文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 RTC 应用指南

目录

1 概述

本文介绍了在APT32F110x中RTC模块的应用。

2. 适用的硬件

该例程使用于 APT32F110x 系列学习板

3. 应用方案代码说明

基于 APT32F110x 完整的 CSI 库文件系统，进行 RTC 配置

3.1 RTC 基本特性

RTC 一旦初始化成功并开始工作，任何复位信号均不能影响其工作，除非重新上电

RTC 在所有低功耗模式下均可独立运行，并支持系统唤醒。

必须保证 PCLK 的频率为所选 RTC 时钟源频率的 2 倍以上

- 支持万年历，自动闰年判定。计时器包括小时，分钟，秒。BCD 格式计数。
- 十二小时或者二十四小时制。支持星期判断。
- 多时钟源，包括外部晶振，内部低速振荡器和内部主震荡器。
- 支持低功耗唤醒
- 两个可编程闹钟
- 支持周期事件触发

3.2 功能框图

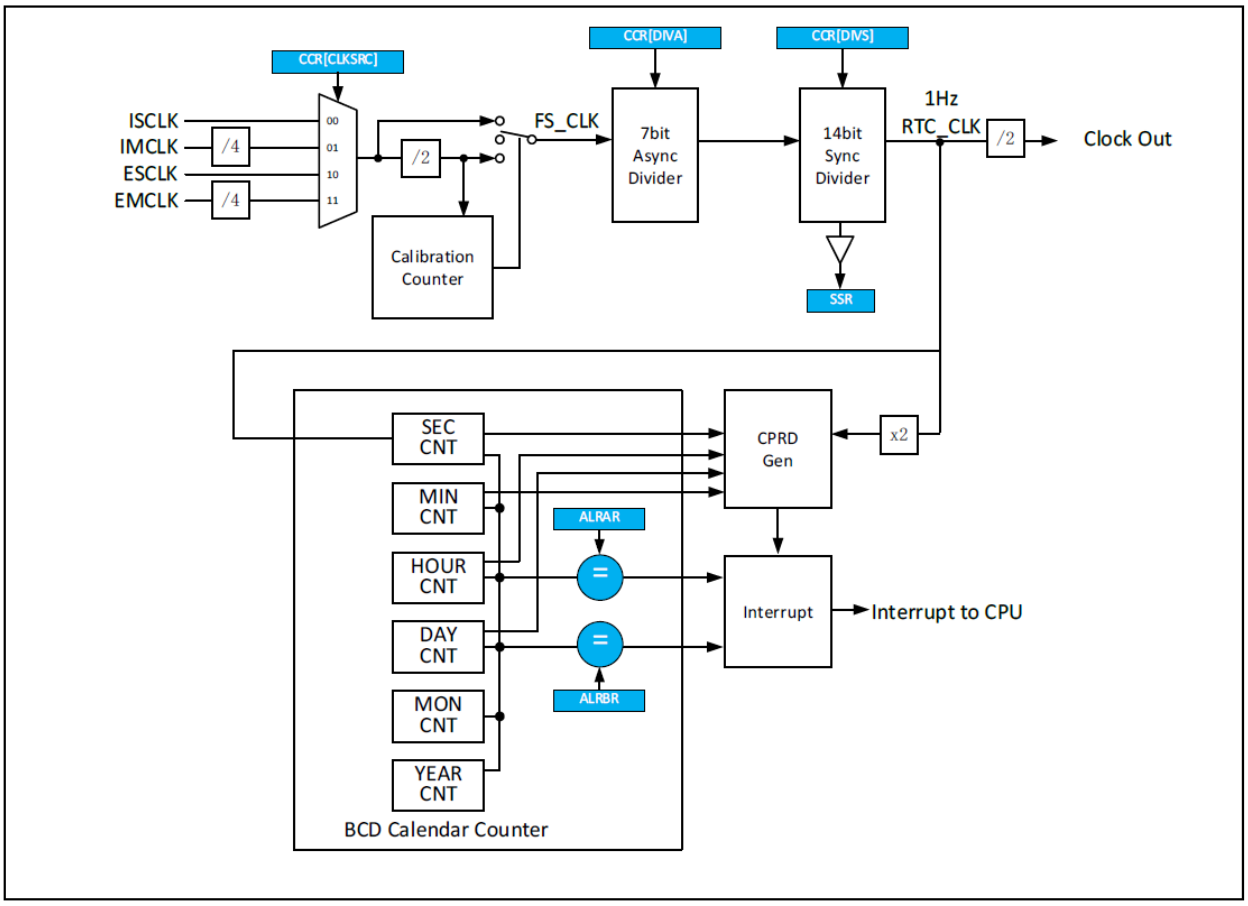

图 3.2.1RTC 框图

3.3 时钟控制

周期事件在 RTC_CLK 为 2Hz 的条件下，支持如下周期事件：每 0.5 秒、每 1 秒、每分钟、每小时、每天、每个月。

RTC_CLK 配置为 2Hz，两级分频器参数如下：7 位的 DIVA 和 15 位的 DIVS

CLKSRC	DIVA	Fasync (Hz)	DIVS
ISOSC：27KHz	49	540	269
IMOSC/4：5.556MHz/4	124	11112	5555
IMOSC/4：4.194MHz/4	124	8388	4193
IMOSC/4：2.097MHz/4	124	4194	2096
IMOSC/4：131.072KHz/4	3	8192	4095
EMOSC/4：(eg：12MHz/4)	99	16000	14999
ESOSC：32.768KHz	3	8192	4095

图 3.3.1 分频参数

3.4 RTC 初始化

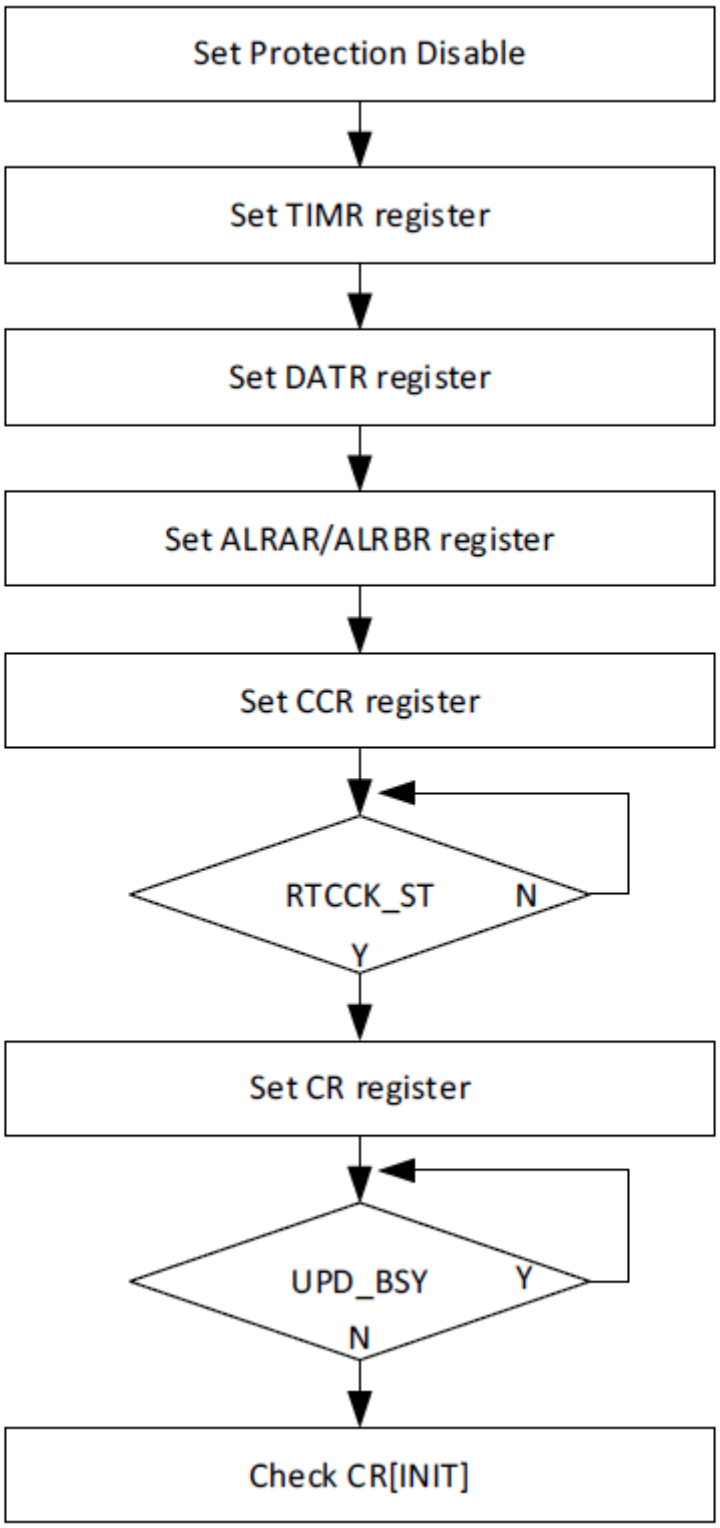

图 3.4.1RTC 初始化流程

3.5 闹钟设置

RTC 支持两路闹钟设置，当 RTC 中的 BCD 计数器的当前计数值满足 ALRAR 或者 ALRBR 的设置值，且 CR 寄存器中的 ALRAE 或者 ALRBE 同时有效，则 RTC 会产生一个闹钟事件。该闹钟事件可以通过系统中断将系统从低功耗模式唤醒，并执行相应中断处理。

3.6 周期事件触发

RTC 支持周期性事件触发。周期事件在 RTC_CLK 为 2Hz 的条件下，支持如下周期事件：每 0.5 秒、每 1 秒、每分钟、每小时、每天、每个月。

3.7 同步事件

RTC 支持同步事件触发输出到 ETCB 模块，通过 EVTRG 寄存器可以配置最大两个同步触发事件的输出。任意 RTC 的中断均支持同步输出。

同步输出接口支持事件计数和软件触发。只有计数值满足 PRD 条件时，才会向 ETCB 输出同步信号。

- **软件配置：**

可在 system.c 文件中 rtc_config 函数进行初始化的配置。

本例程实现功能 RTC 的起始时间设置为 AM 10:00:00，闹钟 A 定时为 10：00：30，触发 BT0 从 PA1.9 输出频率为 1KHZ，占空比为 50%的 PWM 信号，闹钟 B 定时为 10：01：00，同步触发 BT1 从 PA1.10 输出频率为 2KHZ，占空比为 50%的 PWM 信号。

```
//-------------------------------------------------------------
void rtc_config(void)
{
    csi_rtc_time_t tRtcTime;
    csi_rtc_time_t tAlmTime;
    csi_rtc_time_t tBlmTime;
    csi_rtc_config_t tRtcConfig;
    csi_bt_pwm_config_t tPwmCfg;
    csi_etb_config_t tEtbConfig1,tEtbConfig2;
    volatile uint8_t ch;
```

```
//------------时钟源配置----------------------------
    //csi_pin_set_mux(PA03, PA03_OSC_XI);
    //csi_pin_set_mux(PA04, PA04_OSC_XO);

    csi_pin_set_mux(PA01,PA01_OSC_SXI);
    csi_pin_set_mux(PA02,PA02_OSC_SXO);

    tRtcConfig.byClkSrc = RTC_CLKSRC_ESOSC;
    tRtcConfig.byFmt = RTC_12FMT;
    csi_rtc_init(RTC, &tRtcConfig);
//------------时钟配置-------------------------------------------------------------

    tRtcTime.iYear = 22;
    tRtcTime.iMon  = 8;
    tRtcTime.iMday = 10;
    tRtcTime.iWday = 4;
    tRtcTime.iHour = 10;
    tRtcTime.iMin  = 0;
    tRtcTime.iSec  = 0;
    tRtcTime.iPm   = RTC_AM;
    csi_rtc_set_time(RTC, &tRtcTime);
    csi_rtc_start(RTC);
//--------------周期事件定时-----------------------------------------------------
    csi_rtc_start_as_timer(RTC, RTC_TIMER_1MIN);

//-------  闹铃 A 设置-----------------------------------------------
    tAlmTime.iWday = EVERYDAY;
    tAlmTime.iHour = 10;
    tAlmTime.iMin  = 0;
    tAlmTime.iSec  = 30;
    tAlmTime.iPm   = RTC_AM;
    csi_rtc_set_alarm(RTC, RTC_ALMA,&tAlmTime);

//--------闹铃 B 设置-----------------------------------------------
    tBlmTime.iWday =  SATURDAYS | SUNDAYS;
    tBlmTime.iWday =  EVERYDAY;
    tBlmTime.iHour = 10;
    tBlmTime.iMin  = 1;
    tBlmTime.iSec  = 0;
    tBlmTime.iPm   = RTC_AM;
    csi_rtc_set_alarm(RTC,RTC_ALMB,&tBlmTime);

//----------RTC 触发输出配置----------------------------
    csi_rtc_set_evtrg(RTC, RTC_TRGSEL0, RTC_TRGOUT_ALRMA, 0);
```

```
    csi_rtc_set_evtrg(RTC, RTC_TRGSEL1, RTC_TRGOUT_ALRMB, 0);


//---------- 触发目标事件 BT0 配置-------------------------------
    csi_pin_set_mux(PA19, PA19_BT0_OUT);
    tPwmCfg.byIdleLevel = BT_PWM_IDLE_HIGH;
    tPwmCfg.byStartLevel= BT_PWM_START_HIGH;
    tPwmCfg.byDutyCycle = 50;
    tPwmCfg.wFreq               = 1000;
    tPwmCfg.byInt               = BT_INTSRC_NONE;
    csi_bt_pwm_init(BT0, &tPwmCfg);
    csi_bt_set_sync(BT0, BT_TRG_SYNCIN0, BT_TRG_CONTINU, ENABLE);
//------------ 触发目标事件 BT1 配置 ----------------------
    csi_pin_set_mux(PA110, PA110_BT1_OUT);
    tPwmCfg.byIdleLevel = BT_PWM_IDLE_HIGH;
    tPwmCfg.byStartLevel= BT_PWM_START_HIGH;
    tPwmCfg.byDutyCycle = 50;
    tPwmCfg.wFreq               = 2000;
    tPwmCfg.byInt               = BT_INTSRC_NONE;
    csi_bt_pwm_init(BT1, &tPwmCfg);
    csi_bt_set_sync(BT1, BT_TRG_SYNCIN0, BT_TRG_CONTINU, ENABLE);


//------------ ETCB0 配置-------------------------------
    tEtbConfig1.byChType = ETB_ONE_TRG_ONE;
    tEtbConfig1.bySrcIp   = ETB_RTC_TRGOUT0 ;
    tEtbConfig1.bySrcIp1 = 0xff;
    tEtbConfig1.bySrcIp2 = 0xff;
    tEtbConfig1.byDstIp   = ETB_BT0_SYNCIN0;
    tEtbConfig1.byDstIp1 = 0xff;
    tEtbConfig1.byDstIp2 = 0xff;
    tEtbConfig1.byTrgMode = ETB_HARDWARE_TRG;
    csi_etb_init();
    ch = csi_etb_ch_alloc(tEtbConfig1.byChType);
    csi_etb_ch_config(ch,&tEtbConfig1);


//---------------ETCB1 配置 ------------------------------
    tEtbConfig2.byChType = ETB_ONE_TRG_ONE;
    tEtbConfig2.bySrcIp   = ETB_RTC_TRGOUT1 ;
    tEtbConfig2.bySrcIp1 = 0xff;
    tEtbConfig2.bySrcIp2 = 0xff;
    tEtbConfig2.byDstIp   =   5;
    tEtbConfig2.byDstIp1 = 0xff;
    tEtbConfig2.byDstIp2 = 0xff;
    tEtbConfig2.byTrgMode = ETB_HARDWARE_TRG;
    csi_etb_init();
```

```
    ch = csi_etb_ch_alloc(tEtbConfig2.byChType);
    csi_etb_ch_config(ch,&tEtbConfig2);
}
```

- **代码说明：**

GPIO 部分

1) csi_pin_set_mux(PA01,PA01_OSC_SXI);

设置特定的 GPIO 口为外部晶振输入输出口，分外部主晶振和外部辅晶振

时间设置部分

2) csi_rtc_init(RTC, &tRtcConfig);

根据结构体变量 **tRtcConfig** 初始化 RTC

tRtcConfig.byClkSrc---RTC 模块的时钟源选择，当选择 EMOSC 或者 EIOSC 时，需配置 GPIO 口部分

tRtcConfig.byFmt---RTC 时间的格式，12 小时 AM,PM 或者 24 小时制

3) csi_rtc_set_time(RTC, &tRtcTime);

根据结构体变量 **tRtcTime** 设置 RTC 的初始时间，包括年，月，日，星期，小时，分钟，秒

tRtcTime.iYear ---年 0~99

tRtcTime.iMon---月 1~12

tRtcTime.iMday ---日 1~31

tRtcTime.iWday ---星期 1-7

tRtcTime.iHour ---小时 12 小时 0-11, 24 小时 0~23

tRtcTime.iMin ---分钟

tRtcTime.iSec ---秒

tRtcTime.iPm ---12 小时制 AM PM

4) **csi_rtc_start(RTC);**

启动 RTC

周期事件部分

5) **csi_rtc_start_as_timer(RTC, RTC_TIMER_1MIN);**

定时功能设置，启用此功能时，RTC_CLK 需配置为 2HZ，否则定时时间不准，也可另外计算。

RTC_TIMER_1MIN---定时时间选择

闹钟部分

6) **csi_rtc_set_alarm(RTC, RTC_ALMA,&tAlmTime);**

csi_rtc_set_alarm(RTC,RTC_ALMB,&tBlmTime);

根据结构体 tAlmTime 和 tBlmTime 配置闹钟 A 和闹钟 B 时间

同步触发部分

7) **csi_rtc_set_evtrg(RTC, RTC_TRGSEL0, RTC_TRGOUT_ALRMA, 0);**

同步触发输出配置

RTC_TRGSEL0---触发通道选择

RTC_TRGOUT_ALRMA---触发源选择：1.闹钟 A 2.闹钟 B 3.闹钟 A 或 B 4 定时功能中断

0---触发周期设定 0~15 次

ETCB 部分

见 ETCB 应用指南

BT 部分

见 BT 应用指南

在 main.c 和中断函数中，

```
int main()
{
    uint32_t wSec=0;
    csi_rtc_time_t tRtcTimeRdbk;
    mdelay(100);
    system_init();                                  //时钟配置
    board_init();                                   //配置 UART2 作为外部调试口
  csi_pin_set_mux(PA00, PA00_RTC_ALM);
    my_printf("Hello World-~~~~~~~\n");             //print message
    while(1)
    {
        csi_rtc_get_time(RTC,   &tRtcTimeRdbk);

        if(wSec != tRtcTimeRdbk.iSec )
        {
            wSec = tRtcTimeRdbk.iSec;
            my_printf("20%d.%d.%d %d   %d:%d:%d pm= %d\n",tRtcTimeRdbk.iYear,tRtcTimeRdbk.iMon,
                    tRtcTimeRdbk.iMday, tRtcTimeRdbk.iWday,tRtcTimeRdbk.iHour, tRtcTimeRdbk.iMin,
                    tRtcTimeRdbk.iSec, tRtcTimeRdbk.iPm);
        }
        if(csi_rtc_get_int_msg(RTC_INTSRC_ALMA,1))
            my_printf("RTC AlarmA has triggered!!\n");
        if(csi_rtc_get_int_msg(RTC_INTSRC_ALMB,1))
            my_printf("RTC AlarmB has triggered!!\n");
        if(csi_rtc_get_int_msg(RTC_INTSRC_CPRD,1))
            my_printf("RTC CPRD has triggered!!\n");

    }
```

```
    return 0;
}



__attribute__((weak)) void rtc_irqhandler(csp_rtc_t *ptRtcBase)
{
    if(((csp_rtc_get_int_st(ptRtcBase) & RTC_INTSRC_ALMA))==RTC_INTSRC_ALMA)
    {
        s_hwRtcMsg |= RTC_INTSRC_ALMA;
        csp_rtc_int_clr(ptRtcBase,RTC_INTSRC_ALMA);
    }
    if(((csp_rtc_get_int_st(ptRtcBase) & RTC_INTSRC_ALMB))==RTC_INTSRC_ALMB)
    {
        s_hwRtcMsg |= RTC_INTSRC_ALMB;
        csp_rtc_int_clr(ptRtcBase,RTC_INTSRC_ALMB);
    }
    if(((csp_rtc_get_int_st(ptRtcBase) & RTC_INTSRC_CPRD))==RTC_INTSRC_CPRD)
    {
        s_hwRtcMsg |= RTC_INTSRC_CPRD;
        csp_rtc_int_clr(ptRtcBase,RTC_INTSRC_CPRD);
    }
    if(((csp_rtc_get_int_st(ptRtcBase)&RTC_INTSRC_TRGEV0))==RTC_INTSRC_TRGEV0)
    {
        s_hwRtcMsg |= RTC_INTSRC_TRGEV0;
        csp_rtc_int_clr(ptRtcBase,RTC_INTSRC_TRGEV0);
    }
    if(((csp_rtc_get_int_st(ptRtcBase)&RTC_INTSRC_TRGEV1))==RTC_INTSRC_TRGEV1)
    {
        s_hwRtcMsg |= RTC_INTSRC_TRGEV1;
```

```
            csp_rtc_int_clr(ptRtcBase,RTC_INTSRC_TRGEV1);

        }

}
```

- **调试代码说明**

1) csi_rtc_get_time(RTC, &tRtcTimeRdbk);

回读 RTC 中的时间，放在结构体变量 **tRtcTimeRdbk 中，**主循环不停打印时间信息。

2) csi_rtc_get_int_msg(RTC_INTSRC_ALMA,1)

判断中断源是否被触发过，打印出来信息。

3) csi_pin_set_mux(PA00, PA00_RTC_ALM);

设置 PA00 为闹钟脉冲的输出口

- **验证方法：**

借助串口调试工具和万用表，根据串口打印信息和 GPIO 口的电压，50%占空比输出，是 VCC 的一半电压值，略低于 2.5V。PA1.9 在 10：00：30 时输出 2.5V，PA1.10 在 10：01：00 时输出 2.5V。

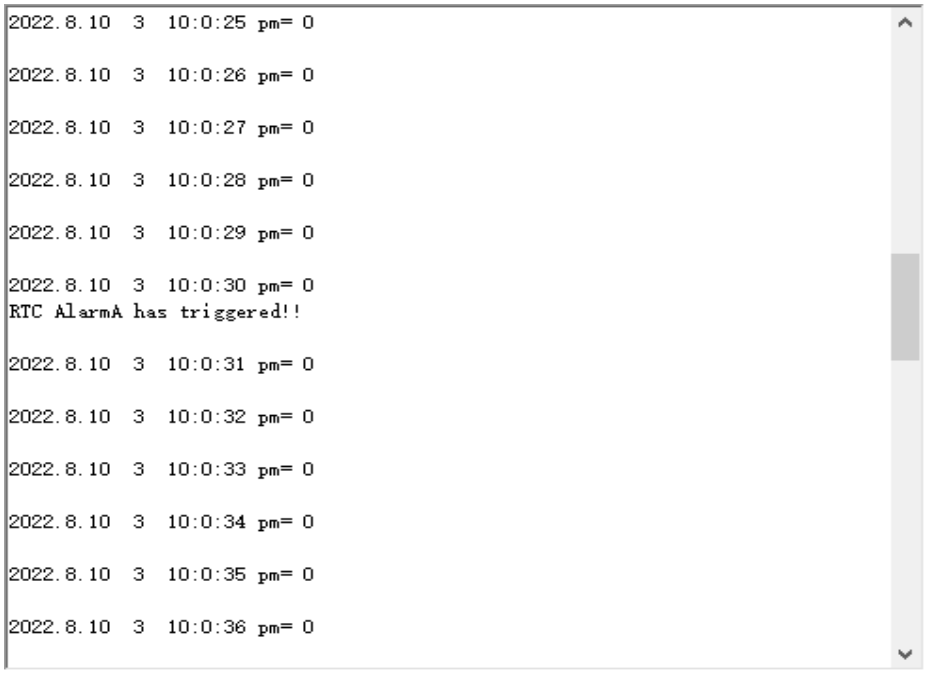

图 3.7.1RTCAlarmA 事件

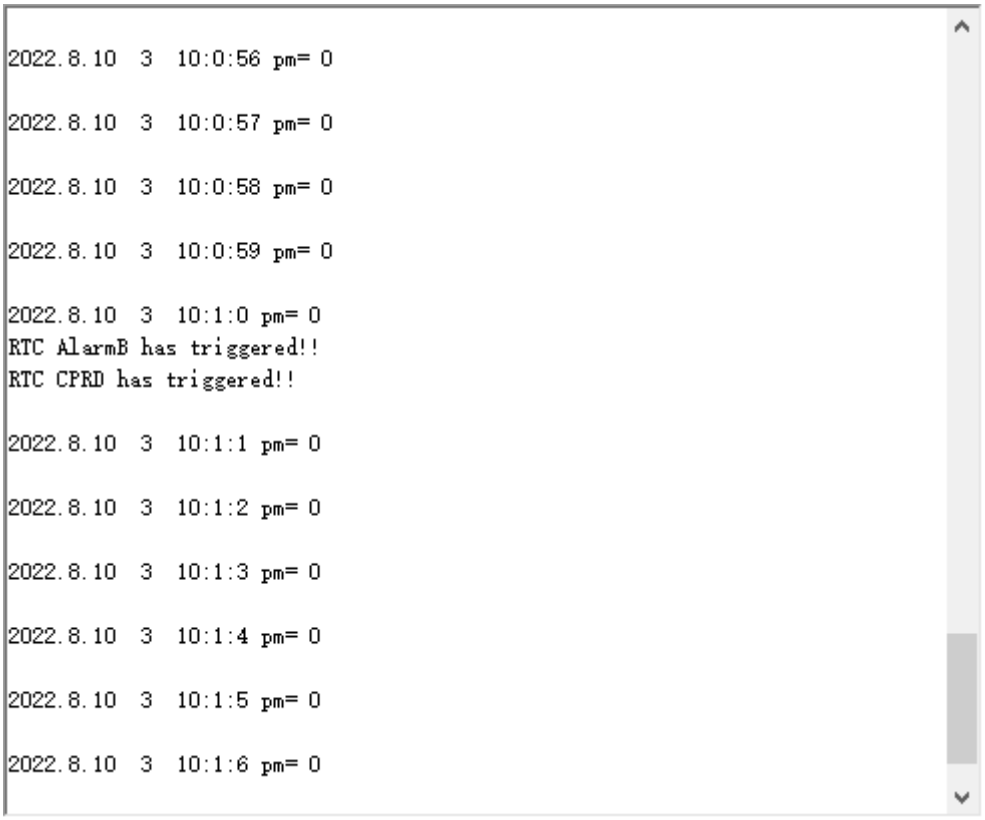

图 3.7.2RTCAlarmB 事件

4. 程序下载和运行

1．将目标板与仿真器连接，分别为 VDD SCLK SWIO GND

2. 程序编译后仿真运行或下载进芯片

3. 通过示波器查看 GPIO 口输出波形或通过万用表量电压