**mini2440裸机试炼之—RTC闹钟中断，节拍中断**

[环境搭建 2](#_Toc392019618)

[硬件环境： 2](#_Toc392019619)

[软件环境： 2](#_Toc392019620)

[RTC实现功能 3](#_Toc392019621)

[RTC概述 3](#_Toc392019622)

[RTC实时时钟 4](#_Toc392019623)

[节拍中断使能 4](#_Toc392019624)

[闹钟中断使能 5](#_Toc392019625)

[中断函数 5](#_Toc392019626)

[调试总结 7](#_Toc392019627)

[RTC闹钟中断，节拍中断代码 9](#_Toc392019628)

[Main函数代码 9](#_Toc392019629)

[RTC\_Display\_TICK\_ALM函数代码 10](#_Toc392019630)

[输出截图 14](#_Toc392019631)

**By：梁惠涌**

**2014年7月1日**

# 环境搭建

# 硬件环境：J-link v8、mini2440、J-link转接板、串口转USB线

# 软件环境：windows7（32位）、开发板uboot（NandFlash）、J-link驱动（J-Link ARM V4.10i）、SecureCRT、ADS1.2

**其中ADS里的AXD设置：**加载JlinkRDI.dll+Options－>Configure Interface...，在Session File一页中选择“Run Configuration Script”，将该name.txt文本文件作为一个脚本加进来，确定。

|  |
| --- |
| Setmem 0x53000000 0x00000000 32  Setmem 0x4A000008 0xFFFFFFFF 32  Setmem 0x4A00001C 0x000007FF 32  Setmem 0x53000000 0x00000000 32  Setmem 0x56000050 0x000055AA 32  Setmem 0x4C000014 0x00000007 32  Setmem 0x4C000000 0x00FFFFFF 32  Setmem 0x4C000004 0x00061012 32  Setmem 0x4C000008 0x00040042 32  Setmem 0x48000000 0x22111120 32  Setmem 0x48000004 0x00002F50 32  Setmem 0x48000008 0x00000700 32  Setmem 0x4800000C 0x00000700 32  Setmem 0x48000010 0x00000700 32  Setmem 0x48000014 0x00000700 32  Setmem 0x48000018 0x0007FFFC 32  Setmem 0x4800001C 0x00018005 32  Setmem 0x48000020 0x00018005 32  Setmem 0x48000024 0x008E0459 32  Setmem 0x48000028 0x00000032 32  Setmem 0x4800002C 0x00000030 32  Setmem 0x48000030 0x00000030 32 |

name.txt内容

# RTC实现功能

RTC开节拍中断、闹钟中断。

节拍中断——串口输出时间 XXXX年XX月XX日XX时XX分XX秒 和 LED闪亮

闹钟中断——beep声 和 LED亮 5秒

# RTC概述

实时时钟（RTC）单元可以在当系统电源关闭后通过备用电池工作。RTC可以通过使用STRB/LDRB ARM操

作发送8位二-十进制交换码（BCD）值数据给CPU。这些数据包括年、月、日、星期、时、分和秒的时间信息。

RTC单元工作在外部32.768kHz晶振并且可以执行闹钟功能。

特性

– BCD数：年、月、日、星期、时、分和秒

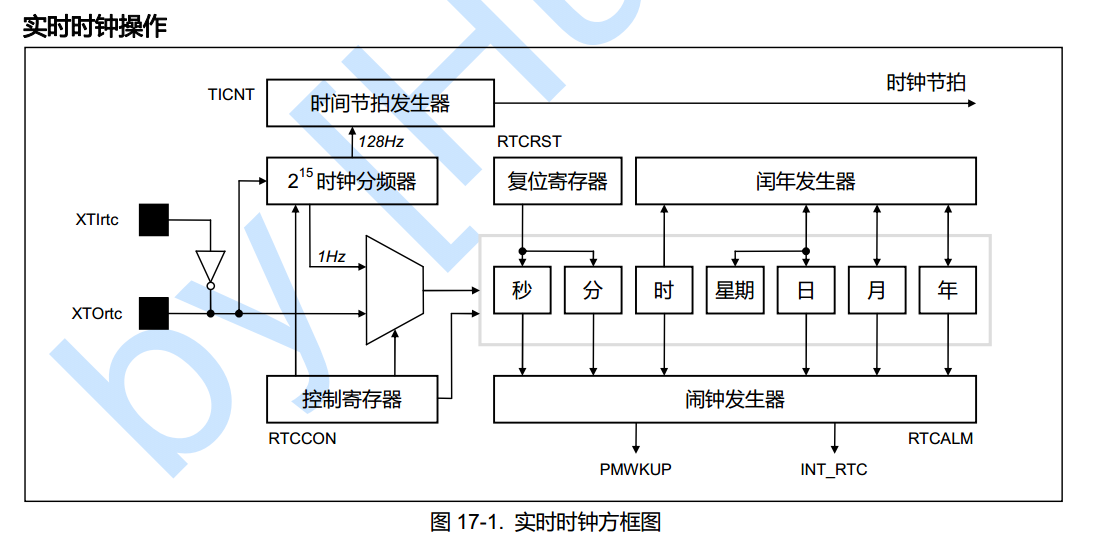
– 闰年发生器

– 闹钟功能：闹钟中断或从掉电模式唤醒

– 已解决的2000年问题

– 独立电源引脚（RTCVDD）

– 支持RTOS内核时钟节拍（tick）的毫秒节拍时间中断



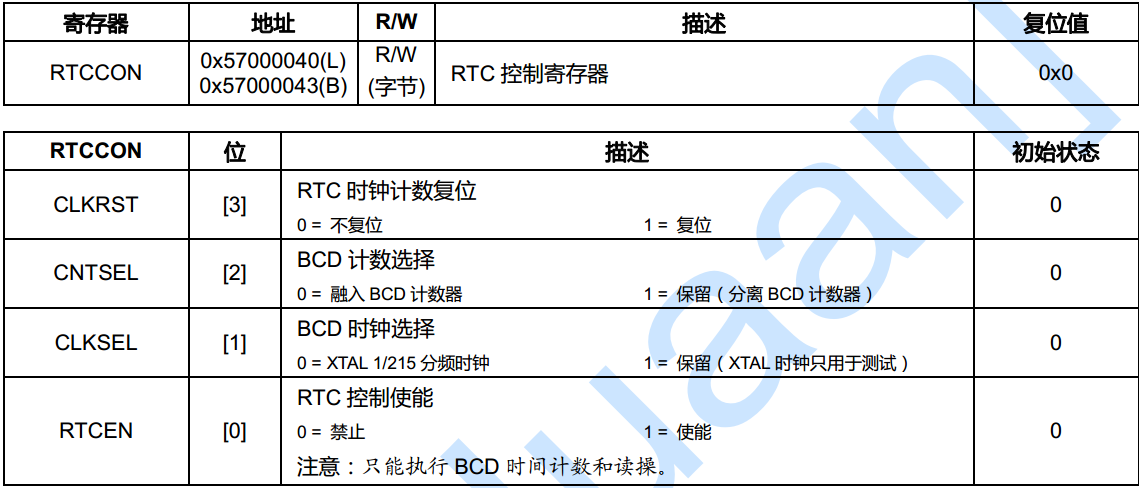
由图可知由于RTC秒有RTCRST复位寄存器，当RTC寄存器被赋值后，秒寄存器数值自动1s（1s=（1/1HZ），1HZ为2^15时钟分频器产生的1HZ）加一,秒寄存器到60时被复位寄存器置零，分寄存器加一，以此类推。

## RTC实时时钟

实时时钟的值是存放在BCD寄存器里（BCD码），BCD码是4位二进制码表示1位十进制数。

实时时钟初始化（赋值） RTCCON寄存器的使用

读写BCD 寄存器的时候只需要将RTCCON寄存器RTCEN（读写）使能，其他为初始值



实时时钟初始化代码

开BCD读写使能

|  |
| --- |
| rRTCCON **|=**0x01**;** //RTCCON只控制BCD寄存器，ALM数据寄存器就不需要读写使能了 |

BCD寄存器赋值。

|  |
| --- |
| rBCDSEC **=**0x0**;** //14年6月12日1点1分0秒 星期4  rBCDMIN **=**0x01**;**  rBCDHOUR **=**0x01**;**  rBCDDAY **=**0x4**;** //星期  rBCDDATE **=**0x12**;**  rBCDMON **=**0x6**;**  rBCDYEAR **=**0x14**;** |

BCD寄存器赋值完成后必须将读写关闭，防止数据异常

|  |
| --- |
| rRTCCON **&=~**0x01**;** |

之后BCD寄存器里的值开始计时

## 节拍中断使能

始终节拍发生器可以产生节拍中断，据用户手册：

|  |
| --- |
| RTC节拍时间是用于中断请求。TICNT寄存器有一个中断使能位和中断的计数值。当节拍时间中断发生时计  数值达到'0'。然后中断周期如下：  — 周期 = ( n+1 ) / 128 秒  — n：节拍时间计数值（1至127）  此RTC时间节拍可能被用于实时操作系统（RTOS）内核时间节拍。如果时间节拍是由RTC时间节拍所产生的，RTOS与时间的功能将通常同步到实际时间。 |

由此可知节拍中断产生的时间可以精确到( n+1 ) / 128 秒，n为1-127区间的取值。

TICNT [7] 节拍时间中断使能。

TICNT [6:0] 节拍时间计数值（1至127）。

节拍中断使能代码

|  |
| --- |
| rTICNT **|=** **(**1**<<**7**)** **|** 127**;** //节拍中断使能 计时寄节拍设置1s 1s=（127+1）/128 |

注意：这里只是开了中断使能产生节拍中断INT\_TICK，并没有开中断

## 闹钟中断使能

|  |
| --- |
| 闹钟功能  RTC在掉电模式中或正常工作模式中的指定时间产生一个闹钟信号。在正常工作模式中，只激活闹钟中断（INT\_RTC）信号。在掉电模式中，除了INT\_RTC 之外还激活电源管理唤醒（PMWKUP）信号。RTC闹钟寄存器（RTCALM）决定了闹钟使能/禁止状态和闹钟时间设置的条件。 |

我们是在正常模式中产生INT\_RTC（闹钟）中断信号源的。使能闹钟中断，当ALM寄存器（年、月、日、时、分、秒）的值与BCD寄存器（年、月、日、时、分、秒）值相等时产生中断信号。RTCALM 寄存器是决定ALM闹钟使能和闹钟时间的，当RTCALM使能分秒的时候，只确认ALM寄存器（分、秒）的值与BCD寄存器（分、秒）值是否相等即产生中断信号。

ALM寄存器是需要赋值的，但并没有BCD寄存器那样的读写使能，而是直接赋值的。

闹钟中断使能

|  |
| --- |
| rRTCALM **=** 0x41**;** //全局闹钟使能，秒闹钟使能(0b1000001) |

ALM闹钟寄存器赋值

|  |
| --- |
| rALMYEAR **=**0x14**; //年**  rALMMON **=**0x06**; //月**  rALMDATE **=**0x12**; //日**  rALMHOUR **=**0x01**; //时**  rALMMIN **=**0x01**; //分**  rALMSEC **=**0x03**; //秒** |

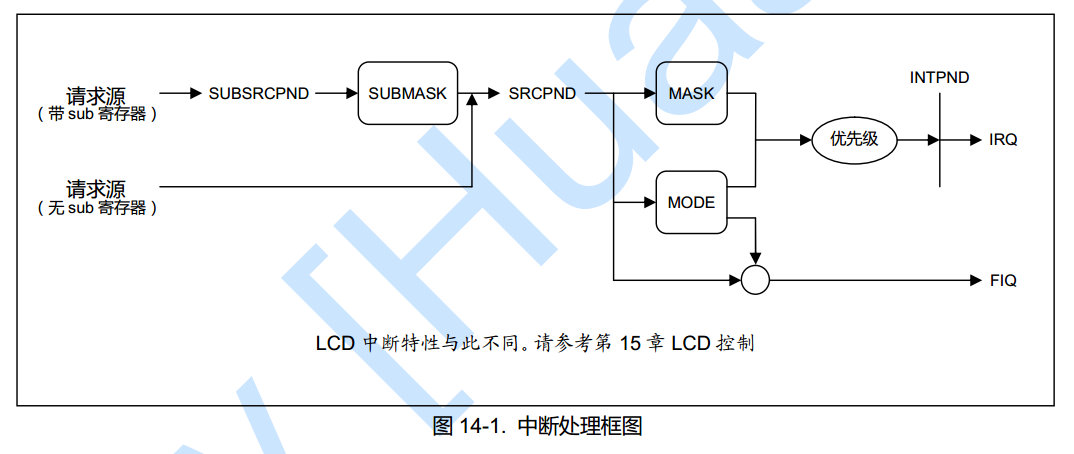
# 中断函数

概述

S3C2440A中的中断控制器接受来自60个中断源的请求。提供这些中断源的是内部外设，如DMA控制器、UART、IIC等等。在这些中断源中，UARTn、AC97和EINTn中断对于中断控制器而言是“或”关系。

当从内部外设和外部中断请求引脚收到多个中断请求时，中断控制器在仲裁步骤后请求ARM920T内核的FIQ或IRQ。

仲裁步骤由硬件优先级逻辑决定并且写入结果到帮助用户通告是各种中断源中的哪个中断发生了的中断挂起寄存器中。



闹钟中断和节拍中断是没有sub寄存器的，通过图14-1可以看出中断发生是：

中断请求中断服务>>SRCPND>>仲裁>> INTPND>>irq中断

中断挂起寄存器

|  |
| --- |
| S3C2440A有两个中断挂起寄存器：源挂起寄存器（SRCPND）和中断挂起寄存器（INTPND）。这些挂起寄存器表明一个中断请求是否为挂起。当中断源请求中断服务，SRCPND寄存器的相应位被置位为1，并且同时在仲裁步骤后INTPND 寄存器仅有1位自动置位为1。如果屏蔽了中断，则SRCPND寄存器的相应位被置位为1。这并不会引起INTPND 寄存器的位的改变。当INTPND 寄存器的挂起位为置位，每当I 标志或F标志被清除为0中断服务程序将开始。SRCPND和INTPND寄存器可以被读取和写入，因此服务程序必须首先通过写1到SRCPND寄存器的相应位来清除挂起状态并且通过相同方法来清除INTPND寄存器中挂起状态。 |

最后一句话可以知道开启中断需要SRCPND和INTPND寄存器清除挂起状态

中断屏蔽开启可服务代码

|  |
| --- |
| // 注意点（二）  rINTMSK **&=** **~(**0x1**<<**8**);** // 中断屏蔽开启可服务(此中断使能不能放在中断入口函数内，因  // 为中断使能并不是中断的操作，而是进中断前的一个控制) |

节拍中断函数入口代码

|  |
| --- |
| pISR\_TICK**=(**unsigned**)**tick\_interrupt**;** //告诉中断处理中断函数的入口地址 |

|  |
| --- |
| void \_\_irq alm\_interrupt**(){** //中断入口函数  rSRCPND**|=**0x1**<<**30**;** //清除中断挂起状态  rINTPND**|=**0x1**<<**30**;** //清除中断挂起状态  //中断功能代码块  Uart\_Printf**(**"\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* "**);**  Uart\_Printf**(**"\n 闹钟时刻！5秒 "**);**  Uart\_Printf**(**"\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n"**);**  rGPBDAT**=**LED\_ON**;** //点亮LED  Beep**(**2000**,** 4000**);** // 由于产生中断会有一秒的时间，所以实际是用了4+1秒这个中断    // 注意点（一）  rSRCPND **&=(**0x1**<<**30**);** // 当产生中断的时候，必须在开了中断功能后关闭中断清除  rINTPND **&=(**0x1**<<**30**);** // 而且关闭中断清除的控制必须在此中断入口函数由此中断控制  // (rSRCPND rINTPND置零,包括没用到的rSUBSRCPND)都应该在中断  // 入口函数里操作，而且入口函数最后也别忘了关闭中断清除  // （rSRCPND rINTPND,rSUBSRCPND置零）  // 如果没有关闭中断，将无限实现闹钟中断功能，其他中断不能进行  **}** |

# 调试总结

1. RTC非中断串口打印时间不连续问题（BCD码）

由来：之前写了一个非中断打印RTC实时时钟的代码，发现时间由9跳到16，后面发现是打印格式是十进制的原因，

解决方法：数据在RTC里是BCD码存储，十进制的，但是取出来的时候，BCD码是四位二进制表示，最高位只是9，10时BCD码(大端存储)是0001 0000，在这时如果采取进制转换成10进制就是16，但是如果继续使用16进制显示就没问题了，还是10

（二）调试程序，中断没问题却发生串口代码 while(!(rUTRSTAT0 & 0x2))；出不来问题

解决方法：

main函数增加：

|  |
| --- |
| U32 mpll\_val **=** 0**,**consoleNum**;**  Port\_Init**(); //定义在2440lib.c**  mpll\_val **=** **(**92**<<**12**)|(**1**<<**4**)|(**1**);**  //init FCLK=400M,  ChangeMPllValue**((**mpll\_val**>>**12**)&**0xff**,** **(**mpll\_val**>>**4**)&**0x3f**,** mpll\_val**&**3**); //定义在2440lib.c**  ChangeClockDivider**(**14**,** 12**);** //the result of rCLKDIVN [0:1:0:1] 3-0 bit **定义在2440lib.c**  cal\_cpu\_bus\_clk**();** //HCLK=100M PCLK=50M    consoleNum **=** 0**;** // Uart 1 select for debug.  Uart\_Init**(** 0**,**115200 **); //定义在2440lib.c**  Uart\_Select**(** consoleNum **); //定义在2440lib.c** |

**cal\_cpu\_bus\_clk()定义如下：**

|  |
| --- |
| static U32 UPLL**;**  static U32 cpu\_freq**;** |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[ HCLK=100M PCLK=50M ]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void cal\_cpu\_bus\_clk**(**void**)**  **{**  U32 val**;**  U8 m**,** p**,** s**;**    val **=** rMPLLCON**;**  m **=** **(**val**>>**12**)&**0xff**;**  p **=** **(**val**>>**4**)&**0x3f**;**  s **=** val**&**3**;**  //(m+8)\*FIN\*2 不要超出32位数!  FCLK **=** **((**m**+**8**)\*(**FIN**/**100**)\***2**)/((**p**+**2**)\*(**1**<<**s**))\***100**;**    val **=** rCLKDIVN**;**  m **=** **(**val**>>**1**)&**3**;**  p **=** val**&**1**;**  val **=** rCAMDIVN**;**  s **=** val**>>**8**;**    **switch** **(**m**)** **{**  **case** 0**:**  HCLK **=** FCLK**;**  **break;**  **case** 1**:**  HCLK **=** FCLK**>>**1**;**  **break;**  **case** 2**:**  **if(**s**&**2**)**  HCLK **=** FCLK**>>**3**;**  **else**  HCLK **=** FCLK**>>**2**;**  **break;**  **case** 3**:**  **if(**s**&**1**)**  HCLK **=** FCLK**/**6**;**  **else**  HCLK **=** FCLK**/**3**;**  **break;**  **}**    **if(**p**)**  PCLK **=** HCLK**>>**1**;**  **else**  PCLK **=** HCLK**;**    **if(**s**&**0x10**)**  cpu\_freq **=** HCLK**;**  **else**  cpu\_freq **=** FCLK**;**    val **=** rUPLLCON**;**  m **=** **(**val**>>**12**)&**0xff**;**  p **=** **(**val**>>**4**)&**0x3f**;**  s **=** val**&**3**;**  UPLL **=** **((**m**+**8**)\***FIN**)/((**p**+**2**)\*(**1**<<**s**));**  UCLK **=** **(**rCLKDIVN**&**8**)?(**UPLL**>>**1**):**UPLL**;**  **}** |

# RTC闹钟中断，节拍中断代码

## Main函数代码

|  |
| --- |
| #define GLOBAL\_CLK 1  #include <stdlib.h>  #include <string.h>  #include "def.h"  #include "option.h"  #include "2440addr.h"  #include "2440lib.h" //函数声明  #include "2440slib.h"  #include "mmu.h"  #include "profile.h"  //功能代码声明处  extern void RTC\_Display\_TICK\_ALM**(**void**);**  void Main**(**void**)**  **{**  U32 mpll\_val **=** 0**,**consoleNum**;**  Port\_Init**();**    mpll\_val **=** **(**92**<<**12**)|(**1**<<**4**)|(**1**);**    //init FCLK=400M,  ChangeMPllValue**((**mpll\_val**>>**12**)&**0xff**,** **(**mpll\_val**>>**4**)&**0x3f**,** mpll\_val**&**3**);**  ChangeClockDivider**(**14**,** 12**);** //the result of rCLKDIVN [0:1:0:1] 3-0 bit  cal\_cpu\_bus\_clk**();** //HCLK=100M PCLK=50M    consoleNum **=** 0**;** // Uart 1 select for debug.  Uart\_Init**(** 0**,**115200 **);**  Uart\_Select**(** consoleNum **);**    Beep**(**2000**,** 100**);**  //>>>>>>>>>>>>>>以下是功能代码块入口<<<<<<<<<<<<<<<<<<<  RTC\_Display\_TICK\_ALM**();** //闹钟中断显示实时时钟，报警中断(调试成功！)  **}** |

## RTC\_Display\_TICK\_ALM函数代码

|  |
| --- |
| #include "2440addr.h" //引脚宏定义  #include "def.h" // U8 U32宏定义  #include "2440lib.h" //使用Uart\_Printf,Dalay声明，Uart\_Printf定义在2440lib.c文件  #include "option.h"  #include "mmu.h"  #define LED\_OFF (0x0f<<5)  #define LED\_ON (~0x0f<<5)  struct Time**{** //RTC时间结构  U32 year**;**  U8 month**;**  U8 day**;**  U8 week**;**  U8 hour**;**  U8 mi**;**  U8 sec**;**  **}**ttime\_rtc**;**  void RTC\_set**()**  **{**  rRTCCON **|=**0x01**;**  rBCDSEC **=**0x0**;** //14年6月12日1点1分0秒 星期4  rBCDMIN **=**0x01**;**  rBCDHOUR **=**0x01**;**  rBCDDAY **=**0x4**;** //星期  rBCDDATE **=**0x12**;**  rBCDMON **=**0x6**;**  rBCDYEAR **=**0x14**;**  rRTCCON **&=~**0x01**;**  **}**  void Led\_Init**(){**  rGPBCON**=**0x015400**;** //GPB5 GPB6 GPB7 GPB8 初始化为输出  rGPBDAT**=**LED\_OFF**;** //熄灭状态  **}**  //从RTC读取值  void read\_for\_rtc**()**  **{**  rRTCCON **|=**0x01**;** //RTCCON只控制BCD寄存器，ALM数据寄存器就不需要读写控制了  ttime\_rtc**.**year **=**0x2000**+**rBCDYEAR**;**  ttime\_rtc**.**month **=**rBCDMON**;**  ttime\_rtc**.**day **=**rBCDDATE**;**  ttime\_rtc**.**week **=**rBCDDAY**;**  ttime\_rtc**.**hour **=**rBCDHOUR**;**  ttime\_rtc**.**mi **=**rBCDMIN**;**  ttime\_rtc**.**sec **=**rBCDSEC**;**  rRTCCON **&=~**0x01**;**  **}**  void ALM\_set**(){** // 闹钟赋值    rALMYEAR **=**0x14**;**  rALMMON **=**0x06**;**  rALMDATE **=**0x12**;**  rALMHOUR **=**0x01**;**  rALMMIN **=**0x01**;**  rALMSEC **=**0x03**;**    **}**  void RTC\_display**(){**  Uart\_Printf**(**"\n"**);**  Uart\_Printf**(**"\n<><><><><><><><><><><><><><><><><><><><><><><><><><><><>\n"**);**  Uart\_Printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*HELLO RTC闹钟中断、节拍中断\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"**);**  Uart\_Printf**(**"#rTICNT: (1<<7) | 127 允许节拍使能\n"**,**rTICNT**);**  Uart\_Printf**(**"#rRTCALM: 0b1000001 允许闹钟使能 秒精度闹钟\n"**,**rRTCALM**);**  Uart\_Printf**(**"#rINTMSK: ~(0x1<<8) 开启INT\_TICK中断源服务\n"**);**  Uart\_Printf**(**"#rINTMSK: ~(0x1<<30) 开启INT\_RTC 中断源服务\n"**);**  Uart\_Printf**(**"#rSRCPND rINTPND 中断控制器清除相应位再关闭\n"**);**  Uart\_Printf**(**"#\n#通过RTC节拍中断，串口输出开发板系统时间\n"**);**  Uart\_Printf**(**"#XXXX年XX月XX日XX时XX分XX秒 | LED闪亮\n"**);**  Uart\_Printf**(**"#并且可设置闹钟，闹钟到时，beep声|LED亮\n"**);**  Uart\_Printf**(**"<><><><><><><><><><><><><><><><><><><><><><><><><><><><>\n\n\n\n"**);**  **}**  void \_\_irq tick\_interrupt**(){**  rSRCPND**|=**0x1**<<**8**;** //清除中断挂起状态  rINTPND**|=**0x1**<<**8**;**    //中断功能代码块      rGPBDAT **=~(**rGPBDAT**>>**5**)<<**5 **;** //LED灯 亮闪状态切换(这样运算不会影响到蜂鸣器)  read\_for\_rtc**();**  Uart\_Printf**(**"\n RTC time: %x年%2x月%02x日--%02x:%02x:%02x 星期%d"**,**ttime\_rtc**.**year**,**ttime\_rtc**.**month**,**ttime\_rtc**.**day**,**ttime\_rtc**.**hour**,**ttime\_rtc**.**mi**,**ttime\_rtc**.**sec**,**ttime\_rtc**.**week**);**    //rTICNT &= ~(1<<7); //当只使用一次中断的时候关闭模块使能  rSRCPND **&=(**0x1**<<**8**);**  rINTPND **&=(**0x1**<<**8**);**    **}**  void \_\_irq alm\_interrupt**(){** //中断入口函数    rSRCPND**|=**0x1**<<**30**;**  rINTPND**|=**0x1**<<**30**;**    //中断功能代码块    Uart\_Printf**(**"\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* "**);**  Uart\_Printf**(**"\n 闹钟时刻！5秒 "**);**  Uart\_Printf**(**"\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n"**);**  rGPBDAT**=**LED\_ON**;** //点亮LED  Beep**(**2000**,** 4000**);** // 由于产生中断会有一秒的时间，所以实际是用了4+1秒这个中断      // 注意点（一）  rSRCPND **&=(**0x1**<<**30**);** // 当产生中断的时候，必须在开了中断功能后关闭中断清除  rINTPND **&=(**0x1**<<**30**);** // 而且关闭中断清除的控制必须在此中断入口函数由此中断控制  // (rSRCPND rINTPND置零,包括没用到的rSUBSRCPND)都应该在中断  // 入口函数里操作，而且入口函数最后也别忘了关闭中断清除  // （rSRCPND rINTPND,rSUBSRCPND置零）  // 如果没有关闭中断，将无限实现闹钟中断功能，其他中断不能进行  **}**  void RTC\_TICK**(){**    rTICNT **|=** **(**1**<<**7**)** **|** 127**;** //节拍中断使能 计时寄节拍设置1s    // 注意点（二）  rINTMSK **&=** **~(**0x1**<<**8**);** // 中断屏蔽开启可服务(此中断使能不能放在中断入口函数内，因  // 为中断使能并不是中断的操作，而是进中断前的一个控制)    pISR\_TICK**=(**unsigned**)**tick\_interrupt**;** //告诉中断处理中断函数的入口地址  **}**    void RTC\_ALM**(){**  rRTCALM **=** 0x41**;** //全局闹钟使能，秒闹钟使能(0b1000001)    rINTMSK **&=** **~(**0x1**<<**30**);**  pISR\_RTC **=(**unsigned**)**alm\_interrupt**;**    **}**    void RTC\_Display\_TICK\_ALM**()** // 子main函数  **{**  //MMU\_EnableICache();这一点至关重要， 不要关闭mmu，否则中断不能正常使用  MMU\_Init**();**  Led\_Init**();** //led1 2 3 4输出，初始化熄灭状态  RTC\_display**();** //打印相应信息    RTC\_set**();** //rtc赋值  ALM\_set**();** // ALM闹钟赋值    RTC\_TICK**();** //节拍中断  RTC\_ALM**();** //闹钟中断    **while(**1**)**  **{**    **}**  **}** |

## 输出截图

蜂鸣器和LED的图可就没了

