同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2251499

姓　　名 桂欣远

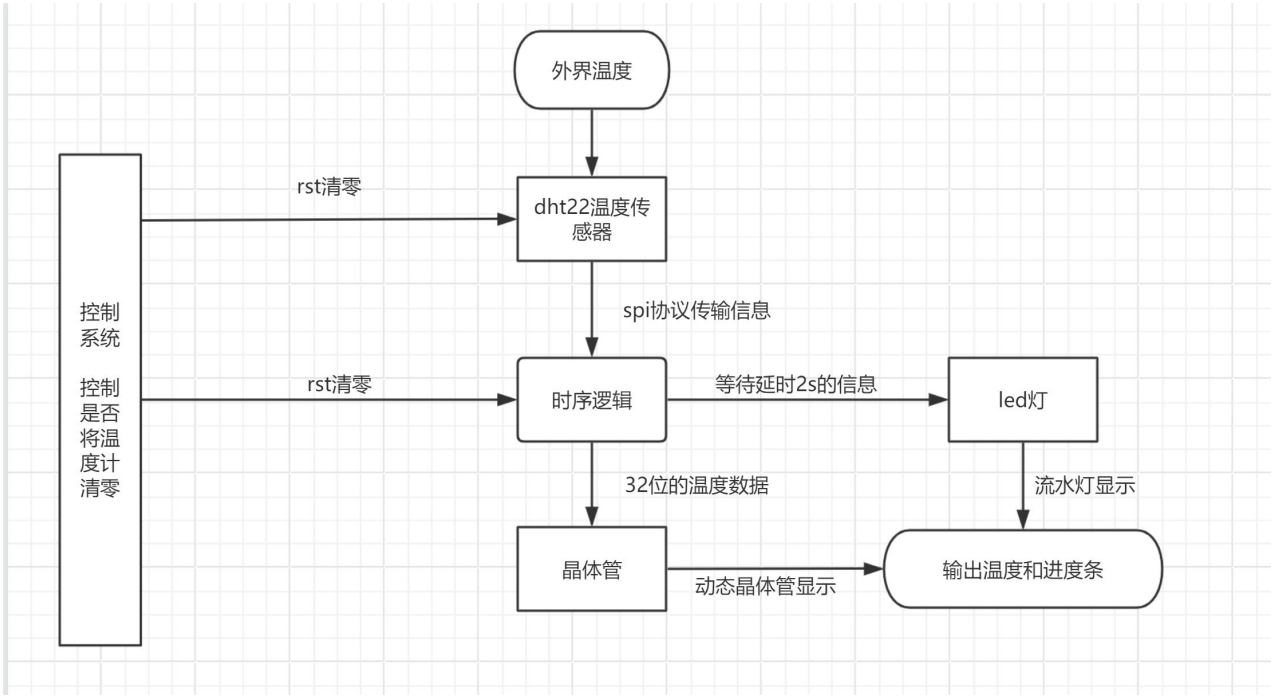
专　　业 计算机科学与技术

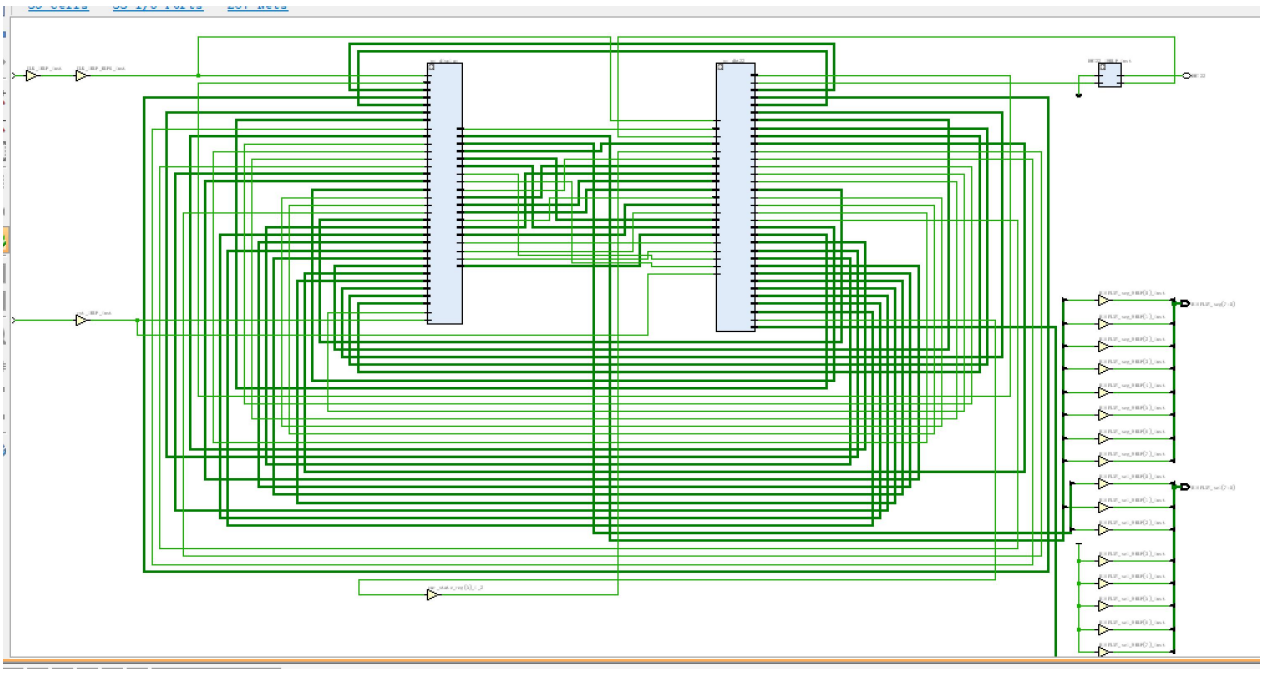
授课老师 郭玉臣

一、实验内容

基于 DHT22 的温度显示器系统

二、温度显示器数字系统总框图



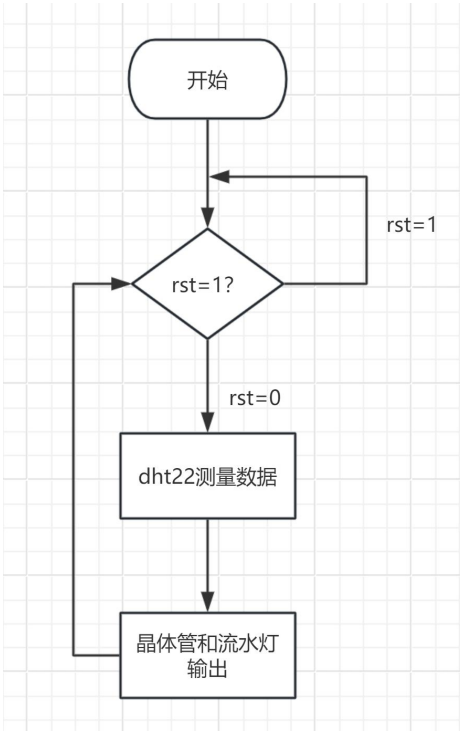


三、系统控制器设计

顶层模块：

连通电源后整个系统一直处于读取温度，显示温度的状态。只有按下 reset 键后才会温度清零，重新测温。

1. ASM 流程图



2. 状态转移真值表

现态输入/rst	次态/结果
rst=1	数据清零，等待 rst 为 1
rst=0	开始测量数据并输出

3. 次态函数表达式：

All_data=!rst&&data_in(顶层模块中无时序逻辑，只有组合逻辑)

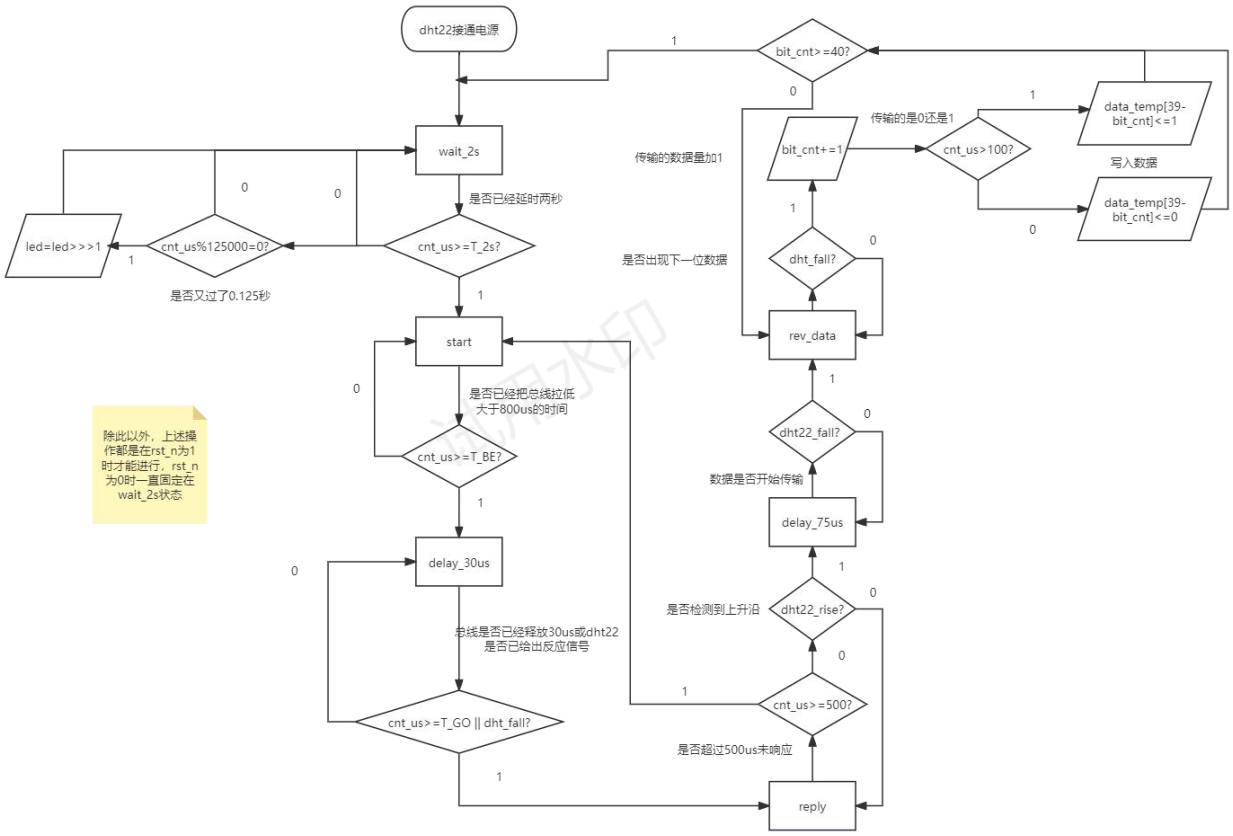
4. 逻辑表达式：

All_data=!rst&&data_in

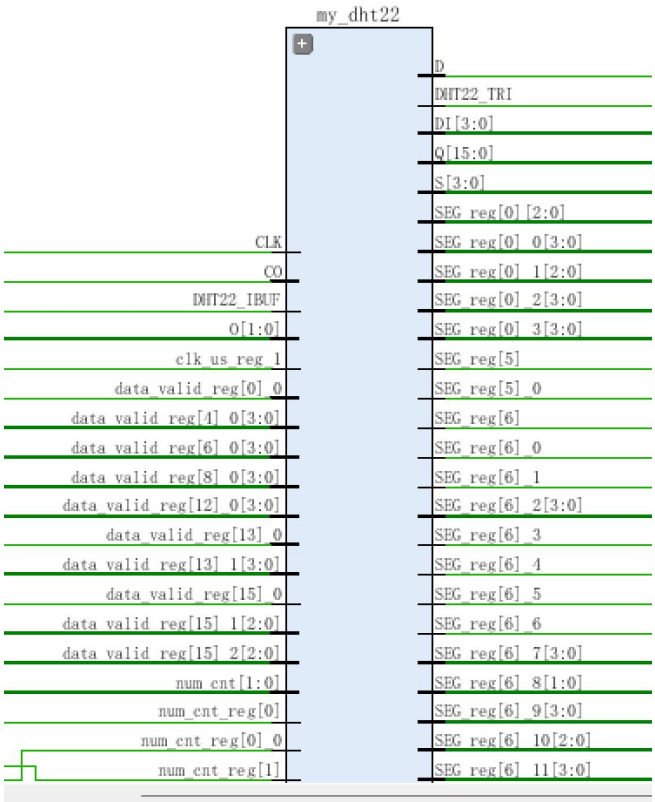
四、子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模，图文描述，描述清楚设计及实现的思路，不要在此处放 verilog 代码，所有的 verilog 代码在附录部分）

以下为 dht_22 模块总的 ASM 流程图：



1　dht22 通信协议状态机:



（1）功能简述

完成与 dht22 的通信，通过 inout 端口 dht 的总线获取 dht22 检测出来的温湿度数据。并将最后的数据输入到 data_valid 的网线中。

（2）端口介绍

名称	类型	功能
dht	Inout	数据传输的总线
Rst_n	input	复位按钮
Sys_clk	input	Fpga 的 100MHz 时钟
Data_valid	output	接收来自传感器的数据
LED	output	根据延时时间显示进度条

（3）参数介绍

状态参数（使用独热码）

名称	值	功能
WAIT_2S	6'b000001	通信前延时的部分
START	6'b000010	主机发出低电平，唤醒 dht22 的部分
DELAY_30us	6'b000100	主机释放总线，等待的 dht22 回应
REPLY	6'b001000	Dht22 回应 75us 低电平
DELAY_75us	6'b010000	Dht22 回应 75us 高电平
REV_data	6'b100000	Dht22 开始传输数据，主机接受数据

时间参数定义

名称	值	功能
T_2S	1999999	上电 1s 延时计数，单位 us
T_BE	999	主机起始信号拉低时间，单位 us
T_GO	30	主机释放总线时间，单位 us

（4）变量介绍

名称	类型	功能
cur_state	reg[6:0]	现态
next_state	reg [6:0]	次态
cnt	reg[5:0]	50 分频计数器，1Mhz(1us)
dht22_out	reg	双向总线输出
dht22_en	reg	双向总线输出使能，1 则输出，0 则高阻态
dht22_d1	reg	总线信号打 1 拍
dht22_d2	reg	总线信号打 2 拍
clk_us	reg	us 时钟
cnt_us	reg[21:0]	us 计数器,最大可表示 4.2s
bit_cnt	reg[5:0]	接收数据计数器，最大可以表示 64 位
data_temp	reg[39:0]	包含校验的 40 位输出

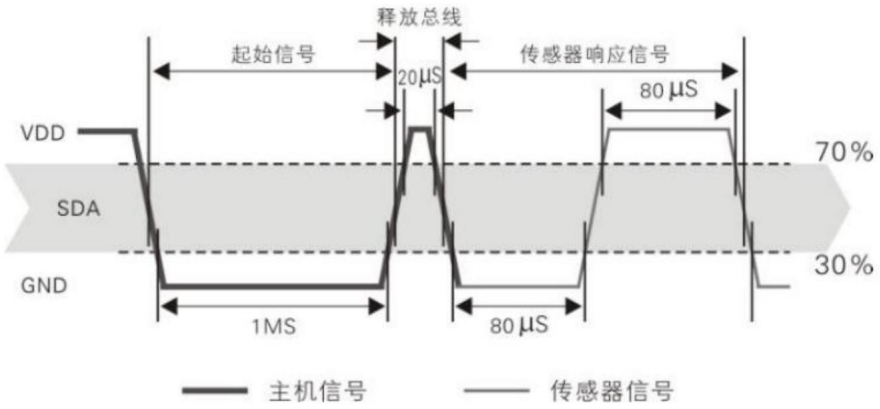
progress_bar	reg signed [15:0]	等待的进度条
dht22_in	wire	双向总线输入
dht22_rise	wire	上升沿
dht22_fall	wire	下降沿

（5）设计思路

首先要先确定总的状态数量和每个状态之间的联系，这就需要分析 dht22 的通信时序了：

AM2302 上电后要等待 2S 以越过不稳定状态，在此期间读取设备不能发送任何指令），测试环境温湿度数据，并记录数据，此后传感器自动转入休眠状态。

之后就是与其响应的信号，电位图如下：



由此分析，要进行通信首先要经历几个阶段：

WAIT_1S：上电后就进入这个状态，因为上电后需要 2s 的是时间稳定 DHT22 的状态。在这个状态使用计数器计时，满足 2s 就跳转到下一个状态 START，并且计数清零；不满足就停留在这个状态一直计数

START：在这个状态 DHT22 进入工作状态。主机将总线拉低 1ms 左右后再拉高。在这个状态仍使用计数器计时，拉低总线后计数开始，计数满足条件后清零并跳转到下一个状态 DELAY_30us；不满足就停留在这个状态一直计数

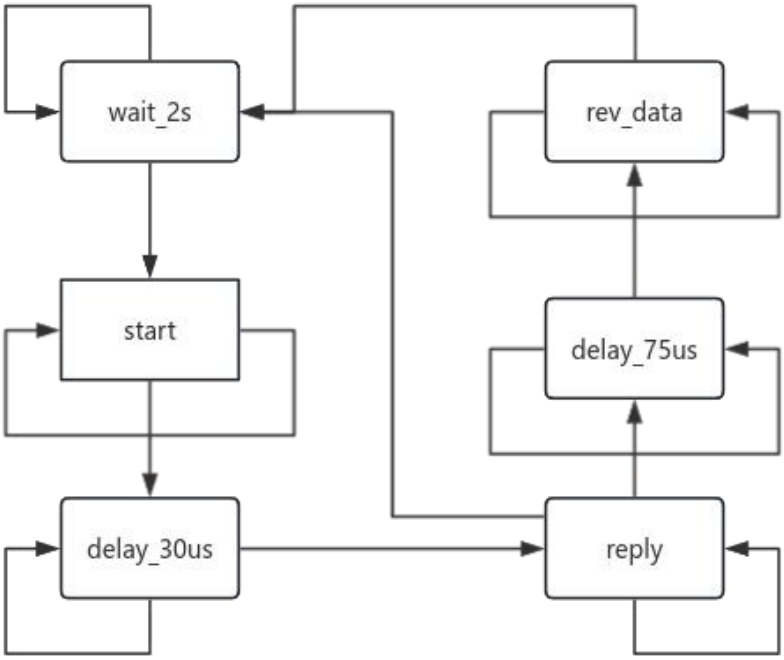
DELAY_30us：在这个状态主机应拉高 30us 后释放总线进入等待状态等待从机（DHT22）拉低作为回应。在这个状态仍使用计数器，拉低总线后计数开始，计数满足条件后清零并跳转到下一个状态 REPLY；不满足就停留在这个状态一直计时

REPLY：这个状态是检查从机是否回应以及回应是否符合时序。一旦主机检测到上升沿且此时计数器介于 75us~85us，则说明从机回应了一个 75us~85us 的低电平信号，就跳转到下一个状态 DELAY_75us；一旦主机检测到

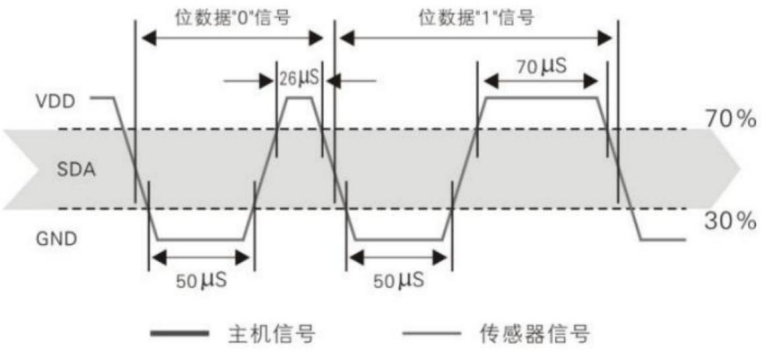
上升沿但此时计数器大于 85us 或者计数器大于 500us 时仍没有检测到上升沿，则说明从机的应答是不符合时序的，那么状态机跳转到 START 状态（不跳转到 WAIT_2S 状态，是因为只有在每次上电后才需要延迟 2s）并将计数器清零，在这个状态计数器一直计数知道跳转到另一个状态才清零（边沿检测可以参考：边沿检测电路（上升沿、下降沿））

DELAY_75us：这个状态是从机输出 75us~85us 的高电平通知主机准备接收数据。因为后续发送数据时一定是以低电平开始，所以在这里一定会检测到数据线上的一个下降沿。所以一旦检测到下降沿且计数器大于 75us，那么跳转到下一个状态 REV_data 并将计数器清零；不然就停留在这个状态一直计时

可得状态函数为：NS=PS<<1
状态转移图如下：



对于 REV_data 状态中的数据处理，dht22 发送的信息模式如下：



位数据“0”的格式为： 50 微秒的低电平加 26-28 微秒的高电平；
位数据“1”的格式为： 50 微秒的低电平加 70 微秒的高电平；
所以只需要在两个下降沿之间计时，若间隔时间大于 100us 则是 1，否则为 0。
记录完所有的 40 位数据后，要对数据进行检验，判别式为：
data_temp[7:0] == data_temp[39:32] + data_temp[31:24] +data_temp[23:16] + data_temp[15:8]
若该等式成立则可以将数据输入到 data_valid 中。由于两次测量之间至少要间隔 2s，就直接回到 wait_2s 阶段。

2 微秒分频器/计数器

（1）功能简述

Dht_22 的使用和传输信号的协议使用的是 spi 协议，这需要将 fpga 自带的 100MHz 的时钟放慢到 1MHz，才能对 us 进行计时。为了方便直接使用微秒计数器，该部分的模块写到了 dht22_drive module 中了。

（2）端口介绍

名称	类型	功能
clk	Input	系统时钟，100MHz
Rst	input	复位按钮
Clk_us	output	1us 的时钟，1MHz
Cnt_us	output	1us 的脉冲波

（3）设计思路

就是基本的一个分频器，倍数为 100 倍，同时再加一个 1us 的计数器，每一百个周期，cnt_us 就加一。

3 上升/下降沿检测

（1）功能简述

因为要检测 dht22 的回复数据，所以要能够检测出连接总线中的上升沿和下降沿。该部分输入一个总线数据，输出上升沿和下降沿的信号。为了方便直接使用微秒计数器，该部分的模块也写到了 dht22_drive module 中了。

（2）端口介绍

名称	**类型**	**功能**
dht	Inout	数据传输的总线

Rst	input	复位按钮
Dht_rise	output	检测上升沿，出现上升沿则置 1，否则置 0
Dht_fall	output	检测下降沿，出现下降沿则置 1，否则置 0

（3）设计思路

在总线有起伏时打拍，一次记录双拍，分别即为 dht_d1 和 dht_d2，如果 dht_d1 为 0，但 dht_d2 为 1，则为上升沿；如果 dht_d1 为 1，dht_d2 为 0，则为下降沿。

4 进度条/流水灯输出

（1）功能简述

因为在每两次测量之间有一个两秒的间隔，可以利用 fpga 上自带的 led 灯设计一个进度条。输入的是系统的 100MHz 的时钟，输出为每 0.125 秒进行算数右移的 16 位数据。该模块主要的功能与 wait_2s 的状态相关，且为了方便直接使用微秒计数器，该部分的模块也写到了 dht22_drive module 中了。

（2）端口介绍

名称	类型	功能
clk	Input	系统时钟，100MHz
LED	Ouput	实现流水灯（进度条）的效果

（3）设计思路

利用微秒计数器，刚进入 wait_2s 状态时对 LED 清零，在首次计数为 125000 时给输出赋值为 16'b1000000000000000，之后每次计时器的值%125000==0 时将输出的值算数右移。

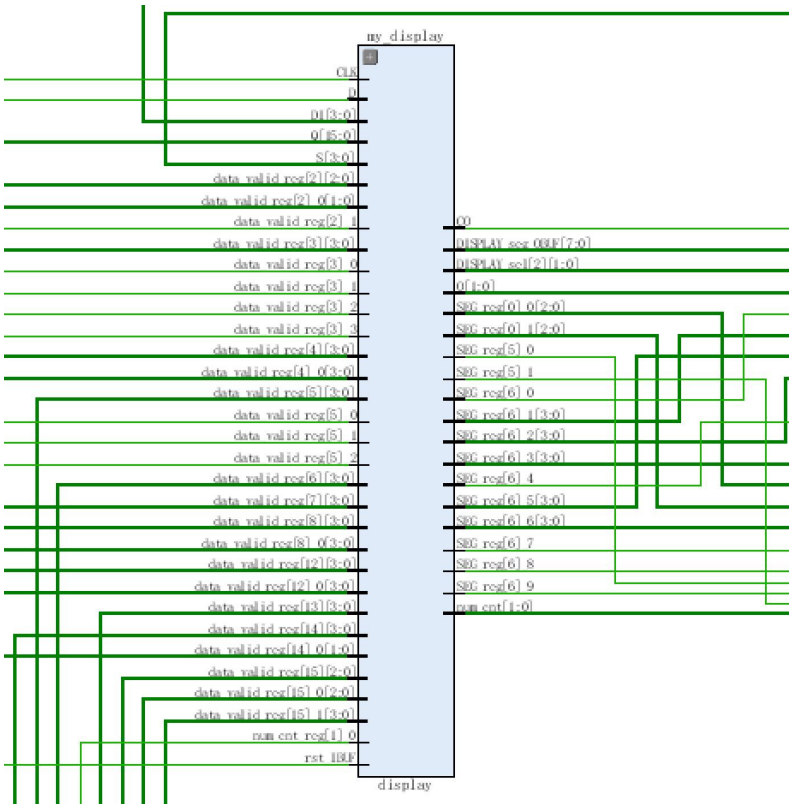
5 晶体管显示数字

（1）功能简述

输入一个 16 位的温度数据，输出对晶体管的控制信号，即位选和段选信号，存储在两个 8 位数据中。

（2）端口介绍

名称	类型	功能
Clk	Input	系统时钟，100MHz
Reset_n	input	复位按钮
Disp_data	input	16 位的温度数据
SEL	output	位选信号
SEG	output	段选信号



（4）变量介绍

名称	类型	功能
num_cnt	reg [1:0]	状态标号
clk_5ms	reg	5ms 的脉冲信号
div_cnt	reg [19:0]	计数器
number	integer[2:0]	储存十进制数字的整形

（5）设计思路

因为只要控制三个数字的显示（不考虑温度大于 100 度和小于 0 度的情况），所以状态就转移很明了：00->01->10->00

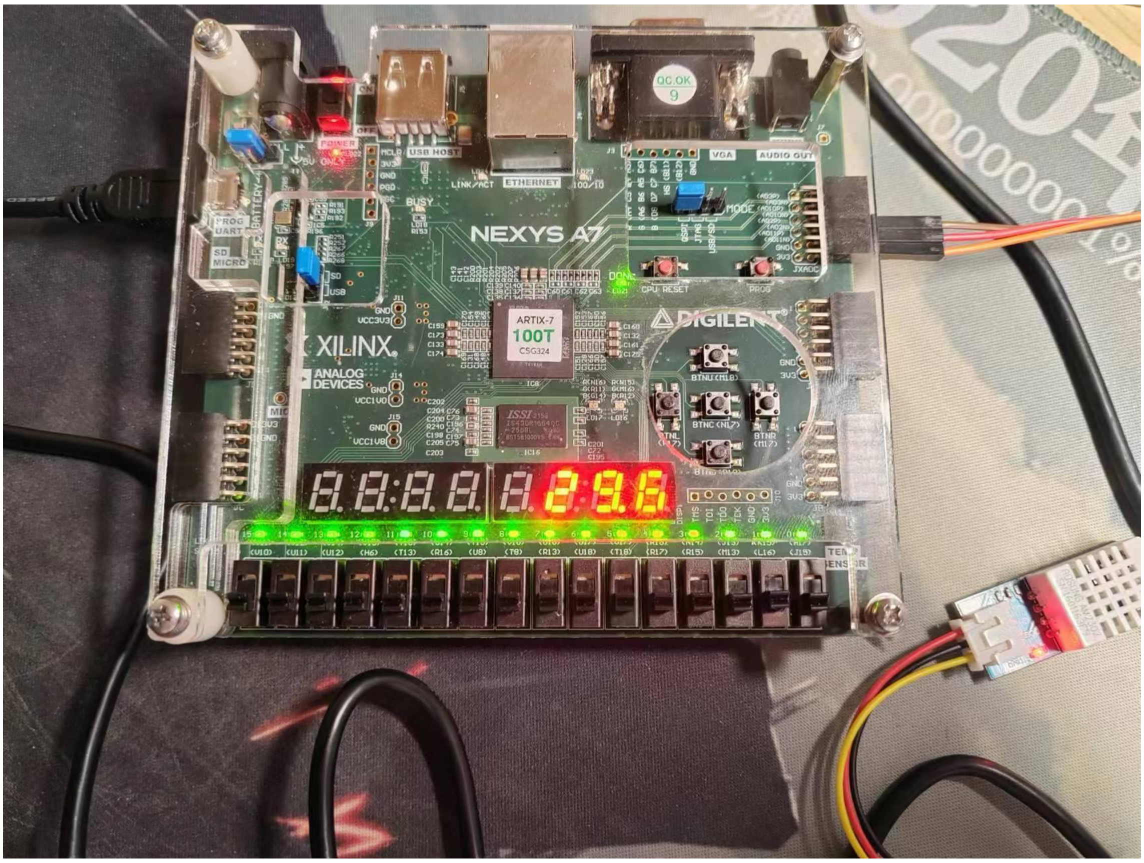
因为dht22 的温度数据存储是将温度的十倍的值转为二进制存储在一个两字节的数据内。所以可以先将这个数整除 100，得到百位数，整除 10 再模 10，得到十位数，直接模 10，得到个位数，存储到数组中。因为多位动态晶体管的引脚方式是段选的片段是串联在一起的，所以要利用人的视觉暂留效果（大概 16ms）来遍历每一位数字，可以将每一位遍历的数字的时间控制在 5ms，使用一个分频器来得到 5ms 的时钟。之后根据位选的数字来确定段选的信号。在 15ms 内刷新一次数据就在肉眼看起来是一直连续亮着的。

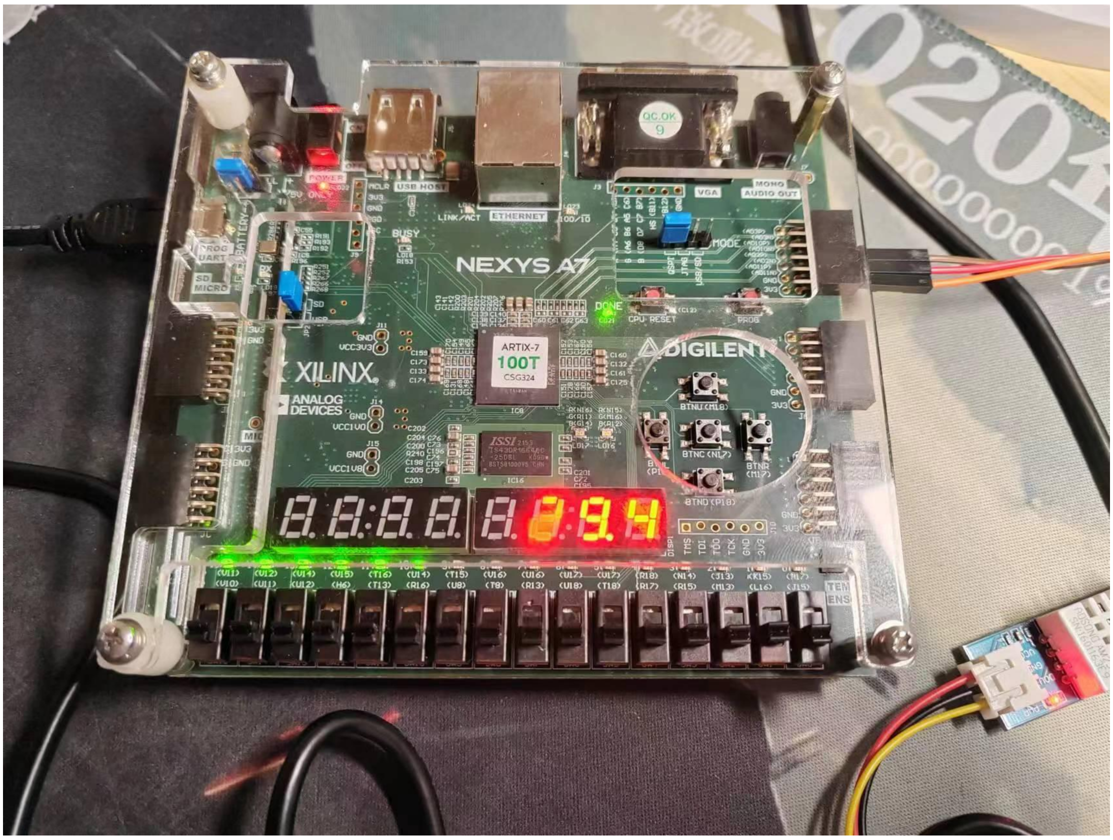
五、测试模块建模

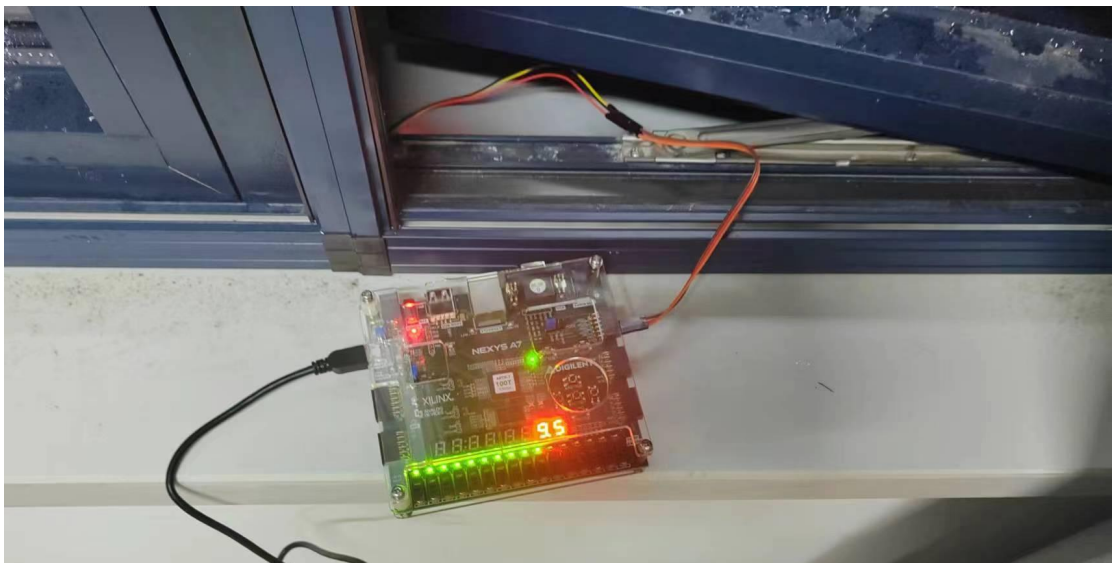
本次实验没有使用 testbanch，是直接下版，通过将部分变量和状态的值输出到 fpga 板上的 led 灯来观察结果并调试的，因此无 tb 文件和仿真结果。

六、实验结果

实时温度显示的效果（刷新频率：0.5Hz）


NEXYS A7
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
USB HOST
ETHERNET
VGA
AUDIO OUT


NEXYS A7
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
USB HOST
ETHERNET
VGA



七、结论

该数字系统使用了 dht22 温湿度传感器和 fpga 开发板实现了电子温度计的功能。总体上分析，先要对 dht22 的通信协议进行解析，之后使用有顶向下的设计方法设计出一套通信时序逻辑的状态机，之后得到了温度的数据后利用开发板上已有的能够显示输出信号的器件将数据输出出来，这里使用了多位动态晶体管来输出温度的数字和 led 灯来显示进度条。完成了由输入信号到输出的过程。

八、心得体会及建议

1.如果时间充裕的话应该去学习一下 vivado 的调试和 debug 功能

温度传感器的通信数据变化多，而且有较高的不确定性，所以人工设置外部输入的信号会比较麻烦而且不可靠，就只能将模块中的过程量和一些变量输出到 led 灯上来反应逻辑运行时的情况。但这样需要不停地调整输出和约束文件，而且还要不停的综合实现看下板结果，效率非常低。如果可以边下板边在电脑上看到不同状态的转移和变量的值，debug 会轻松的多。

2.dht22 传感器的一些通信数据不是绝对，往往是一个范围

在 debug 时发现最开始总是卡在了 delay_75us 的状态，无法进入最终读入数据的阶段。之后发现是因为设置的状态转移的条件太苛刻了，说明文档中说 dht22 响应的高电平是 75us~85us,实际我 debug 出来感觉范围要大得多。设置成了 55us~105us 后，才能终于进入到了 rev_data 状态。但是又发现卡在了 rev_data 状态，且显示收到的数据一共只有 22 个（收到 40 个数据才转移状态）推测原因可能还是状态转移的条件每设定好，导致一部分要接受的数据在之前的状态没有被接收到。将 dht22 响应的两个状态的转移条件去除时间限制后就可以正常运行了。

九、附录

（该部分放 verilog 代码，包括所有设计文件和测试文件）

顶层模块：

因为微秒计数器模块、边沿检测模块、流水灯输出模块比较简短且与 dht22 通信协议状态机密切相关就将这些模块的内容都写入到了 dht22_drive 模块中了，这样的参数的转移更少，使用起来更方便。所以在顶层模块中只有两个子模块。

```
module main(
    input CLK,//系统时钟
    input rst,//复位
    output [7:0] DISPLAY_sel,//晶体管位选输出
    output [7:0] DISPLAY_seg,//晶体管段选输出
    output signed [15:0] led,//流水灯输出
    inout DHT22//与 dht22 传输数据的总线
    );
    wire [31:0]temperature;//从 dht22 得到的数据
    dht22_drive my_dht22(CLK,!rst,DHT22,temperature,led);//dht22 温度传感器
    display my_display(CLK,!rst,temperature[15:0],DISPLAY_sel,DISPLAY_seg);//晶体管显示
Endmodule
```

dht22_drive 通信协议状态机驱动模块+微秒计数器模块+边沿检测模块+流水灯输出模块：

```
//------------<模块及端口声明>----------------------------------------
module dht22_drive(
    input               sys_clk     ,       //系统时钟，100M
    input               rst_n       ,       //低电平有效的复位信号
    inout               dht22       ,       //单总线（双向信号）

    output  reg [31:0]  data_valid  ,       //输出的有效数据，位宽 32
    output  wire signed [15:0] LED
);

//------------<参数定义>----------------------------------------------
//状态机状态定义，使用独热码（onehot code）
localparam  WAIT_2S     = 6'b000001 ,
            START       = 6'b000010 ,
            DELAY_30us  = 6'b000100 ,
            REPLY       = 6'b001000 ,
            DELAY_75us  = 6'b010000 ,
            REV_data    = 6'b100000 ;
//时间参数定义
localparam  T_2S = 1999999  ,               //上电 1s 延时计数，单位 us
            T_BE = 999  ,                   //主机起始信号拉低时间，单位 us
            T_GO = 30       ;               //主机释放总线时间，单位 us

//------------<reg 定
义>----------------------------------------------
```

```
reg [6:0]   cur_state   ;                   //现态
reg [6:0]   next_state  ;                   //次态
reg [5:0]   cnt         ;                   //50 分频计数器，1Mhz(1us)
reg         dht22_out   ;                   //双向总线输出
reg         dht22_en    ;                   //双向总线输出使能，1 则输出，0 则高阻态
reg         dht22_d1    ;                   //总线信号打 1 拍
reg         dht22_d2    ;                   //总线信号打 2 拍
reg         clk_us      ;                   //us 时钟
reg [21:0]  cnt_us      ;                   //us 计数器,最大可表示 4.2s
reg [5:0]   bit_cnt     ;                   //接收数据计数器，最大可以表示 64 位
reg [39:0]  data_temp   ;                   //包含校验的 40 位输出
reg signed [15:0]  progress_bar;            //等待的进度条
//------------<wire 定义>-------------------------------------------
wire        dht22_in    ;                   //双向总线输入
wire        dht22_rise  ;                   //上升沿
wire        dht22_fall  ;                   //下降沿
//-----------------------------------------------------------------------
//--双向端口使用方式和进度条
//-----------------------------------------------------------------------
assign  dht22_in = dht22;                           //高阻态的话，则把总线上的数据赋给 dht22_in
assign  dht22 =  dht22_en ? dht22_out : 1'bz;       //使能 1 则输出，0 则高阻态
assign  LED = progress_bar;
//-----------------------------------------------------------------------
//--us 时钟生成，因为时序都是以 us 为单位，所以生成一个 1us 的时钟会比较方便
//-----------------------------------------------------------------------
//100 分频计数
always @(posedge sys_clk or negedge rst_n)begin
    if(!rst_n)
        cnt <= 6'd0;
    else if(cnt == 6'd49)               //每 50 个时钟 500ns 清零
        cnt <= 6'd0;
    else
        cnt <= cnt + 1'd1;
end
//生成 1us 时钟
always @(posedge sys_clk or negedge rst_n)begin
    if(!rst_n)
        clk_us <= 1'b0;
    else  if(cnt == 6'd49)              //每 500ns
        clk_us <= ~clk_us;              //时钟反转
    else
        clk_us <= clk_us;
end
//-----------------------------------------------------------------------
```

```
//--上升沿与下降沿检测电路
//------------------------------------------------------------------------
//检测总线上的上升沿和下降沿
assign  dht22_rise = ~dht22_d2 && dht22_d1;         //上升沿
assign  dht22_fall = ~dht22_d1 && dht22_d2;         //下降沿
//dht22 打拍，捕获上升沿和下降沿
always @(posedge clk_us or negedge rst_n)begin
    if(!rst_n)begin
        dht22_d1 <= 1'b0;               //复位初始为 0
        dht22_d2 <= 1'b0;               //复位初始为 0
    end
    else begin
        dht22_d1 <= dht22;              //打 1 拍
        dht22_d2 <= dht22_d1;           //打 2 拍
    end
end
//------------------------------------------------------------------------
//--三段式状态机
//------------------------------------------------------------------------
//状态机第一段：同步时序描述状态转移
always @(posedge clk_us or negedge rst_n)begin
    if(!rst_n)
        cur_state <= WAIT_2S;
    else
        cur_state <= next_state;
end
//状态机第二段：组合逻辑判断状态转移条件，描述状态转移规律以及输出
always @(*)begin
    next_state = WAIT_2S;
    case(cur_state)
        WAIT_2S     :begin
            if(cnt_us == T_2S)              //满足上电延时的时间
                next_state = START;         //跳转到 START
            else
                next_state = WAIT_2S;       //条件不满足状态不变
        end
        START       :begin
            if(cnt_us == T_BE)              //满足拉低总线的时间
                next_state = DELAY_30us;    //跳转到 DELAY_30us
            else
                next_state = START;         //条件不满足状态不变
        end
        DELAY_30us  :begin
            if(cnt_us == T_GO || dht22_fall)//满足主机释放总线时间
```

```
                next_state = REPLY;         //跳转到 REPLY
            else
                next_state = DELAY_30us;    //条件不满足状态不变
        end
        REPLY       :begin
            if(cnt_us <= 'd500)begin        //不到 500us
                if(dht22_rise)                      //上升沿响应
                    next_state = DELAY_75us;        //跳转到 DELAY_75us
                else
                    next_state = REPLY;             //条件不满足状态不变
            end
            else
                next_state = START;                 //超过 500us 仍没有上升沿响应则跳转到 START
        end
        DELAY_75us  :begin

            if(dht22_fall)                          //下降沿响应
                next_state = REV_data;              //跳转到 REV_data
            else
                next_state = DELAY_75us;            //条件不满足状态不变
        end
        REV_data    :begin
            if( bit_cnt >= 'd40)                    //接收完了所有 40 个数据后会拉低一段时间作为结束
                                                    //捕捉到上升沿且接收数据个数为 40
                next_state = WAIT_2S;               //状态跳转到 WAIT_2S，重新开始新一轮采集
            else
                next_state = REV_data;              //条件不满足状态不变
        end
        default:next_state = WAIT_2S;               //默认状态为 WAIT_2S
    endcase
end

//状态机第三段：时序逻辑描述输出
always @(posedge clk_us or negedge rst_n)begin
    if(!rst_n)begin                                 //复位状态下输出如下
        dht22_en <= 1'b0;
        dht22_out <= 1'b0;
        cnt_us <= 22'd0;
        bit_cnt <=  6'd0;
        data_temp <= 40'd0;
        progress_bar <= 16'd0;
    end
    else
        case(cur_state)
```

```
        WAIT_2S     :begin
            dht22_en <= 1'b0;                       //释放总线，由外部电阻拉高
            if(cnt_us == T_2S)
                cnt_us <= 22'd0;                    //计时满足条件则清零
            else
            begin
               cnt_us <= cnt_us + 1'd1;             //计时不满足条件则继续计时
               if(cnt_us % 125000 == 0)             //每隔 12.5ms，进度条加一
                  if(cnt_us == 125000)
                    progress_bar <= 16'b1000000000000000;
                  else
                    progress_bar <= progress_bar >>> 1;
            end

        end
        START       :begin
            dht22_en <= 1'b1;                       //占用总线
            dht22_out <= 1'b0;                      //输出低电平
            if(cnt_us == T_BE)
                cnt_us <= 22'd0;                    //计时满足条件则清零
            else
                cnt_us <= cnt_us + 1'd1;            //计时不满足条件则继续计时
        end
        DELAY_30us  :begin
            dht22_en <= 1'b0;                       //释放总线，由外部电阻拉高
            if(cnt_us == T_GO || dht22_fall)
                cnt_us <= 22'd0;                    //计时满足条件则清零
            else
                cnt_us <= cnt_us + 1'd1;            //计时不满足条件则继续计时
        end
        REPLY       :begin
            dht22_en <= 1'b0;                       //释放总线，由外部电阻拉高
            if(cnt_us <= 'd500)begin                //计时不到 500us
                if(dht22_rise)                      //上升沿响应
                    cnt_us <= 22'd0;                //计时清零
                else
                    cnt_us <= cnt_us + 1'd1;        //计时不满足条件则继续计时
            end
            else
                cnt_us <= 22'd0;                    //超过 500us 仍没有上升沿响应，则计数清零
        end
        DELAY_75us  :begin
            dht22_en <= 1'b0;                       //释放总线，由外部电阻拉高
            if(dht22_fall)                          //下降沿响应
```

```
                    begin
                    cnt_us <= 22'd0;                    //计时清零
                    bit_cnt <=  6'd0;                   //清空数据接收计数器
                    end
                else
                    cnt_us <= cnt_us + 1'd1;            //计时不满足条件则继续计时
            end
            REV_data    :begin
                dht22_en <= 1'b0;                       //释放总线，由外部电阻拉高，进入读取状态
                if(bit_cnt >= 'd40)begin                //数据接收完毕
                    cnt_us <= 22'd0;                    //清空计时器
                end
                else if(dht22_fall)begin                //检测到低电平，则说明接收到一个数据
                    bit_cnt <= bit_cnt + 1'd1;          //数据接收计数器+1
                    cnt_us <= 22'd0;                    //计时器重新计数
                    if(cnt_us <= 'd100)
                        data_temp[39-bit_cnt] <= 1'b0;  //总共所有的时间少于 100us,则说明接收到"0"
                    else
                        data_temp[39-bit_cnt] <= 1'b1;  //总共所有的时间大于 100us,则说明接收到"1"
                end
                else begin                              //所有数据没有接收完，且正处于 1 个数据的接收
进程中
                    bit_cnt <= bit_cnt;
                    data_temp <= data_temp;
                    cnt_us <= cnt_us + 1'd1;            //计时器计时
                end
            end

            default:;
        endcase
end

//校验读取的数据是否符合校验规则
always @(posedge clk_us or negedge rst_n)begin
    if(!rst_n)
        data_valid <= 32'd0;
    else if((data_temp[7:0] == data_temp[39:32] + data_temp[31:24] +
    data_temp[23:16] + data_temp[15:8]))
        data_valid <= data_temp[39:8];      //符合规则，则把有效数据赋值给输出
    else
        data_valid <= data_valid;           //不符合规则，则舍弃这次读取的数据，输出仍保持上次的状态
不变
end
endmodule
```

晶体管显示模块：

```
module display(
    Clk,
    Reset_n,
    Disp_Data,
    SEL,
    SEG
);
    //--------------------------
    //      输入输出定义
    //--------------------------
    input Clk;//系统时钟
    input Reset_n;//重置
    input [15:0]Disp_Data;//输入的数据
    output reg[7:0]SEL;//晶体管位选
    output reg[7:0]SEG;//晶体管段选

    //--------------------------
    //        变量定义
    //--------------------------
    reg [1:0]num_cnt;//状态标号
    reg clk_5ms;//5ms 的脉冲信号
    reg [19:0]div_cnt;//计数器
    integer number[2:0];//储存十进制数字的整形
    //---------------------------
    //取三位数的不同位
    //---------------------------
    always@*
    begin
    number[0]<=Disp_Data%10;
    number[1]<=Disp_Data/10%10;
    number[2]<=Disp_Data/100;
    end
    //---------------------------
    //形成 5ms 的脉冲波
    //---------------------------
    always@(posedge Clk or negedge Reset_n)
    begin
    if(!Reset_n)
        begin
        clk_5ms<= 0;
        div_cnt <= 0;
        end
```

```
else if(div_cnt == 499_999)//每 500000 个周期就生成一次脉冲
begin
    clk_5ms <= 1;
    div_cnt <= 0;
    end
else
    begin
    clk_5ms <= 0;
    div_cnt <= div_cnt + 1'b1;
    end
end

//----------------------------
//状态转移：00-》01-》10-》00
//----------------------------
always@(posedge Clk or negedge Reset_n)
if(!Reset_n)
    num_cnt <= 0;
else
if(clk_5ms==1)
    begin
    if(num_cnt==2)
    num_cnt <= 0;
    else
    num_cnt <= num_cnt + 1'b1;
    end
else
    num_cnt <= num_cnt;
//---------------------------------
//设置不同状态的位选信号和小数点是否显示
//---------------------------------
always@(posedge Clk)
    case(num_cnt)
        0: begin
        SEL <= 8'b11111110;
        SEG[7]=1;
        end
        1: begin
        SEL <= 8'b11111101;
        SEG[7]=0;
        end
        2: begin
        SEL <= 8'b11111011;
        SEG[7]=1;
```

```
            end
        endcase
    //---------------------------------
    //设置不同状态的段选信号
    //---------------------------------
    always@(posedge Clk)
    case(number[num_cnt])
    0:if(num_cnt!=2)
      SEG[6:0]<=7'b1000000;
      else//若温度为个位数则不用显示十位数上的“0”
      SEG[6:0]<=7'b1111111;
    1:SEG[6:0]<=7'b1111001;
    2:SEG[6:0]<=7'b0100100;
    3:SEG[6:0]<=7'b0110000;
    4:SEG[6:0]<=7'b0011001;
    5:SEG[6:0]<=7'b0010010;
    6:SEG[6:0]<=7'b0000010;
    7:SEG[6:0]<=7'b1111000;
    8:SEG[6:0]<=7'b0000000;
    9:SEG[6:0]<=7'b0010000;
    endcase
endmodule
```