Ameba-Z WAKEUP ON UART

This document introduces usage of UART & LOGUART WAKEUP.

Realtek

Table of Contents

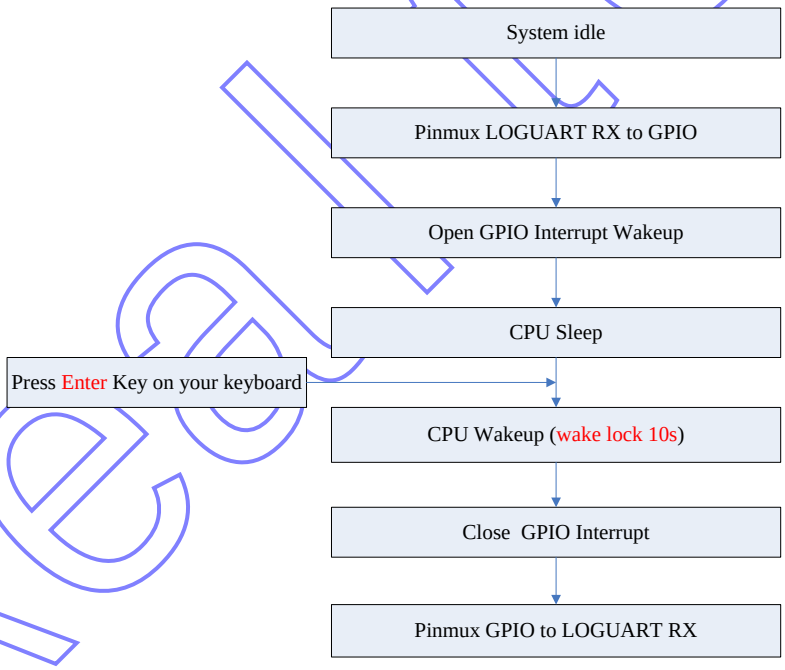
Realtek

1. Summary

UART Index	*CPU Sleep RX*	*Wakeup Solution*
UART0	Y	UART RX Wakeup
UART1	Y	UART RX Wakeup
UART2_LOG	N	GPIO Interrupt Wakeup

2. LOGUART wakeup

LOGUART is not in wakeup sources of sleep mode. To support wake from LOGUART, we can treat LOGUART signal as GPIO interrupt signal and then wake system by GPIO interrupt.
SDK will auto switch LOGUART pin to GPIO pin and open GPIO wakeup function when enter sleep mode, and switch back to LOGUART pin when resume, so you can wakeup sleep mode when press "Enter" key two times.



SDK set 10 seconds wake_lock for LOGUART wakeup, so system will enter sleep mode again after 10 seconds.

Realtek

3. Normal UART wakeup

UART0 and UART1 can receive data under sleep mode. You should open UART wakeup event in SDK wevent_config, then UART will wakeup system when receive data.

```
const PWRCFG_TypeDef wevent_config[]=
{
//   Module                                Status
     {BIT_SYSON_WEVT_GPIO_DSTBY_MSK,       ON},    /* dstandby:  wakepin 0~3 wakeup */
     {BIT_SYSON_WEVT_A33_AND_A33GPIO_MSK,  OFF},   /* dsleep:    REGU A33 Timer & A33 wakepin wakeup*/
     {BIT_SYSON_WEVT_ADC_MSK,              OFF},   /* sleep:     ADC Wakeup */
     {BIT_SYSON_WEVT_SDIO_MSK,             OFF},   /* sleep:     SDIO Wakeup */
     {BIT_SYSON_WEVT_RTC_MSK,              OFF},   /* dstandby:  RTC Wakeup */
     {BIT_SYSON_WEVT_UART1_MSK,            OFF},   /* sleep:     UART1 Wakeup */
     {BIT_SYSON_WEVT_UART0_MSK,            OFF},   /* sleep:     UART0 Wakeup */
     {BIT_SYSON_WEVT_I2C1_MSK,             OFF},   /* sleep:     I2C1 Wakeup */
     {BIT_SYSON_WEVT_I2C0_MSK,             OFF},   /* sleep:     I2C0 Wakeup */
     {BIT_SYSON_WEVT_WLAN_MSK,             ON},    /* sleep:     WLAN Wakeup */
     {BIT_SYSON_WEVT_I2C1_ADDRMATCH_MSK,   OFF},   /* sleep:     ADC Wakeup */
     {BIT_SYSON_WEVT_I2C0_ADDRMATCH_MSK,   OFF},   /* sleep:     I2C1 Slave RX address Wakeup */
     {BIT_SYSON_WEVT_USB_MSK,              OFF},   /* sleep:     I2C0 Slave RX address Wakeup */
     {BIT_SYSON_WEVT_GPIO_MSK,             ON},    /* sleep:     USB Wakeup */
     {BIT_SYSON_WEVT_CHIP_EN_MSK,          OFF},   /* sleep:     ChipEN Wakeup */
     {BIT_SYSON_WEVT_OVER_CURRENT_MSK,     OFF},   /* sleep:     REGU OVER_CURRENT Wakeup */
     {BIT_SYSON_WEVT_GTIM_MSK,             ON},    /* sleep:     Gtimer 4/5 Wakeup */
     {BIT_SYSON_WEVT_SYSTIM_MSK,           OFF},   /* dstandby:  SYS Timer(ANA Timer) Wakeup */

     {0xFFFFFFFF,                          OFF},   /* Table end */
};
```

There are two solutions used for UART RX when CPU sleeps:

3.1. High Speed Mode

If you use UART under high speed mode, you should configure SDK like following:

```
const PWRCFG_TypeDef sleep_pwrmgt_config[]=
{
//   Module                                Status
     {BIT_SYSON_PMOPT_SLP_XTAL_EN,         ON},    /* XTAL: 2.2mA */
     {BIT_SYSON_PMOPT_SNZ_XTAL_EN,         OFF},   /* ADC power save use it */
     {BIT_SYSON_PMOPT_SNZ_SYSPLL_EN,       OFF},   /* ADC power save use it */
     {0xFFFFFFFF,                          OFF},   /* Table end */
};

/* if X can wakeup dsleep, it can wakeup dstandby & sleep */
/* if X can wakeup dstandby, it can wakeup sleep */
const PWRCFG_TypeDef sleep_wevent_config[]=
{
//   Module                                Status
     {BIT_SYSON_WEVT_GPIO_DSTBY_MSK,       ON},    /* dstandby:  wakepin 0~3 wakeup */
     {BIT_SYSON_WEVT_A33_AND_A33GPIO_MSK,  ON},    /* dsleep:    REGU A33 Timer(1K low precision timer) & A33 wakepin wakeup*/
     {BIT_SYSON_WEVT_ADC_MSK,              OFF},   /* sleep:     ADC Wakeup */
     {BIT_SYSON_WEVT_SDIO_MSK,             OFF},   /* sleep:     SDIO Wakeup */
     {BIT_SYSON_WEVT_RTC_MSK,              ON},    /* dstandby:  RTC Wakeup */
     {BIT_SYSON_WEVT_UART1_MSK,            ON},    /* sleep:     UART1 Wakeup */
     {BIT_SYSON_WEVT_UART0_MSK,            ON},    /* sleep:     UART0 Wakeup */
     {BIT_SYSON_WEVT_I2C1_MSK,             OFF},   /* sleep:     I2C1 Wakeup */
     {BIT_SYSON_WEVT_I2C0_MSK,             OFF},   /* sleep:     I2C0 Wakeup */
     {BIT_SYSON_WEVT_WLAN_MSK,             ON},    /* sleep:     WLAN Wakeup */
     {BIT_SYSON_WEVT_I2C1_ADDRMATCH_MSK,   OFF},   /* sleep:     I2C1 Slave RX address Wakeup */
     {BIT_SYSON_WEVT_I2C0_ADDRMATCH_MSK,   OFF},   /* sleep:     I2C0 Slave RX address Wakeup */
     {BIT_SYSON_WEVT_USB_MSK,              OFF},   /* sleep:     USB Wakeup */
     {BIT_SYSON_WEVT_GPIO_MSK,             ON},    /* sleep:     GPIO Wakeup */
     {BIT_SYSON_WEVT_OVER_CURRENT_MSK,     OFF},   /* sleep:     REGU OVER_CURRENT Wakeup */
     {BIT_SYSON_WEVT_SYSTIM_MSK,           ON},    /* dstandby:  250K SYS Timer(ANA Timer) Wakeup */

     {0xFFFFFFFF,                          OFF},   /* Table end */
};

const UARTCFG_TypeDef uart_config[2]=
{
    /* UART0 */
    {
        .LOW_POWER_RX_ENABLE = DISABLE, /*Enable low power RX*/
    },
    /* UART1 */
    {
        .LOW_POWER_RX_ENABLE = DISABLE,
    },
};
```

3.2. Low Power Mode

If you use UART under low power mode, you should configure SDK like following:

```
const PWRCFG_TypeDef sleep_pwrmgt_config[]=
{
//   Module                                 Status
    {BIT_SYSON_PMOPT_SLP_XTAL_EN,          OFF},    /* XTAL: 2.2mA */
    {BIT_SYSON_PMOPT_SNZ_XTAL_EN,          OFF},    /* ADC power save use it */
    {BIT_SYSON_PMOPT_SNZ_SYSPLL_EN,        OFF},    /* ADC power save use it */
    {0xFFFFFFFF,                           OFF},    /* Table end */
};

/* if X can wakeup dsleep, it can wakeup dstandby & sleep */
/* if X can wakeup dstandby, it can wakeup sleep */
const PWRCFG_TypeDef sleep_wevent_config[]=
{
//   Module                                 Status
    {BIT_SYSON_WEVT_GPIO_DSTBY_MSK,        ON},     /* dstandby:  wakepin 0~3 wakeup */
    {BIT_SYSON_WEVT_A33_AND_A33GPIO_MSK,   ON},     /* dsleep:    REGU A33 Timer(1K low precision timer) & A33 wakepin wakeup*/
    {BIT_SYSON_WEVT_ADC_MSK,               OFF},    /* sleep:     ADC Wakeup */
    {BIT_SYSON_WEVT_SDIO_MSK,              OFF},    /* sleep:     SDIO Wakeup */
    {BIT_SYSON_WEVT_RTC_MSK,               ON},     /* dstandby:  RTC Wakeup */
    {BIT_SYSON_WEVT_UART1_MSK,             ON},     /* sleep:     UART1 Wakeup */
    {BIT_SYSON_WEVT_UART0_MSK,             ON},     /* sleep:     UART0 Wakeup */
    {BIT_SYSON_WEVT_I2C1_MSK,              OFF},    /* sleep:     I2C1 Wakeup */
    {BIT_SYSON_WEVT_I2C0_MSK,              OFF},    /* sleep:     I2C0 Wakeup */
    {BIT_SYSON_WEVT_WLAN_MSK,              ON},     /* sleep:     WLAN Wakeup */
    {BIT_SYSON_WEVT_I2C1_ADDRMATCH_MSK,    OFF},    /* sleep:     I2C1 Slave RX address Wakeup */
    {BIT_SYSON_WEVT_I2C0_ADDRMATCH_MSK,    OFF},    /* sleep:     I2C0 Slave RX address Wakeup */
    {BIT_SYSON_WEVT_USB_MSK,               OFF},    /* sleep:     USB Wakeup */
    {BIT_SYSON_WEVT_GPIO_MSK,              ON},     /* sleep:     GPIO Wakeup */
    {BIT_SYSON_WEVT_OVER_CURRENT_MSK,      OFF},    /* sleep:     REGU OVER_CURRENT Wakeup */
    {BIT_SYSON_WEVT_SYSTIM_MSK,            ON},     /* dstandby:  250K SYS Timer(ANA Timer) Wakeup */

    {0xFFFFFFFF,                           OFF},    /* Table end */
};

const UARTCFG_TypeDef uart_config[2]=
{
    /* UART0 */
    {
        .LOW_POWER_RX_ENABLE = ENABLE, /*Enable low power RX*/
    },
    /* UART1 */
    {
        .LOW_POWER_RX_ENABLE = DISABLE,
    },
};
```

Rea

4. Example

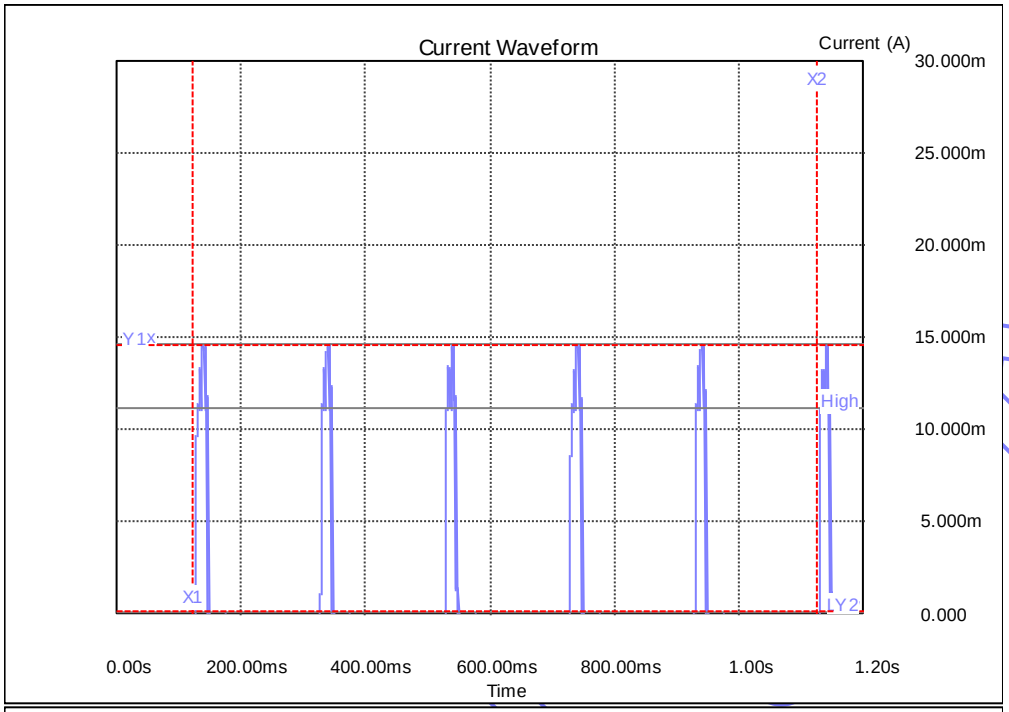
Step1: Register suspend/resume/late_resume callback functions
Step2: Set system active for 5000ms
Step3: Release os wakelock
Step4: After 5000ms system will enter sleep, and “uart_suspend” will print
Step5: Uart Rx wakeup system and “uart_resume” will print
Step6: “uart_late_resume” will print
Step7: After 5000ms system will enter sleep again

```
u32 uart_suspend(u32 expected_idle_time, void *param)
{
        DBG_8195A("uart_suspend \n");
}

u32 uart_resume(u32 expected_idle_time, void *param)
{
        DBG_8195A("uart_resume \n");
}

u32 uart_lateresume(u32 expected_idle_time, void *param)
{
        DBG_8195A("uart_late_resume \n");

        pmu_set_sysactive_time(PMU_LOGUART_DEVICE, 5000);
}
```

```
void psm_sleep_uart(void)
{
    // mbed uart test
    serial_init(&sobj_g,UART_TX,UART_RX);
    serial_baud(&sobj_g,38400);
    serial_format(&sobj_g, 8, ParityNone, 1);

    uart_send_string(&sobj_g, "UART IRQ API Demo...\r\n");
    uart_send_string(&sobj_g, "Enter sleep!!\n");
    uart_send_string(&sobj_g, "\r\n8195a$");

    serial_irq_handler(&sobj_g, uart_irq, (uint32_t)&sobj_g);
    serial_irq_set(&sobj_g, RxIrq, 1);
    serial_irq_set(&sobj_g, TxIrq, 1);

    pmu_sysactive_timer_init();

    pmu_register_sleep_callback(PMU_UART0_DEVICE, uart_suspend, NULL, uart_resume, NULL);
    pmu_register_delay_callback(PMU_UART0_DEVICE, uart_lateresume, NULL);

    pmu_set_sysactive_time(PMU_LOGUART_DEVICE, 5000);
    pmu_release_wakelock(PMU_OS);

    vTaskDelete(NULL);
}
```

5. Power Consumption

UART receive data and wakeup CPU every 200ms.



Calculated Measurements (3374Hz sample rate)

Dc 1.1978mA	Rms 3.6333mA	X1 123.7500ms	Y1 14.6101m
Low 145.3806uA	Min -35.6241uA	X2 1.1231s	Y2 155.9792u
High 11.2563mA	Max 14.6531mA	dX 0.9994s	dY -14.4541m

Realtek