目录

测温

DS18B20的核心功能就是数字化的温度读数, 可以设置为9, 10, 11, 12位分辨率, 缺省分辨率是12位. 各分辨率对应的读数, 温度分辨率分别是0.5, 0.25, 0.125, 0.0625摄氏度.

在执行温度转换命令Convert T `0x44` 后, 温度会被转换并存储在一个2字节的内存单元, 然后通过读取命令Read Scratchpad `0xBE` 读出.

转换时间

在温度转换命令Convert T `0x44` 发起到采集完成需要的时间可能会长达750 ms. 实际从 400ms 至 1s 都有可能.

读数结构

这两个字节各个bit分别代表的数字含义如下, 高字节的高5位仅用于表示温度的正负, 正温度是0, 负温度是1, 后面11个bit表示的数字, 负值使用的是补码, 读数用 (0xFF - 读数)

- 正温度时, 将16位整数乘以对应的温度分辨率
- 负温度时, 将16位整数取反加1后, 乘以对应的温度分辨率

7	6	5	4	3	2	1	0		7	
S	S	S	S	S	2^6	2^5	2^4		2^3	2^2
MSB							LSB		MSB	

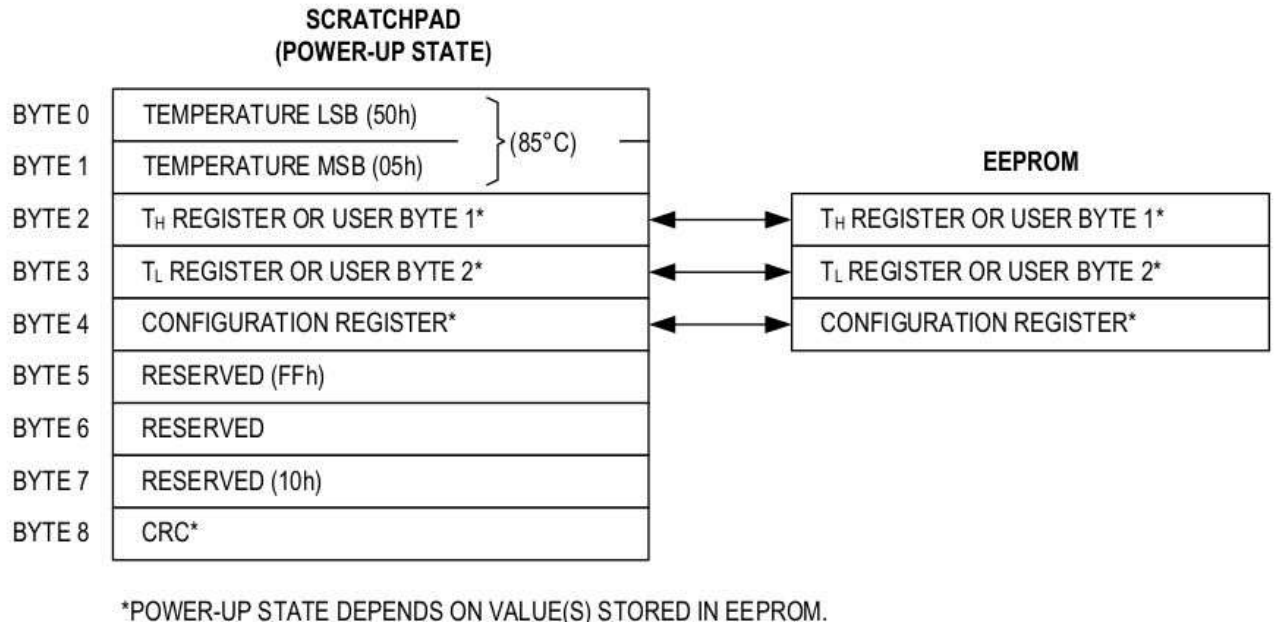
读数快查表

上电后的缺省值为0x0550, 对应85°C, 如果一直读出都是这个值, 需要检查接线

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C	0000 0101 0101 0000	0550h*
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FF6Fh
-55°C	1111 1100 1001 0000	FC90h

测温存储结构

DS18B20内部有9字节的暂存器和3个字节的EEPROM存储, 测温的结果存在暂存器的前两个字节, 整体结构如下, 可以通过读取命令Read Scratchpad `0xBE` 读出全部9个字节



ROM读数

每个 DS18B20 包含一个唯一的只读的64bit编码, 其结构为

1. 最初 8 bits 为固定的 0x28, 1-Wire family code
2. 接下来的 48 bits 是唯一序列号
3. 最后的 8 bits 是前面 56 bits 的 CRC 校验码.

这个 64-bit ROM 和 ROM 方法允许在单线(1-Wire)总线上运行多个 DS18B20, 使用单线总线需要使用下面的方法之一发起:

1. Read ROM,
2. Match ROM,
3. Search ROM,
4. Skip ROM, or
5. Alarm Search.

After a ROM function sequence has been successfully executed, the functions specific to the DS18B20 are accessible and the
bus master may then provide one of the six memory and control function commands.

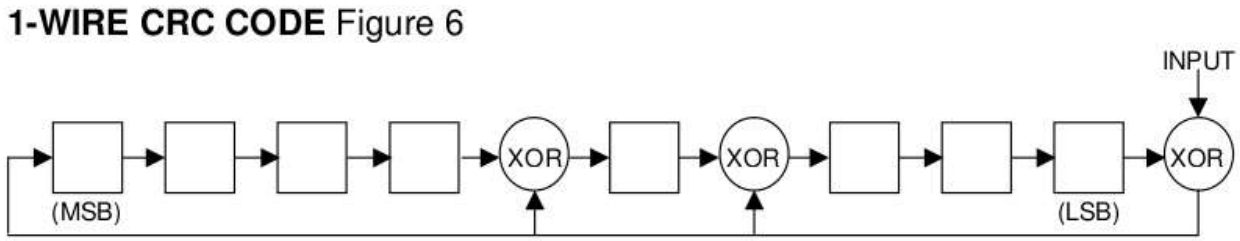
CRC 计算

无论是读取8字节ROM, 还是读取9字节暂存器, 最后一个字节都是前面所有字节的CRC校验值. CRC值的比较与是否继续操作完全由总线控制端决定, DS18B20 内部仅计算CRC, 并不会对CRC不匹配的情况进行处理, 需要总线控制端主动判断.

计算CRC的等效多项式函数为(这是datasheet中的式子, 并非幂运算, 要结合后面的流程图理解)

$$CRC = X^8 + X^5 + X^4 + 1$$

1-Wire总线的CRC计算由移位寄存器和异或门组成的多项式发生器来执行: 移位寄存器位初始化为0, 然后从第一个字节的最低位开始, 一次移入一位, 根据计算结果决定是否与第4, 第5位作异或, 然后CRC也往右移, 最后移位寄存器的值就是CRC.



CRC 计算代码

8位CRC的C语言计算代码为

```
1  uint8_t DS18B20_Crc(uint8_t *addr, uint8_t len)
2  {
3      uint8_t crc = 0, inbyte, i, mix;
```

```
4
5       while (len--)
6       {
7           // inbyte 存储当前参与计算的新字节
8           inbyte = *addr++;
9
10          for (i = 8; i; i--)
11          {
12              // 将新字节与CRC从低位到高位，依次做异或运算，每次运算完CRC右移一位
13              // 如果运算结果值为1，则将CRC与 1000 1100 作异或
14              // 第3,4位代表流程图中的异或运算，第7位其实就是运算结果移入的1
15              mix = (crc ^ inbyte) & 0x01;
16              crc >>= 1;
17              if (mix)
18              {
19                  crc ^= 0x8C;
20              }
21              inbyte >>= 1;
22          }
23      }
24      return crc;
25  }
```