第 29 章 实时时钟和日历

目录

本章包括下列主题：

29.1 简介

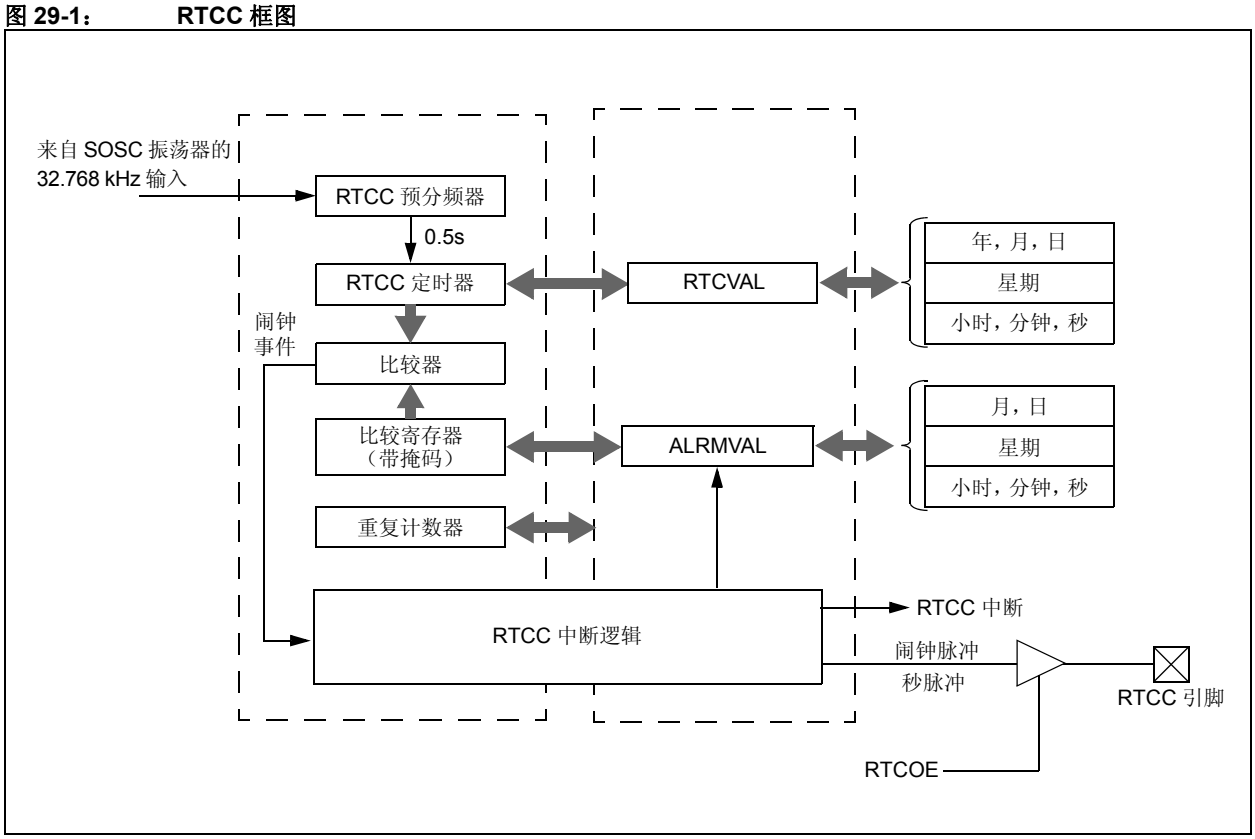
本章将讨论 PIC32MX 器件上提供的实时时钟和日历（Real-Time Clock and Calendar，RTCC）硬件模块及其操作。下面列出了该模块的部分主要特性：

- 时间：时、分和秒
- 24 小时格式 （军用时间）
- 可分辨半秒的时长
- 提供日历：星期、日、月和年
- 闹钟间隔可配置为 0.5 秒、1 秒、10 秒、1 分钟、10 分钟、1 小时、1 天、1 周、1 月或 1 年
- 闹钟使用递减计数器进行重复
- 可无限重复的闹钟：响铃 （chime）
- 年份范围：2000 至 2099
- 闰年修正
- BCD 格式以减少固件开销
- 为长时间电池工作进行了优化
- 小数秒同步
- 用户可使用自动调节功能校准时钟晶振频率
- 校准范围：每月 ± 0.66 秒误差
- 最多校准 260 ppm 的晶振误差
- 要求：外部 32.768 kHz 时钟晶振
- RTCC 引脚上的闹钟脉冲或秒时钟输出

该模块提供实时时钟和日历功能。RTCC 是为需要长时间维持精确时间的应用设计的，无需或只需最少的 CPU 干预。该模块为低功耗使用进行了优化，以便在跟踪时间的同时延长电池的使用寿命。

RTCC 模块具有 100 年的时钟和日历，能自动检测闰年。时钟范围从 2000 年 1 月 1 日 00:00:00（午夜）到 2099 年 12 月 31 日 23:59:59。小时数以 24 小时 （军用时间）格式提供。该时钟提供一秒的时间粒度，用户可看到半秒的时间间隔。

图 29-1： RTCC 框图


来自 SOSC 振荡器的
32.768 kHz 输入
RTCC 预分频器
0.5s
RTCC 定时器
RTCVAL
年，月，日
星期
小时，分钟，秒
闹钟
事件
比较器
比较寄存器
（带掩码）
ALRMVAL
月，日
星期
小时，分钟，秒
重复计数器
RTCC 中断逻辑
RTCC 中断
闹钟脉冲
秒脉冲
RTCC 引脚
RTCOE

29.2 状态和控制寄存器

RTCC 模块寄存器包含以下特殊功能寄存器 （Special Function Register， SFR）：

RTCCON 和 RTCALRM 寄存器控制 RTCC 模块的工作。

- RTCCON：RTCC 模块的控制寄存器
  RTCCONCLR、 RTCCONSET 和 RTCCONINV：RTCCON 的原子级位操作只写寄存器
- RTCALRM：RTCC 模块闹钟功能的控制寄存器
  RTCALRMCLR、 RTCALRMSET 和 RTCALRMINV：RTCALRM 的原子级位操作只写寄存器
- RTCTIME：RTCC 时间寄存器，包括小时、分钟和秒字段
  RTCTIMECLR、 RTCTIMESET 和 RTCTIMEINV：RTCTIME 的原子级位操作只写寄存器
- RTCDATE：RTCC 日期寄存器，包括年、月、日和星期字段
  RTCDATECLR、 RTCDATESET 和 RTCDATEINV：RTCDATE 的原子级位操作只写寄存器
- ALRMTIME：RTCC 闹钟时间寄存器，包括闹钟的小时、分钟和秒字段
  ALRMTIMECLR、ALRMTIMESET 和 ALRMTIMEINV：ALRMTIME 的原子级位操作只写寄存器
- ALRMDATE：RTCC 闹钟日期寄存器，包括闹钟的月、日和星期字段
  ALRMDATECLR、 ALRMDATESET 和 ALRMDATEINV：ALRMDATE 的原子级位操作只写寄存器
- IFS1：INT 控制器寄存器，用于发出有效 RTCC 中断信号
  IFS1CLR、 IFS1SET 和 IFS1INV：IFS1 的原子级位操作只写寄存器
- IEC1：INT 控制器寄存器，用于允许 RTCC 中断
  IEC1CLR、 IEC1SET 和 IEC1INV：IEC1 的原子级位操作只写寄存器
- IPC8：INT 控制器寄存器，用于设定 RTCC 中断优先级和子优先级
  IPC8CLR、 IPC8SET 和 IPC8INV：IPC8 的原子级位操作只写寄存器

下表汇总了所有与 RTCC 相关的寄存器。该汇总表之后列出了相应的寄存器，并且每个寄存器均附有详细的说明。

表 29-1： RTCC SFR 汇总

名称		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
RTCCON	31:24	—	—	—	—	—	—	CAL9:8>	
	23:16	CAL<7:0>							
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	RTSECSEL	RTCCLKON	—	—	RTCWREN	RTCSYNC	HALFSEC	RTCOE
RTCCONCLR	31:0	写入时会将 RTCCON 中的选定位清零，读取时获得的值未定义							
RTCCONSET	31:0	写入时会将 RTCCON 中的选定位置 1，读取时获得的值未定义							
RTCCONINV	31:0	写入时会将 RTCCON 中的选定位取反，读取时获得的值未定义							
RTCALRM	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ALRMEN	CHIME	PIV	ALRMSYNC	AMASK<3:0>			
	7:0	ARPT<7:0>							
RTCALRMCLR	31:0	写入时会将 RTCALRM 中的选定位清零，读取时获得的值未定义							
RTCALRMSET	31:0	写入时会将 RTCALRM 中的选定位置 1，读取时获得的值未定义							
RTCALRMINV	31:0	写入时会将 RTCALRM 中的选定位取反，读取时获得的值未定义							
RTCTIME	31:24	HR10<3:0>				HR01<3:0>			
	23:16	MIN10<3:0>				MIN01<3:0>			
	15:8	SEC10<3:0>				SEC01<3:0>			
	7:0	—	—	—	—	—	—	—	—
RTCTIMECLR	31:0	写入时会将 RTCTIME 中的选定位清零，读取时获得的值未定义							
RTCTIMESET	31:0	写入时会将 RTCTIME 中的选定位置 1，读取时获得的值未定义							
RTCTIMEINV	31:0	写入时会将 RTCTIME 中的选定位取反，读取时获得的值未定义							
RTCDATE	31:24	YEAR10<3:0>				YEAR01<3:0>			
	23:16	MONTH10<3:0>				MONTH01<3:0>			
	15:8	DAY10<3:0>				DAY01<3:0>			
	7:0	—	—	—	—	WDAY01<3:0>			
RTCDATECLR	31:0	写入时会将 RTCDATE 中的选定位清零，读取时获得的值未定义							
RTCDATESET	31:0	写入时会将 RTCDATE 中的选定位置 1，读取时获得的值未定义							
RTCDATEINV	31:0	写入时会将 RTCDATE 中的选定位取反，读取时获得的值未定义							
ALRMTIME	31:24	HR10<3:0>				HR01<3:0>			
	23:16	MIN10<3:0>				MIN01<3:0>			
	15:8	SEC10<3:0>				SEC01<3:0>			
	7:0	—	—	—	—	—	—	—	—
ALRMTIMECLR	31:0	写入时会将 ALRMTIME 中的选定位清零，读取时获得的值未定义							
ALRMTIMESET	31:0	写入时会将 ALRMTIME 中的选定位置 1，读取时获得的值未定义							
ALRMTIMEINV	31:0	写入时会将 ALRMTIME 中的选定位取反，读取时获得的值未定义							
ALRMDATE	31:24	—	—	—	—	—	—	—	—
	23:16	MONTH10<3:0>				MONTH01<3:0>			
	15:8	DAY10<3:0>				DAY01<3:0>			
	7:0	—	—	—	—	WDAY01<3:0>			
ALRMDATECLR	31:0	写入时会将 ALRMDATE 中的选定位清零，读取时获得的值未定义							
ALRMDATESET	31:0	写入时会将 ALRMDATE 中的选定位置 1，读取时获得的值未定义							
ALRMDATEINV	31:0	写入时会将 ALRMDATE 中的选定位取反，读取时获得的值未定义							

表 29-1： RTCC SFR 汇总 （续）

名称		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	写入时会将 IFS1 中的选定位清零，读取时获得的值未定义							
IFS1SET	31:0	写入时会将 IFS1 中的选定位置 1，读取时获得的值未定义							
IFS1INV	31:0	写入时会将 IFS1 中的选定位取反，读取时获得的值未定义							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	写入时会将 IEC1 中的选定位清零，读取时获得的值未定义							
IEC1SET	31:0	写入时会将 IEC1 中的选定位置 1，读取时获得的值未定义							
IEC1INV	31:0	写入时会将 IEC1 中的选定位取反，读取时获得的值未定义							
IPC8	31:24	—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
	23:16	—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
	15:8	—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
	7:0	—	—	—	U2IP<2:0>			U2IS<1:0>	
IPC8CLR	31:0	写入时会将 IPC8 中的选定位清零，读取时获得的值未定义							
IPC8SET	31:0	写入时会将 IPC8 中的选定位置 1，读取时获得的值未定义							
IPC8INV	31:0	写入时会将 IPC8 中的选定位取反，读取时获得的值未定义							

寄存器 29-1： RTCCON：RTC 控制寄存器 (1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	CAL<9:8>	
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CAL<7:0>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

R/W-0	R-0	r-x	r-x	R/W-0	R-0	R-0	R/W-0
RTSECSEL	RTCCLKON	—	—	RTCWREN	RTCSYNC	HALFSEC	RTCOE
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-26 **保留：**写入 0；忽略读操作

bit 25-16 **CAL<9:0>：**RTC 漂移校准位，包含一个有符号 10 位整数值
0111111111 = 最大正向调整，每分钟增加 511 个 RTC 时钟脉冲
...
0000000001 = 最小正向调整，每分钟增加 1 个 RTC 时钟脉冲
0000000000 = 无调整
1111111111 = 最小负向调整，每分钟减少 1 个 RTC 时钟脉冲
...
1000000000 = 最大负向调整，每分钟减少 512 个 RTC 时钟脉冲

bit 15 **ON：**RTCC 使能位
1 = 使能 RTCC 模块
0 = 禁止 RTCC 模块
注 1： 仅当 RTCWREN = 1 时， ON 位才可写。
2： 使用 1:1 PBCLK 分频比时，在清零模块 ON 位的指令之后，用户的软件不应立即在 SYSCLK 周期中读 / 写外设的 SFR。

bit 14 **FRZ：**DEBUG （调试）模式冻结位
1 = 仿真器处于 DEBUG （调试）模式时，模块停止工作
0 = 仿真器处于 DEBUG （调试）模式时，模块继续工作
注： FRZ 仅在调试异常模式下可写，在正常模式下强制为 0。

bit 13 **SIDL：**IDLE （空闲）模式停止位
1 = 在 CPU 进入 IDLE （空闲）模式时，禁止送到 RTCC 的 PBCLK
0 = 在 IDLE （空闲）模式下继续正常工作

bit 12-8 **保留：**写入 0；忽略读操作

bit 7 **RTSECSEL：**RTCC 秒时钟输出选择位
1 = 选择 RTCC 引脚输出 RTCC 秒时钟
0 = 选择 RTCC 引脚输出 RTCC 闹钟脉冲
注： 要使输出有效，要求 RTCOE （RTCCON<0>） = 1。

寄存器 29-1：　RTCCON：RTC 控制寄存器 (1)（续）

bit 6　**RTCCLKON**：RTCC 时钟使能状态位

1 = RTCC 时钟正在运行
0 = RTCC 时钟不在运行

bit 5-4　**保留**：写入 0；忽略读操作

bit 3　**RTCWREN**：RTC 值寄存器写使能位

1 = RTC 值寄存器可由用户写入
0 = RTC 值寄存器已锁定，不可由用户写入
注：　只有在使能写序列时，才能将 RTCWREN 位置 1。在任意时刻都可以向该寄存器中写入 0。

bit 2　**RTCSYNC**：RTCC 值寄存器读同步位

1 = 由于计满返回的波及， RTC 值寄存器可能会在读取时改变，从而导致读取数据无效。
如果两次读取寄存器得到的数据相同，可认为数据是有效的
0 = 可以无需考虑计满返回波动而读取 RTC 值寄存器

bit 1　**HALFSEC**：半秒状态位

1 = 一秒的后半秒
0 = 一秒的前半秒
注：　该位是只读位。它在写 SECONDS 寄存器时清零。

bit 0　**RTCOE**：RTCC 输出使能位

1 = 使能 RTCC 时钟输出——时钟送到 I/O 上
0 = 禁止 RTCC 时钟输出
注：　该位与 ON （RTCCON<15>）进行与运算，产生有效的 RTCC 输出使能。

注　1：该寄存器只能通过上电复位 （Power-on Reset， POR）进行复位。

寄存器 29-2： RTCCONCLR：RTCCON 清零寄存器

写入时会将 RTCCON 中的选定位清零，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCCON 中的选定位清零**

在一个或多个位中写入 1 会将 RTCCON 寄存器中的相应位清零，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCCONCLR = 0x00008001 时，会将 RTCCON 寄存器中的 bit 15 和 bit 0 清零。

寄存器 29-3： RTCCONSET：RTCCON 置 1 寄存器

写入时会将 RTCCON 中的选定位置 1，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCCON 中的选定位置 1**

在一个或多个位中写入 1 会将 RTCCON 寄存器中的相应位置 1，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCCONSET = 0x00008001 时，会将 RTCCON 寄存器中的 bit 15 和 bit 0 置 1。

寄存器 29-4： RTCCONINV：RTCCON 取反寄存器

写入时会将 RTCCON 中的选定位取反，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCCON 中的选定位取反**

在一个或多个位中写入 1 会将 RTCCON 寄存器中的相应位取反，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCCONINV = 0x00008001 时，会将 RTCCON 寄存器中的 bit 15 和 bit 0 取反。

寄存器 29-5： RTCALRM：RTC 闹钟控制寄存器 (1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
ALRMEN	CHIME	PIV	ALRMSYNC	AMASK<3:0>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ARPT<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-16 **保留**：写入 0；忽略读操作

bit 15 **ALRMEN**：闹钟使能位
1 = 使能闹钟
0 = 禁止闹钟
注： 当 ARPT<7:0> = 00 且 CHIME = 0 时，只要发生闹钟事件，硬件就会清零 ALRMEN。
当 RTCC ON （RTCCON<15>） = 1 且 ALRMSYNC = 1 时，不应写入该字段。

bit 14 **CHIME**：响铃使能位
1 = 使能响铃——ARPT<7:0> 允许从 00 返回到 FF
0 = 禁止响铃——ARPT<7:0> 到达 00 就停止
注： 当 RTCC ON （RTCCON<15>） = 1 且 ALRMSYNC = 1 时，不应写入该字段。

bit 13 **PIV**：闹钟脉冲初始值位
当 ALRMEN = 0 时， PIV 可写并决定闹钟脉冲的初始值。
当 ALRMEN = 1 时， PIV 只读并返回闹钟脉冲的状态。
注： 当 RTCC ON （RTCCON<15>） = 1 且 ALRMSYNC = 1 时，不应写入该字段。

bit 12 **ALRMSYNC**：闹钟同步位
1 = ARPT<7:0> 和 ALRMEN 可能会由于读操作期间半秒计满返回而改变。
必须重复读取 ARPT，直到两次读到相同值为止。由于可能会有多个位发生改变，然后与 PB 时钟域进行同步，所以必须执行该操作。
0 = 由于预分频器与半秒计满返回相距的时间大于32个RTC时钟，所以可以无需考虑计满返回而读取 ARPT<7:0> 和 ALRMEN。
注： 以上假定 CPU 读操作的执行时间小于 32 个 PBCLK。

寄存器 29-5：　RTCALRM：RTC 闹钟控制寄存器 (1)（续）

bit 11-8　**AMASK<3:0>：**闹钟掩码配置位

0000 = 每半秒
0001 = 每秒
0010 = 每 10 秒
0011 = 每分钟
0100 = 每 10 分钟
0101 = 每小时
0110 = 一天一次
0111 = 一周一次
1000 = 一月一次
1001 = 一年一次（配置在 2 月 29 日除外，这种情况下每 4 年一次）
1010 = 保留——不使用
1011 = 保留——不使用
11XX = 保留——不使用

注：　当 RTCC ON（RTCCON<15>）= 1 且 ALRMSYNC = 1 时，不应写入该字段。

bit 7-0　**ARPT<7:0>：**闹钟重复计数器值位

11111111 = 闹钟将触发 256 次
...
00000000 = 闹钟将触发 1 次
每当发生闹钟事件时计数器就递减 1。除非 CHIME = 1，否则计数器不能从 00 返回到 FF。

注：　当 RTCC ON（RTCCON<15>）= 1 且 ALRMSYNC = 1 时，不应写入该字段。

注　1：该寄存器只能通过 POR 进行复位。

寄存器 29-6： RTCALRMCLR：RTCALRM 清零寄存器

写入时会将 RTCALRM 中的选定位清零，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCALRM 中的选定位清零**

在一个或多个位中写入 1 会将 RTCALRM 寄存器中的相应位清零，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCALRMCLR = 0x0000c000 时，会将 RTCALRM 寄存器中的 bit 15 和 bit 14 清零。

寄存器 29-7： RTCALRMSET：RTCALRM 置 1 寄存器

写入时会将 RTCALRM 中的选定位置 1，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCALRM 中的选定位置 1**

在一个或多个位中写入 1 会将 RTCALRM 寄存器中的相应位置 1，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCALRMSET = 0x0000c000 时，会将 RTCALRM 寄存器中的 bit 15 和 bit 14 置 1。

寄存器 29-8： RTCALRMINV：RTCALRM 取反寄存器

写入时会将 RTCALRM 中的选定位取反，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCALRM 中的选定位取反**

在一个或多个位中写入 1 会将 RTCALRM 寄存器中的相应位取反，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCALRMINV = 0x0000c000 时，会将 RTCALRM 寄存器中的 bit 15 和 bit 14 取反。

寄存器 29-9： RTCTIME：RTC 时间值寄存器 (1)

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
HR10<3:0>				HR01<3:0>			
bit 31							bit 24

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MIN10<3:0>				MIN01<3:0>			
bit 23							bit 16

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SEC10<3:0>				SEC01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-28 **HR10<3:0>**：小时的十位数的二 - 十进制码 （Binary-Coded Decimal， BCD）值位；值为 0 到 2
注： HR10<3:2> 位总是读为 0。

bit 27-24 **HR01<3:0>**：小时的个位数的 BCD 值位；值为 0 到 9

bit 23-20 **MIN10<3:0>**：分钟的十位数的 BCD 值位；值为 0 到 5
注： MIN10<3> 位总是读为 0。

bit 19-16 **MIN01<3:0>**：分钟的个位数的 BCD 值位；值为 0 到 9

bit 15-12 **SEC10<3:0>**：秒的十位数的 BCD 值位；值为 0 到 5
注： SEC10<3> 位总是读为 0。

bit 11-8 **SEC01<3:0>**：秒的个位数的 BCD 值位；值为 0 到 9

bit 7-0 **保留**：写入 0；忽略读操作

注 1：仅当 RTCWREN （RTCCON<3>） = 1 时，该寄存器才可写。

寄存器 29-10： RTCTIMECLR：RTCTIME 清零寄存器

写入时会将 RTCTIME 中的选定位清零，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCTIME 中的选定位清零**

在一个或多个位中写入 1 会将 RTCTIME 寄存器中的相应位清零，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCTIMECLR = 0x0000ff00 时，会将 RTCTIME 寄存器中的 bit 15:8 清零。

寄存器 29-11： RTCTIMESET：RTCTIME 置 1 寄存器

写入时会将 RTCTIME 中的选定位置 1，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCTIME 中的选定位置 1**

在一个或多个位中写入 1 会将 RTCTIME 寄存器中的相应位置 1，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCTIMESET = 0x00005900 时，会将 RTCTIME 寄存器中的秒设置为 59。

寄存器 29-12： RTCTIMEINV：RTCTIME 取反寄存器

写入时会将 RTCTIME 中的选定位取反，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCTIME 中的选定位取反**

在一个或多个位中写入 1 会将 RTCTIME 寄存器中的相应位取反，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCTIMEINV = 0x00000300 时，会将 RTCTIME 寄存器中的 bit 9 和 bit 8 取反。

寄存器 29-13： RTCDATE：RTC 日期值寄存器 (1)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
YEAR10<3:0>				YEAR01<3:0>			
bit 31							bit 24

R-0	R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MONTH10<3:0>				MONTH01<3:0>			
bit 23							bit 16

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DAY10<3:0>				DAY01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	R-0	R/W-x	R/W-x	R/W-x
—	—	—	—	WDAY01<3:0>			
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-28 **YEAR10<3:0>：**年份的十位数的 BCD 值位

bit 27-24 **YEAR01<3:0>：**年份的个位数的 BCD 值位

bit 23-20 **MONTH10<3:0>：**月份的十位数的 BCD 值位；值为 0 到 1

注： MONTH10<3:1> 位总是读为 0。

bit 19-16 **MONTH01<3:0>：**月份的个位数的 BCD 值位；值为 0 到 9

bit 15-12 **DAY10<3:0>：**日的十位数的 BCD 值位；值为 0 到 3

注： DAY10<3:2> 位总是读为 0。

bit 11-8 **DAY01<3:0>：**日的个位数的 BCD 值位；值为 0 到 9

bit 7-4 **保留：**写入 0；忽略读操作

bit 3-0 **WDAY01<3:0>：**星期的个位数的 BCD 值位；值为 0 到 6

注： WDAY01<3> 位总是读为 0。

注 1：仅当 RTCWREN（RTCCON<3>）= 1 时，该寄存器才可写。

寄存器 29-14： RTCDATECLR：RTCDATE 清零寄存器

写入时会将 RTCDATE 中的选定位清零，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCDATE 中的选定位清零**

在一个或多个位中写入 1 会将 RTCDATE 寄存器中的相应位清零，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCDATECLR = 0x00000007 时，会将 RTCDATE 寄存器中的 bit 2:0 清零。

寄存器 29-15： RTCDATESET：RTCDATE 置 1 寄存器

写入时会将 RTCDATE 中的选定位置 1，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCDATE 中的选定位置 1**

在一个或多个位中写入 1 会将 RTCDATE 寄存器中的相应位置 1，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCDATESET = 0x00000003 时，会将 RTCDATE 寄存器中的星期设置为 3。

寄存器 29-16： RTCDATEINV：RTCDATE 取反寄存器

写入时会将 RTCDATE 中的选定位取反，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 RTCDATE 中的选定位取反**

在一个或多个位中写入 1 会将 RTCDATE 寄存器中的相应位取反，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：RTCDATEINV = 0x00000003 时，会将 RTCDATE 寄存器中的 bit 1 和 bit 0 取反。

寄存器 29-17： ALRMTIME：闹钟时间值寄存器

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
HR10<3:0>				HR01<3:0>			
bit 31							bit 24

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MIN10<3:0>				MIN01<3:0>			
bit 23							bit 16

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SEC10<3:0>				SEC01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-28 **HR10<3:0>**：小时的十位数的 BCD 值位；值为 0 到 2
注： HR10<3:2> 位总是读为 0。

bit 27-24 **HR01<3:0>**：小时的个位数的 BCD 值位；值为 0 到 9

bit 23-20 **MIN10<3:0>**：分钟的十位数的 BCD 值位；值为 0 到 5
注： MIN10<3> 位总是读为 0。

bit 19-16 **MIN01<3:0>**：分钟的个位数的 BCD 值位；值为 0 到 9

bit 15-12 **SEC10<3:0>**：秒的十位数的 BCD 值位；值为 0 到 5
注： SEC10<3> 位总是读为 0。

bit 11-8 **SEC01<3:0>**：秒的个位数的 BCD 值位；值为 0 到 9

bit 7-0 **保留**：写入 0；忽略读操作

寄存器 29-18： ALRMTIMECLR：ALRMTIME 清零寄存器

写入时会将 ALRMTIME 中的选定位清零，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 ALRMTIME 中的选定位清零**

在一个或多个位中写入 1 会将 ALRMTIME 寄存器中的相应位清零，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：`ALRMTIMECLR = 0x0000ff00` 时，会将 ALRMTIME 寄存器中的 bit 15:8 清零。

寄存器 29-19： ALRMTIMESET：ALRMTIME 置 1 寄存器

写入时会将 ALRMTIME 中的选定位置 1，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 ALRMTIME 中的选定位置 1**

在一个或多个位中写入 1 会将 ALRMTIME 寄存器中的相应位置 1，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：`ALRMTIMESET = 0x00330000` 时，会将 ALRMTIME 寄存器中的闹钟分钟设置为 33。

寄存器 29-20： ALRMTIMEINV：ALRMTIME 取反寄存器

写入时会将 ALRMTIME 中的选定位取反，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 ALRMTIME 中的选定位取反**

在一个或多个位中写入 1 会将 ALRMTIME 寄存器中的相应位取反，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：`ALRMTIMEINV = 0x00000300` 时，会将 ALRMTIME 寄存器中的 bit 9 和 bit 8 取反。

寄存器 29-21： ALRMDATE：闹钟日期值寄存器

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

R-0	R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MONTH10<3:0>				MONTH01<3:0>			
bit 23							bit 16

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DAY10<3:0>				DAY01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	R-0	R/W-x	R/W-x	R/W-x
—	—	—	—	WDAY01<3:0>			
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-24 **保留：**写入 0；忽略读操作

bit 23-20 **MONTH10<3:0>：**月份的十位数的 BCD 值位；值为 0 到 1

注： MONTH10<3:1> 位总是读为 0。

bit 19-16 **MONTH01<3:0>：**月份的个位数的 BCD 值位；值为 0 到 9

bit 15-12 **DAY10<3:0>：**日的十位数的 BCD 值位；值为 0 到 3

注： DAY10<3:2> 位总是读为 0。

bit 11-8 **DAY01<3:0>：**日的个位数的 BCD 值位；值为 0 到 9

bit 7-4 **保留：**写入 0；忽略读操作

bit 3-0 **WDAY01<3:0>：**星期的个位数的 BCD 值位；值为 0 到 6

注： WDAY01<3> 位总是读为 0。

寄存器 29-22： ALRMDATECLR：ALRMDATE 清零寄存器

写入时会将 ALRMDATE 中的选定位清零，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 ALRMDATE 中的选定位清零**

在一个或多个位中写入 1 会将 ALRMDATE 寄存器中的相应位清零，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：ALRMDATECLR = 0x00000007 时，会将 ALRMDATE 寄存器中的 bit 2:0 清零。

寄存器 29-23： ALRMDATESET：ALRMDATE 置 1 寄存器

写入时会将 ALRMDATE 中的选定位置 1，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 ALRMDATE 中的选定位置 1**

在一个或多个位中写入 1 会将 ALRMDATE 寄存器中的相应位置 1，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：ALRMDATESET = 0x00003100 时，会将 ALRMDATE 寄存器中的闹钟日设置为 31。

寄存器 29-24： ALRMDATEINV：ALRMDATE 取反寄存器

写入时会将 ALRMDATE 中的选定位取反，读取时获得的值未定义	
bit 31	bit 0

bit 31-0 **将 ALRMDATE 中的选定位取反**

在一个或多个位中写入 1 会将 ALRMDATE 寄存器中的相应位取反，但不会影响未实现位或只读位。写入 0 不会影响该寄存器。

示例：ALRMDATEINV = 0x00000003 时，会将 ALRMDATE 寄存器中的 bit 1 和 bit 0 取反。

寄存器 29-25： IFS1：中断标志状态寄存器 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-26 **保留：**写入 0；忽略读操作

bit 25-24 其他外设器件的中断标志

bit 23-20 **保留：**写入 0；忽略读操作

bit 19-16 其他外设器件的中断标志

bit 15 **RTCCIF**：RTCC 中断标志位
1 = RTCC 中断待处理
0 = 没有待处理的 RTCC 中断
在 RTCC 产生事件时由硬件置 1。
用软件清零，通常在 ISR 中清零。

bit 14-0 其他外设器件的中断标志

注 1：该中断寄存器中的阴影位名称用于控制其他 PIC32MX 外设，与 RTCC 无关。

寄存器 29-26： IEC1：中断允许控制寄存器 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-26 未实现：读为 0

bit 25-24 其他外设器件的中断允许位

bit 23-20 **保留**：写入 0；忽略读操作

bit 19-16 其他外设器件的中断允许位

bit 15 **RTCCIE**：RTCC 中断允许位
1 = 允许 RTCC 中断
0 = 禁止 RTCC 中断
用软件置 1/ 清零，以允许 / 禁止在 RTCC 产生事件时发生中断。

bit 14-0 其他外设器件的中断允许位

注 1： 该中断寄存器中的阴影位名称用于控制其他 PIC32MX 外设，与 RTCC 无关。

寄存器 29-27： IPC8：中断优先级控制寄存器 8(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FCSMIP<2:0>			FCSMIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U2IP<2:0>			U2IS<1:0>	
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-29 **保留：**写入 0；忽略读操作

bit 28-26 **RTCCIP<2:0>：**RTCC 中断向量优先级位
111 = RTCC 中断优先级为 7（最高优先级）
•
•
•
001 = RTCC 中断优先级为 1
000 = 禁止 RTCC 中断

bit 25-24 **RTCCIS<1:0>：**RTCC 中断向量子优先级位
11 = RTCC 中断子优先级为 3（最高子优先级）
•
•
00 = RTCC 中断子优先级为 0（最低子优先级）

bit 23-21 **保留：**写入 0；忽略读操作

bit 20-16 **其他外设器件的中断优先级控制**

bit 15-13 **保留：**写入 0；忽略读操作

bit 12-8 **其他外设器件的中断优先级控制**

bit 7-5 **保留：**写入 0；忽略读操作

bit 4-0 **其他外设器件的中断优先级控制**

注 1： 该中断寄存器中的阴影位名称用于控制其他 PIC32MX 外设，与 RTCC 无关。

29.3 工作模式

RTCC 模块提供以下工作模式：

- 实时时钟和日历 （RTCC）功能
- 闹钟功能

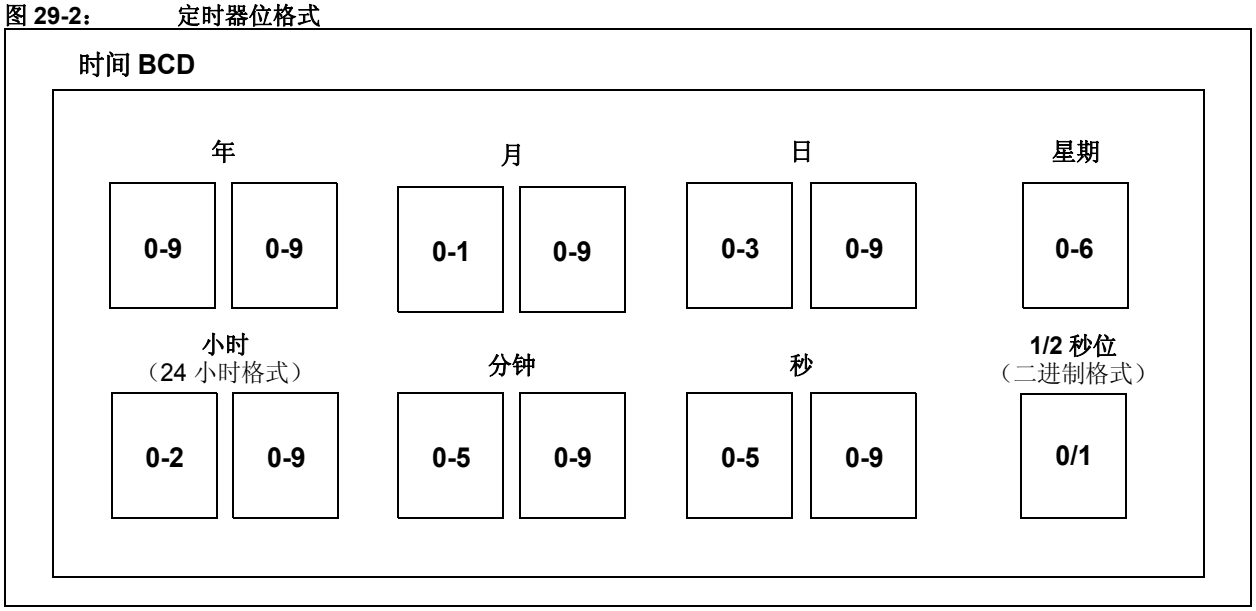
29.3.1 RTCC 工作原理

RTCC 具有 100 年的时钟和日历，能自动检测闰年。时钟范围从 2000 年 1 月 1 日 00:00:00 （午夜）到 2099 年 12 月 31 日 23:59:59。小时使用 24 小时时间格式 （军用时间），未提供常规时间格式 （AM/PM）的硬件。

RTCC 提供了 1 秒的编程粒度，但可分辨半秒的时长。

RTCC 值 （RTCTIME 和 RTCDATE）的寄存器接口是用二 - 十进制码 （BCD）格式实现的。这在使用该模块时简化了固件，因为每个位值都包含在它自己的 4 位值中了 （见图 29-2）。

图 29-2： 定时器位格式



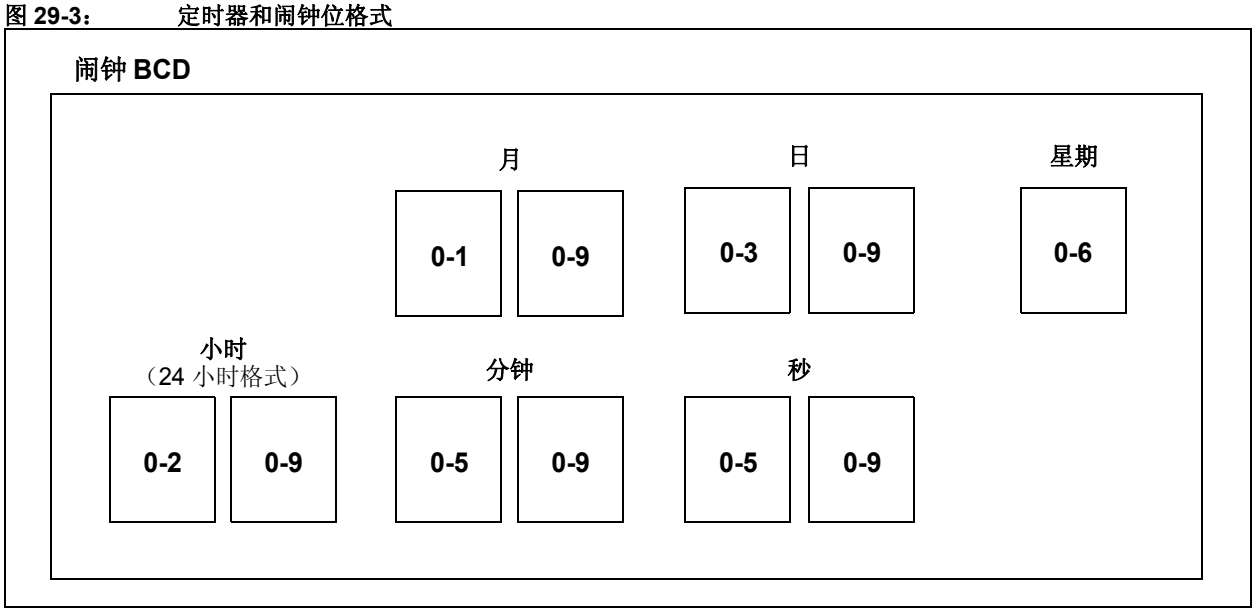
29.3.2 闹钟工作原理

模块提供了闹钟功能，可配置范围为半秒至一年。但是，只有半秒闹钟具有半秒的分辨率。模块可以配置为在使能闹钟之后，以预先配置的间隔重复闹钟。闹钟的无限重复通过响铃功能实现。

模块可以在每次发生闹钟脉冲事件时提供中断。除了闹钟中断之外，还提供了频率为闹钟频率一半的闹钟脉冲输出（闹钟脉冲在每次闹钟匹配时翻转）。该输出与 RTCC 时钟完全同步，可用于为其他器件提供触发时钟。该输出的初始值通过 PIV （RTCALRM<13>）进行控制。更多信息，请参见寄存器 29-5。

闹钟值 （ALRMTIME 和 ALRMDATE）的寄存器接口是用 BCD 格式实现的。这在使用该模块时简化了固件，因为每个位值都包含在它自己的 4 位值中了 （见图 29-3）。

图 29-3： 定时器和闹钟位格式

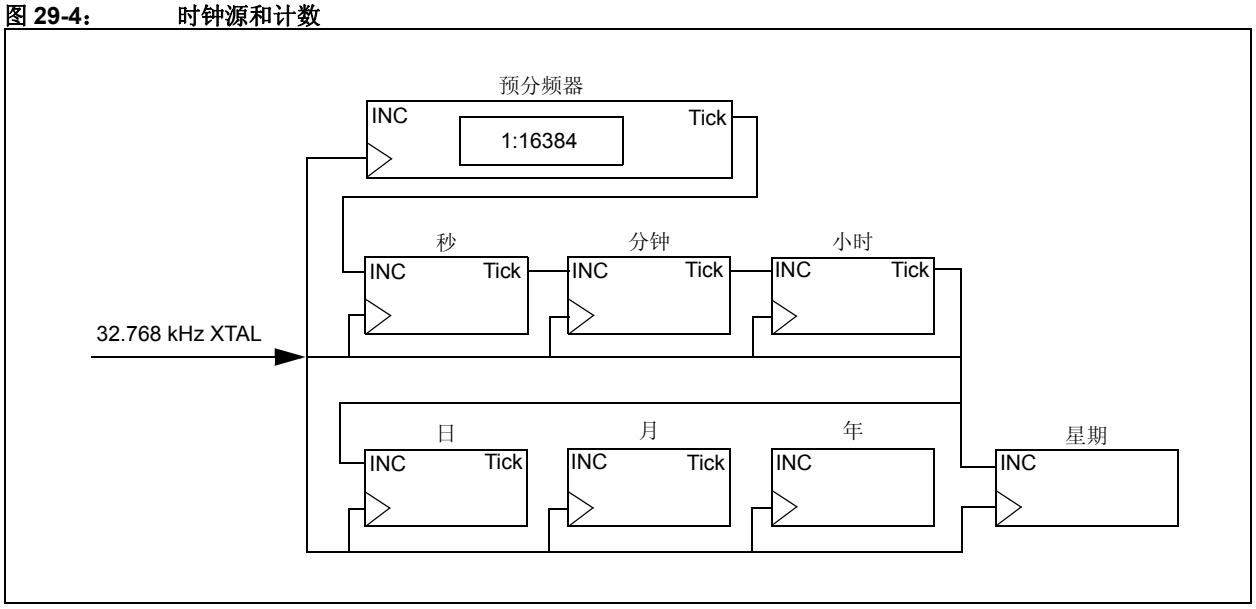


29.3.3 时钟源

RTCC 模块应由 32.768 kHz 的外部实时时钟晶振提供时钟源。晶振校准可通过该模块实现，达到的精度为每月 +/-0.66 秒（更多详细信息，请参见**第 29.3.10 节 “校准”**）。

要允许 RTCC 由外部 32.768 kHz 晶振提供时钟源，SOSCEN 位（OSCCON<1>）必须置 1（见**第 6 章 “振荡器”**（DS61112）的寄存器 6-1）。要使能 RTCC，这是用户在 RTCC 模块之外唯一要注意的位。状态位 SOSCRDY （OSCCON<22>）可用于检查辅助振荡器是否运行。

图 29-4： 时钟源和计数



29.3.4 进位规则

本节说明发生计满返回时，会影响哪些定时器值。

- 时间——从 23:59:59 到 00:00:00，向日字段进位
- 日——从日字段到月字段的进位取决于当前月份 （关于日到月的计满返回设定，请参见表 29-3。）
- 月——从 12/31 到 01/01，向年字段进位
- 星期——从 6 到 0，无进位 （见表 29-2）
- 年——从 99 到 00，无进位 （这超出了 RTCC 的适用范围）

考虑到以下值是 BCD 格式，向 BCD 高位的进位将在计数为 10 时，而不是在计数为 16 时发生（秒数、分钟数、小时数、星期、日和月）。

表 29-2： 星期设定

星期	
星期日	0
星期一	1
星期二	2
星期三	3
星期四	4
星期五	5
星期六	6

表 29-3： 日到月计满返回设定

月	最大日字段
01 （一月）	31
02 （二月）	28 或 29(1)
03 （三月）	31
04 （四月）	30
05 （五月）	31
06 （六月）	30
07 （七月）	31
08 （八月）	31
09 （九月）	30
10 （十月）	31
11 （十一月）	30
12 （十二月）	31

注 1：请参见**第 29.3.5 节 “闰年”**。

29.3.5 闰年

由于 RTCC 模块的年份范围是 2000 到 2099，闰年是通过以上范围内的年份能否被 4 整除来确定的。闰年中唯一受影响的月份是二月。（闰年中二月有 29 天，其他所有年份中二月只有 28 天。）

29.3.6 RTCC 的一般功能

所有包含秒或更大时间值的定时器寄存器都可写。用户只需将所需的年、月、日、小时、分钟和秒写入这些寄存器，就可以配置当前时间。随后定时器就会用新写入的值从所需起点继续计数。

请注意，如果通过设置 ON （RTCCON<15>） = 1 而使能了 RTCC，则即使在调节寄存器时，定时器也会继续递增。但是，每次写入 SECONDS 寄存器 （RTCTIME<15:8>）时，预分频器会复位为 0。这可以在进行定时器调节之后提供已知的预分频值。

如果发生定时器寄存器更新（CPU 写操作），用户需要负责确保在 ON （RTCCON<15>） = 1 的情况下，正在更新的寄存器不会发生定时器递增操作。这可以通过以下方式实现：观察 RTCSYNC 位 （RTCCON<2>）的值或观察可发生进位的先前位，或者紧接在秒脉冲 （或闹钟中断）之后立即更新寄存器。请注意，相应的计数器将按照为它们定义的间隔发送时钟脉冲，即，对于 DAYS 寄存器每天发送一次时钟脉冲，对于 MONTHS 寄存器每月仅发送一次时钟脉冲，如此类推。这留出很大的时间窗，让用户可以安全地更新寄存器。

定时器还提供了分辨计数器半秒字段的功能。但是，该值是只读值，只能通过写入 SECONDS 寄存器 （RTCTIME<15:8>）进行复位。

29.3.7 寄存器读写安全窗口

RTCSYNC 位 （RTCCON<2>）指示对应于以下情况的时间窗：短时间内 RTCC 时间寄存器（RTCTIME 和 RTCDATE）不会发生更新，可以安全地读取和写入寄存器。当 RTCSYNC = 0 时，CPU 可安全访问这些寄存器。当 RTCSYNC = 1 时，用户必须采用固件方法确保数据读取没有发生在更新边界，从而导致无效或部分读取。

RTCSYNC 位会在更新发生的 32 个 RTCC 时钟边沿之前置 1。它在发生更新的一个时钟之后清零（从而 RTCSYNC 置 1 的时间为 33 个时钟）。最坏情况下，如果 CPU 内核使用 32.768 kHz 振荡器作为时钟， PBCLK 为 SYSCLK/8，在读取到 RTCSYNC = 0 之后，至少有 22 个 CPU 时钟可用于执行指令 （由于同步和总线延时的原因，可能会丢失一些时钟周期）。

请注意，不论 RTCSYNC 值如何，用户都可以通过两次读取并比较定时器寄存器值，在代码中确保寄存器读操作的跨度不会涵括 RTCC 时钟更新操作。

在 RTCSYNC = 1 时，不应对时间和日期寄存器执行写操作。存在该限制有两个原因：

1. 写操作可能导致闹钟匹配逻辑中发生时序违例，从而导致无效的闹钟事件和 ARPT 寄存器数据损坏。在 RTCC 时钟的低电平期间，紧随在计满返回事件之后执行写操作会产生该事件。
2. 在 RTCC 时钟为高电平时，在计满返回事件期间执行的写操作会被硬件忽略。

例 29-1： 更新 RTCC 时间和日期

```
/*
    The following code example will update the RTCC time and date.
*/

    assume the secondary oscillator is enabled and ready, i.e. OSCCON<1>=1, OSCCON<22>=1, and
    RTCC write is enabled i.e. RTCWREN (RTCCON<3>) =1;

    unsigned long time=0x04153300;// set time to 04 hr, 15 min, 33 sec
    unsigned long date=0x06102705;// set date to Friday 27 Oct 2006

    RTCCONCLR=0x8000;             // turn off the RTCC
    while(RTCCON&0x40);           // wait for clock to be turned off
    RTCTIME=time;                 // safe to update the time
    RTCDATE=date;                 // update the date
    RTCCONSET=0x8000;             // turn on the RTCC
    while(!(RTCCON&0x40));        // wait for clock to be turned on

                                  // can disable the RTCC write
```

例 29-2： 使用 RTCSYNC 窗口更新 RTCC 时间

```
/*
    The following code example will update the RTCC time and date.
*/

    assume RTCC write is enabled i.e. RTCWREN (RTCCON<3>) =1;

    unsigned long time=0x04153300;// set time to 04 hr, 15 min, 33 sec
    unsigned long date=0x06102705;// set date to Friday 27 Oct 2006

    asm volatile ("di");          // disable interrupts, critical section follows
    while((RTCCON&0x4)!=0);       // wait for not RTCSYNC
    RTCTIME=time;                 // safe to update the time
    RTCDATE=date;                 // update the date
    asm volatile ("ei");          // restore interrupts, critical section ended

                                  // can disable the RTCC write
```

29.3.8 同步

模块提供了单个 RTCSYNC 位 （RTCCON<2>），用户必须使用该位来确定何时可以安全地读取和更新时间和日期寄存器。此外，模块还可以对复位条件 （即，写入秒寄存器）和 ON （RTCCON<15>）进行同步。

29.3.8.1 RTCSYNC 位产生

RTCSYNC 位是只读位，它在 ON = 1，并且 RTCC 预分频器计数器等于 0x7FE0 （与一秒计满返回相距 32 个时钟）时置 1。对于以下任意条件，逻辑会将 RTCSYNC 位清零：

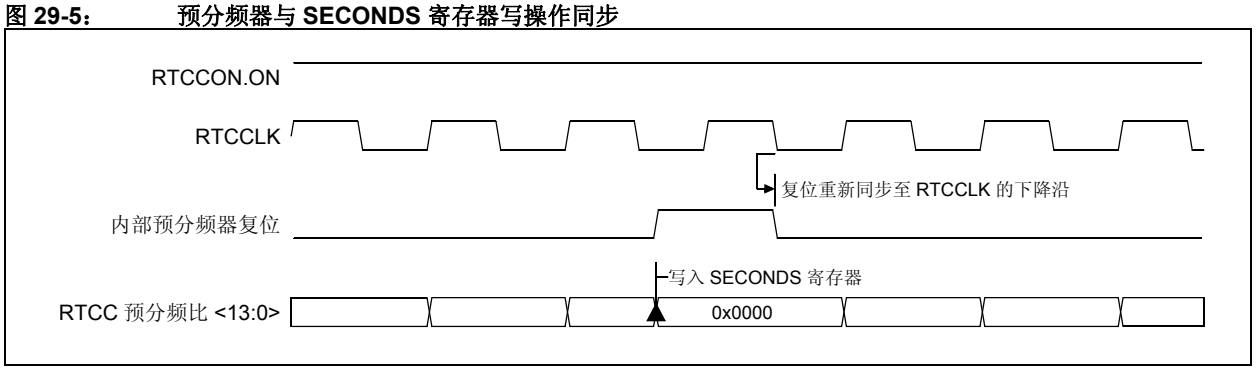
- POR
- 每当 ON = 0 时
- 写入 SECONDS （RTCTIME<15:8>）寄存器时
- 在 RTCC 时钟的上升沿，当预分频器等于 0x0000 时

关于 RTCSYNC 位的时序，请参见图 29-6。

29.3.8.2 预分频器复位同步

写入 SECONDS 寄存器 （RTCTIME<15:8>）会异步复位 RTCC 预分频器 （包括 HALFSEC 寄存器）。复位一直保持有效，直到检测到 RTCCLK 下降沿为止 （见图 29-5）。

图 29-5： 预分频器与 SECONDS 寄存器写操作同步



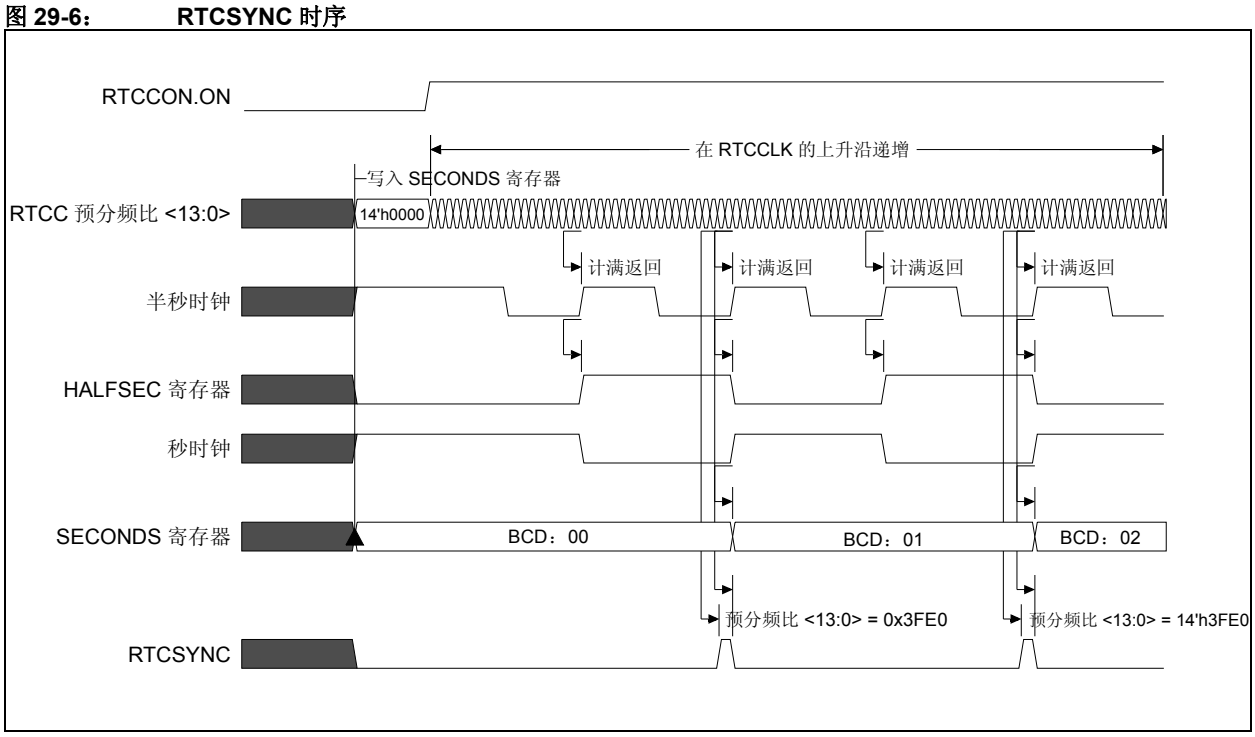
29.3.8.3 断开 RTCC 时钟

在两种情况下，内部 RTCC 时钟会断开：

- ON （RTCON<15>） = 0
- 器件处于 DEBUG （调试）模式，并且 FRZ （RTCCON<14>）= 1

停止 RTCC 时钟不会影响通过外设总线接口读写寄存器。

图 29-6： RTCSYNC 时序



29.3.9 写锁定

为执行对任何 RTCC 定时器寄存器的写操作，RTCWREN 位（RTCCON<3>）必须置 1。只有在执行器件电平解锁序列之后，才能将 RTCWREN 位置 1。解锁序列如下：

1. 将 0xAA996655 装入 CPU 寄存器 X。
2. 将 0x556699AA 装入 CPU 寄存器 Y。
3. 将 0x00000008 装入 CPU 寄存器 Z （RTCWREN 位编号）。
4. 暂停或禁止所有可访问外设总线和中断解锁序列的发起器 （即， DMA 和中断）。
5. 将 CPU 寄存器 X 存储到 SYSKEY 中。
6. 将 CPU 寄存器 Y 存储到 SYSKEY 中。
7. 将 CPU 寄存器 Z 存储到 RTCCONSET 中。
8. 重新使能 DMA 和允许中断。

请注意，必须严格遵循步骤 5 至 7 来解锁 RTCC 写操作。如果未严格遵循该序列， RTCWREN 位将不会置 1。

关于写解锁操作的汇编语言实现，请参见例 29-3。

例 29-3： 写解锁序列

```
# assume interrupts are disabled
# assume the DMA controller is suspended
# assume the device is locked

#starting critical sequence
SYSKEY = 0xaa996655;                // write first unlock key to SYSKEY
SYSKEY = 0x556699aa;                // write second unlock key to SYSKEY
RTCCONSET = 0x8;                    // set RTCWREN in RTCCONSET
#end critical sequence

# re-enable interrupts
# re-enable the DMA controller
```

注： 为避免意外写入 RTCC 时间值，建议其他任何时候 RTCWREN 位 （RTCCON<3>）都保持为零。要将 RTCWREN 置 1，在 key1/key2 序列和 RTCWREN 置 1 之间只允许 1 个指令周期的时间窗。因此，建议遵循例 29-3 中的代码示例。

29.3.10 校准

实时晶振输入可用周期性自动调节功能校准。正确校准后， RTCC 可提供每月小于 0.66 秒的误差。校准最高可以消除 260 ppm 的误差。

校准的实现方式是，先确定误差时钟脉冲数量，然后将该值写入 RTCCON 寄存器的 CAL 字段（RTCCON<9:0>）。这个 10 位有符号值将每分钟与 RTCC 定时器相加或相减一次。关于 RTCC 校准，请参见以下步骤：

1. 使用器件上的其他定时器资源，用户必须找出 32.768 kHz 晶振的误差。
2. 知道误差后，必须将它转换为每分钟的误差时钟脉冲数。

公式 29-1： 公式框：

(理想频率 （32,758） – 测得频率) * 60 = 每分钟误差时钟数

3. a) 如果振荡器频率*快于*理想频率（从步骤 2 得出负的结果），CAL 寄存器值必须为负。这会导致每分钟从定时器计数器中减去指定的时钟脉冲数。
   b) 如果振荡器频率*慢于*理想频率（从步骤 2 得出正的结果），CAL 寄存器值必须为正。这会导致每分钟在定时器计数器中加上指定的时钟脉冲数。
4. 将正确的值装入 CAL 寄存器 （RTCCON<9:0>）。

CAL 寄存器写操作只能在定时器关闭时，或紧接在秒脉冲上升沿之后进行（由于可能发生自动调节事件， SECONDS （RTCTIME<15:8>）字段为 00 时除外）。

注： 是否在误差值中包含晶振初始误差、温度造成的漂移和晶振老化造成的漂移，由用户自行决定。

SECONDS 寄存器写操作会复位校准的状态 （不是它的值）。如果刚刚发生调整，则会由于分钟计满返回而再次发生。

例 29-4： 更新 RTCC 校准值

```
/*
    The following code example will update the RTCC calibration.
*/



    int cal=0x3FD;                // 10 bits adjustment, -3 in value

    if(RTCCON&0x8000)
    {                             // RTCC is ON
        unsigned intt0, t1;
        do
        {
            t0=RTCTIME;
            t1=RTCTIME;
        }while(t0!=t1);           // read valid time value
        if((t0&0xFF)==00)
        {                         // we're at second 00, wait auto-adjust to be performed
            while(!(RTCCON&0x2)); // wait until second half...
        }
    }

    RTCCONCLR=0x03FF0000;         // clear the calibration
    RTCCONSET=cal;
```


29.4 闹钟

RTCC 模块提供了具有以下特性的闹钟功能：

- 可在半秒到一年的范围内配置
- 使用 ALRMEN 位 （RTCALRM<15>）使能
- 提供一次闹钟、重复闹钟，以及无限重复的闹钟

29.4.1 配置闹钟

闹钟功能通过 ALRMEN 位使能。

时间间隔根据闹钟掩码位 AMASK （RTCALRM<11:8>）的设置进行选择。 AMASK 位决定要触发闹钟的哪些位以及多少位必须和时钟值匹配 （见图 29-7）。

注：	在定时器值达到闹钟设置时，在设置闹钟中断之前会经过一个 RTCC 时钟周期。产生的结果就是，用户会在一个很短的时间内看到闹钟设置的定时器值，在此期间不会产生中断。

29.4.1.1 配置一次闹钟

发出闹钟之后，如果ARPT（RTCALRM<7:0>）= 0且CHIME（RTCALRM<14>）= 0，则ALRMEN位会自动清零。

例 29-5： 配置 RTCC 来产生每日一次的单次闹钟

```
/*
    The following code example will update the RTCC one-time alarm.
    Assumes the interrupts are disabled.
*/


    unsigned long alTime=0x16153300;// set time to 04 hr, 15 min, 33 sec
    unsigned long alDate=0x06102705;// set date to Friday 27 Oct 2006

                                    // turn off the alarm, chime and alarm repeats; clear
                                    // the alarm mask

    while(RTCALRM&0x1000);          // wait ALRMSYNC to be off
    RTCALRMCLR=0xCFFF;              // clear ALRMEN, CHIME, AMASK and ARPT;
    ALRMTIME=alTime;
    ALRMDATE=alDate;                // update the alarm time and date

    RTCALRMSET=0x8000|0x00000600;   // re-enable the alarm, set alarm mask at once per day
```

29.4.1.2 配置重复闹钟

除了提供一次闹钟之外，模块还可以配置为以预先配置的时间间隔重复闹钟。ARPT 寄存器中包含在使能闹钟后闹钟的重复次数。当 ARPT = 0 且 CHIME = 0 时，重复功能会被禁止，并且将仅产生单个闹钟脉冲。通过设置 ARPT = 0xFF，闹钟最多可以重复 256 次。

每次闹钟发出后， ARPT 寄存器都递减 1。寄存器的值达到 0 后，将产生最后一次闹钟；此后 ALRMEN 位将自动清零，闹钟将关闭。

例 29-6： 配置 RTCC 来产生每小时一次，重复十次的闹钟

```
/*
    The following code example will update the RTCC repeat alarm.
    Assumes the interrupts are disabled.
*/


    unsigned long alTime=0x23352300;  // set time to 23hr, 35 min, 23 sec
    unsigned long alDate=0x06111301;  // set date to Monday 13 Nov 2006

                                      // turn off the alarm, chime and alarm repeats; clear
                                      // the alarm mask
    while(RTCALRM&0x1000);            // wait ALRMSYNC to be off
    RTCALRMCLR=0xCFFF;                // clear the ALRMEN, CHIME, AMASK and ARPT;
    ALRMTIME=alTime;
    ALRMDATE=alDate;                  // update the alarm time and date
    RTCALRMSET=0x8000|0x0509;         // re-enable the alarm, set alarm mask at once per hour
                                      // for 10 times repeat
```

29.4.1.3 配置无限闹钟

要提供无限重复闹钟，可以使用 CHIME（RTCALRM<14>）位使能响铃功能。当 CHIME = 1 时，在执行最后一次重复之后，ARPT将从0x00计满返回至0xFF，并继续无限计数，而不是禁止闹钟。

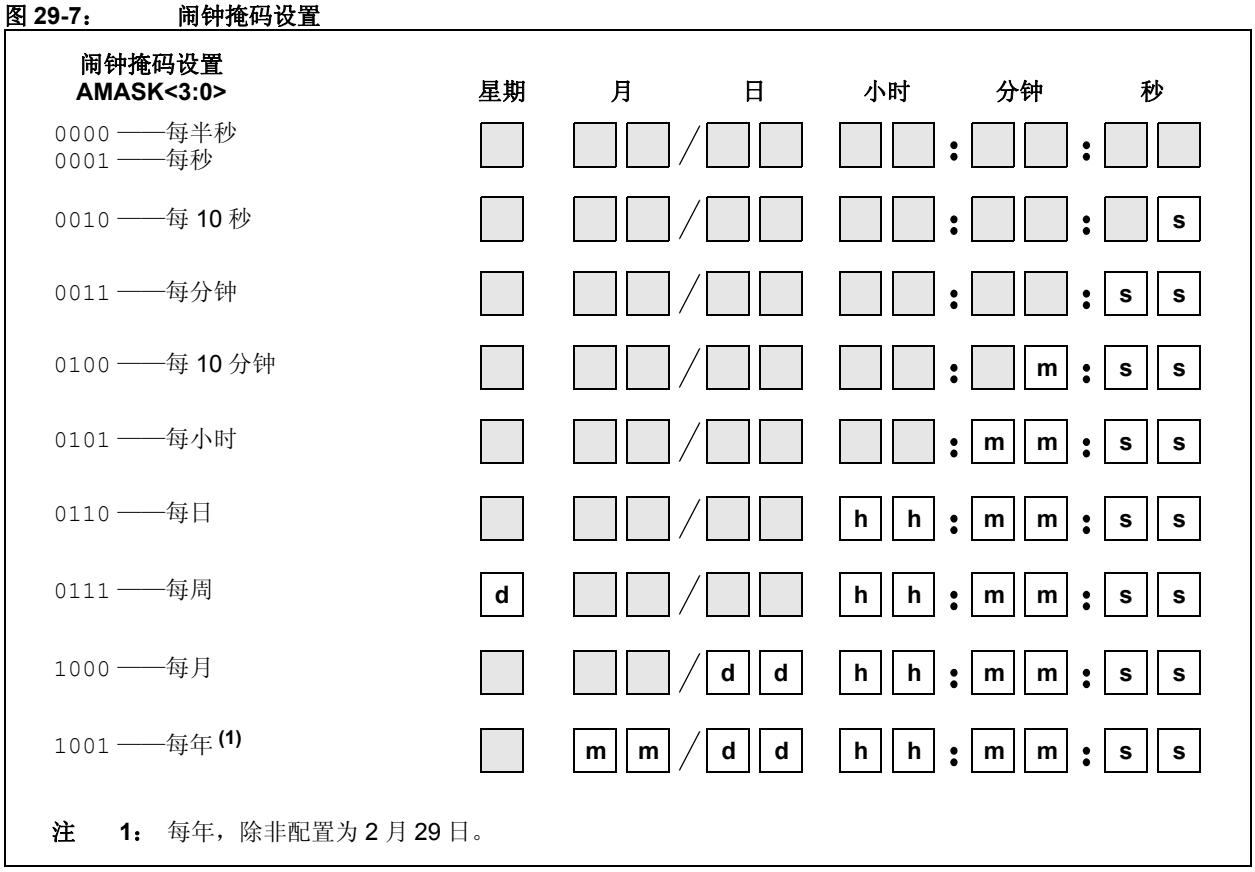
例 29-7： 配置 RTCC 来产生每日一次，无限次重复的闹钟

```
/*
    The following code example will update the RTCC indefinite alarm.
    Assumes the interrupts are disabled.
*/


    unsigned long alTime=0x23352300;  // set time to 23hr, 35 min, 23 sec
    unsigned long alDate=0x06111301;  // set date to Monday 13 Nov 2006

                                      // turn off the alarm, chime and alarm repeats; clear
                                      // the alarm mask
    while(RTCALRM&0x1000);            // wait ALRMSYNC to be off
    RTCALRMCLR=0xCFFF;                // clear ALRMEN, CHIME, AMASK, ARPT;
    ALRMTIME=alTime;
    ALRMDATE=alDate;                  // update the alarm time and date
    RTCALRMSET=0xC600;                // re-enable the alarm, set alarm mask at once per
                                      // hour, enable CHIME
```


图 29-7：　闹钟掩码设置



注　1： 每年，除非配置为 2 月 29 日。

29.4.2 闹钟中断

当RTCC定时器与闹钟寄存器匹配时，会产生闹钟事件。模块必须根据AMASK（RTCALRM<11:8>）位设置，只与时间 / 日期寄存器的无掩码部分进行匹配 （见图 29-7）。

每个闹钟事件发生时，都会产生中断。此外会提供闹钟脉冲输出，其频率是闹钟频率的一半。该输出与 RTCC 时钟完全同步，可用作其他外设的触发时钟。可在 RTCC 引脚上输出闹钟脉冲。输出脉冲是占空比为 50% 的时钟，频率为闹钟事件频率的一半 （见图 29-8）。只有使能闹钟时，脉冲才会有效（ALRMEN （RTCALRM<15>） = 1）。RTCC 输出引脚上的闹钟输出的初始值可使用 PIV 位 （RTCALRM<13>）设定。

RTCC 引脚也能输出秒时钟。用户可在 RTCC 模块产生的闹钟脉冲或秒时钟输出之间选择。RTSECSEL 位 （RTCCON<7>）在这两个输出之间进行选择。当 RTSECSEL = 0 时，选择闹钟脉冲。当 RTSECSEL = 1 时，选择秒时钟 （见图 29-9）。

注：	在闹钟使能 （ALRMEN = 1）时，更改闹钟时间、日期和闹钟寄存器 （RTCOE （RTCCON<0>）除外）中的任一寄存器会产生错误的闹钟事件，导致错误的闹钟中断。为了避免错误的闹钟事件，并对闹钟寄存器执行安全的写操作，只有在RTCC被禁止 （RTCCON<15> = 0）时，或者 ALRMSYNC （RTCALRM<12>） = 0 时，才能更改定时器和闹钟值。

图 29-8： 闹钟事件产生

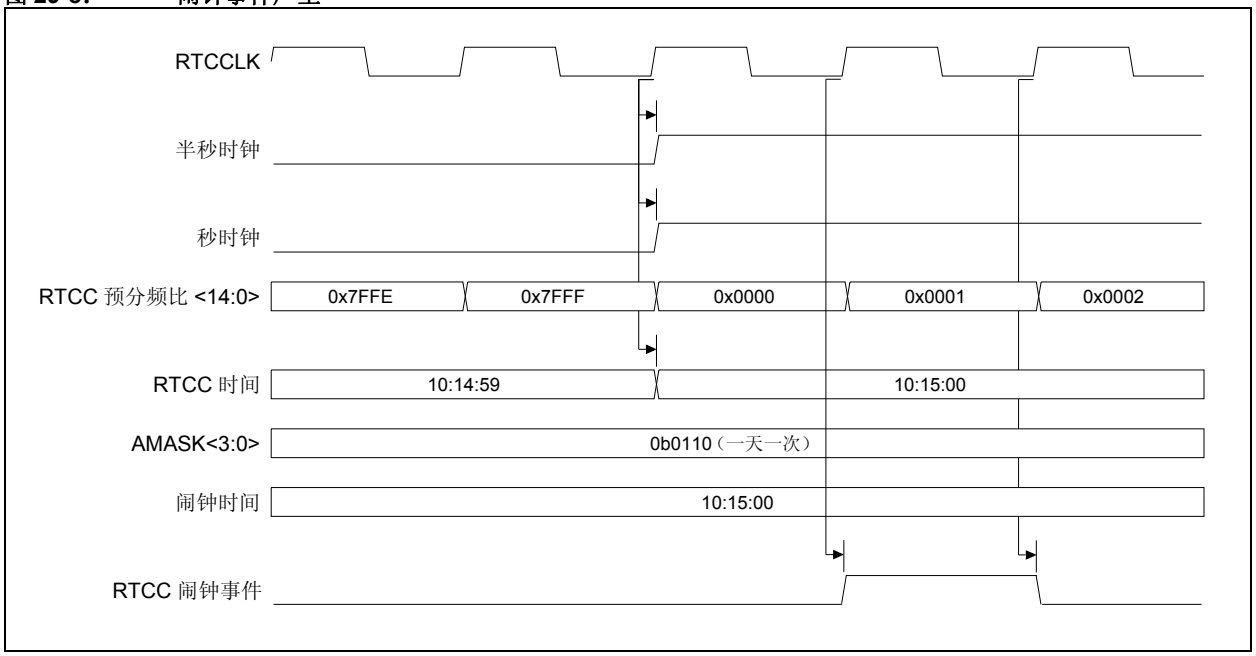
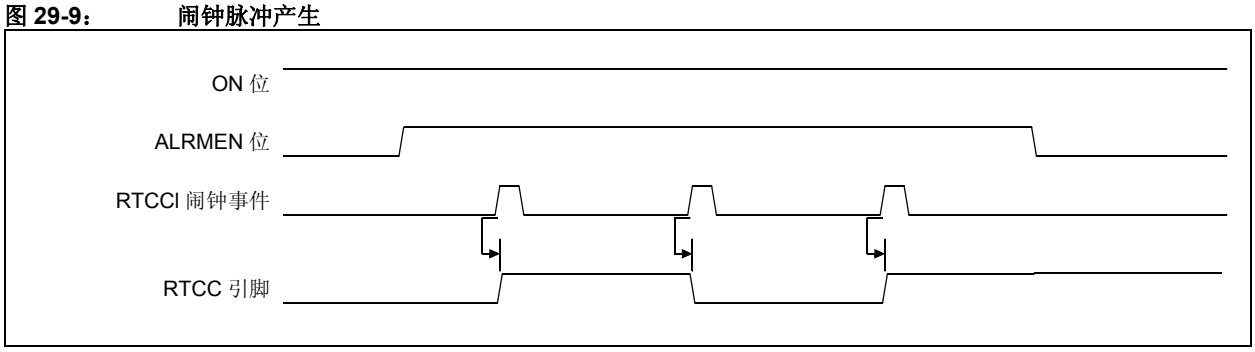


图 29-9：　　闹钟脉冲产生



29.5 中断

RTCC 器件可以产生一些中断，反映在 RTCC 定时器与闹钟寄存器匹配时发生的闹钟事件。模块会根据 AMASK （RTCALRM<11:8>）位设置，与时间 / 日期寄存器的无掩码部分进行匹配。

每个闹钟事件发生时，都能产生中断：

• 闹钟中断通过 RTCCIF （IFS1<15>）指示。该中断标志必须用软件清零。

为了允许 RTCC 中断，请使用相应的 RTCC 中断允许位：

• RTCCIE （IEC1<15>）

此外，还必须配置中断优先级位和中断子优先级位：

• RTCCIP （IPC8<28:26>）和 RTCCIS （IPC8<25:24>）

更多详细信息，请参见**第 8 章 “中断”**（DS61108）。

29.5.1 中断配置

RTCC 具有一个专用的中断标志位 RTCCIF 和相应的中断允许 / 屏蔽位 RTCCIE。 RTCCIE 用于允许或禁止 RTCC 中断源。存在一个特定的 RTCC 中断向量。

当 RTCC 闹钟寄存器与 RTCC 时间寄存器匹配时， RTCCIF 会置 1。

请注意， RTCCIF 位是否置 1 与相应允许位的状态无关。如果需要，可以用软件查询 IF 位。

RTCCIE 位用于定义在相应 RTCCIF 位置 1 时，中断控制器（INT）的行为。当 RTCCIE 位清零时， INT 模块不会为事件产生 CPU 中断。如果 RTCCIE 位置 1，则 INT 模块会在 RTCCIF 位置 1 时产生 CPU 中断 （受以下段落中概述的优先级和子优先级制约）。

处理特定中断的用户软件程序负责在服务程序完成之前清零相应的中断标志位。

RTCC 外设的优先级可以使用 RTCCIP<2:0> 位设置。该优先级定义了中断源将分配到的优先级组。优先级组值的范围为 7 （最高优先级）到 0 （不产生中断）。较高优先级组中的中断会抢占正在处理、但优先级较低的中断。

子优先级位用于设置中断源在优先级组中的优先级。子优先级 RTCCIS<1:0> 值的范围为 3 （最高优先级）到 0 （最低优先级）。处于相同优先级组，但具有更高子优先级值的中断不会抢占子优先级较低、但正在进行的中断。

优先级组和子优先级位让多个中断源可以共用相同的优先级和子优先级。如果在该配置下同时发生若干个中断，则中断源在优先级 / 子优先级组对中的自然顺序将决定所产生的中断。自然优先级基于中断源的向量编号。向量编号越小，中断的自然优先级就越高。在当前中断的中断标志清零之后，所有不按照自然顺序执行的中断会基于优先级、子优先级和自然顺序产生相应的中断。

产生允许的中断之后， CPU 将跳转到为该中断分配的向量处。该中断的向量编号与自然顺序编号相同。然后， CPU 将在向量地址处开始执行代码。该向量地址处的用户代码应执行特定于应用程序的操作、清零 RTCCIF （IFS1<15>）中断标志，然后退出。关于中断的更多信息，请参见**第 8 章 “中断”**（DS61108）中的向量地址表详细信息。

表 29-4： 各种偏移量的 RTCC 中断向量 （EBASE = 0x8000:0000）

中断	向量 / 自然顺序	IRQ 编号	向量地址 IntCtl.VS = 0x01	向量地址 IntCtl.VS = 0x02	向量地址 IntCtl.VS = 0x04	向量地址 IntCtl.VS = 0x08	向量地址 IntCtl.VS = 0x10
RTCC 闹钟	35	47	8000 0660	8000 0AC0	8000 1380	8000 2500	8000 4800

例 29-8： 允许中断的 RTCC 初始化代码示例

```
/*
    The following code example illustrates an RTCC initialization with interrupts enabled.
    When the RTCC alarm interrupt is generated, the cpu will jump to the vector assigned to
    RTCC interrupt.
*/
                                    // assume RTCC write is enabled i.e. RTCWREN (RTCCON<3>) =1;
    IEC1CLR=0x00008000;             // disable RTCC interrupts

    RTCCONCLR=0x8000;               // turn off the RTCC
    while(RTCCON&0x40);             // wait for clock to be turned off

    IFS1CLR=0x00008000;             // clear RTCC existing event
    IPC8CLR=0x1f000000;             // clear the priority
    IPC8SET=0x0d000000;             // Set IPL=3, subpriority 1
    IEC1SET=0x00008000;             // Enable RTCC interrupts

    RTCTIME=0x16153300;             // safe to update time to 16 hr, 15 min, 33 sec
    RTCDATE=0x06102705;             // update the date to Friday 27 Oct 2006

    RTCALRMCLR=0xCFFF;              // clear ALRMEN, CHIME, AMASK and ARPT;
    ALRMTIME=0x16154300;            // set alarm time to 16 hr, 15 min, 43 sec
    ALRMDATE=0x06102705;            // set alarm date to Friday 27 Oct 2006

    RTCALRMSET=0x8000|0x00000600; // re-enable the alarm, set alarm mask at once per day

    RTCCONSET=0x8000;               // turn on the RTCC
    while(!(RTCCON&0x40));          // wait for clock to be turned on
```

例 29-9： RTCC ISR 代码示例

```
/*
    The following code example demonstrates a simple interrupt service routine for RTCC
    interrupts.The user's code at this vector should perform any application specific
    operations and must clear the RTCC interrupt flag before exiting.
*/

void__ISR(_RTCC_VECTOR, ipl3) __RTCCInterrupt(void)
{
                                        // ... perform application specific operations
                                        // in response to the interrupt

    IFS1CLR=0x00008000;                 // be sure to clear RTCC interrupt flag
                                        // before exiting the service routine.
}
```

注： RTCC ISR 代码示例显示的是 MPLAB® C32 C 编译器的特定语法。关于对 ISR 的支持，请参见编译器手册。

29.6 节能和调试模式下的操作

注： 在本手册中，对于特定模块中使用的功耗模式和器件使用的功耗模式进行了区分；例如，比较器的 Sleep（休眠）模式和 CPU 的 SLEEP（休眠）模式。为了指示所期望功耗模式的类型，模块功耗模式使用大写字母加小写字母（Sleep，Idle，Debug）（休眠、空闲和调试）来表示，器件功耗模式使用全大写字母（SLEEP，IDLE，DEBUG）（休眠、空闲和调试）来表示。

29.6.1 SLEEP（休眠）模式下的 RTCC 操作

当器件进入 SLEEP（休眠）模式时，系统时钟被禁止。RTCC 和闹钟在 SLEEP（休眠）模式下继续工作。闹钟的操作不会受 SLEEP（休眠）模式影响。如果闹钟中断的优先级高于当前 CPU IPL，则闹钟事件可以唤醒 CPU。

29.6.2 IDLE（空闲）模式下的 RTCC 操作

当器件进入 IDLE（空闲）模式时，系统时钟源继续保持工作。RTCC 和闹钟在 IDLE（空闲）模式下继续工作。闹钟的操作不会受 IDLE（空闲）模式影响。如果闹钟中断的优先级高于当前 CPU IPL，则闹钟事件可以唤醒 CPU。

SIDL 位（RTCCON<13>）用于选择模块 IDLE（空闲）模式的行为。

- 如果 SIDL = 1，则到 RTCC 的 PBCLK 会被禁止。PBCLK 是以下对象的时钟源：AMASK（RTCALRM<11:8>）、CHIME（RTCALRM<14>）、ALRMTIME、ALRMDATE，以及为 RTCTIME 提供读取数据的同步器，以及一些其他位，诸如 ALRMSYNC（RTCALRM<12>）、ALRMEN（RTCALRM<15>）和 RTCSYNC（RTCCON<2>）。因而，SIDL 功能可以用于降低 RTCC 功耗，而不影响 RTCC 的功能。
- 如果 SIDL = 0，则模块将在 IDLE（空闲）模式下继续正常工作。

29.6.3 DEBUG（调试）模式下的 RTCC 操作

FRZ 位（RTCCON<14>）决定 CPU 在 DEBUG（调试）模式下执行调试异常代码（即，应用程序暂停）时，RTCC 模块是继续运行还是停止。当 FRZ = 0 时，即使应用程序在 DEBUG（调试）模式下暂停，RTCC 模块也会继续运行。当 FRZ = 1 且应用程序在 DEBUG（调试）模式下暂停时，模块将冻结其操作，并且不会更改 RTCC 模块的状态。预分频器和 RTCC 定时器不会递增。如果某个配置寄存器在正常情况下会导致模块状态在读操作时改变，则在冻结期间会禁止该功能。在 CPU 继续开始执行代码之后，模块将继续工作。

注： 只有 CPU 在调试异常模式下执行时，FRZ 位才可读写。在所有其他模式下，FRZ 位读为 0。如果 FRZ 位在 DEBUG（调试）模式期间发生改变，则只有退出当前调试异常模式并重新进入该模式之后，新值才会生效。在调试异常模式期间，在进入 DEBUG（调试）模式时 FRZ 位会读取外设状态。

29.7 各种复位的影响

29.7.1 器件复位

发生器件复位时，RTCALRM 寄存器强制进入其复位状态，导致闹钟被禁止（如果复位之前已使能）。但请注意，如果 RTCC 已使能，则发生器件复位后将继续工作。

29.7.2 上电复位

RTCTIME 和 RTCDATE 寄存器不受 POR 影响。POR 强制器件进入无效状态。器件退出 POR 状态后，时钟寄存器应重新装入所需值。

定时器预分频值只能通过写入 SECONDS 寄存器（RTCTIME<15:8>）复位。器件复位不会影响预分频器。

29.7.3 看门狗定时器复位

看门狗定时器复位等效于器件复位。

29.7.4 ON 位的影响

当 ON 位（RTCCON<15>）= 0 时，RTCSYNC（RTCCON<2>）、HALFSEC（RTCCON<1>）和 ALRMSYNC （RTCALRM<4>）等位会异步复位，并保持复位状态。此外， RTCC 引脚输出由 RTCOE 位 （RTCCON<0>）决定，通过 ON 位进行门控。

29.8 使用 RTCC 模块的外设

没有任何其他外设模块使用 RTCC。

29.9 I/O 引脚控制

使能 RTCC 模块时会配置 I/O 引脚方向。使能、配置 RTCC 模块和使能模块输出时， I/O 引脚方向会正确配置为数字输出。

表 29-5： 用于 RTCC 模块的 I/O 引脚配置

模块引脚控制所需的设置							
IO 引脚名称	**必需**	**模块控制**	**位域**	**TRIS(4)**	**引脚类型**	**缓冲器类型**	**说明**
RTCC	是 (1)	ON 和 RTCOE(2)	RTSECSEL = 1	X	O	CMOS	RTCC 秒时钟
RTCC	是 (1)	ON 和 RTCOE(2)	RTSECSEL = 0 以及 ALRMEN 和 PIV(3)	X	O	CMOS	RTCC 闹钟脉冲

图注： CMOS = CMOS 兼容输入或输出
ST = 带 CMOS 电平的施密特触发器输入
I = 输入
O = 输出

注 1： 只有在需要秒时钟或闹钟脉冲输出时，才需要 RTCC 引脚。其他情况下，该引脚可以用作通用 I/O，并要求用户设置相应的 TRIS 控制寄存器位。

2： 要验证RTCC引脚的输出功能（秒时钟或闹钟脉冲），总是需要ON（RTCCON<15>）和RTCOE（RTCCON<0>）位。

3： 当 RTSECSEL（RTCCON<7>）= 0 时，RTCC 引脚输出为闹钟脉冲。如果 ALRMEN（RTCALRM<15>）= 0，则 PIV（RTCALRM<13>）会选择 RTCC 引脚上的值。当 ALRMEN = 1 时，RTCC 引脚会反映闹钟脉冲的状态。

4： TRIS 位的设置无关。

29.10 设计技巧

问 1： ***如果不使用 RTCC 模块的 RTCC 输出，该 I/O 引脚是否可用作通用 I/O 引脚？***

答 1： 可以。如果不需要输出秒时钟或闹钟脉冲，在禁止 RTCC 输出（RTCCON<0> = 0）的情况下，可以使用 RTCC 引脚作为通用 I/O 引脚。请注意，在用作通用 I/O 引脚时，用户负责为输入或输出配置相应的数据方向寄存器 （TRIS）。

问 2： ***我该如何确保在读取 RTCC 时间值时获得正确的值，而不受计满返回 （秒至分，分至小时）影响？***

答 2： 读取正确当前时间的最简单方式是对 RTCTIME 寄存器执行两次读操作。在两个连续读数相同时，说明得到的时间值是正确的。读取 RTCDATE 寄存器时，也是如此。

问 3： ***在设置特定日期（例如2006年1月18日）时，RTCC器件是否会自动计算星期几？***

答 3： 不会，器件不会自动执行该计算。在写入 RTCDATE 寄存器时，必须为 WDAY01 字段 （RTCDATE<3:0>）提供一个有效值，即提供 3 来代表 1 月 18 日星期三。但是，从此时开始， RTCC 器件会负责正确地更新星期几。

问 4： ***器件是否会自动执行闰年计算，还是我必须在 RTCC 日期中执行一些修正？***

答 4： RTCC 器件会自动执行闰年检测。在 2000 至 2099 的年份范围内，不需要执行更新。但是，在 RTCDATE 寄存器中设定日期时，必须输入一个有效日期。例如，不要将 RTCC 的日期设定为 2007 年 2 月 29 日。

问 5： ***我是否可以自由地写入 RTCTIME 和 RTCDATE 寄存器来更新当前时间或日期？***

答 5： 简单地说，不行；您不能直接写入 RTCTIME 和 RTCDATE 寄存器。实际上，如果 RTCC 被禁止 （RTCCON<15> = 0），则可以在任意时刻更新时间和日期值。但是，如果 RTCC 开启，则必须采取进一步的预防措施。在一个安全时间窗中，可以安全地对时间和日期寄存器执行写操作。该窗口通过 RTCSYNC 位（RTCCON<2>）指示。对于 RTCTIME 或 RTCDATE 寄存器的任意更新都应在 RTCSYNC = 0 时进行。（在 RTCC 时钟为高电平时，硬件实际上会忽略在计满返回期间对时间和日期寄存器发生的写操作，所以写操作不会被检测到） 。不仅如此，如果使能了闹钟，并且 AMASK 对应于半秒 （RTCALRM<11:8>），则对于 RTCTIME 和 RTCDATE 的任意更新都应在 ALRMSYNC （RTCALRM<12>）为 0 时发生。这可以确保闹钟触发机制正确工作，不会产生虚假的闹钟事件。

注： 为了能够更新 RTCTIME 和 RTCDATE 寄存器，必须将 RTCWREN（RTCCON<3>）置 1。

在尝试更新 RTCTIME 时通常应禁止中断。如果不禁止，则无法确保对 RTCTIME 寄存器的写操作在 RTCSYNC/ALRMSYNC 清零时发生。

对系统时间和日期执行更新的另一种方式是先关闭 RTCC，执行写操作，然后再次开启 RTCC。

问 6： *我是否可以写入 ALRMTIME 和 ALRMDATE 寄存器，以自由地更新闹钟时间或日期？*

答 6： 简单地说，不行；您不能直接更新闹钟时间和日期寄存器。需要执行以下步骤：

- 如果 RTCC 被禁止 （即， ON （RTCCON<15>） = 0），则可以在任意时刻对 ALRMTIME 和 ALRMDATE 执行写操作。

否则，写操作只能在 ALRMSYNC （RTCALRM<12>） = 0 时进行。

注： 在尝试更新闹钟时间和日期时，通常应禁止中断。如果不禁止，则无法确保对 ALRMTIME/ALRMDATE 寄存器的写操作在 ALRMSYNC 清零时发生。

对闹钟时间和日期执行更新的另一种方式是先关闭 RTCC，执行写操作，然后再次开启 RTCC。请注意，这种方式会对时序精度产生影响，因为 RTCC 在停止时不会进行计数。

同一方法适用于 RTCALRM 寄存器的可写字段：CHIME （RTCALRM<14>） 、AMASK （RTCALRM<11:8>） 、ALRMEN （RTCALRM<15>） 、ARPT （RTCALRM<7:0>）和 PIV （RTCALRM<13>） 。如果 RTCC 开启，则只能在 ALRMSYNC = 0 时进行写操作。

问 7： *我是否可以自由地翻转 RTCCON 寄存器中的 RTCWREN 位？*

答 7： 您总是可以清零 RTCWREN 位 （RTCCON<3>）。但是，为了使能对于 RTCCTIME 和 RTCCDATE 寄存器的写操作，必须执行正确的操作序列。详情请参见**第 29.3.9 节“写锁定”**。

29.11 相关应用笔记

本节列出了与手册本章内容相关的应用笔记。这些应用笔记可能并不是专为 PIC32MX 器件系列而编写的，但其概念是相近的，通过适当修改并受到一定限制即可使用。当前与实时时钟和日历（RTCC）模块相关的应用笔记有：

标题	应用笔记编号
目前没有相关的应用笔记。	N/A

注： 如需获取更多 PIC32MX 系列器件的应用笔记和代码示例，请访问 Microchip 网站（www.microchip.com）。

29.12 版本历史

版本 A （2007 年 10 月）

这是本文档的初始版本。

版本 B （2007 年 10 月）

更新了文档 （删除了 “机密”状态）。

版本 C （2008 年 4 月）

将状态修改为 “初稿”； 将 U-0 修改为 r-x。

版本 D （2008 年 6 月）

修改了寄存器 29-1 的 bit 14 ； 修改了寄存器 29-26 和 29-27 的注释； 修改了例 29-1 和 29-9 ； 将保留位从 “保持为”更改为 “写入”； 为 ON 位 （RTCCON 寄存器）增加了注释。

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

ISBN：978-1-60932-526-8

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2002 认证。公司在 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://support.microchip.com
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

克里夫兰 Cleveland
Independence, OH
Tel: 1-216-447-0464
Fax: 1-216-447-0643

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Farmington Hills, MI
Tel: 1-248-538-2250
Fax: 1-248-538-2260

科科莫 Kokomo
Kokomo, IN
Tel: 1-765-864-8360
Fax: 1-765-864-8387

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

圣克拉拉 Santa Clara
Santa Clara, CA
Tel: 1-408-961-6444
Fax: 1-408-961-6445

加拿大多伦多 Toronto
Mississauga, Ontario,
Canada
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 – 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 – 重庆
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

中国 – 香港特别行政区
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

中国 – 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 – 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 – 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 – 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 – 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 – 西安
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

中国 – 厦门
Tel: 86-592-238-8138
Fax: 86-592-238-8130

中国 – 珠海
Tel: 86-756-321-0040
Fax: 86-756-321-0049

台湾地区 – 高雄
Tel: 886-7-213-7830
Fax: 886-7-330-9305

台湾地区 – 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

亚太地区

台湾地区 – 新竹
Tel: 886-3-6578-300
Fax: 886-3-6578-370

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

印度 India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

印度 India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韩国 Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

07/15/10