用 NTC 热敏电阻做温度采集

V1.1 - Dec 8, 2005

中文版

19, Innovation First Road • Science Park • Hsin-Chu • Taiwan 300 • R.O.C.

Tel: 886-3-578-6005 Fax: 886-3-578-4418 E-mail: mcu@sunplus.com.cn

http://www.sunplusmcu.com http://mcu.sunplus.com

版权声明

凌阳科技股份有限公司保留对此文件修改之权利且不另行通知。凌阳科技股份有限公司所提供之信息相信为正确且可靠之信息，但并不保证本文件中绝无错误。请于向凌阳科技股份有限公司提出订单前，自行确定所使用之相关技术文件及规格为最新之版本。若因贵公司使用本公司之文件或产品，而涉及第三人之专利或著作权等智能财产权之应用及配合时，则应由贵公司负责取得同意及授权，本公司仅单纯贩售产品，上述关于同意及授权，非属本公司应为保证之责任。又未经凌阳科技股份有限公司之正式书面许可，本公司之所有产品不得使用于医疗器材，维持生命系统及飞航等相关设备。

目录

修订记录

版本	日期	编写及修订者	编写及修订说明
1.0	2004/01/27		初版
1.1	2005/12/08		错误校正

1 系统概要

1.1 系统说明

本应用例实现 NTC 热敏电阻器对温度的测量。热敏电阻器把温度的变化转换为电阻阻值的变化，再应用相应的测量电路把阻值的变化转换为电压的变化；SPMC75F2413A 内建 8 路 ADC 可以把模拟的电压值转换为数字信号，对数值信号进行处理可以得到相应的温度值。

1.2 热敏电阻器

热敏电阻有电阻值随温度升高而升高的正温度系数（Positive Temperature Coefficient 简称 PTC）热敏电阻和电阻值随温度升高而降低的负温度系数（Negative Temperature Coefficient 简称 NTC）热敏电阻。

NTC 热敏电阻器，是一种以过渡金属氧化物为主要原材料，采用电子陶瓷工艺制成的热敏半导体陶瓷组件。这种组件的电阻值随温度升高而降低，利用这一特性可制成测温、温度补偿和控温组件，又可以制成功率型组件，抑制电路的浪涌电流。

电阻温度特性可以近似地用下式来表示：

$$R_T = R_N * EXP[B*(1/T-1/T_N)]$$

式中：R_T、R_N分别表示NTC在温度T(K)和额定额定温度T_N(K)下的电阻值，单位Ω，T、T_N为温度，单位K（T_N(k)=273.15+T_N(℃)）。B，称作B值，NTC热敏电阻特定的材料常数(Beta)。由于B值同样是随温度而变化的，因此NTC热敏电阻的实际特性，只能粗略地用指数关系来描述，所以这种方法只能以一定的精度来描述额定温度或电阻值附近的有限的范围。

但是在实际应用中，要求有比较精确的 R-T 曲线。要用比较复杂的方法(例如用 the steinhart-Hart 方程)，或者用表格的形式来给定电阻/温度关系。

应用例选用 NTC 热敏电阻器 CWF2-502F3950，基于精确的 R-T 曲线，来对温度进行精确的测量。

1.2.1 电阻－温度关系

如表 1－1 所示，NTC 热敏电阻器 CWF2-502F3950 各温度点的电阻值，即电阻－温度关系表。从提供的电阻－温度关系表中可以看出 NTC 热敏电阻器 CWF2-502F3950 的测温范围为[-55℃,125℃]，其电阻值的变化范围为[250062Ω,242.64Ω]。

表 1－1 电阻－温度关系表

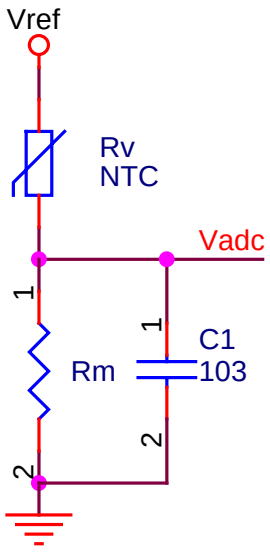
温度(℃)	电阻值(Ω)	温度(℃)	电阻值(Ω)	温度(℃)	电阻值(Ω)
-55	250062	-54	237404	-53	225239
-52	213575	-51	202412	-50	191750
-49	181580	-48	171895	-47	162684
-46	153933	-45	145638	-44	137753
-43	130293	-42	123231	-41	116550

-40	110232	-39	104261	-38	98621.7
-37	93295.5	-36	88267.4	-35	83521.8
-34	79043.9	-33	74819.2	-32	70833.9
-31	67074.7	-30	63529	-29	60184.6
-28	57030.2	-27	54054.7	-26	51247.9
-25	48600	-24	46101.6	-23	43744
-22	41519	-21	39418.8	-20	37435.9
-19	35563.5	-18	33795	-17	32124.4
-16	30545.8	-15	29053.8	-14	27643.3
-13	26309.5	-12	25047.9	-11	23854.2
-10	22724.6	-9	21655.3	-8	20642.7
-7	19683.6	-6	18774.9	-5	17913.6
-4	17097.1	-3	16332.9	-2	15588.4
-1	14891.5	0	14230	1	13601.9
2	13005.4	3	12438.7	4	11900.1
5	11388.2	6	10901.3	7	10438.3
8	9997.74	9	9578.41	10	9181
11	8799	12	8436.83	13	8091.73
14	7762.78	15	7449.16	16	7150.04
17	6864.7	18	6592.4	19	6332.49
20	6084.32	21	5847.31	22	5620.89
23	5404.53	24	5197.72	25	5000
26	4810.9	27	4630.01	28	4456.93
29	4291.28	30	4132.69	31	3980.83
32	3835.38	33	3696.03	34	3562.49
35	3434.5	36	3311.78	37	3194.1
38	3081.22	39	2972.92	40	2869
41	2769.24	42	2673.47	43	2581.5
44	2493.17	45	2408.3	46	2326.76
47	2248.38	48	2173.04	49	2100.6
50	2032	51	1963.92	52	1899.44
53	1837.4	54	1777.68	55	1720.2
56	1664.85	57	1611.54	58	1560.2
59	1510.74	60	1463.08	61	1417.14
62	1372.87	63	1330.18	64	1289.02
65	1249.32	66	1211.03	67	1174.09
68	1138.44	69	1104.04	70	1070.83
71	1038.78	72	1007.82	73	977.93
74	949.06	75	921.17	76	894.22
77	868.18	78	843.02	79	818.69
80	795.17	81	772.43	82	750.44
83	729.17	84	708.6	85	688.7
86	669.44	87	650.8	88	632.76

89	615.3	90	598.39	91	582.02
92	566.17	93	550.81	94	535.94
95	521.53	96	507.57	97	494.05
98	480.94	99	468.23	100	453.3
101	443.97	102	432.38	103	421.15
104	410.26	105	399.69	106	389.44
107	379.5	108	369.85	109	360.48
110	351.4	111	342.57	112	334.01
113	325.69	114	317.62	115	309.77
116	302.16	117	294.76	118	287.57
119	280.59	120	273.8	121	267.21
122	260.8	123	254.58	124	248.52
125	242.64				

1.3 数值处理

通过表 1－1 电阻－温度关系表可以很直观的看到电阻的变化范围从 242.64Ω到 250062Ω，在－55℃的时候其表现出的电阻值是 125℃时所表现的电阻值的 1030 倍，这幺大的变化范围也为 ADC 测量带来了困难。测量电路如图 1－1 所示。



如图 1－1 测量电路

如上图所示 NTC 热敏电阻 Rv 和测量电阻 Rm（精密电阻）组成一个简单的串联分压电路，参考电压 VCC_Ref 经过分压可以得到一个电压值随着温度值变化而变化的数值，这个电压的大小将反映出 NTC 电阻的大小，从而也就是相应温度值的反映。

通过欧姆定律可以得到输出电压值 Vadc 和 NTC 电阻值的一个关系表达式：

$$Vadc = Vref*Rm/(Rv+Rm) \qquad (1)$$

那幺接下来的数据处理将基于式（1）展开：SPMC75F2413A 的 ADC 为 10-Bit 的精度，其参考电

压为 5V，因此这里可以选择 Vref＝5V。各温度点对应的 ADC 转换后的数字量可以计算。

Dadc ＝ 1024*Vadc/5V　　（2）

式（1）、（2）结合可以得到：

Dadc ＝ 1024*Rm/(Rv+Rm)　　（3）

如果这里取测量电阻 Rm 选择 4.7KΩ，那么可以计算出在－55℃时所对应的 Dadc ＝ 1024*1000/(250062+1000)＝ 4；在 125℃时所对应的 Dadc＝ 1024*1000/(242.64+1000)＝ 824。根据这样的对应关系对数据进行预处理，得到如下处理结果如表 1－2 所示：

表 1－2

```
tatic const Int16 NTCTAB2[181] =
{
   19,20,21,22,23,24,26,27,29,30,32,34,
   36,38,40,42,44,47,49,52,55,57,61,64,
   67,71,74,78,82,86,90,95,99,104,109,114,
   120,150,156,161,168,172,180,187,194,201,208,215,
   222,230,238,247,255,264,272,280,291,302,310,319,
   328,338,347,357,367,376,384,395,405,414,424,434,
   444,453,464,474,484,494,502,512,522,531,540,551,
   560,569,579,586,595,604,613,624,633,642,650,658,
   666,673,680,688,696,704,712,719,726,733,741,749,
   755,760,767,774,780,785,791,798,804,811,816,821,
   827,832,837,842,847,851,856,862,868,873,856,860,
   864,868,872,876,879,883,886,890,893,896,899,902,
   905,908,911,914,917,919,922,924,927,929,931,934,
   936,938,940,942,944,946,947,949,951,953,954,956,
   958,959,961,962,964,965,966,968,969,970,971,973,
   974
};//4.7K
```

当然这也是应用例中所需要的一个很重要的转换表，这一部分是事先制作好的表格，将为接下来的处理提供参考依据。

测量电阻 Rm 的选取是有一定的规律的，在实际的应用中不一定都需要测量全程温度，可以估算出大致的温度范围。本着提高测量精度的宗旨：如果是应用在测量低温的系统中建议 Rm 选择较大的电阻（10KΩ），如果在测量较高温的系统中建议 Rm 选择较小的电阻（1KΩ）等。

1.4　线性插值

在 ADC 进行数据采集的过程中不可能每一个数值都在整温度所对应的 ADC 数值上，所以如果在两个数据的中间一段就要对其进行进一步的精确定位。这样就必须知道采集到的数据在表 1－2 中的具体位置，因此要对数据表进行搜索、查找。线性表的查找（也称检索），可以有比较常见的顺序查找、折半查找及分块查找等方法，分析线性表 1－2 可以得到折半查找的算法是比较高效的。

Eg.如果 ADC 采样的数值为 Dadc ＝ 360，即 357<Dadc<367，那么温度值就绝对不是一个整数值了，怎么来得到具体的温度值呢！可以运用简单可行的线性插值来对付类似的情况。

插值求得温度值实际就是用直线 L 拟和温度曲线 T，这样的做法虽然难免的有一定的误差，但是可以控制在允许的范围内的，线性插值原理如图 1－2 所示。

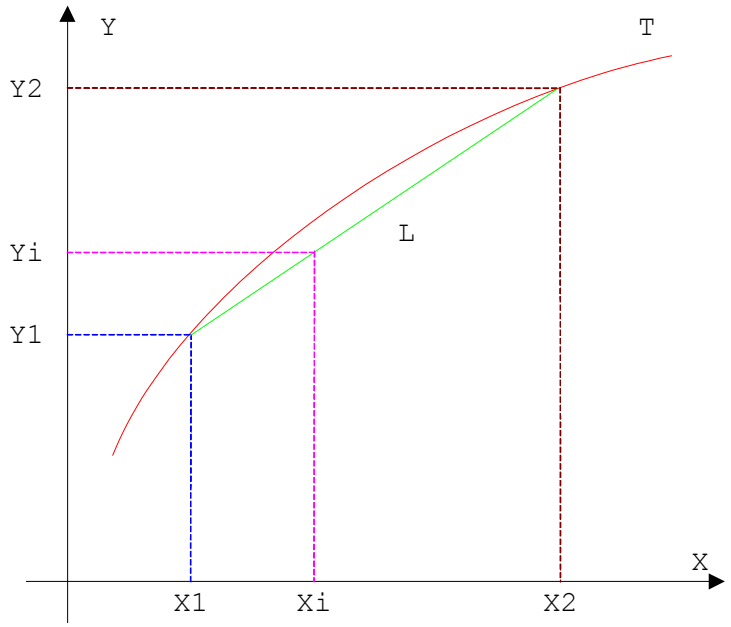


图 1－2 线性插值原理

已知点（X1,Y1）和点（X2,Y2）求（Xi,Yi）。

由两点可以得到直线 L 的方程：

$$X = \frac{X_2 - X_1}{Y_2 - Y_1}(Y - Y_1) + X_1 \qquad (4)$$

点（X1,Y1）和点（X2,Y2）为相邻两温度点，所以 X2－X1＝1 那幺由式（4）可得：

$$X = \frac{1}{Y_2 - Y_1}(Y - Y_1) + X_1 \qquad (5)$$

这样通过 ADC 采样来的 Dadc（Y1）数值带入式（5）可以求得相应的温度值。

插值计算出来的数值肯定是小数，那幺需要对数值进行特殊的处理：基于定点计算的思想，把数据首先规格化，把小数点定在第六位即计算数值放大 64 倍参与计算，当然在计算后的温度数据也应该是真实数值的 64 倍，所以需要 X/64 得到的数值为实际测量到的温度值。把小数点定的位数越高表示的精度越高。

这样的插值计算实际上是分段的，用直线段来拟和温度曲线，因此在处理的过程中分段越细致拟和的曲线就越接近实际温度曲线。

2 软件说明

2.1 软件说明

应用例程序部分主要针对 NTC 热敏电阻测量温度的应用，其中最主要的是使用 ADC 模块对信号的采集和处理，从而得到温度数值。

2.2 档案构成

文件名称	功能	类型
Main.C	完成初始化	C
NTC.C	ADC 采集、查表、插值	C
AN_SPMC75_0101.H	API 引用和相关参数定义	H

2.3 子程序说明

Spmc75_ADC_Init()

原　　形　　void Spmc75_ADC_Init(void)

描　　述　　初始化 ADC，选择信道 0 用于采集温度

输入参数　　无

输出参数　　无

头 文 件　　AN_SPMC75_0101.H

库 文 件　　无

注意事项　　默认 ADC 信道 0

例　子 1　　Spmc75_ADC_Init();

Temperature_Measure()

原　　形　　Int16 Temperature_Measure(void)

描　　述　　ADC 转换、查温度表、插值计算

输入参数　　无

输出参数　　0x7FFF：超出量程，温度小于－55℃
0xFFFF：超出量程，温度大于 125℃
else：实际测量温度的 64 倍（规格化的定点小数）

头 文 件　　AN_SPMC75_0101.H

库 文 件　　无

注意事项　返回数值除以 64 可以得到一定精度的实际测量温度值

例　　子

```
float Temp;
Temp = Temperature_Measure()/64.0;
```

3 程序范例

3.1 DEMO 程序

范例程序将演示温度的采集，温度值在 4*8seg LED 上显示的功能。

```
#include  "AN_SPMC75_0101.H"
//=====================================================================
main()
{
   Spmc75_ADC_Init(); //ADC Initial
   Spmc75_Led_Init(); //Show LED GPIO Init
   IRQ_ON();
   while(1)
   {
       NOP();
       NOP();
   }
}
```

```
void IRQ7(void) __attribute__ ((ISR));
void IRQ7(void)
{
   static Int16 temp;
   if(P_INT_Status->B.CMTIF)
   {
       if(P_CMT_Ctrl->B.CM0IF && P_CMT_Ctrl->B.CM0IE)
       {
           temp = Temperature_Measure();          //温度测量
           if(temp != 0xFFFF && temp != 0x7FFF) //?超量程?
           {
               temp = temp/6.4;                   //保留一位小数
           }
       }
       if(P_CMT_Ctrl->B.CM1IF && P_CMT_Ctrl->B.CM1IE)
       {
           Spmc75_Led_Show(temp);                //LED 显示温度值
       }
       P_CMT_Ctrl->W = P_CMT_Ctrl->W;
   }
}
```

```
void Spmc75_Led_Init(void)
{
   P_IOB_SPE->W    = 0x0000;   //GPIO 用于 LED 显示驱动
```

```
   P_IOB_Buffer->W = 0xffff;
   P_IOB_Attrib->W = 0xffff;
   P_IOB_Dir->W    = 0xffff;
   P_IOB_Data->W   = 0x0000;

   P_CMT_Start->W = 0x0000;
   P_CMT_Ctrl->W  = 0x0000;

   P_CMT_Ctrl->B.CKA  = CB_CKA_FCK_256;
   P_CMT_Ctrl->B.CM0IE = CB_CMT0_INT_EN;
   P_CMT_Ctrl->B.CM0IF = CB_CLEAR_CM0IF;

   P_CMT0_TPR->W = 11718;     //CMT0－8Hz，温度采集

   P_CMT_Ctrl->B.CKB  = CB_CKB_FCK_16;
   P_CMT_Ctrl->B.CM1IE = CB_CMT1_INT_EN;
   P_CMT_Ctrl->B.CM1IF = CB_CLEAR_CM1IF;

   P_CMT1_TPR->W = 5859;      //CMT1－256Hz，LED 动显

   P_CMT_Start->B.ST0 = CB_CMT0_Start;
   P_CMT_Start->B.ST1 = CB_CMT1_Start;
}
```

```
typedef union
{
    UInt16     W;
    struct
    {
        UInt16  _byte0     : 8;                    /* LSB byte code */
        UInt16  _byte1     : 4;                    /* MSB byte code */
        UInt16  _byte2     : 4;
    }B;
}TYPELEDDEF;
#define LED_DATA      ((volatile TYPELEDDEF *)(P_IOB_Buffer_ADDR))
const UInt16 SEGARRAY[14] = {0x3F,0x06,0x5B,0x4F,0x66,
                             0x6D,0x7D,0x07,0x7F,0x6F,
                           /* -   +   E   r   */
                             0x40,0x46,0x79,0x50}; //段码
const UInt16 DIGARRAY[4]  = {0x01,0x02,0x04,0x08}; //位码
UInt16 SHOWBUFFER[4] = {0};  //显示缓存
void Spmc75_Led_Show(Int16 data)
{
   static UInt16 i = 0;
```

```
    if(data == 0x7FFF)      //-55℃超量程错误显示
    {
        SHOWBUFFER[0] = 11;
        SHOWBUFFER[1] = 13;
        SHOWBUFFER[2] = 13;
        SHOWBUFFER[3] = 12;
    }
    else if(data == 0xFFFF)    //125℃超量程错误显示
    {
        SHOWBUFFER[0] = 10;
        SHOWBUFFER[1] = 13;
        SHOWBUFFER[2] = 13;
        SHOWBUFFER[3] = 12;
    }
    else if(data >= 0)      //零上温度显示
    {
        SHOWBUFFER[0] = data%10;
        SHOWBUFFER[1] = (data/10)%10;
        SHOWBUFFER[2] = (data/100)%10;
        SHOWBUFFER[3] = (data/1000)%10;
    }
    else                    //零下温度显示
    {
        data = 0xffff-data;
        SHOWBUFFER[0] = data%10;
        SHOWBUFFER[1] = (data/10)%10;
        SHOWBUFFER[2] = (data/100)%10;
        SHOWBUFFER[3] = 10;
    }

    LED_DATA->B._byte1 = 0x00;
    if(i == 1)              //显示小数点
        LED_DATA->B._byte0 = 0x80+SEGARRAY[SHOWBUFFER[i]];
    else
        LED_DATA->B._byte0 = SEGARRAY[SHOWBUFFER[i]];
    LED_DATA->B._byte1 = DIGARRAY[i++];    //位扫描
    if(i>3) i = 0;
}
```

部分子函数：

```
#define Q_Data          64  //温度使用 Q5 格式的补码
#define Data_125  NTCTAB[180]
#define Data_N55  NTCTAB[0]
static const Int16 NTCTAB[181] =
```

```
{
   4, 4, 5, 5, 5, 5, 6, 6, 6, 7,
   7, 7, 8, 8, 9, 9, 10, 10, 11, 11,
   12, 13, 14, 14, 15, 16, 17, 18, 19, 20,
   21, 22, 23, 24, 25, 27, 28, 29, 31, 32,
   34, 36, 37, 39, 41, 43, 45, 47, 50, 52,
   54, 57, 59, 62, 64, 67, 70, 73, 76, 79,
   83, 86, 90, 93, 97, 101, 105, 109, 113, 117,
   121, 126, 130, 135, 140, 145, 150, 155, 160, 165,
   171, 176, 182, 188, 194, 200, 206, 212, 218, 224,
   231, 237, 244, 251, 258, 265, 272, 279, 286, 293,
   300, 308, 315, 323, 330, 338, 345, 353, 361, 369,
   376, 384, 392, 400, 408, 416, 424, 432, 439, 447,
   455, 463, 471, 479, 487, 494, 502, 510, 518, 525,
   533, 541, 548, 556, 563, 570, 578, 585, 592, 599,
   606, 613, 620, 627, 634, 641, 647, 654, 660, 667,
   673, 679, 685, 691, 697, 705, 709, 715, 721, 726,
   732, 737, 742, 748, 753, 758, 763, 768, 772, 777,
   782, 786, 791, 795, 800, 804, 808, 812, 816, 820,
   824
};//测量电阻为 1K 数据表
```

```
//折半查表
static void Lookup_TAB(UInt16 data,const Int16 *TABLE,Int16 *aptr)
{
   register UInt16 i;
   Int16 *eptr=(Int16 *)&TABLE[180];  //高端指针
   Int16 *sptr=(Int16 *)TABLE;        //低端指针
   Int16 *ptr;                        //查数指针

   for(i=0;i<8;i++)                   //搜索全表
   {
       ptr = (Int16 *)((Int16)sptr+(((Int16)(eptr-sptr))>>1));
       if(*ptr<data)   sptr = ptr;
       else if(*ptr>data) eptr = ptr;
       else               //查到相等的节点
       {
           aptr[2] = *ptr;         //Y1
           aptr[1] = *(ptr+1);     //Y2
           aptr[0] = (Int16)(ptr-TABLE);//X1
           break;
       }

       if(eptr-sptr==1)    //查到节点的范围
       {
```

```
            aptr[2] = *sptr;        //Y1
            aptr[1] = *eptr;        //Y2
            aptr[0] = (Int16)(sptr-TABLE);//X1
            break;
        }
    }
}
```

```
//分段线性插值
static Int16 LinearInsert(UInt16 data,const Int16 *TABLE)
{
   int Temp,Array[3];

   if(data > Data_N55)            //?Beyond Lowest Temperature
   {
       if(data < Data_125)        //?Beyond Highest Temperature
       {
           Lookup_TAB(data,TABLE,Array);  //Lookup data
           if(data == Array[2])   Temp = (Array[0] - 55)*Q_Data;
           else Temp = (Array[0]-55)*Q_Data+
                       (data-Array[2])*Q_Data/(Array[1]-Array[2]);
       }
       else    Temp = 0x7FFF;     //mark Underflow
   }
   else    Temp = 0xFFFF;         //mark Overflow
   return(Temp);
}
```

3.2 硬件原理图

DEMO 硬件原理图，如图 3－1 所示。显示部分电路原理图为示意图。

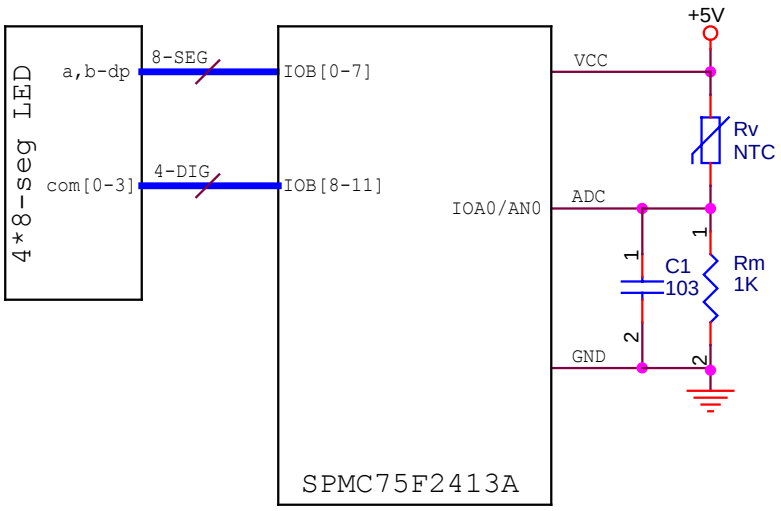


图 3－1 DEMO 硬件原理图

4 MCU 使用资源

4.1 MCU 硬件使用资源说明

CPU 型号	SPMC75F2413A	封装	QFP80-1.0
振荡器	☒ crystal	频率	6MHz
	☐外部	输入频率	
WATCHDOG	☒ 有 ☐ 无	☐ 启用 ☒ 未启用	
IO 口使用情况	IOB[0-11]	LED 显示	
	剩余 IO 及处理方式	主控应用/GND	
Timer 使用情况	CMT0	系统定时	
中断使用情况	CMT0(IRQ7)	数据采集、显示中断	
ROM 使用情况	1.79K Word		

5 参考文献

【1】SUNPLUS　SPMC75F2413A 编程指南 V1.1　Jan、03、2005