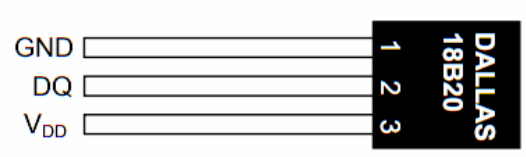
温度传感器测温

这节我们介绍一下 DS18B20 的操作，这也是应用比较广泛的温度传感器，它采用了1-wire(一线)协议。学会了它，我们就又学会一种协议的操作。(由于该器件涉及的命令较多，此文无法面面俱到，建议若有不明白的地方可参考 DS18B20 的器件手册)

DS18B20 是 DALLAS 公司生产的一线式数字温度传感器，采用 3 引脚 TO-92 型小体积封装；温度测量范围为-55℃～+125℃，可编程为 9 位～12 位 A/D 转换精度，测温分辨率可达 0.0625℃，被测温度用符号扩展的 16 位数字量方式串行输出。

其封装图如下：



各引脚定义如下表：

VDD	电压输入
GND	地
DQ	数据输入/输出

我们有必要简单了解一下其内部结构：

DS18B20 的内部结构如下图所示，主要由以下几部分组成：64 位 ROM、温度传感器、非挥发的温度报警触发器 TH(温度高)和 TL(温度低)、配置寄存器、暂存寄存器(SCRATCHPAD)、存储器控制逻辑。DQ 为数字信号输入/输出端。

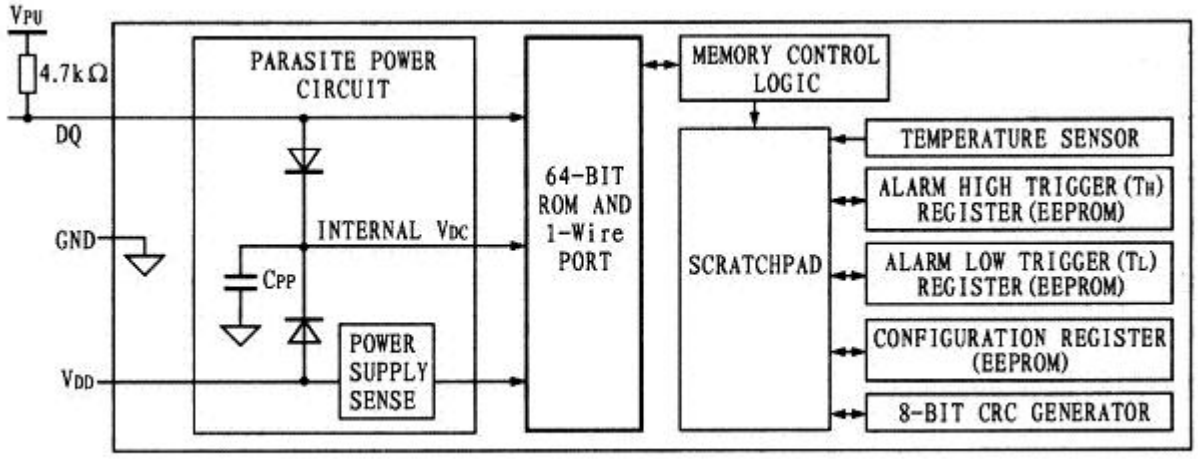


图 1　DS18B20 的内部结构框图

ROM 中的 64(8 位产品家族编号、48 位 ID 号、8 位 CRC)位序列号是出厂前刻好的，这 64 位序列号具有惟一性，每个 DS18B20 的 64 位序列号均不相同，我们通过这个序列号在总线上寻址器件。

8 位 CRC 生成器可以完成通信时的校验。

暂存寄存器有 9 个字节，包含温度测量结果、温度报警寄存器、CRC 校验码等内容。

操作 DS18B20 的命令大致可分为两大类：

ROM 命令：

这些命令完成 FPGA 与总线上的某一具体 DS18B20 建立联系。ROM 命令有搜寻 ROM(SEARCH ROM)、读 ROM(READ ROM)、匹配 ROM(MATCH ROM)、忽略 ROM(SKIP ROM)、报警查找(ALARM SEARCH)等命令。

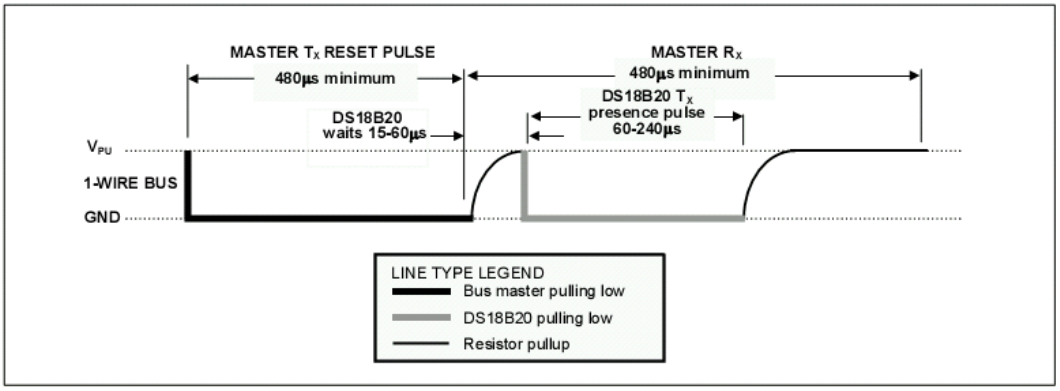
功能命令：（又称 RAM 命令）

这些命令完成温度转换(CONVERTT)、写暂存寄存器(WRITE SCRATCHPAD)、读暂存寄存器(READ SCRATCHPAD)、拷贝暂存寄存器(COPYSCRATCHPAD)、装载暂存器寄存器(RECALL E2)、读供电模式命令(READ POWER SUPPLY)等。

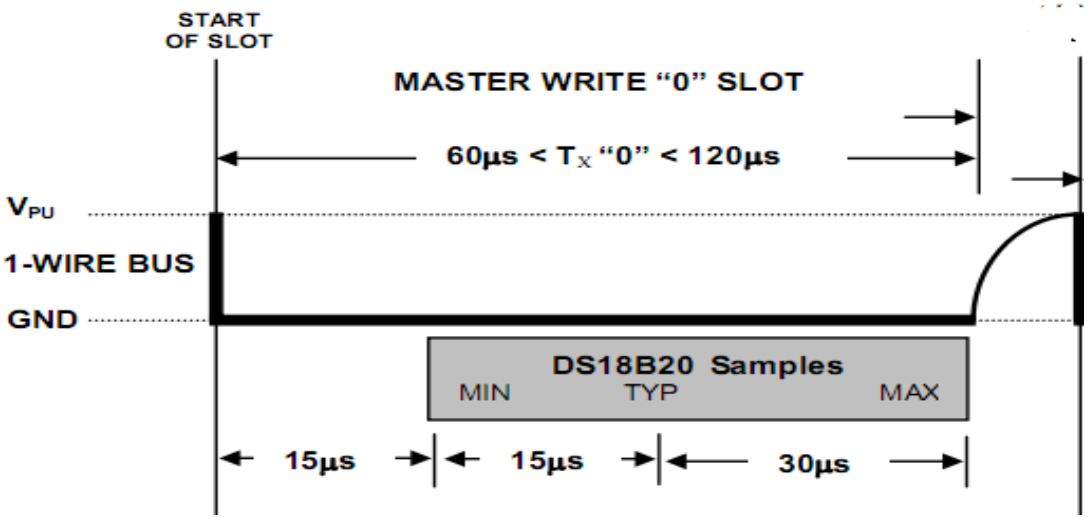
一线通信协议需要严格遵从时序要求，对时间的掌控很严格。CPLD 对时序的操作可以做的很精确，所以用 CPLD 做 DS18B20 的时序是绰绰有余的。

我们先看一下 DS18B20 操作的几个常用的、重要的时序。

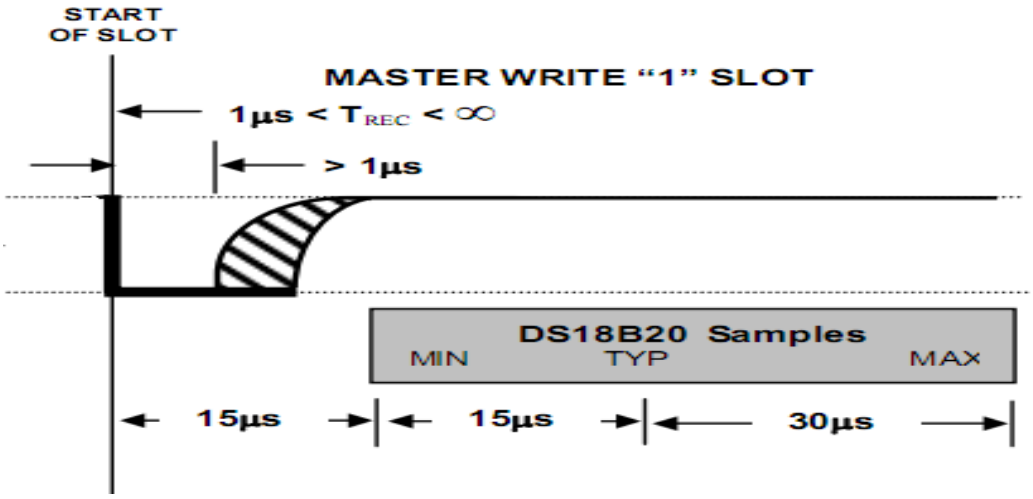
Figure 13. Initialization Timing



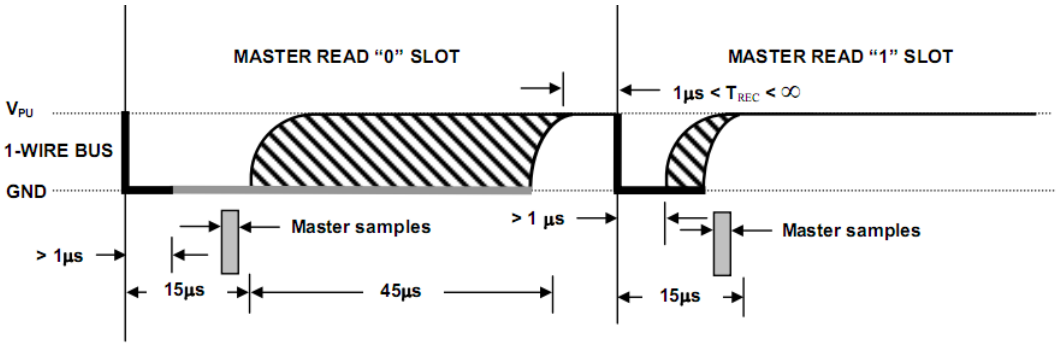
总线复位：置总线为低电平并保持至少 480us,然后拉高电平，等待从端（DS18B20）拉低电平作为响应，则总线复位完成。



写数据 0：置总线为低电平并保持至少 15us，然后保持低电平 15us-45us 等待从端(DS18B20)采样电平，最后拉高电平完成写操作。



写数据 1：置总线为低电平并保持至少 1us-15us，然后拉高电平并保持 15us-45us 等待从端（DS18B20）采样电平，完成写操作。



读数据位：置总线为低电平并保持至少 1us,然后拉高电平保持至少 1us，在 15us 内采样总线电平获得数据，延时 45us 完成读操作。

对 DS18B20 的操作分为 3 个步骤：初始化->ROM 命令->功能命令

其具体操作流程如下表所示（各种指令的代码以及详细介绍见手册 Page10-Page22）：

复位流程：

CPLD 状态	命令/数据	说明
发送	Reset	复位
接受	Presence	DS18B20 应答
发送	0XCC	忽略 ROM 匹配(因为只有一个 DS18B20 挂在总线上)
发送	0X4E	写暂存寄存器
发送	连续发送 3 字节数据	分别为设置温度边界值 TH，设置温度边界值 TL 和设置配置控制字

下面解析几个问题：

暂存寄存器的结构如下图：

字节 0	温度低字节（ＬＳＢ）

字节 1	温度高字节（ＭＳＢ）
字节 2	温度边界值上限ＴＨ
字节 3	温度边界值下限ＴＬ
字节 4	配置控制字
字节 5	保留字
字节 6	保留字
字节 7	保留字
字节 8	ＣＲＣ校验字

当我们发送写暂存寄存器指令时，其后续的第一个，第二个，第三个字节分别写入字节 2，字节 3 和字节 4（跳过字节 0 和 1）。

再了解一下配置寄存器的结构：

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

整个配置寄存就是 BIT6 和 BIT5 有用，它们的组合定义了温度转换的精度：

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	(t_{CONV}/8)
0	1	10	187.5ms	(t_{CONV}/4)
1	0	11	375ms	(t_{CONV}/2)
1	1	12	750ms	(t_{CONV})

转换精度越高，其转换所需要的时间就越长。这个在我们编程的时候需要注意，要根据你设置的转化精度选择合适的等待时间。这个精度的设置就相应的就定义了温度寄存器中温度值的有效小数位的位数，这个我们在后面介绍。

我们简单介绍下温度下限寄存器 TL 和温度上限寄存器 TH 的结构：

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0

其高位是符号位，后面 7 位就是可设置的温度值了。

温度转换以及读取流程：

CPLD 状态	**命令/数据**	**说明**
发送	**Reset**	**复位**
接受	**Presence**	**DS18B20 应答**
发送	**0XCC**	**忽略 ROM 匹配(因为只有一个**

		DS18B20 挂在总线上）
发送	0X44	温度转换命令
等待		等待时间根据你在配置寄存器中设置的转换精度来定
发送	Reset	复位
接受	Presence	DS18B20 应答
发送	0XCC	忽略 ROM 匹配(因为只有一个 DS18B20 挂在总线上）
发送	0XBE	读取暂存寄存器
读取	连续的 9 字节数据	前 2 字节为温度值

需要了解的几个问题：

温度字节的结构（处于暂存寄存器的前两个字节）

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

BIT15-BIT11 都是符号位，其中 BIT3-BIT0 是小数位，该小数位的个数是由配置控制字中设置的转换精度决定的：

12 位精度（精度为 0.0625°）:BIT3-BIT0 有效

11 位精度（精度为 0.125°）:BIT3-BIT1 有效，BIT0 无效

10 位精度（精度为 0. 25°）:BIT3-BIT2 有效，BIT1-BIT0 无效

9 位精度（精度为 0. 5°）:BIT3 有效，BIT2-BIT0 无效

也就是说，温度的转换精度越高，小数位就越多，这个也是很合理的。

在发送完读取暂存寄存器命令 0XBE 后，其后面**顺序读取**的是暂存寄存器的字节 0--字节 8，若我们只需要其中的部分字节，那就在读取到该部分字节后，通过一个复位操作停止读取。按照暂存寄存器的排列顺序，我们先读到的是温度的低字节（LSB）,然后是温度的高字节（MSB）。

若需要参考详细的资料，可见其手册，里面介绍的非常详细，是最好的参考资料了。

下面我们根据上面描述的操作流程来解析一下代码：

本代码的功能就是：操作 DS18B20,实时读取温度值，然后通过串口发送到 PC 显示

```
--DS18B20的top控制模块
--控制DS18B20，实时读取数据，然后通过串口发送到PC机上
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity top is
    port
    (
        clkin,resetin:in std_logic;--40M系统时钟和复位信号
        DQ:inout std_logic;--数据端
        txd_out:out std_logic
    );
end top;

architecture behav of top is
    component gen_div is--分频元件调用声明
    generic(div_param:integer:=2);--默认是4分频
    port
    (
        clk:in std_logic;
        bclk:out std_logic;
        resetb:in std_logic
    );
    end component;
    ------------------------
    component uart_t is--串口发送元件调用
    port
    (
        bclkt,resett,xmit_cmd_p:in std_logic;
        txdbuf:in std_logic_vector(7 downto 0);
        txd:out std_logic;
        txd_done:out std_logic
    );
    end component;


signal clk_us:std_logic;--分频的得到的us脉冲

signal  data_w:std_logic_vector(7 downto 0);--写数据暂存信号
signal  Rxdata_r:std_logic_vector(7 downto 0);--读数据暂存信号
signal  Rxdata_r_l:std_logic_vector(7 downto 0);--温度低位字节
signal  Rxdata_r_h:std_logic_vector(7 downto 0);--温度高位字节
--
signal uart_clk:std_logic;--串口时钟
signal txd_done_iner:std_logic;--串口发送完毕
signal tx_start:std_logic;--转换完成标志，上升沿
signal tx_data:std_logic_vector(7 downto 0);--待发送数据

type state is (sys_init,wait_conver,reg_w,reg_r,trasmit_data);--DS18B20操作状态
signal ds18b20_state:state:=sys_init;
```

```
begin
---
gen_us: --分频产生1M脉冲
        gen_div generic map(20)--40分频的,产生us脉冲
        port map--分频元件例化
        (
            clk=>clkin,
            resetb=>not resetin,
            bclk=>clk_us
        );
----
gen_uart_clk: --分频产生115200*16脉冲
        gen_div generic map(11)--22分频
        port map--分频元件例化
        (
            clk=>clkin,
            resetb=>not resetin,
            bclk=>uart_clk
        );
uart_transmit:
        uart_t port map--串口发送元件调用
        (
            bclkt=>uart_clk,
            resett=>not resetin,
            xmit_cmd_p=>tx_start,
            txdbuf=>tx_data(7 downto 0),
            txd=>txd_out,
            txd_done=>txd_done_iner
        );

DS18b20_operation:
   process(resetin,clk_us)
   variable cnt:integer range 0 to 1023:=0;--延时计数器，做18b20的时序用
   variable cnt_1:integer range 0 to 15:=0;--发往DS18B20的字节计数器
   variable cnt_2:integer range 0 to 7:=0;--读取DS18B20的字节计数器
   variable cnt_3:integer range 0 to 7:=0;--串口发送数据计数器
   variable cnt_data:integer range 0 to 8:=0;--8bit计数器
   variable delay_cnt2:integer range 0 to 127:=0;--温度转换时的等待延时计数器
   begin
       if resetin='0' then--系统复位期间
           cnt:=0;
           cnt_1:=0;
           cnt_2:=0;
           cnt_3:=0;
           cnt_data:=0;
           delay_cnt2:=0;
           DQ<='Z';--总线高阻
           ds18b20_state<=sys_init;
       elsif rising_edge(clk_us) then
           case ds18b20_state is
               -------------
               when sys_init=>
                   cnt:=cnt+1;
                   if cnt=1 then--DQ保持总线低电平500us(至少480us)
                       DQ<='0';
                   elsif cnt=501 then--释放总线120us,自动拉高(等待15~60us之间,从端拉低维持60~240us之间)
                       DQ<='Z';
                   elsif cnt=621 then--检测总为低作为从端的初始完毕响应
                       if DQ='0' then--为低，初始完毕
                           cnt:=621;--跳出
                       else --继续等待检测
                           cnt:=620;
                       end if;
                   elsif cnt=1010 then-- 这个时间要足够长，让这个复位应答阶段结束后再进行下个操作
                       cnt:=0;
                       ds18b20_state<=reg_w;
                       data_w<=X"CC";--忽略ROM匹配，复位后接着必是写该ROM命令
                   end if;
               ------------
```

```
122                 when reg_w=>
123                     cnt:=cnt+1;
124                     if cnt=1 then--DQ保持总线低电平10us
125                         DQ<='0';
126                     elsif cnt=10 then--数据维持80us,让18b20采样(采样时间是第15～第45us)
127                         if data_w(cnt_data)='1' then--LSB first
128                             DQ<='Z';--释放
129                         elsif data_w(cnt_data)='0' then
130                             DQ<='0';--维持低
131                         end if;
132                         cnt_data:=cnt_data+1;
133                     elsif cnt=101 then--拉高DQ,准备下一个BIT发送
134                         DQ<='Z';
135                     elsif cnt=105 then
136                         if cnt_data=8 then--8bit数据发完，跳出
137                             cnt:=105;
138                             cnt_data:=0;
139                         else
140                             cnt:=0;--循环发送数据
141                         end if;
142                     elsif cnt=106 then--8bit发生完毕，修改状态机和待写入DS18B20数据
143                         --DQ<='Z';--释放总线,自动拉高
144                         cnt:=0;
145                         cnt_1:=cnt_1+1;
146                         case cnt_1 is
147                         when 1 =>
148                             data_w<=X"4E";--设置暂存reg
149                         when 2 =>
150                             data_w<=X"64";--温度上限
151                         when 3 =>
152                             data_w<=X"8A";--温度下限
153                         when 4 =>
154                             data_w<=X"1F";--配置reg
155                         when 5 =>
156                             ds18b20_state<=sys_init;--复位
157                         when 6 =>
158                             data_w<=X"44";--温度转换
159                         when 7 =>
160                             ds18b20_state<=wait_conver;--等待转换结束
161                         when 8 =>
162                             data_w<=X"BE";--读取reg
163                         when 9 =>
164                             ds18b20_state<=reg_r;--读温度寄存器
165                             cnt_1:=0;--清零了
166                         when others=>
167                             null;
168                         end case;
169                     end if;
170                 when wait_conver=>  --延时100ms,等待转换完毕
171                     DQ<='Z';
172                     cnt:=cnt+1;
173                     if delay_cnt2>=127 then
174                         ds18b20_state<=sys_init;--跳到init,准备第三次复位
175                         cnt:=0;
176                         delay_cnt2:=0;
177                     else
178                         if cnt>=1023 then
179                             delay_cnt2:=delay_cnt2+1;
180                             cnt:=0;
181                         end if;
182                     end if;
```

```
183                 when reg_r=>
184                     cnt:=cnt+1;
185                     if cnt=1 then--DQ保持总线低电平5us(拉低至少1us)
186                         DQ<='0';
187                     elsif cnt=5 then--释放总线，自动拉高电平，让18B20送数到总线上
188                         DQ<='Z';
189                     elsif cnt=14 then--在第14us开始采样数据,并延时大约40us完成读操作
190                         Rxdata_r(cnt_data)<=DQ;--LSB first
191                         cnt_data:=cnt_data+1;
192                     elsif cnt=61 then--延时完毕，拉高DQ，准备下一次采样数据
193                         DQ<='Z';
194                     elsif cnt=65 then
195                         if cnt_data=8 then--8bit数据接完，跳出
196                             cnt:=65;
197                             cnt_data:=0;
198                         else
199                             cnt:=0;--循环接收数据
200                         end if;
201                     elsif cnt=66 then--一字节数据接收完毕，保存接受数据
202                         cnt:=0;
203                         cnt_2:=cnt_2+1;
204                         case cnt_2 is
205                         when 1 =>
206                             Rxdata_r_l<=Rxdata_r;
207                         when 2 =>
208                             Rxdata_r_h<=Rxdata_r;
209                             cnt_2:=0;--清零了
210                             ds18b20_state<=trasmit_data;
211                             tx_data<=Rxdata_r;--先发送高温度字节
212                         when others=>
213                             null;
214                         end case;
215                     end if;

216                 when trasmit_data=>--发送温度的高字节
217                     cnt:=cnt+1;
218                     if cnt=1 then
219                         tx_start<='1';
220                     elsif cnt=50 then
221                         tx_start<='0';
222                     elsif cnt=51 then
223                         if txd_done_iner='0' then
224                             cnt:=50;--继续等待
225                         else
226                             cnt:=51;--跳出
227                         end if;
228                     elsif cnt=52 then
229                         cnt:=0;
230                         cnt_3:=cnt_3+1;
231                         case cnt_3 is
232                         when 1 =>
233                             tx_data<=Rxdata_r_l;--发送LSB
234                         when 2 =>
235                             ds18b20_state<=sys_init;--重新开始下一次采样
236                             cnt_3:=0;
237                         when others=>
238                             null;
239                         end case;
240                     end if;
241                     ------
242                     when others=>
243                         ds18b20_state<=sys_init;
244                         cnt:=0;
245                         cnt_1:=0;
246                         cnt_2:=0;
247                         cnt_3:=0;
248                         cnt_data:=0;
249                         delay_cnt2:=0;
250                         DQ<='Z';--总线空闲
251                 end case;
252         end if;
253     end process;
254 end behav;
```

逐行解释：

11：40M 输入时钟，复位信号输入（低电平复位）
12：数据输入/输出端口。因为是双向的，所以用 inout 类型
13：串口发送输出
18-25：分频元件声明
28-35：串口发送元件声明
38：分频出的 us 时钟,做为产生 DS18B20 时序的基准时钟
40：写到 DS18B20 数据的暂存寄存器
41：从 DS18B20 读出数据的暂存寄存器
42-43：分别是从 DS18B20 读出温度的低字节和高字节
45：串口发送时钟，分频得来，采用 115200 的比特率，所以该时钟的频率为 16 倍波特率，即 115200*16Hz
46：串口发送一字节完毕标志
47：启动串口发送信号
48：待发送的数据
50：自定义一种枚举类型 state，表示本程序 DS18B20 操作的各个状态。sys_init--DS18B20 复位； wait_conver--等待 DS18B20 温度转换完毕； reg_w—写 DS18B20;reg_r—读 DS18B20;trasmit_data—串口发送温度值
51:定义 state 类型的信号 ds18b20_state
55-62：分频元件实例化，产生 T=1us 的脉冲，用作产生 DS18B20 时序的基准时钟
64-70：分频元件实例化，产生用于串口发送的时钟，f=115200*16Hz
72-81：串口发送元件实例化
83-255：**DS18B20 的操作过程:配置 DS18B20，实时读取数据，然后通过串口发送到上位机**
85：延时计数器，对 clk_us 进行计数，用于产生 DS18B20 时序
86：发往 DS18B20 的字节计数器,用该计数器来标识下一个要发送到 DS18B20 的数据
87：读取 DS18B20 的字节计数器,标识当前读到的 DS18B20 的数据
88：串口发送数据计数器，标识下一个需要发送的串口数据
89：8bit 数据计数器，用来计数从 DS18B20 顺序读出的 BIT 位
90：延时计数器，DS18B20 温度转换时的延时等待用
92-100：系统复位。清零所有的计数器，然后把总线 DQ 拉成高阻，表示无数据，状态机 ds18b20_state 置为 sys_init，即 DS18B20 复位状态，准备在系统复位后立即进入 DS18B20 的复位操作。
101-253：操作 DS18B20。用 case 语句形成的状态机来实现整个操作流程。
104-121：**DS18B20 复位操作**。按复位操作时序进行控制就可以了。
105：计数器加 1
106-107：置总线 DQ 为低电平，并保持至少 480us，这里我们控制 DQ 为低并保持 500us
108-109:置 DQ 为高阻 Z，即释放总线（因为外界上拉电阻的作用，DQ 会变成高电平），并等待 120us,让 DS18B20 拉低电平作为相应
110-115:检测 DQ 上的状态，判断 DS18B20 是否拉低总线作为应答。若检测到 DS18B20 的应答，则跳到 116 继续执行，否则回到 110 继续检测 DS18B20 的应答信号。
116-119:在检测到 DS18B20 的应答信号后需要再等待一段时间，保证 DS18B20 的应答信号完全结束。然后把发送到 DS18B20 的数据置为 0xCC,并把状态机置

reg_w--写 DS18B20，准备下一步往 DS18B20 写入忽略 ROM 匹配命令。

122-169：**写 DS18B20**。在该段程序中，做一个写操作的时序，并通过 cnt_1 的值不断的修改要写入 DS18B20 的数据。

123：计数器加 1

124-125：拉低 DQ 并保持 10us

126-132:根据 data_w 里每一 bit 的值置 DQ 为不同的电平状态:若为 1，则置 DQ 为高阻（实质就是为高电平，因为有上拉电阻的作用，以下同）；若为 0，则置 DQ 为低电平。把 cnt_data 加 1，使指向下一个需要发送的 bit 位。每一 bit 数据的保持时间为 80us

133-134：置 DQ 为高阻，准备下一 BIT 的发送

135-141：判断一字节的数据是否发送完毕，若发送完毕，下一步就跳到 142，否则下一步就跳到 124，继续发送剩下的 bit 位

142-168：发送完毕一个字节后，修改下一个需要发送的数据 data_w 和状态机状态。cnt_1 为 1,2,3,4 时，都是复位操作里写 DS18B20 的操作，所以并不改变状态机的状态，使其一直在 DS18B20 的写状态--reg_w 里。当 cnt 为 5 时，跳到 DS18B20 的复位状态，整个操作执行到温度转换过程中。当 cnt 为 6 时，写 DS18B20,发送温度转换命令。当 cnt 为 7 时，跳到等待温度转换状态--wait_conver 里。当 cnt 为 8 时，写 DS18B20，发送读暂存寄存器命令。当 cnt 为 9 时，跳到 DS18B20 读状态 reg_r 里，开始读取暂存寄存器中的温度字。

170-182：**等待 DS18B20 温度转换完成的延时状态**。延时完毕后跳入 DS18B20 复位状态 sys_init，进行第三次复位。

183-200：**读 DS18B20**。在该段程序中，做一个读 DS18B20 的时序，只读出前两个温度字。

184：计数器加 1

184-185：拉低 DQ 并保持 5us

187-188:释放总线 DQ，让 DS18B20 把数据放到 DQ 上

189-191:读取 DQ 上的数据，并根据 cnt_data 的值依次放入接收寄存器 Rxdata_r 中

192-193：释放总线，完成 1bit 的读取

194-200：判断一字节的数据是否接收完毕，若接收完毕，下一步就跳到 201，否则下一步就跳到 185，继续接收剩下的 bit 位

201-215：把接收到的一字节数据保存起来。并判断两字节的温度字节是否已经接收进来，若两字节的温度字节已经接收进来，则准备好要往串口发送的数据，并将状态机置为串口发送状态--trasmit_data。首先接收到的是温度低字节，然后是温度高字节。但串口发送的时候是先发送温度高字节。

216-240：**串口发送温度字节**。先发送温度高字节。

218-219：置高 tx_start,启动串口发送。该次要发送的数据已经在第 211 句中准备好了

220-221：拉低 tx_start

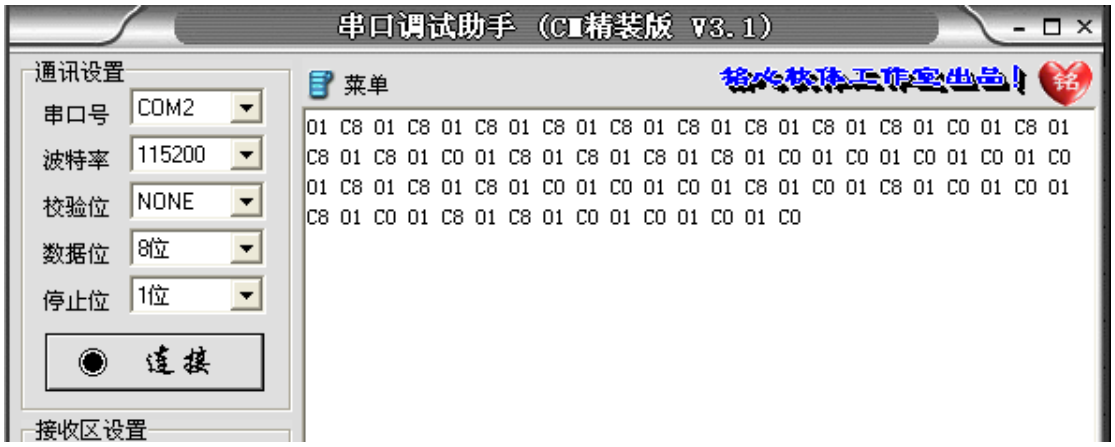
222-227：通过 txd_done_iner 信号判断一字节的串口发送是否完毕。若发送完毕，则跳到 228，否则跳到 222 继续等待。

228-240：若两字节的温度值没有发送完毕，则下一步会跳到 216 继续发送数

据，若已经把两字节数据发送出去了，则将状态机置为 sys_init 状态，进行新一轮的 DS18B20 的操作，这样就形成了实时温度的采集了。

242-250：**其它状态**。清零计数器，高阻 DQ,并将状态机置为 sys_init 状态，使在下一个时刻状态机回到正常状态里。

因为串口发送出来的是未经处理过的温度值,所以在串口调试助手中得到的数据还需要手动转换一下才能变成温度值。比如这是串口收到的温度值：



我们可以得知 01 C8 是一个整体的温度值，01 是高字节温度值，C8 是低字节温度值，我们把两者都转换为二进制值：0000 0001 1100 1000 ，其中高 5 位是符号位，因为高 5 位全是 0，所以温度值是正值。由于我们设置的是 9 位精度（精度为 0. 5°），也就是说低 4 位的小数值中只有 BIT3 有效，而且这里 BIT3=1，所以小数位就是 0.5。我们再把高 5 位和低 4 位去掉，得到这个二进制值：0011100，转换成十进制就是 28，这个就是温度的整数值，所以合起来的温度值就是 28.5 度。

大家可以用这种方法计算出温度值，也算是对先前知识的复习巩固了！