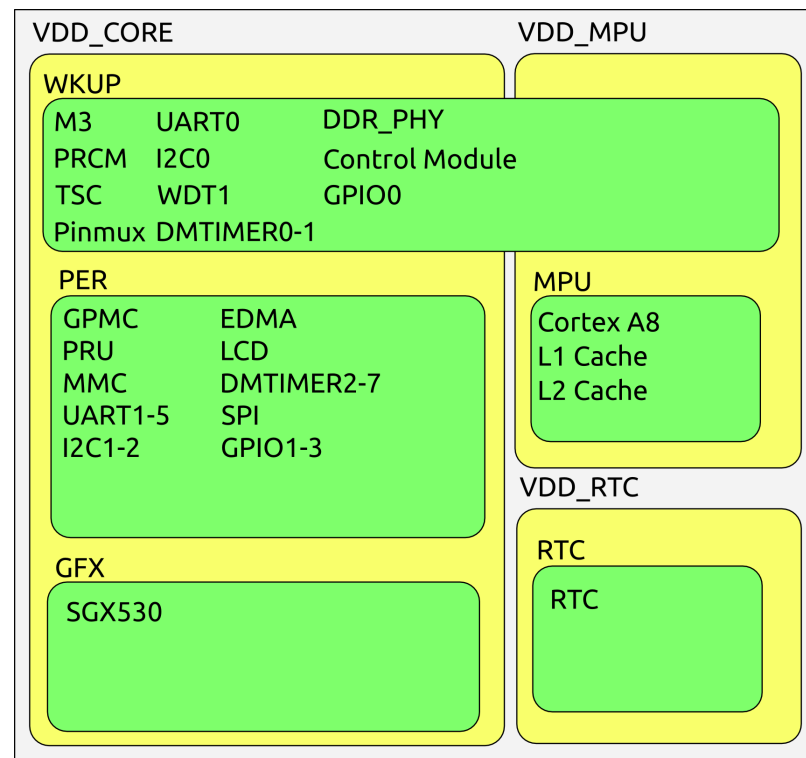


Figure 8-4. Generic Power Domain Architecture



AM33xx PM Overview

- MPU, PER, and GFX power domains can be turned off during suspend
- Current OMAP PM core assumes WKUP domain will always have power



WKUP Context

- Used for:
 - Power, reset, and clock management (PRCM)
 - Pin mux configuration
 - modules that wake up the processor from suspend
 - Such as UART0, GPIO0
 - modules that should continue running when processor is in suspend
 - Such as DMTIMER0
- After hibernation, we need to restore this state
 - Many critical portions of kernel code depend on devices within the WKUP domain
 - sched_timeout

PRCM – Power Domains

- Represented by arch/arm/mach-omap2/powerdomain.c
 - Power state configuration

```
static void am33xx_pwrldm_save_context(struct powerdomain *pwrldm)
{
    pwrldm->context = am33xx_prm_read_reg(pwrldm->prcm_offs,
                                          pwrldm->pwrstctrl_offs);

    /*
     * Do not save LOWPOWERSTATECHANGE, writing a 1 indicates a request,
     * reading back a 1 indicates a request in progress.
     */
    pwrldm->context &= ~AM33XX_LOWPOWERSTATECHANGE_MASK;
}

static void am33xx_pwrldm_restore_context(struct powerdomain *pwrldm)
{
    am33xx_prm_write_reg(pwrldm->context, pwrldm->prcm_offs,
                        pwrldm->pwrstctrl_offs);
    am33xx_pwrldm_wait_transition(pwrldm);
}
```

PRCM – Reset State/Module State

- Represented by omap_hwmod, leverage it

```
for (i = 0; i < oh->rst_lines_cnt; i++)
    if (oh->rst_lines[i].context)
        _assert_hardreset(oh, oh->rst_lines[i].name);
    else if (oh->_state == _HWMOD_STATE_ENABLED)
        _deassert_hardreset(oh, oh->rst_lines[i].name);

if (oh->_state == _HWMOD_STATE_ENABLED) {
    if (soc_ops.enable_module)
        soc_ops.enable_module(oh);
} else {
    if (soc_ops.disable_module)
        soc_ops.disable_module(oh);
}
```

PRCM – Clock Domains

- Represented by arch/arm/mach-omap2/clockdomain.c

```
static int am33xx_clkdm_save_context(struct clockdomain *clkdm)
{
    clkdm->context = am33xx_cm_read_reg_bits(clkdm->cm_inst,
                                              clkdm->clkdm_offs, AM33XX_CLKTRCTRL_MASK);

    return 0;
}

static int am33xx_clkdm_restore_context(struct clockdomain *clkdm)
{
    switch (clkdm->context) {
        case OMAP34XX_CLKSTCTRL_DISABLE_AUTO:
            am33xx_clkdm_deny_idle(clkdm);
            break;
        case OMAP34XX_CLKSTCTRL_FORCE_SLEEP:
            am33xx_clkdm_sleep(clkdm);
            break;
        case OMAP34XX_CLKSTCTRL_FORCE_WAKEUP:
            am33xx_clkdm_wakeup(clkdm);
            break;
        case OMAP34XX_CLKSTCTRL_ENABLE_AUTO:
            am33xx_clkdm_allow_idle(clkdm);
            break;
    }
    return 0;
}
```

PRCM - Clocks

- Leverage the clock tree by adding context save/restore callbacks

```
static int clk_save_context(struct clk *clk)
{
    struct clk *child;
    struct hlist_node *tmp;
    int ret = 0;

    hlist_for_each_entry(child, tmp, &clk->children, child_node) {
        ret = clk_save_context(child);
        if (ret < 0)
            return ret;
    }

    if (clk->ops && clk->ops->save_context)
        ret = clk->ops->save_context(clk->hw);

    return ret;
}

static void clk_restore_context(struct clk *clk)
{
    struct clk *child;
    struct hlist_node *tmp;

    if (clk->ops && clk->ops->restore_context)
        clk->ops->restore_context(clk->hw);

    hlist_for_each_entry(child, tmp, &clk->children, child_node)
        clk_restore_context(child);
}
```

```
int clk_divider_save_context(struct clk_hw *hw)
{
    struct clk_divider *divider = to_clk_divider(hw);
    u32 val;

    val = readl(divider->reg) >> divider->shift;
    divider->context = val & div_mask(divider);

    return 0;
}

void clk_divider_restore_context(struct clk_hw *hw)
{
    struct clk_divider *divider = to_clk_divider(hw);
    u32 val;

    val = readl(divider->reg);
    val &= ~(div_mask(divider) << divider->shift);
    val |= divider->context << divider->shift;
    writel(val, divider->reg);
}
```

pinctrl

- Controls how internal signals are routed to external pins
- Contains memory map of register area, but no complete description of registers
- AM335X errata complicates the situation, certain registers lose context when the PER domain powers during suspend
- The pinctrl subsystem needs knowledge of which registers are available, and which domain they are in.

pinctrl

- Temporary measure, list each power domain register set as a pinconf function

```
am33xx_pinmux: pinmux@44e10800 {
    compatible = "pinctrl-single";
    reg = <0x44e10800 0x0238>;
    #address-cells = <1>;
    #size-cells = <0>;
    pinctrl-single,register-width = <32>;
    pinctrl-single,function-mask = <0x7f>;

    pinctrl-names = "default", "context";
    pinctrl-0 = <>;
    pinctrl-1 = <&am33xx_pmx_wkup &am33xx_pmx_per>;

    am33xx_pmx_wkup: am33xx_pmx_wkup {
        pinctrl-single,pins = <
            0x000 0x00 /* GPMC_AD0 */
            0x004 0x00 /* GPMC_AD1 */
            0x008 0x00 /* GPMC_AD2 */
            0x00c 0x00 /* GPMC_AD3 */
            0x010 0x00 /* GPMC_AD4 */
            0x014 0x00 /* GPMC_AD5 */

            0x1e8 0x00 /* EMU1 */
            0x1f8 0x00 /* RTC_PWRONRSTN */
            0x1fc 0x00 /* PMIC_POWER_EN */
            0x200 0x00 /* EXT_WAKEUP */
            0x204 0x00 /* RTC_KALDO_ENN */
            0x21c 0x00 /* USB0_DRVVBUS */
            0x234 0x00 /* USB1_DRVVBUS */

        >;
    };
};
```

```
am33xx_pmx_per: am33xx_pmx_per {
    pinctrl-single,pins = <
        0x040 0x00 /* GPMC_A0 */
        0x044 0x00 /* GPMC_A1 */
        0x048 0x00 /* GPMC_A2 */
        0x04c 0x00 /* GPMC_A3 */
        0x050 0x00 /* GPMC_A4 */
        0x054 0x00 /* GPMC_A5 */
        0x058 0x00 /* GPMC_A6 */

        0x130 0x00 /* MII1_RXCLK */
        0x134 0x00 /* MII1_RXD3 */
        0x138 0x00 /* MII1_RXD2 */
        0x13c 0x00 /* MII1_RXD1 */
        0x140 0x00 /* MII1_RXD0 */
        0x144 0x00 /* MII1_REFCLK */
        0x148 0x00 /* MDIO_DATA */
        0x14c 0x00 /* MDIO_CLK */

    >;
};
```

pinctrl

- Code added to pinctrl to save/restore a pinctrl function group

```
int pinmux_save_context(struct pinctrl_dev *pctldev, const char *function)
{
    const struct pinmux_ops *pmxops = pctldev->desc->pmxops;
    int ret;

    ret = pinmux_func_name_to_selector(pctldev, function);
    if (ret < 0) {
        dev_err(pctldev->dev, "invalid function %s\n", function);
        return ret;
    }

    if (!pmxops || !pmxops->save_context)
        return -EINVAL;

    return pmxops->save_context(pctldev, ret);
}
EXPORT_SYMBOL(pinmux_save_context);

void pinmux_restore_context(struct pinctrl_dev *pctldev, const char *function)
{
    const struct pinmux_ops *pmxops = pctldev->desc->pmxops;
    int ret;

    ret = pinmux_func_name_to_selector(pctldev, function);
    if (ret < 0) {
        dev_err(pctldev->dev, "invalid function %s\n", function);
        return;
    }

    if (!pmxops || !pmxops->restore_context)
        return;

    pmxops->restore_context(pctldev, ret);
}
EXPORT_SYMBOL(pinmux_restore_context);
```

```
static int pcs_save_context(struct pinctrl_dev *pctldev, unsigned fselector)
{
    struct pcs_device *pcs;
    struct pcs_function *func;
    int i;

    pcs = pinctrl_dev_get_drvdata(pctldev);
    func = radix_tree_lookup(&pcs->ftree, fselector);
    if (!func) {
        dev_err(pcs->dev, "%s could not find function%i\n",
                __func__, fselector);
        return -ENODEV;
    }

    for (i = 0; i < func->nvals; i++) {
        struct pcs_func_vals *vals;

        vals = &func->vals[i];
        vals->val = pcs->read(vals->reg);
    }

    return 0;
}

static void pcs_restore_context(struct pinctrl_dev *pctldev, unsigned fselector)
{
    struct pcs_device *pcs;
    struct pcs_function *func;
    int i;

    pcs = pinctrl_dev_get_drvdata(pctldev);
    func = radix_tree_lookup(&pcs->ftree, fselector);
    if (!func) {
        dev_err(pcs->dev, "%s could not find function%i\n",
                __func__, fselector);
        return;
    }

    for (i = 0; i < func->nvals; i++) {
        struct pcs_func_vals *vals;
        unsigned val, mask;

        vals = &func->vals[i];
        val = pcs->read(vals->reg);
        if (!vals->mask)
            mask = pcs->fmask;
        else
            mask = pcs->fmask & vals->mask;

        val &= ~mask;
        val |= (vals->val & mask);
        pcs->write(val, vals->reg);
    }
}
```


pinctrl

- Current solution is a bit of a hack and likely not upstreamable.
- Possible solution?
 - New type of pinctrl register grouping
 - Would contain reference to power domain register group is contained in
 - Code could use syscore suspend/resume callbacks to save and restore context
- Problem
 - omap2+ power domains are currently arch specific

clocksource/clockevent

- clockevent is already handled properly, disabling on suspend and reprogramming on resume
 - Located in PER power domain
- clocksource is assumed to be always running and within a domain that does not lose power
- clocksource is also required for many kernel delay calculations. Must be restored before most other kernel code

```
static cycle_t clksrc_suspend_cyc;

static void omap_clksrc_suspend(struct clocksource *cs)
{
    char name[10];
    struct omap_hwmod *oh;

    sprintf(name, "timer%d", clksrc.id);
    oh = omap_hwmod_lookup(name);
    if (!oh)
        return;

    clksrc_suspend_cyc = (cycle_t)__omap_dm_timer_read_counter(&clksrc, 1);
    clksrc.ctx_loss_count = omap_hwmod_get_context_loss_count(oh);
}
```

```
static void omap_clksrc_resume(struct clocksource *cs)
{
    char name[10];
    struct omap_hwmod *oh;
    u32 ctx_loss_cnt_after;

    sprintf(name, "timer%d", clksrc.id);
    oh = omap_hwmod_lookup(name);
    if (!oh)
        return;

    ctx_loss_cnt_after = omap_hwmod_get_context_loss_count(oh);
    if (ctx_loss_cnt_after != clksrc.ctx_loss_count) {
        omap2_dflt_clk_restore_context(__clk_get_hw(clksrc.fclk));
        omap_hwmod_reset(oh);
        __omap_dm_timer_load_start(&clksrc,
                                   OMAP_TIMER_CTRL_ST | OMAP_TIMER_CTRL_AR,
                                   clksrc_suspend_cyc, 1);
        __omap_dm_timer_int_enable(&clksrc, OMAP_TIMER_INT_OVERFLOW);
    }
}
```

SRAM

- Internal memory on many OMAP processors used to run suspend resume code or code that modifies memory controller registers or clocking
- Currently restored only for OMAP3, but in an OMAP3 specific way – Make it more general instead

```
void omap_sram_save_context(void)
{
    if (omap_sram_backup) {
        unsigned long start = omap_sram_ceil - omap_sram_base;
        memcpy(omap_sram_backup, omap_sram_base, omap_sram_skip);
        memcpy(omap_sram_backup + start, omap_sram_ceil,
               omap_sram_size - start);
    }
}

void omap_sram_restore_context(void)
{
    if (omap_sram_backup) {
        unsigned long start = omap_sram_ceil - omap_sram_base;
        memcpy(omap_sram_base, omap_sram_backup, omap_sram_skip);
        memcpy(omap_sram_ceil, omap_sram_backup + start,
               omap_sram_size - start);
    }
}
```

Other Devices

- Many drivers just need to know that their power domain lost context
- Teach arch/arm/mach-omap2/powerdomain.c about hibernation induced off modes.

```
static int pwrldm_lost_power(struct powerdomain *pwrldm, void *unused)
{
    enum pwrldm_func_state fpwrst;

    /*
     * Power has been lost across all powerdomains, increment the
     * counter.
     */
    if (pwrldm->fpwrst == PWRDM_FUNC_PWRST_OFF)
        return 0;

    pwrldm->fpwrst_counter[PWRDM_FUNC_PWRST_OFF - PWRDM_FPWRST_OFFSET]++;

    fpwrst = _pwrldm_read_fpwrst(pwrldm);
    if (fpwrst != PWRDM_FUNC_PWRST_OFF)
        pwrldm->fpwrst_counter[fpwrst - PWRDM_FPWRST_OFFSET]++;
    pwrldm->fpwrst = fpwrst;

    return 0;
}
```

Other Devices

- Many drivers that depend on a context loss count function pointer do not get that pointer under DT based systems
 - gpio-omap
 - omap_hsmmc
 - omap-serial
- Currently a hack fix with a pointer to `omap_pm_get_dev_context_loss_count`
 - [HACK]: Crosses arch/arm/mach-omap2, drivers/ boundary
- There is a need for a generic framework to inform drivers when their device has lost power

Other Devices

- Some drivers are misconfigured in such a way to prevent suspend/resume callbacks during hibernation
- When not using `dev_pm_ops`, the platform_driver `.suspend/.resume` callbacks are used for hibernation thaw/freeze/restore/poweroff functionality
- However, when using `dev_pm_ops` these must be filled in. The helper macro, `SET_SYSTEM_SLEEP_PM_OPS` should be used to fill in the thaw/freeze/restore/poweroff callbacks (unless special thaw/freeze/restore/poweroff behavior is required).

```
static struct dev_pm_ops omap_hsmmc_dev_pm_ops = {
    .suspend      = omap_hsmmc_suspend,
    .resume       = omap_hsmmc_resume,
    .prepare      = omap_hsmmc_prepare,
    .complete     = omap_hsmmc_complete,
    .runtime_suspend = omap_hsmmc_runtime_suspend,
    .runtime_resume = omap_hsmmc_runtime_resume,
};

static struct platform_driver omap_hsmmc_driver = {
    .probe        = omap_hsmmc_probe,
    .remove       = omap_hsmmc_remove,
    .driver       = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .pm = &omap_hsmmc_dev_pm_ops,
        .of_match_table = of_match_ptr(omap_mmc_of_match),
    },
};
```

Other Devices

- Some drivers *do* need special hibernation callbacks
- The omap watchdog requires special handling because the state of the watchdog under the boot kernel is not known

```
static int omap_wdt_restore(struct device *dev)
{
    struct watchdog_device *wdog = dev_get_drvdata(dev);
    struct omap_wdt_dev *wdev = watchdog_get_drvdata(wdog);

    omap_wdt_resume(dev);

    /*
     * We don't know what the resume kernel last pinged the WDT with. If
     * it pinged it with the same value we ping it with, the ping will be
     * ignored. Double ping to be sure we reset the timer.
     */
    if (wdev->omap_wdt_users)
        omap_wdt_ping(wdog);

    return 0;
}

static const struct dev_pm_ops omap_wdt_pm_ops = {
    .suspend      = omap_wdt_suspend,
    .freeze       = omap_wdt_suspend,
    .poweroff     = omap_wdt_suspend,
    .resume       = omap_wdt_resume,
    .thaw         = omap_wdt_resume,
    .restore      = omap_wdt_restore,
};
```

Saving/Restoring WKUP Domain

- Putting it all together in pm33xx.c

```
static int am33xx_wkup_save_context(void)
{
    int ret;

    ret = pinmux_save_context(pmx_dev, "am33xx_pmx_wkup");
    if (ret < 0)
        return ret;

    omap_intc_save_context();
    am33xx_control_save_context();

    clks_save_context();
    pwrdds_save_context();
    omap_hwmods_save_context();
    clkdm_save_context();
    omap_sram_save_context();

    return 0;
}

static void am33xx_wkup_restore_context(void)
{
    clks_restore_context();
    pwrdds_restore_context();
    clkdm_restore_context();
    omap_hwmods_restore_context();
    am33xx_control_restore_context();
    pinmux_restore_context(pmx_dev, "am33xx_pmx_wkup");
    omap_intc_restore_context();
    wkup_m3_reinitialize();
    omap_sram_restore_context();
}
```


Generic ARM Hibernation Support

Hibernation support for ARM

- Minimum implementation
 - `swsusp_arch_suspend`
 - Save current cpu state
 - Call `swsusp_save` to snapshot memory
 - Return control to `swsusp_arch_suspend` caller
 - `swsusp_arch_resume`
 - Perform page copies of pages in the `restore_pbelist`
 - Copies pages from their restored location to their original location
 - Restore cpu state from `swsusp_arch_suspend`
 - Return control to `swsusp_arch_suspend` caller
 - `pfn_is_no_save`
 - Return true if this pfn is not to be saved in the hibernation image
 - `save_processor_state` (Called just before `swsusp_arch_suspend`)
 - Save any extra processor state (fp registers, etc)
 - `restore_processor_state` (Called just after `swsusp_arch_suspend`)
 - Restore extra processor state
- Based on a patchset from Frank Hofmann

Hibernation support - arch_suspend

- swsusp_arch_suspend
 - Utilizes cpu_suspend to save current cpu state
 - Second argument of cpu_suspend is called after state is saved
 - Calling cpu_resume causes execution to return to cpu_suspend caller
 - Utilizing soft_restart disables MMU as cpu_resume expects

```
/*  
 * Snapshot kernel memory and reset the system.  
 * After resume, the hibernation snapshot is written out.  
 */  
static int notrace __swsusp_arch_save_image(unsigned long unused)  
{  
    extern int swsusp_save(void);  
    int ret;  
  
    ret = swsusp_save();  
    if (ret == 0)  
        soft_restart(virt_to_phys(cpu_resume));  
    return ret;  
}  
  
/*  
 * Save the current CPU state before suspend / poweroff.  
 */  
int notrace swsusp_arch_suspend(void)  
{  
    return cpu_suspend(0, __swsusp_arch_save_image);  
}
```

Hibernation support - arch_resume

- swsusp_arch_resume
 - Uses stack allocated in nosave region to prevent ourselves from overwriting our stack
 - We will overwrite our code, but with the same bytes
 - Uses cpu_resume to restore cpu state and return to cpu_suspend caller (swsusp_arch_suspend)

```
/*
 * The framework loads the hibernation image into a linked list anchored
 * at restore_pblist, for swsusp_arch_resume() to copy back to the proper
 * destinations.
 *
 * To make this work if resume is triggered from initramfs, the
 * pagetables need to be switched to allow writes to kernel mem.
 */
static void notrace __swsusp_arch_restore_image(void *unused)
{
    extern struct pbe *restore_pblist;
    struct pbe *pbe;

    cpu_switch_mm(idmap_pgd, &init_mm);
    for (pbe = restore_pblist; pbe; pbe = pbe->next)
        copy_page(pbe->orig_address, pbe->address);

    soft_restart_noirq(virt_to_phys(cpu_resume));
}

static u8 __swsusp_resume_stk[PAGE_SIZE/2] __nosavedata;

/*
 * Resume from the hibernation image.
 * Due to the kernel heap / data restore, stack contents change underneath
 * and that would make function calls impossible; switch to a temporary
 * stack within the nosave region to avoid that problem.
 */
int __naked swsusp_arch_resume(void)
{
    extern void call_with_stack(void (*fn)(void *), void *arg, void *sp);
    cpu_init(); /* get a clean PSR */
    call_with_stack(__swsusp_arch_restore_image, 0,
        __swsusp_resume_stk + sizeof(__swsusp_resume_stk));
    return 0;
}
```

AM33xx Hibernation Support

- With prep work done, adding hibernation support to AM33xx is actually fairly straightforward
- begin/end wrap all hibernation code
- We use disable/enable_hlt to prevent pm_idle from being called
- The enter call back just powers down the machine
- These calls make sure that the hardware is in the same state before running the restored image as when it was made

```
static int am33xx_hibernation_begin(void)
{
    disable_hlt();
    return 0;
}

static void am33xx_hibernation_end(void)
{
    enable_hlt();
}
```

```
static int am33xx_hibernation_enter(void)
{
    machine_power_off();
    return 0;
}
```

```
static int am33xx_hibernation_pre_restore(void)
{
    omap2_gpio_prepare_for_idle(1);
    return 0;
}

static void am33xx_hibernation_restore_cleanup(void)
{
    omap2_gpio_resume_after_idle();
}
```

AM33xx Hibernation Support

- pre_snapshot saves all our state registers and prepares the GPIOs for power loss
- leave is called after restoring an image. We inform the power domains that they have lost power and we restore our wkup context
- finish is called both after restoring an image (after leave) and after snapshotting the system. We continue our context restore and also undo the actions in pre_snapshot

```
static int am33xx_hibernation_pre_snapshot(void)
{
    am33xx_per_save_context();
    omap2_gpio_prepare_for_idle(1);
    am33xx_wkup_save_context();

    return 0;
}

static void am33xx_hibernation_leave(void)
{
    pwrdoms_lost_power();
    am33xx_wkup_restore_context();
}

static void am33xx_hibernation_finish(void)
{
    omap2_gpio_resume_after_idle();
    am33xx_per_restore_context();
}
```



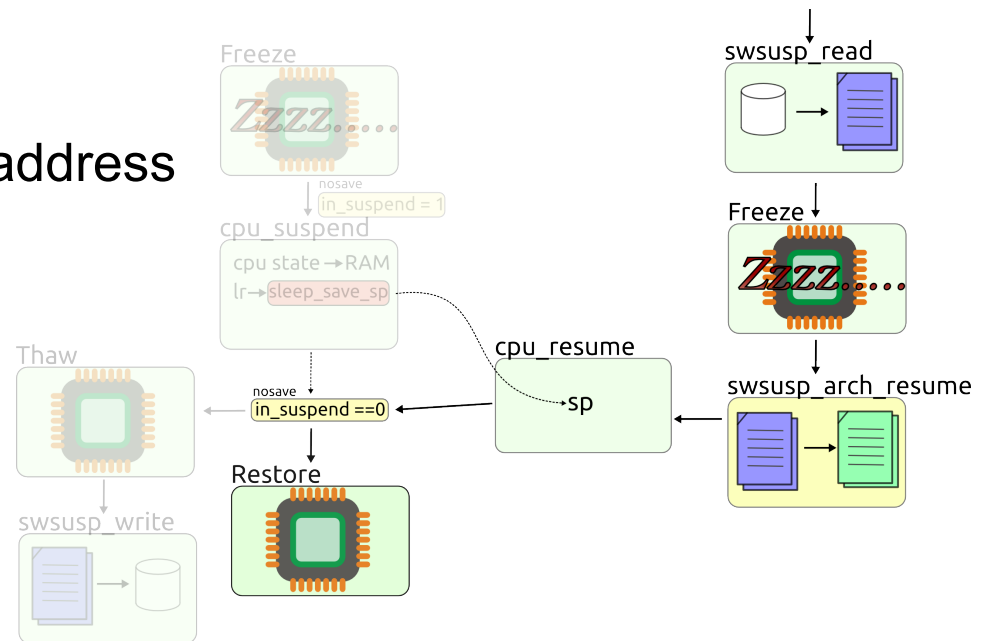
Debugging Methods

- Debugging can be difficult as the hardware is usually in some unknown state.
- Debugging using GPIOs
 - GPIOs are usually pretty easy to configure clocks for and enable with just a few register writes, even from assembly
 - Binary search of where the code is failing can be performed by moving the GPIO enable around
- printk
 - The kernel logging facility is useful so long as you are getting to a point where serial output is enabled
- openocd
 - Usually PC is down in the weeds
 - openocd+gdb?
- Register map comparisons
 - Utilizing devmem2 to snapshot register values before and after a hibernation file is useful to track down missed registers or buggy restore code

Restore from U-Boot

swsusp and U-Boot

- Restoring from hibernation just involves copying pages from disk into memory and jumping to an address
 - That's what U-Boot does!
- Restoring from U-Boot can be faster than booting a kernel just to copy pages
- Issues
 - U-Boot has no idea what address to jump to
 - U-Boot doesn't know the contents or even location of the nosave pages



Kernel Modifications – nosave section

- U-Boot doesn't know about nosave pages or their address
- We instead save and restore them from the kernel
- Backup nosave pages are saved at boot
- Special version of `cpu_resume` is provided that restores nosave pages before calling the real `cpu_resume`

```
static int __init swsusp_arch_init(void)
{
    char *backup;
    size_t len;

    len = &__nosave_end - &__nosave_begin;
    backup = kmalloc(len, GFP_KERNEL);
    if (backup) {
        pr_info("%s: Backed up %d byte nosave region\n", __func__, len);
        memcpy(backup, &__nosave_begin, len);
    }

    __nosave_backup_phys = virt_to_phys(backup);
    __nosave_begin_phys = virt_to_phys(&__nosave_begin);
    __nosave_end_phys = virt_to_phys(&__nosave_end);

    return 0;
}
late_initcall(swsusp_arch_init);
```

```
ENTRY(cpu_resume_restore_nosave)
    ldr    r0, =__nosave_backup_phys
    ldr    r0, [r0]
    ldr    r1, =__nosave_begin_phys
    ldr    r1, [r1]
    ldr    r2, =__nosave_end_phys
    ldr    r2, [r2]
1:      ldmia r0!, {r3-r10}
    stmia  r1!, {r3-r10}
    cmp    r1, r2
    bne    1b
    b      cpu_resume
```

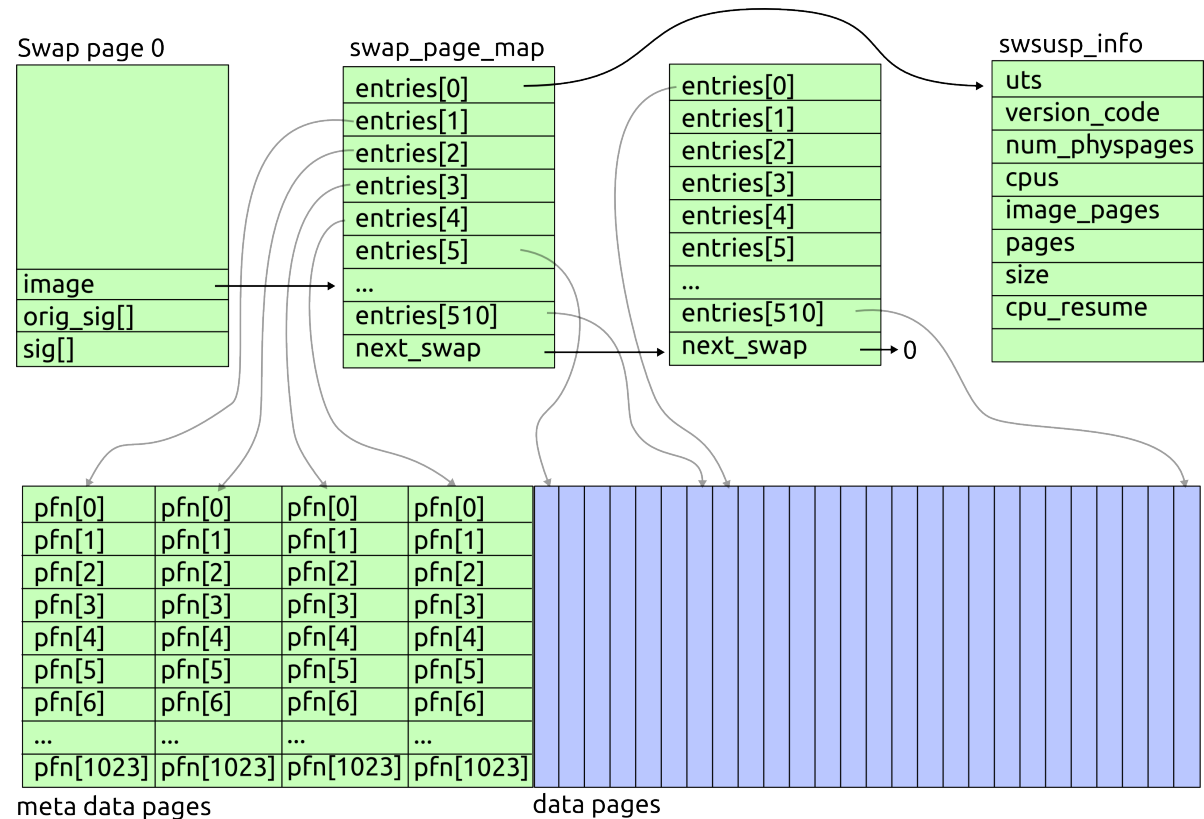
Kernel Modifications - cpu_resume

- Need to pass address of cpu_resume function to U-Boot
 - Store in swsusp_info page
 - Add arch callback for storing that data in the swsusp_info page
- Just stores the physical address of the new version of cpu_resume that first copies the nosave pages

```
void swsusp_arch_add_info(char *archdata, size_t size)
{
    *(u32 *) archdata = virt_to_phys(cpu_resume_restore_nosave);
}
```

swsusp Image Layout

- Each metadata entry is associated with the same numbered data page
- Each data page is to be loaded into memory at the pfn indicated by its metadata pfn entry



U-Boot modifications

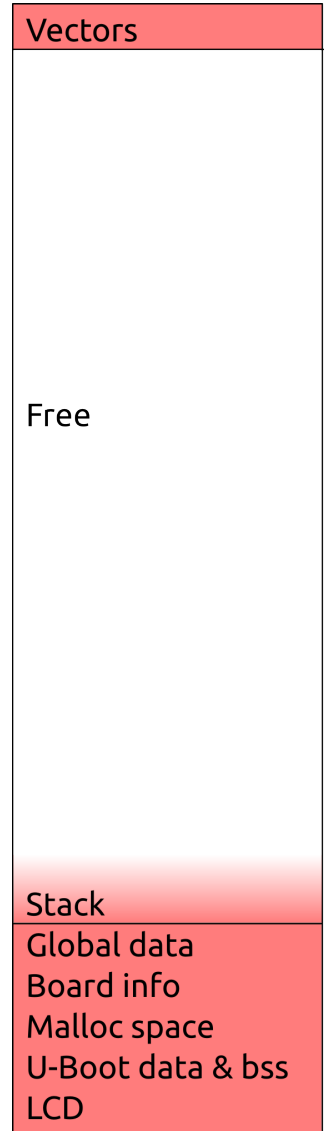
- Provide cmd_swsusp
 - No-op if S1SUSPEND sig does not exist
 - Rewrites sig with orig_sig to prevent boot loop on bad image
 - Snapshot booting can populate orig_sig with S1SUSPEND
 - Reads in metadata pages with pfn mappings
 - Also populates bitmap of used pages for easy access to free pages
 - Copy each data page to memory
 - Original location if it is free
 - Other wise copy to first available free page and update remap list
 - Copy finish function and cpu_resume address to free data page
 - Run finish function from free data page (use stack contained in free page)
 - Copies remapped pages to their correct location
 - Jumps to cpu_resume function

```
U-Boot# help swsusp
swsusp - Restore SWSUSP hibernation image

Usage:
swsusp <interface> [<dev[:part]>] [<offset>]
```

U-Boot Memory Mapping

- The U-Boot memory mapping makes it very easy to see if we can load a page directly into its original location
- If not, we load it into a location not used by U-Boot or the final location of any of the swsusp pages



Loading pfn and Free Page Mapping

- We utilize malloc'd pages to store the pfn index
- Mark used pages as we go

```
memset(pfn_pages, 0, nr_pfn_pages * sizeof(u32 *));
for (i = 0; i < nr_pfn_pages; i++) {
    u32 idx;
    pfn_pages[i] = malloc(PAGE_SIZE);
    if (!pfn_pages[i])
        goto mem_err;
    if (image_page_get_next(pfn_pages[i]) <= 0)
        goto read_err;
    for (idx = 0; idx < PAGE_SIZE / sizeof(u32); idx++) {
        u32 page = pfn_pages[i][idx];
        if (page == ~0UL)
            break;
        free_page_mark_used(page);
    }
}
```

Loading swsusp Pages Into Memory

- Utilize free pages to store remapping lists, malloc'd data will be overwritten
- min_page is first free page in U-Boot memory map
- max_page is last free page in U-Boot memory map (well before stack pointer)
- If a page is to be copied into U-Boot's memory space, it is instead copied into an unused free page

```
remap_orig = pg2addr(free_page_get_next());
remap_temp = pg2addr(free_page_get_next());
remap_idx = 0;

for (i = 0; i < swsusp_info->image_pages; i++) {
    u32 page = pfn_pages[i >> 10][i & 0x3ff];
    if (page < min_page || page > max_page) {
        if (nr_remap == remap_idx + 1)
            goto err;
        remap_orig[remap_idx] = pg2addr(page);
        page = free_page_get_next();
        remap_temp[remap_idx] = pg2addr(page);
        remap_idx++;
    }
    if (image_page_get_next(pg2addr(page)) <= 0)
        goto read_err;
}
```


Prepare to Copy Remapped Pages

- Final copy must happen from memory unused by swsusp or U-Boot
 - remap_orig/remap_temp already exist in free page
 - Utilize free page for final copy of remapped pages
 - Copy swsusp_finish into page
 - Copy context information into page
 - Setup stack pointer at end of page

```
/* put end markers on the remap list */
remap_orig[remap_idx] = (void *) ~0UL;
remap_temp[remap_idx] = (void *) ~0UL;

/* Make a copy of swsusp_finish in a free data page */
data_page = pg2addr(free_page_get_next());
memcpy(data_page, swsusp_finish, PAGE_SIZE);
swsusp_finish_copy = (void *) data_page;

/* Setup context for swsusp_finish */
context = (struct swsusp_finish_context *) (data_page + PAGE_SIZE);
context--;
context->remap_orig = remap_orig;
context->remap_temp = remap_temp;
context->cpu_resume = swsusp_info->cpu_resume;

/* Get a stack pointer for swsusp_finish */
stack_addr = ((char *) context) + PAGE_SIZE - sizeof(u32);

cleanup_before_linux();

/* Copy the final data from a safe place */
call_with_stack(swsusp_finish_copy, context, stack_addr);
```

Copy Remaining Pages

- Moved remapped pages into their originally intended location
- Call `cpu_resume` (actually `cpu_resume_copy_nosave`)

```
static void swsusp_finish (void *userdata)
{
    struct swsusp_finish_context *context = userdata;

    while (*context->remap_orig != (void *) ~0UL) {
        u32 *orig, *temp;
        int count;

        count = PAGE_SIZE / 4;
        orig = *context->remap_orig;
        temp = *context->remap_temp;
        while (count--)
            *orig++ = *temp++;

        context->remap_orig++;
        context->remap_temp++;
    }
    context->cpu_resume();
}
```

Lessons Learned/Open Issues

- Save/Restore context function
 - Bit of a hack, better to keep context as we go and look at context loss count
 - /dev/mem bitbangers?
 - Registers that are initialized at boot and never read by the driver?
- Over zealous restore
 - Don't restore context of clocks who's parents are disabled
 - Check to make sure clock drivers are programming appropriate registers on resume
- Setup debug framework early
- pinctrl
 - Properly integrate with system

Questions?

Introducing: The Next-Gen BeagleBone

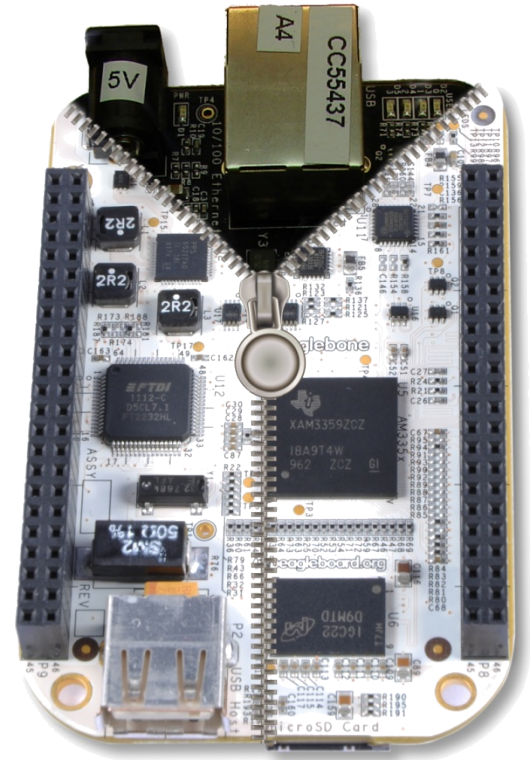
An introduction worthy of a black tie affair.

- New color for Spring
- New and improved features
- Bold move to more performance for lower cost

Want a sneak peek and information on
advanced ordering options?

Make an impression. Register your interest today.

beagleboard.org/unzipped



beagleboard.org