mode name	sleep name	lpp_sleep_level_t	handler
wake	S0	e_lpp_level_wake	disable_all()
light	N/A	e_lpp_level_light	handle_light_sleep()
deep	S1	e_lpp_level_deep	handle_deep_sleep()
suspend	S2	e_lpp_level_suspend	handle_suspend_sleep()
hibernate	S4	e_lpp_level_hibernate	handle_hibernate_sleep()
softoff	S5	e_lpp_level_softoff	handle_softoff_sleep()
rtos	N/A	e_lpp_level_rtos	handle_rtos_sleep()

handler	send command to R4	send value to R4	default value	optional
handle_light_sleep()	e_lpp_sys_sleep_level	e_lpp_sleep_level_suspend_1		N
handle_deep_sleep()	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep		N
	e_lpp_sys_initialize	0xFFFFFFFF		N
	e_lpp_sys_initialize	LOWPOWER_COMMAND_DATECODE	0x20150903	N
	e_lpp_sys_power_service	lpi.lpp[e_lpp_level_deep].service	LP_LPP_FLAGS_DEEP_SERVICE(0)	N
	e_lpp_sys_power_wfi	lpi.lpp[e_lpp_level_deep].wfi	LP_LPP_FLAGS_DEEP_SLEEP	N
	e_lpp_sys_power_global	lpi.lpp[e_lpp_level_deep].global	LP_LPP_FLAGS_DEEP_GLOBAL	N
	e_lpp_sys_ddr_mc_flags	lpi.lpp[e_lpp_level_deep].mc	LP_LPP_FLAGS_DEEP_MC	N
	e_lpp_sys_avs	lpi.lpp[e_lpp_level_deep].avs	LP_LPP_FLAGS_AVS_DEFAULT	N
	e_lpp_sys_power_debug	lpi.lpp[e_lpp_level_deep].debug	LP_LPP_FLAGS_DEEP_DBG	N
	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep_rdy		N
handle_suspend_sleep()	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep		N
	e_lpp_sys_initialize	0xFFFFFFFF		N
	e_lpp_sys_initialize	LOWPOWER_COMMAND_DATECODE	0x20150903	N

	e_lpp_sys_power_service	lpi.lpp[e_lpp_level_suspend].service		N
	e_lpp_sys_power_wfi	lpi.lpp[e_lpp_level_suspend].wfi		N
	e_lpp_sys_power_global	lpi.lpp[e_lpp_level_suspend].global		N
	e_lpp_sys_ddr_mc_flags	lpi.lpp[e_lpp_level_suspend].mc		N
	e_lpp_sys_avs	lpi.lpp[e_lpp_level_suspend].avs		N
	e_lpp_sys_power_debug	lpi.lpp[e_lpp_level_suspend].debug		N
	e_lpp_sys_irq_x	LP_LPP_SUSPEND_DEFAULT_WAKE_INT_0		Y
	e_lpp_sys_irq_x	LP_LPP_SUSPEND_DEFAULT_WAKE_INT_1		Y
	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep_rdy		N
handle_hibernate_sleep()	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep		N
	e_lpp_sys_initialize	0xFFFFFFFF		N
	e_lpp_sys_initialize	LOWPOWER_COMMAND_DATECODE	0x20150903	N
	e_lpp_sys_power_service	lpi.lpp[e_lpp_level_hibernate].service		N
	e_lpp_sys_power_wfi	lpi.lpp[e_lpp_level_hibernate].wfi		N
	e_lpp_sys_power_global	lpi.lpp[e_lpp_level_hibernate].global		N
	e_lpp_sys_d	lpi.lpp[e_lpp_level_hiber		

	e_lpp_sys_ddr_mc_flags	lpi.lpp[e_lpp_level_hibernate].mc		N
	e_lpp_sys_avs	lpi.lpp[e_lpp_level_hibernate].avs		N
	e_lpp_sys_power_debug	lpi.lpp[e_lpp_level_hibernate].debug		N
	e_lpp_sys_alarm_ticks	LP_LPP_TEST_HIBERNATE_WAKE_TIMER		Y
	e_lpp_gpio_pin_x	LP_LPP_TEST_HIBERNATE_WAKE_GPIO_0		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_HIBERNATE_WAKE_INT_0		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_HIBERNATE_WAKE_INT_1		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_HIBERNATE_WAKE_INT_2		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_HIBERNATE_WAKE_INT_3		Y
	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep_rdy		N
handle_softoff_sleep	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep		N
	e_lpp_sys_initialize	0xFFFFFFFF		N
	e_lpp_sys_initialize	LOWPOWER_COMMAND_DATECODE	0x20150903	N
	e_lpp_sys_power_service	lpi.lpp[e_lpp_level_softoff].service		N
	e_lpp_sys_power_wfi	lpi.lpp[e_lpp_level_softoff].wfi		N
	e_lpp_sys_power_global	lpi.lpp[e_lpp_level_softoff].global		N
	e_lpp_sys_d	lpi.lpp[e_lpp_level_softo		

	e_lpp_sys_ddr_mc_flags	lpi.lpp[e_lpp_level_softoff].mc		N
	e_lpp_sys_avs	lpi.lpp[e_lpp_level_softoff].avs		N
	e_lpp_sys_power_debug	lpi.lpp[e_lpp_level_softoff].debug		N
	e_lpp_sys_alarm_ticks	LP_LPP_TEST_SOFTOFF_WAKE_TIMER		Y
	e_lpp_gpio_pin_x	LP_LPP_TEST_SOFTOFF_WAKE_GPIO_0		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_SOFTOFF_WAKE_INT_0		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_SOFTOFF_WAKE_INT_1		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_SOFTOFF_WAKE_INT_2		Y
	e_lpp_sys_irq_x	LP_LPP_TEST_SOFTOFF_WAKE_INT_3		Y
	e_lpp_sys_sleep_level	e_lpp_sleep_level_deep_rdy		N
handle_rtos_sleep	e_lpp_sys_power_wfi	lpi.lpp[e_lpp_level_deep].wfi		N
	e_lpp_sys_ddr_mc_flags	lpi.lpp[e_lpp_level_deep].mc		N
	e_lpp_sys_sleep_level	e_lpp_sleep_level_suspend_2		N

```c
typedef struct lp_control_s
{
    unsigned int     mode;
    char *           mode_name[e_lpp_level_levels];
    char *           sleep_name[e_lpp_level_levels];
    unsigned int     debug;                                     // enable our debug mode
    unsigned int     light_lpi_enable;                          // light sleep is either AP WFI, or our LPI_MODE_LINUX_SLEEP_1
    lpp_control_t    lpp[e_lpp_level_levels];
} lp_control_t;
```

```c
typedef struct lpp_control_s
{
    unsigned int     wake;
    unsigned int     service;
    unsigned int     wfi;
    unsigned int     global;
    unsigned int     avs;
    unsigned int     mc;
    unsigned int     debug;
    unsigned int     asr_clocks;
} lpp_control_t;
```

而lpp_control_t中的的每个field其实都是一个 `lpp_power_control_t`

in ccsgit/r4/common/asic/88pa6220/lowpower/include/lpp_api.h

```
/**
 * lpp_power_config bit flags
 */
typedef struct
{
    uint32_t    core0_pd:1;         // b0     AP asking LPP to power down core0
    uint32_t    core0_down:1;       // b1     LPP has PD core0, internal written
    uint32_t    reserved2:6;        // b7:2   AP asking LPP
    uint32_t    ddr_sr:1;           // b8     AP asking LPP to put DDR into self refresh
    uint32_t    ddr_pll_bp:1;       // b9     AP asking LPP when in self refresh to put DDR PLL into bypass
    uint32_t    ddr_pll_pd:1;       // b10    AP asking LPP when DDR PLL in bypass to power down PLL
    uint32_t    ddr_asr:1;          // b11    AP asking LPP to put DDR into auto self refresh..
    uint32_t    mc_busguard:1;      // b12    AP asking LPP busguard MC
    uint32_t    reserved13:1;       // b13    AP asking LPP think
    uint32_t    ddr_halt_sched:1;   // b14    AP asking LPP to halt MC scheduler prior to DDR SR, monitor MC status to exit
    uint32_t    ddr_mc_pad:1;       // b15    AP asking LPP
    uint32_t    core_pll_bp:1;      // b16    AP asking LPP to put Core PLL into bypass
    uint32_t    core_pll_pd:1;      // b17    AP asking LPP when Core PLL is in bypass, to power down
    uint32_t    reserved18:2;       // b19:18 AP asking LPP
    uint32_t    clk_div:2;          // b21:20 AP asking LPP to change clock div
    uint32_t    avs:1;              // b22    AP asking LPP to change AVS control and/or VddLimit registers, as defined in cmd 'e_lpp_sys_avs' and 'lpp_power_avs_t'
    uint32_t    sys_clk_gate:1;     // b23    AP asking LPP to gate SYS CLK
    uint32_t    sys_pll_pd:1;       // b24    AP asking LPP to gate+bp+pd SYS CLK
    uint32_t    reserved25:7;       // b31:25 AP asking LPP
} lpp_power_control_t;
```

How to interpret the following assignments in lpi global variable ?

in low_power/low-power-mod/low_power_idle.c

```
static lp_control_t lpi = {
    ......
    .lpp[e_lpp_level_light].wfi         = LP_AP_FLAGS_DEFAULT,
    .lpp[e_lpp_level_light].asr_clocks  = LP_AP_AUTO_SR_IDLE_CLKS,
    .lpp[e_lpp_level_light].mc          = LP_LPP_FLAGS_HIBERNATE_MC,

    .lpp[e_lpp_level_deep].avs          = LP_LPP_FLAGS_AVS_DEFAULT,
    .lpp[e_lpp_level_deep].global       = LP_LPP_FLAGS_DEEP_GLOBAL,
    .lpp[e_lpp_level_deep].mc           = LP_LPP_FLAGS_DEEP_MC,
    .lpp[e_lpp_level_deep].service      = LP_LPP_FLAGS_DEEP_SERVICE,
    .lpp[e_lpp_level_deep].wake         = 0,
    .lpp[e_lpp_level_deep].wfi          = LP_LPP_FLAGS_DEEP_SLEEP,
    .lpp[e_lpp_level_deep].debug        = LP_LPP_FLAGS_DEEP_DBG,

    .lpp[e_lpp_level_suspend].avs       = LP_LPP_FLAGS_AVS_DEFAULT,
    .lpp[e_lpp_level_suspend].global    = LP_LPP_FLAGS_SUSPEND_GLOBAL,
    .lpp[e_lpp_level_suspend].mc        = LP_LPP_FLAGS_SUSPEND_MC,
    .lpp[e_lpp_level_suspend].service   = LP_LPP_FLAGS_SUSPEND_SERVICE,
    .lpp[e_lpp_level_suspend].wake      = 0,
    .lpp[e_lpp_level_suspend].wfi       = LP_LPP_FLAGS_SUSPEND_SLEEP,
    .lpp[e_lpp_level_suspend].debug     = LP_LPP_FLAGS_SUSPEND_DBG,

    .lpp[e_lpp_level_hibernate].avs     = LP_LPP_FLAGS_AVS_DEFAULT,
    .lpp[e_lpp_level_hibernate].global  = LP_LPP_FLAGS_HIBERNATE_GLOBAL,
    .lpp[e_lpp_level_hibernate].mc      = LP_LPP_FLAGS_HIBERNATE_MC,
    .lpp[e_lpp_level_hibernate].service = LP_LPP_FLAGS_HIBERNATE_SERVICE,
    .lpp[e_lpp_level_hibernate].wake    = 0,
    .lpp[e_lpp_level_hibernate].wfi     = LP_LPP_FLAGS_HIBERNATE_SLEEP,
    .lpp[e_lpp_level_hibernate].debug   = LP_LPP_FLAGS_HIBERNATE_DBG,

    .lpp[e_lpp_level_softoff].avs       = LP_LPP_FLAGS_AVS_DEFAULT,
    .lpp[e_lpp_level_softoff].global    = LP_LPP_FLAGS_SOFTOFF_GLOBAL,
    .lpp[e_lpp_level_softoff].mc        = LP_LPP_FLAGS_SOFTOFF_MC,
    .lpp[e_lpp_level_softoff].service   = LP_LPP_FLAGS_SOFTOFF_SERVICE,
    .lpp[e_lpp_level_softoff].wake      = 0,
    .lpp[e_lpp_level_softoff].wfi       = LP_LPP_FLAGS_SOFTOFF_SLEEP,
    .lpp[e_lpp_level_softoff].debug     = LP_LPP_FLAGS_SOFTOFF_DBG,
};
```

for example

> .lpp[e_lpp_level_light].wfi = LP_AP_FLAGS_DEFAULT,

```
#define LP_AP_FLAGS_DEFAULT         (LP_FLAGS_DDR_AUTO_SR )
#define LP_FLAGS_DDR_AUTO_SR        (1 << 0x0B)     // tell MC how many CLK of idle to auto enter SR
```

这里lpp[e_lpp_level_light].wfi的 `e_lpp_level_light` 是A53上Linux下low power的一种状态，而 `wfi` 是指lpp中r4运行的一种状态。
下面是lpp定义的r4运行时的几种状态
in lpp_api.h

```
typedef enum
{                                    // The following levels are just a STATE, the definition is ACTUALLY determined dynamically in a schema
    lpp_driver_power_wake,           // WAKE:    ~= ACPI S1:   AP is powered, but some power controls enabled, control is being transitioned AP to/from LPP
    lpp_driver_power_service,        // SERVICE: ~= ACPI S2/3: AP is (usually) NOT powered, DRAM usually in self refresh, LPP is exe from SRAM
    lpp_driver_power_wfi,            // WFI:     ~= ACPI S4:   AP is (usually) NOT powered, DRAM usually in self refresh, PLL bypass, LPP in WFI
    lpp_driver_power_wfi_poll,       // WFI:     ~= ACPI S4:   AP is (usually) NOT powered, DRAM usually in self refresh, PLL bypass, LPP polling for activity
    lpp_driver_power_unknown         // unknown..
} lpp_power_level_t;
```

上面的 `wfi` 就对应lpp中r4运行时的lpp_driver_power_wfi。

> .lpp[e_lpp_level_light].wfi = LP_AP_FLAGS_DEFAULT

即令r4-lpp在lpp_driver_power_wfi state时的第11(0xB) bit为1,对应到lpp_power_control_t中的bitfield就是

> uint32_t ddr_asr:1; // b11 AP asking LPP to put DDR into auto self refresh..

low_power_idle driver中的设置是怎么起作用的?

在Linux low power idle driver中指定设置，但真正对硬件设置确是在r4-lpp上，这中间过程实在有点曲折，大致步骤如下
由于在 `e_lpp_level_light` state下，Linux low power idle driver并不会在handle_light_sleep()中传递config value `lpi.lpp[e_lpp_level_light].wfi` 给lpp，所以下面以 `lpi.lpp[e_lpp_level_deep].wfi` 为例。

step 1:
in handle_deep_sleep()

> send_low_power_cmd(e_lpp_sys_power_wfi,(void*)lpi.lpp[e_lpp_level_deep].wfi,0,false,0);

通过ipc把config vaule传递给r4-threadx。

step 2:
这是在r4上运行的是threadx(还不是lpp).

in asic_pwr_ipc.c/recv_callback()

```
    default:
        // pass cmd through. this is a configuration / context for LP driver ...
        asic_pwr_set_cmd((uint32_t)command,u32);
        break;
```

(command, value) = (e_lpp_sys_power_wfi, lpi.lpp[e_lpp_level_deep].wfi) pair并不被r4-threadx处理，而是被保存到
asic_low_power_cmd_table[] array中。

asic_low_power_cmd_table[] array也就是r4-lpp中的所谓 `mailbox` 。

step 3:
在从r4-threadx切换到r4-lpp时，在DDR RAM中的asic_low_power_cmd_table[] array会被复制到在LCM中的mailbox.
in transition_cpu()

```
    memcpy((void *)LPP_MAILBOX_ADDR,asic_low_power_cmd_table,(sizeof(asic_low_power_table_t)*ASIC_LOW_POWER_MAX_CMDS));

    // handle low level warmboot routine for asic
    WarmBoot_CPU(lpp_ucode_bin,lpp_ucode_bin_length);

    memcpy(asic_low_power_cmd_table,(void *)LPP_MAILBOX_ADDR,(sizeof(asic_low_power_table_t)*ASIC_LOW_POWER_MAX_CMDS));
```

step 4:
r4-lpp在lpp_power_controls()从mailbox中获取该config value

```
        if (OK != lpp_get_mailbox_cmd(e_lpp_sys_power_wfi,(uint32_t*)&lpp_pwr.config_wfi))
        {
            *(uint32_t*)&lpp_pwr.config_wfi = LPP_POWER_DEFAULT_CONFIG_WFI;
        }
```

这里 `lpp_pwr.config_wfi` 就是在Linux low power idle driver中发来的value

> send_low_power_cmd(e_lpp_sys_power_wfi, (void*)lpi.lpp[e_lpp_level_deep].wfi,0,false,0);

即lpp_pwr.config_wfi = LP_LPP_FLAGS_DEEP_SLEEP =
LP_LPP_FLAGS_SLEEP_HSR_DSC =
(LP_LPP_FLAGS_SLEEP_HSR_D | LP_FLAGS_LPP_CLK_DIV_SET(2) |
LP_FLAGS_SYS_CLK_GATE | LP_FLAGS_CORE_PLL_BP)

step 5:
把config value设置到hardware

比如switch r4-lpp to `wfi` state

in lpp_power_controls.c/pwr_wfi()

```
static bool pwr_wfi()
{
    LPP_TRACE_MODE_ENTRY();
    // check if we have reason to NOT enter WFI...
    if (lpp_wfi_skip())
    {
        return true;
    }
    // service --> wfi
    if (lpp_pwr.config_global.wfi_poll_optimize)
    {
#ifndef HAVE_SERENITY
        // bypass normal power sequencing -- be careful!!
        pwr_wfi_poll_optimize();
#endif
    }
    else
    {
        // set any controls associated with being in our defined 'wfi' mode
        pwr_config(lpp_driver_power_wfi, lpp_pwr.config_wfi);

        LPP_TRACE_MODE_NOTE("wfi asm");
        asm volatile
        (
            " wfi ;"
        );
        // exit wfi --> service
        pwr_service();
    }
    return true;
}
```

> pwr_config(lpp_driver_power_wfi, lpp_pwr.config_wfi);

r4-lpp will set hardware according to `lpp_pwr.config_wfi`.

```
static void pwr_config(lpp_power_level_t target_level, lpp_power_control_t target_config)
{
    //DBG_PRINTF(DBG_LOUD,("%s: prev %#x : %#x; target %#x : %#x \r\n",__FUNCTION__,lpp_pwr.state,lpp_pwr.config,target_level,target_config));
    ASSERT(target_level != lpp_pwr.state);
    /*
     * config is valid, apply it.
     * we have 3 levels:
     * S1:      EXIT/WAKE at this level.
     * S2/S3:   SERVICE -- some configurable power controls
     * S4:      WFI ------ some configurable power controls
     * When we enter LP mode, we transition into SERVICE,
     * Then we SERVICE-->[WFI-->SERVICE-->]* WAKE
     */
    // store our previous cfg
    lpp_pwr.prev_cfg = lpp_pwr.hw_cfg;
    lpp_pwr.prev_state = lpp_pwr.state;
    lpp_pwr.target = target_level;
    //tmpLPP_LOG_EVENT(e_lpp_log_power_level,target_level);

    if (*(uint32_t*)&lpp_pwr.config != *(uint32_t*)&target_config)
    {
#ifndef HAVE_SERENITY
        // we only need to change to target config, IF the config is different...  ** TBD **
        if (target_level < lpp_pwr.state)
        {
            // WFI --> SERVICE or SERVICE --> WAKE
            // apply target config, which is delta from the previous config.
            LPP_POWER_TRACE(e_lpp_dbg_power_start,1);
            lpp_prep_cpu_clk_div(target_config, target_level);
            lpp_avs(target_config);             // restore AVS, if appropriate
            lpp_core_pll(target_config);        // bring up cpu_pll, (could be either core/ddr pll), and apply new clk div
            lpp_sys_pll(target_config);         // bring up sys_pll
            lpp_ddr(target_config);             // now bring up ddr, if appropriate
            lpp_ap0(target_config);             // now bring up ap core 0, if appropriate
            LPP_POWER_TRACE(e_lpp_dbg_power_end,1);
        }
        else // (target_level > previous_level)
        {
            // entry --> SERVICE or SERVICE --> WFI
            // apply target config, which is delta from the previous config.
            LPP_POWER_TRACE(e_lpp_dbg_power_start,0);
            lpp_prep_cpu_clk_div(target_config, target_level);
            lpp_ap0(target_config);             // tear down ap core 0, if appropriate
            lpp_ddr(target_config);             // now tear down ddr, if appropriate
            lpp_sys_pll(target_config);         // now tear down sys pll
            lpp_core_pll(target_config);        // now tear down core pll and apply new clk div, if appropriate
            lpp_avs(target_config);             // apply AVS, if appropriate
            LPP_POWER_TRACE(e_lpp_dbg_power_end,0);
        }
#endif
        lpp_pwr.config = target_config;
    }
    lpp_pwr.state = target_level;
    LPP_LOG_EVENT((e_lpp_log_wake_ready+target_level),0);
    lpp_pwr.target = lpp_driver_power_unknown;
    //DBG_PRINTF(DBG_LOUD,("%s: hw_cfg %#08x \r\n",__FUNCTION__,lpp_pwr.hw_cfg));
}
```

lpp_xxx()读取lpp_pwr.config_wfi中各自的关心的bitfield(其实就是对相应hardware的开关)来设置hardware。