

## DS1302 涓流充电时钟保持芯片的原理与应用

**摘要：**本文概括介绍了 DS1302 时钟芯片的特点和基本组成，通过实例详细说明了有关功能的应用软件。关于 DS1302 各寄存器的详细位控功能请参考 DALLAS（达拉斯）公司的相应产品资料。

### 概述

DS1302 是 DALLAS 公司推出的涓流充电时钟芯片，内含有一个实时时钟/日历和 31 字节静态 RAM，通过简单的串行接口与单片机进行通信。实时时钟/日历电路提供秒、分、时、日、日期、月、年的信息，每月的天数和闰年的天数可自动调整，时钟操作可通过 AM/PM 指示决定采用 24 或 12 小时格式。DS1302 与单片机之间能简单地采用同步串行的方式进行通信，仅需用到三个口线：(1) RES（复位），(2) I/O（数据线），(3) SCLK（串行时钟）。时钟/RAM 的读/写数据以一个字节或多达 31 个字节的字符组方式通信。DS1302 工作时功耗很低，保持数据和时钟信息时功率小于 1mW。

DS1302 是由 DS1202 改进而来，增加了以下的特性：双电源管脚用于主电源和备份电源供应，Vcc1 为可编程涓流充电电源，附加七个字节存储器。它广泛应用于电话、传真、便携式仪器以及电池供电的仪器仪表等产品领域。下面将主要的性能指标作一综合：

- 实时时钟具有能计算 2100 年之前的秒、分、时、日、日期、星期、月、年的能力，还有闰年调整的能力
- 31×8 位暂存数据存储 RAM
- 串行 I/O 口方式使得管脚数量最少
- 宽范围工作电压：2.0~5.5V
- 工作电流：2.0V 时，小于 300nA
- 读/写时钟或 RAM 数据时，有两种传送方式：单字节传送和多字节传送（字符组方式）
- 8 脚 DIP 封装或可选的 8 脚 SOIC 封装（根据表面装配）
- 简单 3 线接口
- 与 TTL 兼容（Vcc=5V）
- 可选工业级温度范围：-40℃~+85℃
- 与 DS1202 兼容
- 在 DS1202 基础上增加的特性
  - 对 Vcc1 有可选的涓流充电能力
  - 双电源管用于主电源和备份电源供应
  - 备份电源管脚可由电池或大容量电容输入
  - 附加的 7 字节暂存存储器

### 1. DS1302 的基本组成和工作原理

DS1302 的管脚排列及描述如下图及表所示

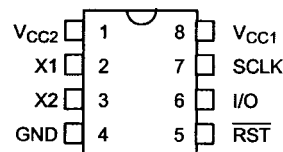
## 管脚描述

X1, X2	——32.768KHz 晶振管脚
GND	——地
RST	——复位脚
I/O	——数据输入/输出引脚
SCLK	——串行时钟
Vcc1,Vcc2	——电源供电管脚

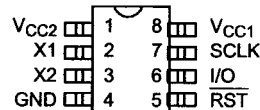
## 订单信息

部分#	描述
DS1302	串行时钟芯片，8 脚 DIP
DS1302S	串行时钟芯片，8 脚 SOIC（200mil）
DS1302Z	串行时钟芯片，8 脚 SOIC（150mil）

## 管脚配置



DS1302  
8-PIN DIP (300 MIL)



DS1302S 8-PIN SOIC (200 MIL)  
DS1302Z 8-PIN SOIC (150 MIL)

## 2. DS1302 内部寄存器

CH: 时钟停止位	寄存器 2 的第 7 位: 12/24 小时标志
CH=0 振荡器工作允许	bit7=1, 12 小时模式
CH=1 振荡器停止	bit7=0, 24 小时模式
WP: 写保护位	寄存器 2 的第 5 位: AM/PM 定义
WP=0 寄存器数据能够写入	AP=1 下午模式
WP=1 寄存器数据不能写入	AP=0 上午模式
TCS: 涓流充电选择	DS: 二极管选择位
TCS=1010 使能涓流充电	DS=01 选择一个二极管
TCS=其它 禁止涓流充电	DS=10 选择两个二极管
DS=00 或 11, 即使 TCS=1010, 充电功能也被禁止	

DS 位	电阻	典型位
00	没有	没有
01	R1	2K $\Omega$
10	R2	4K $\Omega$
11	R3	8K $\Omega$

### A. 时钟

秒	7	6	5	4	3	2	1	0		
	1	0	0	0	0	0	0	RD	W	
分										
	1	0	0	0	0	0	1	RD	W	
小时										
	1	0	0	0	0	1	0	RD	W	
日										
	1	0	0	0	0	1	1	RD	W	
月										
	1	0	0	0	1	0	0	RD	W	
星期										
	1	0	0	0	1	0	1	RD	W	
年										
	1	0	0	0	1	1	0	RD	W	
控制										
	1	0	0	0	1	1	1	RD	W	
涓流充电										
	1	0	0	1	0	0	0	RD	W	
时钟多字节										
	1	0	1	1	1	1	1	RD	W	

00-59	CH	10 秒	秒			
00-59	0	10 分	分			
01-12 00-23	12/ 24	0	10 A/P	小时	小时	
01-28/29 01-30 01-31	0	0	10 日	日		
01-12	0	0	0	10 M	月	
01-07	0	0	0	0	0	星期
00-99	10 年			年		
WP 0 0 0 0 0 0 0						
TCS TCS TCS TCS DS DS RS RS						

### B. RAM

RAM 数据 0

1	1	0	0	0	0	0	RD / $\overline{W}$
---	---	---	---	---	---	---	---------------------

RAM 数据 30

1	1	1	1	1	1	0	RD / $\overline{W}$
---	---	---	---	---	---	---	---------------------

RAM 字符组

1	1	1	1	1	1	1	RD / $\overline{W}$
---	---	---	---	---	---	---	---------------------

RAM 数据 0

--	--	--	--	--	--	--	--

RAM 数据 30

--	--	--	--	--	--	--	--

## DS1302 与微控制器的接口软件及功能应用举例

下面首先给出基本的接口软件，然后举例说明各种功能的应用。

### 1. 写保护寄存器操作

当写保护寄存器的最高位为 0 时，允许数据写入寄存器，写保护寄存器可以通过命令字节 8E、8F 来规定禁止写入/读出。写保护位不能在多字节传送模式下写入。

Write Enable:

```
MOV    Command, #8Eh    ;命令字节为 8E
MOV    ByteCnt, #1      ;单字节传送模式
MOV    R0, #XmtDat      ;数据地址覆给 R0
MOV    XmtDat, #00h     ;数据内容为 0（写入允许）
```

```
ACALL    Send_Byte      ; 调用写入数据子程序
RET      ; 返回调用本子程序处
```

当写保护寄存器的最高位为 1 时，禁止数据写入寄存器，

Write\_Disable:

```
MOV      Command, #8Eh   ; 命令字节为 8E
MOV      ByteCnt, #1     ; 单字节传送模式
MOV      R0, #XmtDat     ; 数据地址覆给 R0
MOV      XmtDat, #80h    ; 数据内容为 80h（禁止写入）
ACALL    Send_Byte      ; 调用写入数据子程序
RET      ; 返回调用本子程序处
```

以上程序调用了基本数据发送(Send\_Byte)模块及一些内存单元定义，其源程序清单在附录中给出。下面的程序亦使用了这个模块。

## 2. 时钟停止位操作

当把秒寄存器的第 7 位（时钟停止位）设置为 0 时，起动时钟开始。

Osc\_Enable:

```
MOV      Command, #80h   ; 命令字节为 80
MOV      ByteCnt, #1     ; 单字节传送模式
MOV      R0, #XmtDat     ; 数据地址覆给 R0
MOV      XmtDat, #00h    ; 数据内容为 0（振荡器工作允许）
ACALL    Send_Byte      ; 调用写入数据子程序
RET      ; 返回调用本子程序处
```

当把秒寄存器的第 7 位（时钟停止位）设置为 1 时，时钟振荡器停止，HT1380 进入低功耗方式，

Osc\_Disable:

```
MOV      Command, #80h   ; 命令字节为 80
MOV      ByteCnt, #1     ; 单字节传送模式
MOV      R0, #XmtDat     ; 数据地址覆给 R0
MOV      XmtDat, #80h    ; 数据内容为 80h（振荡器停止）
ACALL    Send_Byte      ; 调用写入数据子程序
RET      ; 返回调用本子程序处
```

## 3. 多字节传送方式

当命令字节为 BE 或 BF 时，DS1302 工作在多字节传送模式，8 个时钟/日历寄存器从寄存器 0 地址开始连续读写从 0 位开始的数据。当命令字节为 FE 或 FF 时，DS1302 工作在多字节 RAM 传送模式，31 个 RAM 寄存器从 0 地址开始连续读写从 0 位开始的数据。

例如：写入 00 年、6 月 21 日、星期三、13 时、59 分、59 秒，程序设置如下：

Write\_Multiplebyte:

```
MOV      Command, #0BEh  ; 命令字节为 BEh
MOV      ByteCnt, #8     ; 多字节写入模式（此模块为 8 个）
MOV      R0, #XmtDat     ; 数据地址覆给 R0
MOV      XmtDat, #59h    ; 秒单元内容为 59h
```

```
MOV    XmtDat+1, #59h    ; 分单元内容为 59h
MOV    XmtDat+2, #13h    ; 时单元内容为 13h
MOV    XmtDat+3, #21h    ; 日期单元内容为 21h
MOV    XmtDat+4, #06h    ; 月单元内容为 06h
MOV    XmtDat+5, #03h    ; 星期单元内容为 03h
MOV    XmtDat+6, #0      ; 年单元内容为 00h
MOV    XmtDat+7, #0      ; 写保护单元内容为 00h
ACALL  Send_Byte         ; 调用写入数据子程序
RET                                          ; 返回调用本子程序处
```

读出寄存器 0-7 的内容，程序设置如下：

Read\_Multiplebyte:

```
MOV    Command, #0BFh    ; 命令字节为 BFh
MOV    ByteCnt, #8        ; 多字节读出模式（此模块为 8 个）
MOV    R1, #RcvDat       ; 数据地址覆给 R1
ACALL  Receive_Byte      ; 调用读出数据子程序
RET                                          ; 返回调用本子程序处
```

以上程序调用了基本数据接收(Receive\_Byte)模块及一些内存单元定义，其源程序清单在附录中给出。下面的程序亦使用了这个模块。

#### 4. 单字节传送方式

例如：写入 8 时（12 小时模式），程序设置如下：

Write\_Singlebyte:

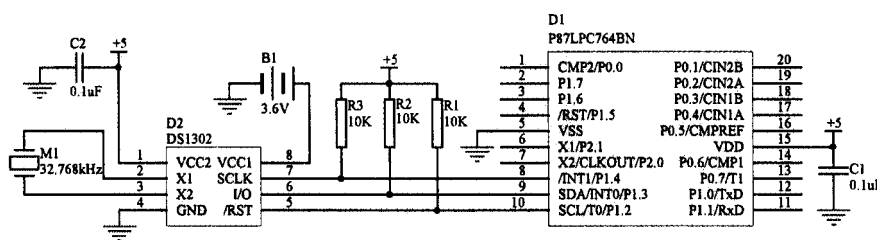
```
MOV    Command, #84h      ; 命令字节为 84h
MOV    ByteCnt, #1        ; 单字节传送模式
MOV    R0, #XmtDat        ; 数据地址覆给 R0
MOV    XmtDat, #88h       ; 数据内容为 88h
ACALL  Send_Byte          ; 调用写入数据子程序
RET                                          ; 返回调用本子程序处
```

上面所列出的程序模块“Write\_Enable”、“Write\_Disable”、“Osc\_Enable”、“Osc\_Disable”与单字节写入模块“Write\_Singlebyte”的程序架构完全相同，仅只是几个入口参数不同，本文是为了强调功能使用的不同才将其分为不同模块，另外，与涓流充电相关的设定也是单字节操作方式，这里就不再单独列出，用户在使用中可灵活简略。

下面模块举例说明如何单字节读出“小时”单元的内容。

Read\_Singlebyte:

```
MOV    Command, #85h      ; 命令字节为 85h
MOV    ByteCnt, #1        ; 单字节传送模式
MOV    R1, #RcvDat       ; 数据地址覆给 R1
ACALL  Receive_Byte      ; 调用读出数据子程序
RET                                          ; 返回调用本子程序处
```



```

        SETB    RST          ;复位引脚为高电平，逻辑控制有效
        NOP
        MOV     A, Command    ; 准备发送命令字节
        MOV     BitCnt, #08h  ; 传送位数为 8
S_Byte0:
        RRC     A             ; 将最低位传送给进位位 C
        MOV     IO_DATA, C    ; 位传送至数据总线
        NOP
        SETB    SCLK          ; 时钟上升沿，发送数据有效
        NOP
        CLR     SCLK          ; 清时钟总线
        DJNZ    BitCnt, S_Byte0 ; 位传送未完则继续
        NOP
S_Byte1:                                ; 准备发送数据
        MOV     A, @R0        ; 传送数据，过程与传送命令相同
        MOV     BitCnt, #08h
S_Byte2:
        RRC     A
        MOV     IO_DATA, C
        NOP
        SETB    SCLK
        NOP
        CLR     SCLK
        DJNZ    BitCnt, S_Byte2
        INC     R0            ; 发送数据的内存地址加 1
        DJNZ    ByteCnt, S_Byte1 ; 字节传送未完则继续
        NOP
        CLR     RST          ; 逻辑操作完毕，清 RST
        RET

```

\*\*\*\*\*

;接收数据程序;

;名称:Receive\_Byte

;描述:从被控器 DS1302 接收 (ByteCnt) 个字节数据

;命令字节地址在 Command 中

;所接收数据的字节数在 ByteCnt 中，接收的数据在 RcvDat 缓冲区中

\*\*\*\*\*

Receive\_Byte:

```

        CLR     RST          ;复位引脚为低电平，所有数据传送终止
        NOP
        CLR     SCLK          ; 清时钟总线
        NOP
        SETB    RST          ;复位引脚为高电平，逻辑控制有效
        MOV     A, Command    ; 准备发送命令字节

```

```

MOV    BitCnt, #08h          ; 传送位数为 8
R_Byte0:
RRC    A                     ; 将最低位传送给进位位 C
MOV    IO_DATA, C            ; 位传送至数据总线
NOP
SETB   SCLK                  ; 时钟上升沿，发送数据有效
NOP
CLR    SCLK                  ; 清时钟总线
DJNZ   BitCnt, R_Byte0       ; 位传送未完毕则继续
NOP
R_Byte1:                      ; 准备接收数据
CLR    A                     ; 清类加器
CLR    C                     ; 清进位位 C
MOV    BitCnt, #08h          ; 接收位数为 8
R_Byte2:
NOP
MOV    C, IO_DATA            ; 数据总线上的数据传送给 C
RRC    A                     ; 从最低位接收数据
SETB   SCLK                  ; 时钟总线置高
NOP
CLR    SCLK                  ; 时钟下降沿接收数据有效
DJNZ   BitCnt, R_Byte2       ; 位接收未完毕则继续
MOV    @R1, A                 ; 接收到的完整数据字节放入接收内存缓冲区
INC    R1                    ; 接收数据的内存地址加 1
DJNZ   ByteCnt, R_Byte1      ; 字节接收未完毕则继续
NOP
CLR    RST                   ; 逻辑操作完毕，清 RST
RET
;=====
END

```