13.一文搞懂Cortex-A9 RTC

原创 土豆居士 一口Linux 2021-01-05 11:45

收录于合集
#所有原创 206 #从0学arm 27

ARM系列文章合集如下：

《从0学arm合集》

一、RTC

RTC(Real-Time Clock）实时时钟。

RTC

RTC是集成电路，通常称为时钟芯片。在一个嵌入式系统中，通常采用RTC来提供可靠的系统时间，包括时分秒和年月日等，而且要求在系统处于关机状态下它也能正常工作（通常采用后备电池供电）。

它的外围也不需要太多的辅助电路，典型的就是只需要一个高精度的32.768kHz 晶体和电阻电容等,并且具有闹钟的功能。。

二、Exynos 4412 RTC

本篇主要以Cortex-A9 soc为例讲解RTC的使用方法。

实时时钟（RTC）单元可以通过备用电池供电，因此，即使系统电源关闭，它也可以继续工作。RTC可以通过STRB/LDRB 指令将8位BCD码数据送至CPU。这些BCD数据包括秒、分、时、日期、星期、月和年。

RTC单元通过一个外部的32.768kHz 晶振提供时钟。

RTC具有定时报警的功能。

其功能说明如下：

1 -- 时钟数据采用BCD编码。

2 -- 能够对闰年的年月日进行自动处理。

3 -- 具有告警功能，当系统处于关机状态时，能产生警告中断。

4 -- 具有独立的电源输入。

5 -- 提供毫秒级时钟中断，该中断可以用于作为嵌入式操作系统的内核时钟。

2. RTC Block

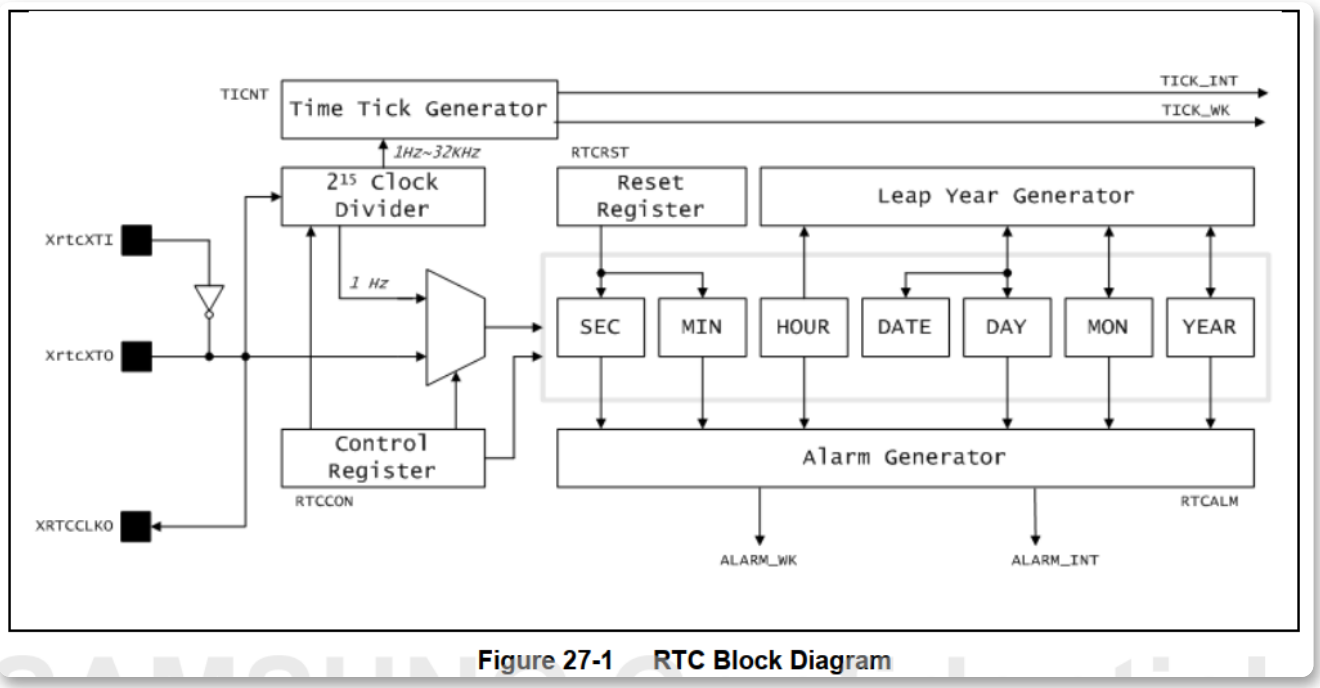

Figure 27-1 RTC Block Diagram

RTC Block Diagram

RTC在Linux中主要实现两种功能，分别是系统掉电后的时间日期维持和时间日期报警(类似

定时器)。

「1）时间日期维持功能：」

主要是由RTC实时时钟控制寄存器RTCCON进行功能的使能控制，由节拍时间计数寄存器TICNT来产生节拍时间中断来实现实时操作系统功能相关的时间和实时同步。其中对时间日期的操作实际上是对BCD码操作，而BCD码则是由一系列的寄存器组成(BCD秒寄存器BCDSEC、BCD分寄存器BCDMIN、BCD小时寄存器BCDHOUR、BCD日期寄存器BCDDATE、BCD日寄存器BCDDAY、BCD月寄存器BCDMON、BCD年寄存器BCDYEAR)。

「2）报警功能：」

主要由RTC报警控制寄存器RTC ALM进行功能使能控制，并产生报警中断。报警时间日期的设置也是对一系列的寄存器进行操作(报警秒数据寄存器ALMSEC、报警分钟数据寄存器ALMMIN、报警小时数据寄存器ALMHOUR、报警日期数据寄存器ALMDATE、报警月数据寄存器ALMMON、报警年数据寄存器ALMYEAR)。

「3）闰年发生器」

可以根据BCDDAY、BCDMON和BCDEEAR的值自动计算闰年。

3. 备用电池

备用电池可以驱动RTC逻辑。备用电池通过RTCVDD引脚向RTC块，即使系统电源关闭。如果系统关闭，您应该阻止CPU和RTC逻辑。为了减少功耗，备用电池单独驱动振荡电路和BCD计数器。

4. Alarm【报警】 功能

RTC在断电模式或正常运行模式都可以在执行的时间产生一个ALARM_INT 和ALARM_WK信号。在正常工作模式下，它会产生ALARM_INT。在断电模式下，它会ALARM_WK以及ALARM_INT信号。RTC报警寄存器（RTCALM）确定报警启用/禁用状态和报警时间设置的条件。

6. 晶振

32.768 kHz X-Tal Connection Example

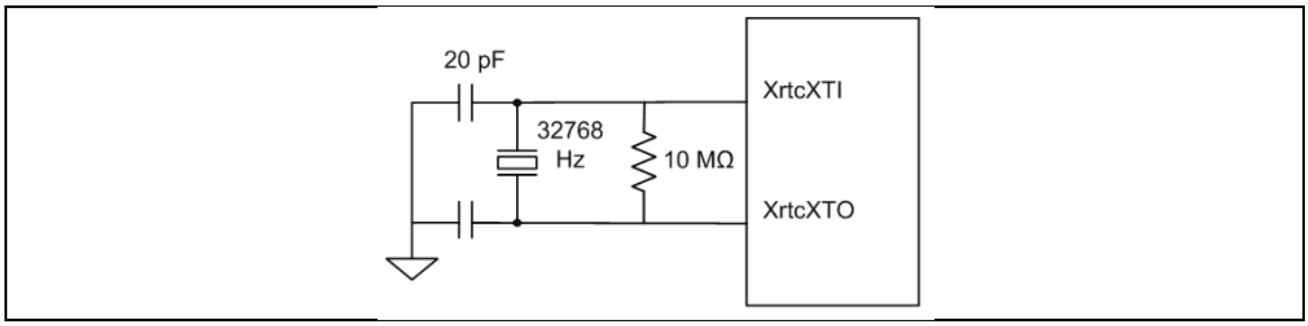

Figure 27-2 Main Oscillator Circuit Example

晶振时钟频率 32.768 kHz。

Signal	I/O	Description	Pad	Type
XT_RTC_I	Input	32.768 kHz RTC Oscillator Clock Input	XrtcXTI	Dedicated
XT_RTC_O	Output	32.768 kHz RTC Oscillator Clock Output	XrtcXTO	Dedicated
XRTCCLKO	Output	32.768 kHz RTC Clock Output (1.8 to 3.3 V). This signal is turned off by default. You can enable this signal by setting 1 in CLKOUTEN field of RTCCON register. NOTE: To use XRTCCLO, it should supply ALIVE power.	XRTCCLKO	Dedicated

1. XT_RTC_I 32.768 kHz RTC振荡器时钟输入
2. XT_RTC_O 32.768 kHz RTC振荡器时钟输出
3. XRTCCLKO 32.768 kHz RTC振荡器时钟输出，此信号默认关闭。可以通过设置寄存器RTCCON的CLKOUTEN字段为1来启用它。

引脚连接图：

由电路图可知，只连接了RTC振荡器时钟输入引脚XT_RTC_I 。

三、寄存器

1. RTC寄存器组：

27.9 Register Description

27.9.1 Register Map Summary

- Base Address: 0x1007_0000

Register	Offset	Description	Reset Value
INTP	0x0030	Specifies the interrupt pending register	0x0000_0000
RTCCON	0x0040	Specifies the RTC control register	0x0000_0000
TICCNT	0x0044	Specifies the tick time count register	0x0000_0000
RTCALM	0x0050	Specifies the RTC alarm control register	0x0000_0000
ALMSEC	0x0054	Specifies the alarm second data register	0x0000_0000
ALMMIN	0x0058	Specifies the alarm minute data register	0x0000_0000
ALMHOUR	0x005C	Specifies the alarm hour data register	0x0000_0000
ALMDAY	0x0060	Specifies the alarm day data register	0x0000_0000
ALMMON	0x0064	Specifies the alarm month data register	0x0000_0000
ALMYEAR	0x0068	Specifies the alarm year data register	0x0000_0000
BCDSEC	0x0070	Specifies the BCD second register	Undefined
BCDMIN	0x0074	Specifies the BCD minute register	Undefined
BCDHOUR	0x0078	Specifies the BCD hour register	Undefined
BCDDAYWEEK	0x007C	Specifies the BCD day of the week register	Undefined
BCDDAY	0x0080	Specifies the BCD day register	Undefined
BCDMON	0x0084	Specifies the BCD month register	Undefined
BCDYEAR	0x0088	Specifies the BCD year register	Undefined
CURTICCNT	0x0090	Specifies the current tick time counter register	0x0000_0000

设置闹钟寄存器组

设置BCD格式的时间寄存器

滴答计时器当前计数值

寄存器组

2. INTP

- Base Address: 0x1007_0000
- Address = Base Address + 0x0030, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:2]	–	Reserved	0
ALARM	[1]	RW	Alarm interrupt pending bit 0 = Interrupt did not occur 1 = Interrupt occurred	0
Time TIC	[0]	RW	Time TIC interrupt pending bit 0 = No interrupt occurred 1 = Interrupt occurred.	0

设置对应的bit为1就可以清除中断。

3. RTCCON

RTCCON	位	描述	复位值
保留	[31:10]	保留	0

RTCCON	位	描述	复位值
CLKOUTEN	[9]	使能RTC通过XRTCCLKO输出 0 disable 1 enable	0
TICEN	[8]	嘀嗒计时器 0 = 禁止 1 = 使能	0
TICCKSEL	[7:4]	嘀嗒计时器子时钟源选择 「4'b0000 = 32768 Hz」 4'b0001 = 16384 Hz 4'b0010 = 8192 Hz 4'b0011 = 4096 Hz 4'b0100 = 2048 Hz 4'b0101 =1024 Hz 4'b0110 =512 Hz 4'b0111 =256 Hz 4'b1000 =128 Hz 4'b1001 =64 Hz 4'b1010 =32 Hz 4'b1011 =16 Hz 4'b1100 =8 Hz 4'b1101 =4 Hz 4'b1110 =2 Hz 4'b1111 =1 Hz	4'b0000
CLKRST	[3]	RTC时钟计数复位 0 = 不复位 1 = 复位 0	
CNTSEL	[2]	BCD计数选择 0 = 分配 BCD 计数 1 = 保留	0
CLKSEL	[1]	BCD 时钟选择 0 = XTAL 1/2 divided clock 1 = 保留（XTAL 供频）	0
RTCEN	[0]	RTC控制使能 0 = 禁止 1 = 使能	0

1. RTCCON寄存器由10位组成，如控制BCD SEL读/写启用的CTLEN， CNTSEL、CLKRST、TICKSEL、TICEN用于测试，CLKOUTEN用于RTC时钟输出控制。
2. CTLEN位控制CPU和RTC之间的所有接口。因此，您应该在RTC控件中将其设置为“1”，在系统重置后启用数据写入的例程。为了防止无意中写入BCD计数器寄存器，应该关闭电源前将CTLEN位清除为0。
3. CLKRST是2^15^时钟分频器的计数器复位。在设置RTC时钟之前，应重置215时钟分频器以获得精确的RTC操作。

四、RTC的操作

1. 设置时间

「举例：」我们要将当前时间设置为 「2020年11月11日，15:24:50」。

1） 先将RTC控制使能开启，即RTCCON[0]置为1；2）然后将时间对应的BCD格式数值，设置到应对的寄存器，BCDYEAR 、BCDMON 、BCDDAY 、BCDHOUR 、BCDMIN 、BCDSEC；3）将RTCCON[0]置为0,防止误操作修改了时间；4）如果我们要访问当前时间，可以直接读取寄存器BCDYEAR 、BCDMON 、BCDDAY 、BCDHOUR 、BCDMIN 、BCDSEC。

```
void rtc_init(void)
{
 RTCCON = 1;//使能RTC控制写功能
 RTC.BCDYEAR = 0x20;// 2020年11月11日, 15:24:50.以BCD码格式写入
 RTC.BCDMON = 0x11;
 RTC.BCDDAY = 0x11;
 RTC.BCDHOUR = 0x15;
 RTC.BCDMIN = 0x24;
 RTC.BCDSEC = 0x50;
 RTCCON = 0;//关闭RTC控制写功能
}
```

2. 操作滴答定时器

TICNT

- Base Address: 0x1007_0000
- Address = Base Address + 0x0044, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
TICK_TIME_ COUNT	[31:0]	RW	32-bit tick time count value. Tick timer is an up-counter. If the current tick count reaches this value, tick time interrupt occurs. NOTE: This value must be greater than 3.	32'b0

TICNT

RTC计时器是一个递增计数器，并引发计时中断。TICNT寄存器包含32位目标计数值，并且CURTICCNT寄存器包含32位当前计时计数。如果当前滴答数达到TICNT中指定的目标值时，计时中断发生。

一秒钟计数的次数，由RTCCON[7:4]即TICCKSEL位决定：

TICCKSEL	[7:4]	RW	Tick timer sub clock selection 4'b0000 = 32768 Hz 4'b0001 = 16384 Hz 4'b0010 = 8192 Hz 4'b0011 = 4096 Hz 4'b0100 = 2048 Hz 4'b0101 = 1024 Hz 4'b0110 = 512 Hz 4'b0111 = 256 Hz 4'b1000 = 128 Hz 4'b1001 = 64 Hz 4'b1010 = 32 Hz 4'b1011 = 16 Hz 4'b1100 = 8 Hz 4'b1101 = 4 Hz 4'b1110 = 2 Hz 4'b1111 = 1 Hz	4'b0000

TICCKSEL

因为我们的晶振频率也是32768，为方便计数，所以我们设置RTCCON[7:4]为0，开启滴答计时器需要设置RTCCON[8]位1：

TICEN	[8]	RW	Enables Tick timer 0 = Disables tick timer 1 = Enables tick timer	0

TICEN

代码如下：

```
RTCCON = RTCCON & (~(0xf << 4)) | (1 << 8);
TICCNT = 32768;
```

3. 操作ALARM闹钟

RTCALM

Name	Bit	Type	Description	Reset Value
RSVD	[31:7]	–	Reserved	0
ALMEN	[6]	RW	Enables Alarm global 0 = Disables alarm global 1 = Enables alarm global NOTE: For using ALARM_INT and ALARM_WK, set ALMEN as 1'b1.	0
YEAREN	[5]	RW	Enables Year alarm 0 = Disables year alarm 1 = Enables year alarm	0
MONEN	[4]	RW	Enables Month alarm 0 = Disables month alarm 1 = Enables month alarm	0
DAYEN	[3]	RW	Enables Day alarm 0 = Disables day alarm 1 = Enables day alarm	0
HOUREN	[2]	RW	Enables Hour alarm 0 = Disables hour alarm 1 = Enables hour alarm	0
MINEN	[1]	RW	Enables Minute alarm 0 = Disables minute alarm 1 = Enables minute alarm	0
SECEN	[0]	RW	Enables Second alarm 0 = Disables second alarm 1 = Enables second alarm	0

RTCALM寄存器控制报警功能的启用和报警时间。请注意，RTCALM寄存器在断电模式下将同时生成ALARM_INT和ALARM_WK信号，但在正常模式下仅生成ALARM_INT信号。设置ALMEN[6]为1以产生ALARM_INT和ALARM_WK信号。

「举例：」

比如我们想每个小时的25分58秒产生一个中断信号，那我们需要设置RTCALM[1]、RTCALM[0]为1，同时设置RTCALM[6]为1以开启alarm功能，然后将BCD格式的时间设置到寄存器ALMSEC、ALMMIN。

代码如下：

```
RTCALM.ALM = (1 << 6)|(1 << 0)|(1 << 1);//使能bite：MINEN、SECEN
RTCALM.SEC = 0x58;
RTCALM.MIN = 0x25;  //每小时25:58产生一次中断
```

alarm功能设置闹钟时间寄存器如下：

27.9.1.5 ALMSEC

- Base Address: 0x1007_0000
- Address = Base Address + 0x0054, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:7]	–	Reserved	0
SECDATA	[6:4]	RW	BCD value for alarm second. 0 to 5	000
	[3:0]		0 to 9	0000

27.9.1.6 ALMMIN

- Base Address: 0x1007_0000
- Address = Base Address + 0x0058, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:7]	–	Reserved	0
MINDATA	[6:4]	RW	BCD value for alarm minute. 0 to 5	000
	[3:0]		0 to 9	0000

27.9.1.7 ALMHOUR

- Base Address: 0x1007_0000
- Address = Base Address + 0x005C, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:6]	–	Reserved	0
HOURDATA	[5:4]	RW	BCD value for alarm hour. 0 to 2	00
	[3:0]		0 to 9	0000

27.9.1.8 ALMDAY

- Base Address: 0x1007_0000
- Address = Base Address + 0x0060, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:6]	–	Reserved	0
DAYDATA	[5:4]	RW	BCD value for alarm day, from 0 to 28, 29, 30, 31. 0 to 3	00
	[3:0]		0 to 9	0000

27.9.1.9 ALMMON

- Base Address: 0x1007_0000
- Address = Base Address + 0x0064, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:5]	–	Reserved	0
MONDATA	[4]	RW	BCD value for alarm month. 0 to 1	0
	[3:0]		0 to 9	0000

27.9.1.10 ALMYEAR

- Base Address: 0x1007_0000
- Address = Base Address + 0x0068, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:8]	–	Reserved	0
YEARDATA	[11:8]	RW	BCD value for alarm year. 0 to 9	0000
	[7:4]		0 to 9	0000
	[3:0]		0 to 9	0000

寄存器操作，采用BCD格式。

五、完整代码实现

滴答计时器和alarm闹钟会产生内部中断信号，所以我们必须给这两个中断信号进行中断相关的初始化，并在中断处理函数中增加相应的处理代码。

中断号

参考datasheet 9.2.2 GIC Interrupt Table

SPI Port No	ID	Int_I_Combiner	Interrupt Source	Source Block
44	76	–	RTC_ALARM	–
45	77	–	RTC_TIC	–

rtc中断号

关于中断的初始化的寄存器配置，我们可以参考《11．从0开始学ARM-基于Exynos4412中断详解、key程序编写》

区别是，key连接在了第一级中断控制器，而rtc的这两个中断则没有。清中断需要设置的寄存器如下：

「滴答计时器清中断：」

```
RTCINTP  = RTCINTP | (1 << 0);
//清GIC中断标志位
ICDICPR.ICDICPR2 = ICDICPR.ICDICPR2 | (0x1 << 13);
```

```
//清cpu中断标志位
CPU0.ICCEOIR = CPU0.ICCEOIR&(~(0x3ff))|irq_num;
```

「alarm计时器清中断：」

```
RTCINTP  = RTCINTP | (1 << 1);
//清GIC中断标志位
ICDICPR.ICDICPR2 = ICDICPR.ICDICPR2 | (0x1 << 12);
//清cpu中断标志位
CPU0.ICCEOIR = CPU0.ICCEOIR&(~(0x3ff))|irq_num;
```

「滴答计时器中断初始化：」

```
void rtc_tic(void)
{
 RTCCON = RTCCON & (~(0xf << 4)) | (1 << 8);
 TICCNT = 32768;
 ICDDCR = 1;  //使能分配器
 ICDISER.ICDISER2 = ICDISER.ICDISER2 | (0x1 << 13); //使能相应中断到分配器
 ICDIPTR.ICDIPTR19 = ICDIPTR.ICDIPTR19 & (~(0xff << 8))|(0x1 << 8); //选择CPU接口
 CPU0.ICCPMR = 255; //中断屏蔽优先级
 CPU0.ICCICR = 1;   //使能中断到CPU
}
```

「alarm初始化」

```
void rtc_alarm(void)
{
 RTCALM.ALM = (1 << 6)|(1 << 0)|(1 << 1);
 RTCALM.SEC = 0x58;
 RTCALM.MIN = 0x25;  //每小时25:58产生一次中断
 ICDDCR = 1;    //使能分配器
  //使能相应中断到分配器
 ICDISER.ICDISER2 = ICDISER.ICDISER2 | (0x1 << 12);
 //选择CPU接口
 ICDIPTR.ICDIPTR19 = ICDIPTR.ICDIPTR19 & (~(0xff << 0))|(0x1 << 0);
```

```
 CPU0.ICCPMR = 255; //中断屏蔽优先级
 CPU0.ICCICR = 1;   //使能中断到CPU
}
```

「中断处理函数」

```
void do_irq(void)
{
 static int a = 1;
 int irq_num;
 irq_num = CPU0.ICCIAR&0x3ff;  //获取中断号
 switch(irq_num)
 {
  case 57: //按键key
   printf("in the irq_handler\n");
   //清GPIO中断标志位
   EXT_INT41_PEND = EXT_INT41_PEND |((0x1 << 1));
   //清GIC中断标志位
   ICDICPR.ICDICPR1 = ICDICPR.ICDICPR1 | (0x1 << 25);
  break;
  case 76:
   printf("in the alarm interrupt!\n");
   RTCINTP  = RTCINTP | (1 << 1);
   //清GIC中断标志位
   ICDICPR.ICDICPR2 = ICDICPR.ICDICPR2 | (0x1 << 12);
  break;
  case 77:
   printf("in the tic interrupt!\n");
   RTCINTP  = RTCINTP | (1 << 0);
    //清GIC中断标志位
   ICDICPR.ICDICPR2 = ICDICPR.ICDICPR2 | (0x1 << 13);
   break;
 }
 //清cpu中断标志位
 CPU0.ICCEOIR = CPU0.ICCEOIR&(~(0x3ff))|irq_num;
}
```

「其他代码：」

```c
void rtc_init(void)
{
 RTCCON = 1;//使能RTC控制写功能
 RTC.BCDYEAR = 0x20;// 2020年11月11日, 15:24:50.以BCD码格式写入
 RTC.BCDMON = 0x11;
 RTC.BCDDAY = 0x11;
 RTC.BCDHOUR = 0x15;
 RTC.BCDMIN = 0x24;
 RTC.BCDSEC = 0x50;
 RTCCON = 0;//关闭RTC控制写功能
}
int main (void)
{     rtc_init();
  rtc_alarm();
  rtc_tic();
  //每隔一秒打印以下当前时间
  while(1)
  {
   printf("%x-%x-%x %x:%x:%x\n",RTC.BCDYEAR,
   RTC.BCDMON,
   RTC.BCDDAY,
   RTC.BCDHOUR,
   RTC.BCDMIN,RTC.BCDSEC);
   delay_ms(1000);
  }
}
```

推荐阅读

【1】【从0学ARM】你不了解的ARM处理异常之道

【2】为什么使用结构体效率比较高？ **必读**

【3】9. 基于Cortex-A9 LED汇编、C语言驱动编写**必读**

【4】一文包你学会网络数据抓包**必读**

【5】10. 基于Cortex-A9的pwm详解 **必读**

【6】11. 基于ARM Cortex-A9中断详解 **必读**

【7】12. 如何基于Cortex-A9的UART详解

【8】网络/命令行抓包工具tcpdump详解

进群，请加一口君个人微信，带你嵌入式入门进阶。

收录于合集 #从0学arm 27

上一篇
12. 如何基于Cortex-A9的UART从头实现printf函数

下一篇
14. 从0学ARM Cortex-A9 看门狗入门

阅读原文

喜欢此内容的人还喜欢

pinia
睡不着所以学编程

Konva实现图片自适应裁剪
A逐梦博客

面试连环问--操作系统
阿Q正砖