查看: 28151 | 回复: 23

电波校时钟 [复制链接]

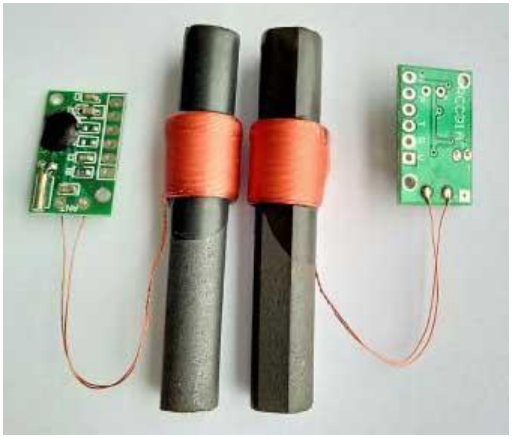
ID:238754 发表于 2017-10-12 01:49 | 显示全部楼层

本内容出自：智慧帽diy（cleverhat）

简介: 手头有一个项目，需要一个“精确”的频率计，作为DIY一族，准备动手制作一个。在制作频率计之前，需要一个能精确测量晶振频率的东西。之前尝试过NTP对时，发现精度太差，不好使。所以准备改用电波对时。百度搜了一下，动手做电波表的朋友还真不少，但讲得不够详细，解码部分还是“保密”的。只好尝试自立更生了。

BPC电波钟模块 电波对时需要一个接收头，接收授时中心发出的电波信号。我们国家的授时信号频率为68.5Khz,发射台在商丘。 接收头实际上是一个窄带信号接收器，其带宽只有几个赫兹，通常采用晶体滤波器来限制接收带宽。由于工作频率比较低，放大部分是比较 容易设计的，有一定无线电基础的都可以做出来。

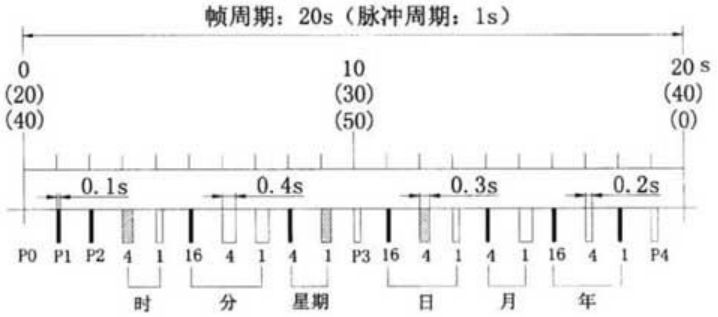
这里我们直接从某宝上买来一个完整的电波钟模块。花了15大洋，省了很多事。



BPC模块上用到的引脚有4条。

- VCC :电源，1.5~3.5V
- GND :地
- SIG :BPC授时信号输出
- EN :模块使能(低电平使能，高电平关闭模块)

BPC解码和校时 电波钟模块输出的BPC信号如下图，1分钟包含3个帧，一个BPC帧的周期为20秒，除了第"0"秒外，其余19秒每秒一个 脉冲。方波秒脉冲有0.1S，0.2S，0.3S，0.4S四种脉冲宽度状态，分别表示四进制的0, 1, 2, 3，现有的时间编码都以二进制表示时间信息， 是为了采用微处理器解码方便。但四进制只是数值的一种表示方式，并不影响微处理器把它作为二进制处理，或者采取简单的变换就可将1位 四进制数变成2位二进制数。



- P0设在每分钟0，20， 40秒，以缺少秒脉冲使帧与帧隔开，同时作为帧起始预告。
- P1为帧标志，P1 = 0表示帧起于第1秒，P1 = 1表示帧起始于21秒，P1 = 2表示帧起始于41秒。帧标志是必需的，它用来确定整分的起始。 例如：当接收完一组包含着"10时38分"的时间编码时，如果帧标志标明该帧为第二帧，就可以把下一帧的P1位置标定为10时38分41秒， 再过20秒便是10时39分的起始。
- P2为预留位。用于需要扩充信息。
- 时&分表示了时间
- 其他各位数据在本案中没有用到，不做详细说明

解码MCU采用了TI公司的MSP430G2211IPW14,MSP系列的MCU以低功耗著称，非常适合于电池供电的应用。本例中， MCU大部分时间工作在约20uA的低功耗模式下。

BPC解码软件使用了MSP430G2211的TimeA，TimeA的计数器由32768Hz的晶振提供时钟，从0~0xFFFF循环计数。每2秒 循环一周。BPC信号接在MCU的中断请求上，在上下跳变沿均产生中断，中断服务程序中读取TimerA计数器。根据脉冲宽度解码BPC编码的信号。 由于BPC信号的最小脉宽为0.1秒，软件中还加入了滤波处理，可以滤除脉宽较窄的干扰信号。

```
01.    if (MCU_INT_GET(BPC_SIGNAL_PORT, BPC_SIGNAL_PIN)) {
02.      static uint16 wPrevToggle = 0; //前一次跳变时刻
03.      static uint16 wLastPulseWidth = 0;  //前一次脉冲宽度
04.      static uint8  bLastPulsePolarity = 0; //前一次脉冲极性(1:正脉冲，0：负脉冲)
05.      static uint8  ucBpcBitPos = 0xFF; //BPC解码位置，0xFF表示解码状态机复位
06.      static uint8  ucBpc[2]; //BPC码数据,只取包含"时:分:秒"信息的前面2字节。
07.      uint16 wCurToggle; //本次跳变时刻
08.      uint16 wCurPulseWidth; //本次脉冲宽度
09.      uint16 wBpcSecond; //BPC解码得到的时间秒数
10.      uint8 bCurPulsePolarity; //本次脉冲极性(1:正脉冲，0：负脉冲)
11.      uint8 ucTmp;
12.
13.      MCU_INT_XOR_EDGE(BPC_SIGNAL_PORT, BPC_SIGNAL_PIN);
14.      MCU_INT_CLEAR(BPC_SIGNAL_PORT, BPC_SIGNAL_PIN);
15.      //正跳变，表示的是负脉冲(结束)
16.      bCurPulsePolarity = MCU_INT_GET_EDGE(BPC_SIGNAL_PORT, BPC_SIGNAL_PIN) ? 0 : 1;
17.      wCurToggle = GetCurTimerA();
18.      wCurPulseWidth =  wCurToggle - wPrevToggle;
19.      if ((wCurPulseWidth < PULSE_FILTER_OUT) || (bCurPulsePolarity == bLastPulsePolarity)) {
20.        //本次跳变脉宽过短，将当作干扰毛刺被过滤掉，脉宽同前一次合并
21.        //本次脉宽极性同前一次相同(跟在干扰毛刺后)，脉宽也同前一次合并
22.        wLastPulseWidth += wCurPulseWidth;
23.      } else {
24.        //前一次脉宽数据有效，可以处理了。
25.        if (!bLastPulsePolarity && (wLastPulseWidth > PULSE_10ms * 100)) {
26.          //每帧数据头部有1秒时间的空档期，表示帧起始
27.          ucBpcBitPos = 14; //只用到前14bit数据
28.          ucBpc[0] = ucBpc[1] = 0;
29.        } else if (ucBpcBitPos != 0xFF) {
30.          if (!bLastPulsePolarity) {
31.            //负脉冲
32.            if (wLastPulseWidth < PULSE_10ms * 55)
33.              //BPC编码正脉冲宽度最小600ms,<550ms为非法，解码状态机复位
34.              ucBpcBitPos = 0xFF;
35.          }
36.          else {
37.            //正脉冲
38.            ucTmp = 0xFF;
39.            if (wLastPulseWidth < PULSE_10ms * 45) {
```

```
40.              if (wLastPulseWidth > PULSE_10ms * 35) //400ms脉宽
41.                ucTmp = 3;
42.              else if (wLastPulseWidth > PULSE_10ms * 25) //300ms脉宽
43.                ucTmp = 2;
44.              else if (wLastPulseWidth > PULSE_10ms * 15) //200ms脉宽
45.                ucTmp = 1;
46.              else if (wLastPulseWidth > PULSE_10ms * 5)  //100ms脉宽
47.                ucTmp = 0;
48.            }
49.            if (ucTmp == 0xFF) {
50.              //脉宽非法，解码状态机复位
51.              ucBpcBitPos = 0xFF;
52.            }
53.            else {
54.              //保存合法数据
55.              ucBpcBitPos -= 2;
56.              ucBpc[ucBpcBitPos >> 3] |= ucTmp << (ucBpcBitPos & 0x07);
57.              if (ucBpcBitPos == 0) { //一帧数据接收完成
58.                ucBpcBitPos = 0xFF; //解码状态机复位，等待下次数据
59.                wBpcSecond = ((ucBpc[1] << 2) | (ucBpc[0] >> 6)) & 0x0F; //小时
60.                wBpcSecond *= 3600;
61.                wBpcSecond += ((uint16)(ucBpc[0] & 0x3F)) * 60; //分钟
62.                wBpcSecond += ((ucBpc[1] >> 4) & 0x03) * 20 + 21; //秒数
63.                //如果相临近的两次BPC校时都是准确的(没有误码)，守时中断应该在BPC
64.                //信号的边界前后，因此，秒数只可能差0或1秒。据此判断校时成功
65.                if ((wBpcSecond - s_wRealSecond) < 2)
66.                  s_wEvent |= BPC_FINISHED;
67.                s_wRealSecond = wBpcSecond;
68.                s_wTarSecond = wCurToggle + COUNT_1S;
69.               } //if (ucBpcBitPos == 0
70.            }
71.          }
72.        }
73.        wPrevToggle = wCurToggle;
74.        wLastPulseWidth = wCurPulseWidth;
75.        bLastPulsePolarity = bCurPulsePolarity;
76.      }
77.    }
```

复制代码

守时信号输出 TimerA的通道0工作在比较器模式，用作守时和UART波特率发生器.时间以12小时内的秒数表示，从00:00:00或12:00:00 开始计数，在每秒开始时将时间读数从UART口发出去。MSP430G2211没有专用的UART,需要软件实现。UART数据格式为8位、无校验、1200波特率.总共 16bit数据，需要用2个字节表示，加上每字节的起始位、停止位，总共20位。

```
01.  #pragma vector=TIMER0_A0_VECTOR
02.  __interrupt void Uart_ISR(void)
03.  {
04.    static int8 iBits = 0;
05.    static uint16 wMask;
06.
07.    TACCTL0 &= ~TAIFG; //清中断
08.    //每秒起始位置把16bits实时时间通过UART发送出去
09.    if ((iBits == 0) || (iBits == 10)) {
10.      MCU_IO_CLR(UART_TX_PORT, UART_TX_PIN); //起始位
11.    } else if ((iBits == 9) || (iBits == 19))
12.      MCU_IO_SET(UART_TX_PORT, UART_TX_PIN); //停止位
13.    else {
14.      //发送数据位
15.      if (s_wRealSecond & wMask)
16.        MCU_IO_SET(UART_TX_PORT, UART_TX_PIN);
17.      else
18.        MCU_IO_CLR(UART_TX_PORT, UART_TX_PIN);
19.      wMask <<= 1;
```

```
    }
    iBits++;

    if (iBits == 20) {
      //实时时间发送完毕，准备在下一秒再次发送
      s_wTarSecond += COUNT_1S;
      TACCR0 = s_wTarSecond;
      s_wRealSecond++;
      if (s_wRealSecond >= ((uint16)12*3600))
        s_wRealSecond = 0;    iBits = 0;
      wMask = 1;
      s_wEvent |= SECOND_EVENT;
    } else
      TACCR0 += UART_BIT_WIDTH;

    __low_power_mode_off_on_exit();
  }

```

复制代码

守时时钟校准　本系统需要依靠标称频率为32768Hz晶振提供时间基准来守时，晶振负载电容对频率有微调作用.为了测定晶振实际工作　　频率，可测量一段时间内的积累误差。例如在10:00:00时进行第一次BPC校时，6个小时后在16:00:00进行第二次BPC校时，用串口工具接收并打印出　　第二次BPC校时前后从UART口输出的守时信号

校时之前

【2017-10-11 15:58:47:850】CB 0D

【2017-10-11 15:58:48:850】CC 0D

校时之后

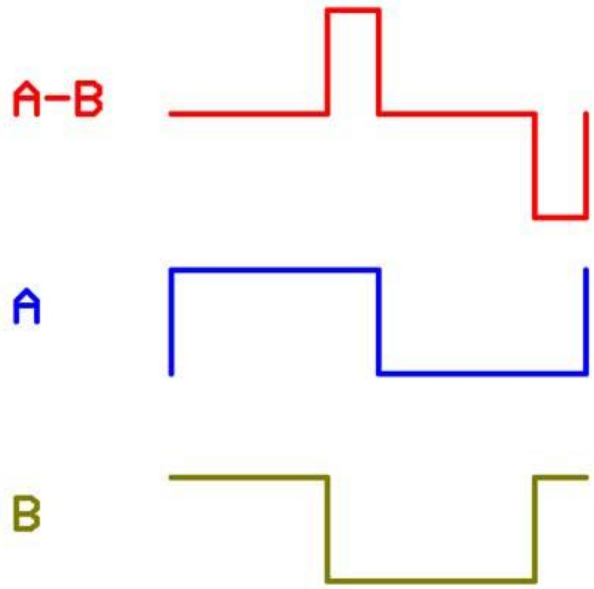
【2017-10-11 15:59:47:827】07 0E

【2017-10-11 15:59:48:827】08 0E

从打印出来的数据可以看出，在校时前后两点(0x0E08-0x0DCC = 60秒)，对应PC机系统时间为59:48:827-58:48:850　　= 59.977秒，也就是说6小时内积累的误差为-0.023秒，可以忽略不计。否则可能需要调整负载电容来对频率进行微调，或者也可以调整代码中的　　COUNT_1S的宏定义值来重新标定“1秒”。

机芯驱动　本设计初衷是要制作一个能够准确计时(无累积误差)的东西.因此完成守时信号输出就OK了，但既然动了手，就准备弄个　　完整的电波钟玩玩。拆解了一个多时不用的石英钟，只将机芯步进马达的线圈引出，其他电路统统拆掉。这里我范了一个错误，本以为马达是1秒　　走一步或二步，按此设计了驱动代码，结果马达跑得非常别扭，走走退退，不知怎回事，弄了半天才发现问题所在，实际上这个机芯步进马达是每秒16步　　的，差得也太远了。因此，建议朋友在拆机前先测一下原机的驱动波形。

这种步进电机的驱动信号为正负交替的脉冲信号。脉冲宽度需要有个合理的范围，拆机时没有先用示波器测一下，只好自己　　凑了。不过，就算测了也只能供参考，因为原机是1.5V供电的，现在改成3V，脉宽肯定需要调窄，理论上升到2倍电压后脉宽应是原来的1/4.我的这个马达　　用12ms脉宽驱动，工作得很Happy.朋友自己制作时可以自行调整MOTOR_PULSE_DUTY。



为了实现正负极性的交替，使用了2个IO端口，输出两路移相的方波信号.两路方波信号之间的相差即为脉冲宽度。当钟面显示　　的时间同实际时间有偏差时，需要改变脉冲周期以调整电机速度，使两者趋于一致。机芯驱动使用了TimeA的通道1，在其中断服务中实现

```
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Motor_drv_ISR(void)
{

  if (TACCTL1 & CCIFG) {
    // 步进电机驱动信号，一个周期分4个Stage，电机走2步。
    // S1，S3提供动力输出。
    //               ------
    //               | S1  |
    //               |     |
    // --------------|     |--------------|     |
    //      S0                   S2       |     |
    //                                    |     |
    //                                    ------
    //                                      S3

    static uint8 ucStage = 0; //步进电机每走一步分2个stage。
    uint16       wS0S2; //S0,S2时间长度

    TACCTL1 &= ~CCIFG; //清中断
    if (ucStage == 2 * STEP_1S - 1) {
      //每秒末进入此处
      ucStage = 0;
      s_wDisplaySecond++;
      if (s_wDisplaySecond >= ((uint16)12 * 3600))
        s_wDisplaySecond = 0;
    }
    else
      ucStage++;

    //先假定钟面时间总是偏"快"的，算一下"快"了多少
    if (s_wDisplaySecond > s_wRealSecond)
      wS0S2 = s_wDisplaySecond - s_wRealSecond;
    else //超了一圈(12小时)
      wS0S2 = ((uint16)12 * 3600) - (s_wRealSecond - s_wDisplaySecond);
    //如果算下来"快"了9小时以内，认为其确实"快"了，否则认为实际是"慢"了3小时不到。
    //之所以不以6小时分界，是因为步进马达可以无限放慢，有限地加快。
    if (wS0S2 < 2) //偏快不多，正常运行
      wS0S2 = MOTOR_PERIOD - MOTOR_PULSE_DUTY; //正常运行
    else if (wS0S2 < ((uint16)9 * 3600)) //偏快，降速运行
```

```
41.         wS0S2 = MOTOR_PERIOD_SLOW - MOTOR_PULSE_DUTY;
42.       else //偏慢，加速运行
43.         wS0S2 = MOTOR_PERIOD_FAST - MOTOR_PULSE_DUTY;
44.
45.       //步进电机驱动信号4个stage一个循环，走2步
46.       switch(ucStage & 0x03) {
47.       case 0:
48.         MCU_IO_CLR(MOTOR_DRVN_PORT, MOTOR_DRVN_PIN);
49.         TACCR1 += wS0S2;
50.         break;
51.       case 1:
52.         MCU_IO_SET(MOTOR_DRVP_PORT, MOTOR_DRVP_PIN);
53.         TACCR1 += MOTOR_PULSE_DUTY;
54.         break;
55.       case 2:
56.         MCU_IO_SET(MOTOR_DRVN_PORT, MOTOR_DRVN_PIN);
57.         TACCR1 += wS0S2;
58.         break;
59.       case 3:
60.         MCU_IO_CLR(MOTOR_DRVP_PORT, MOTOR_DRVP_PIN);
61.         TACCR1 += MOTOR_PULSE_DUTY;
62.         break;
63.       }
64.     }
65.
66.     __low_power_mode_off_on_exit();
67.   }
```

复制代码

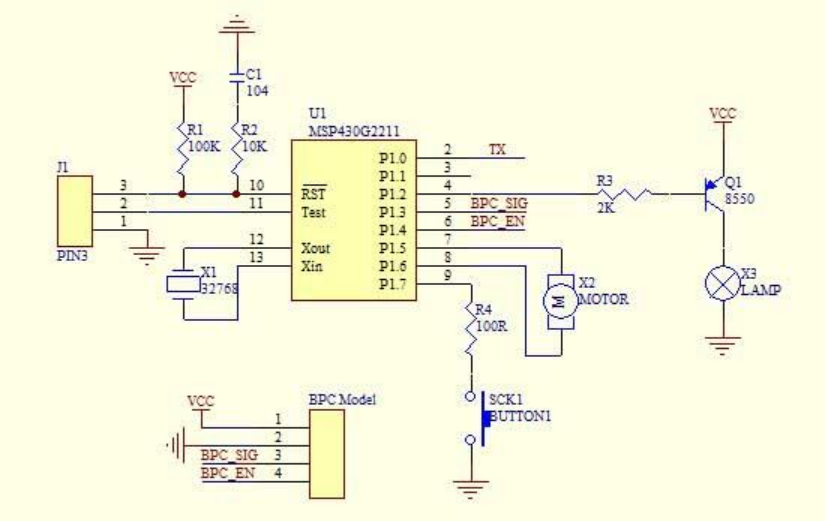
使用方法　由于系统无法读取钟面显示的时间，因此在系统上电启动时，必须先把钟面拨到00:00:00的默认位置.上电时，MCU默认为　　钟面显示时间和系统实际时间为00:00:00.系统在00:00:02启动BPC对时，对时成功后，“实际时间”就准确了，这时，从UART输出的守时信号也是准确的了。　　但钟面时间需要一段时间后才能逐步同实际时间一致。以后，系统会在每天的00:00:02和12:00:02各启动BPC对时一次，如果对时成功或20分钟内不能对时　　则自动关闭BPC模块以节约电池。对时成功，照明等会点亮2秒。

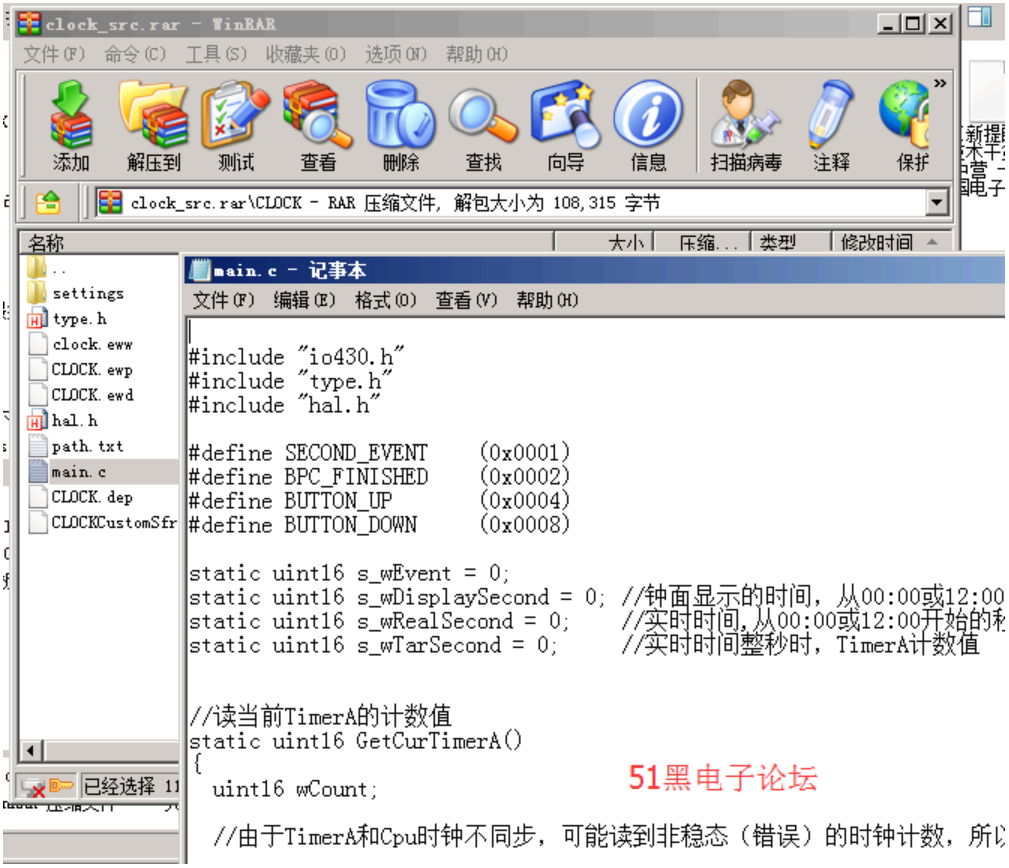
在每天的17:00-21:00, 05:00-9:00两个时段内是BPC发射台是关闭的，在这两个时段内开机是无法对时的。

系统提供了一个按钮，短按按钮可以点灯5秒，以便夜间照明。长按2秒以上可以立即打开BPC对时，对时成功或5分钟内不成功　　则自动关闭BPC模块。这些业务逻辑都在主程序中实现。

实物图

原理图　JI为下载接口。MSP430G2211晶振匹配电容内置，可配置，所以不需要外接电容





全部资料下载地址：

clock_src.rar (15.25 KB, 下载次数: 290)

评分

参与人数 1	黑币 +5	理由	收起
zgs17951	+ 5		

查看全部评分

回复 举报