22.DHT11 温湿度传感器使用实验例程

22.1. 实验目的

使用盘古 PGX-Nano 开发板驱动 DHT11 模块读取环境温湿度度并通过串口打印输出

22.2. 实验简介

22.2.1. DHT11 模块介绍

DHT11 是一种数字温湿度传感器，集成了温度和湿度测量功能。它采用电容式感湿元件和负温度系数电阻测温元件，并结合高性能 8 位单片机进行信号的采集处理和输出。DHT11 通过单线数字接口与微处理器通信，输出经过校准的数字信号，温度测量范围为 0 至 50 摄氏度，精度为±2℃；湿度测量范围为 20%至 90%RH，精度为±5%RH。因其良好的稳定性和性价比，被广泛应用于各种环境监测设备、智能家居系统和自动化控制领域。其模块图如下：

模块引脚定义如下：

接口名称	接口描述
VCC	电源正极 3~5.5V
GND	电源负级
DATA	串行数据，单总线

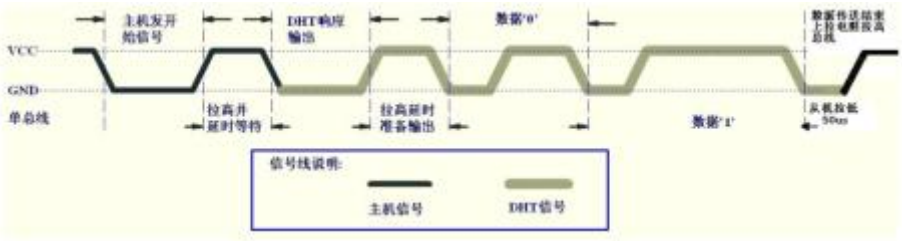
22.2.2. DHT11 通讯时序以及数据格式

DHT11 是一种数字温湿度传感器，它通过单总线协议与微处理器通信。正常情况下，DHT11 处于低功耗模式。当设备需要读取温湿度数据时，首先发送一次复位信号。DHT11 接收到复位信号后，从低功耗模式转换到高速模式，开始执行一次温湿度采集过程，完成后通过单总线发送响应信号，并将总线拉高，准备传输数据。一次完整的数据传输包括 40 位（5 字节）的数据，具体格式为：8 位湿度整数数据 +8 位湿度小数数据 +8 位温度整数数据 +8 位温度小数数据 +8 位校验和。

由于 DHT11 的分辨率限制，湿度和温度的小数部分数据始终为 0。校验和是前 4 个字节数据的累加和，用于验证数据传输的准确性，确保接收端接收到的数据没有错误。

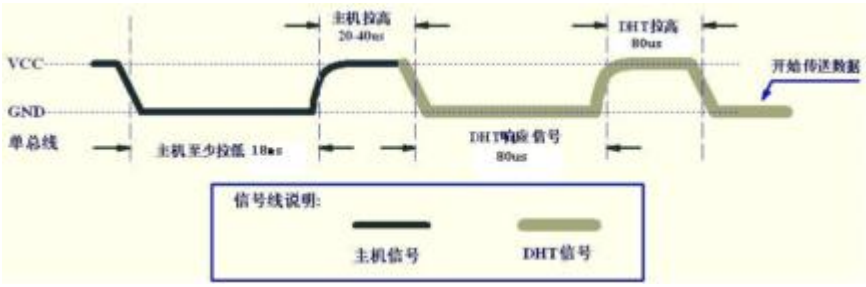
DHT11 只有在接收到开始信号后才会触发温湿度采集，如果没有接收到复位信号，它将保持在低功耗模式，不会主动进行数据采集。当数据采集和传输完成后，DHT11 会自动切换回低功耗模式，等待下一次触发信号。

通讯过程示意图如下：



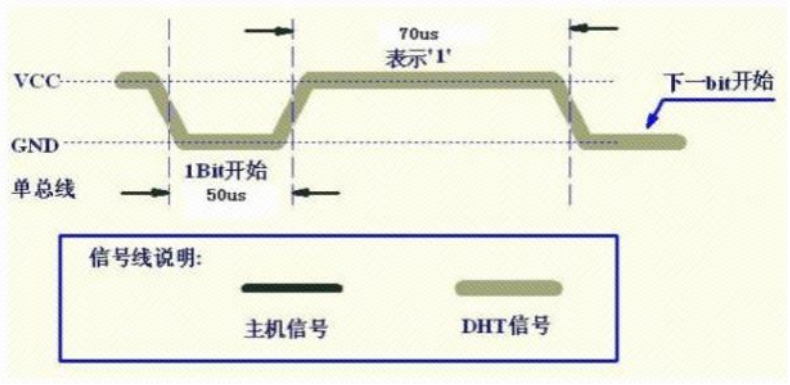
总线空闲状态为高电平,主机把总线拉低等待 DHT11 响应,主机把总线拉低必须大于 18 毫秒，保证 DHT11 能检测到起始信号，然后拉高等待 20-40us, 读取 DHT11 的响应信号，此时主机释放总线。DHT11 接收到主机的开始信号后，发送（70~100）us 低电平响应信号。然后然后 DHT11 拉高总线 （70~100）us，开始传输数据。

主机发送触发信号与 DHT11 响应的过程示意图如下：

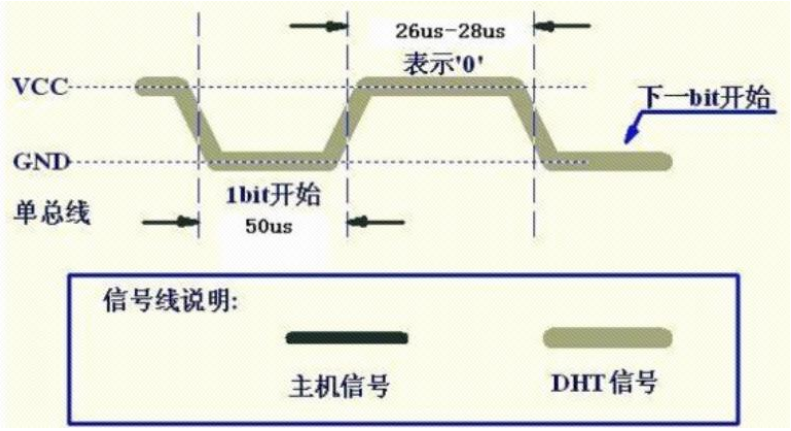


由该图可知触发信号由主机发送，主机需要拉低总线至少 18ms，然后再拉高总线，延时 20~40us。DHT11 检测到触发信号后，开始一次采样，并拉低总线 80us 表示响应信号，告诉主机数据已经准备好了。然后 DHT11 拉高总线 80us，之后开始传输数据。数字 1 与数字 0 的表示方法示意图如下：

数字 1 表示方法示意图



数字 0 表示方法示意图



由示意图可知“0”的高电平持续 26~28us，“1”的高电平持续 70us，每一位数据前都有 50us 的起始时隙。到此 DHT11 的通讯协议以及数据格式就介绍完了。

22.3. 程序设计

DHT11 驱动模块端口描述如下表：

端口	I/O	位宽	描述
clk	input	1	50Mhz 时钟
rst_n	input	1	复位信号
cap_start_flag	input	1	触发一次采样
dht11	inout	1	DHT11 通讯总线
data_vld	output	1	输出温度有效数据标志
temperature_data	output	32	模块测量得到的温湿度数据

驱动部分代码如下，完整源码请查看 demo 源文件

```
//同步时序描述状态转移
always @(posedge clk_1us or negedge rst_n)begin
   if(!rst_n)
```

```
      cur_state <= WAIT_1S;
   else
      cur_state <= next_state;
end
//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_1S;
   case(cur_state)
      WAIT_1S        :begin
         if(us_cnt == TIME_1S)                                  //满足上电延时的时间
            next_state = WAIT_TRI;
         else
            next_state = WAIT_1S;
      end
      WAIT_TRI    :begin
         if(cap_start)
            next_state = START;
         else
            next_state = WAIT_TRI;
      end
      START       :begin
         if(us_cnt == TIME_BE)                                  //满足拉低总线的时间
            next_state = DELAY_30US;
         else
            next_state = START;
      end
      DELAY_30US  :begin
         if(us_cnt == TIME_GO)                                  //满足主机释放总线时
间
            next_state = BUS_REPLY_LOW;
         else
            next_state = DELAY_30US;
      end
      BUS_REPLY_LOW   :begin
         if(us_cnt <= 'd500)begin                               //不到 500us
            if(dht11_rise && us_cnt >= 'd70
            && us_cnt <= 'd100)                                 //上升沿响应，且低电
平时间介于 70~100us
               next_state = BUS_REPLY_HIGH;                     //跳转到 DELAY_75us
            else
               next_state = BUS_REPLY_LOW;                      //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
        BUS_REPLY_HIGH  :begin
            if(dht11_fall && us_cnt >= 'd70)                       //上升沿响应，且低电
平时间大于 70us
                next_state = REV_data;                             //跳转到 REV_data
            else
                next_state = BUS_REPLY_HIGH;                       //条件不满足状态不变
        end
        REV_data    :begin
            if(dht11_rise && 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_TRI;                             //状态跳转到 START，
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 = START;                                //默认状态为 START
    endcase
end

//时序逻辑描述输出
always @(posedge clk_1us or negedge rst_n)begin
    if(!rst_n)begin                                                //复位状态下输出如
下
        dht11_en <= 1'b0;
        dht11_out <= 1'b0;
        us_cnt <= 22'd0;
        bit_cnt <=  6'd0;
        data_temp <= 40'd0;
    end
    else
        case(cur_state)
            WAIT_1S        :begin
                dht11_en <= 1'b0;                                  //释放总线，由外部电
阻拉高
                if(us_cnt == TIME_1S)
                    us_cnt <= 22'd0;
                else
                    us_cnt <= us_cnt + 1'd1;
            end
```

```
        WAIT_TRI    :begin
           dht11_en <= 1'b0;
        end
        START       :begin
           dht11_en <= 1'b1;                                    //占用总线
           dht11_out <= 1'b0;                                   //输出低电平
           if(us_cnt == TIME_BE)
              us_cnt <= 22'd0;
           else
              us_cnt <= us_cnt + 1'd1;
        end
        DELAY_30US    :begin
           dht11_en <= 1'b0;                                    //释放总线，由外部电
阻拉高//这里是否要拉高
           if(us_cnt == TIME_GO)
              us_cnt <= 22'd0;
           else
              us_cnt <= us_cnt + 1'd1;
        end
        BUS_REPLY_LOW       :begin
           dht11_en <= 1'b0;                                    //释放总线，由外部电
阻拉高
           if(us_cnt <= 'd500)begin                              //计时不到 500us
              if(dht11_rise && us_cnt >= 'd70
              && us_cnt <= 'd100)                               //上升沿响应，且低电
平时间介于 70~100us
                 us_cnt <= 22'd0;                               //计时清零
              else
                 us_cnt <= us_cnt + 1'd1;
           end
           else
              us_cnt <= 22'd0;                                  //超过 500us 仍没有上
升沿响应，则计数清零
        end
        BUS_REPLY_HIGH  :begin
           dht11_en <= 1'b0;                                    //释放总线，由外部电
阻拉高
           if(dht11_fall && us_cnt >= 'd70)                     //上升沿响应，且低电
平时间大于 70us
              us_cnt <= 22'd0;                                  //计时清零
           else
              us_cnt <= us_cnt + 1'd1;
        end
        REV_data    :begin
```

```
            dht11_en <= 1'b0;                                    //释放总线，由外部电
阻拉高，进入读取状态
            if(dht11_rise && bit_cnt == 'd40)begin                //数据接收完毕
               bit_cnt <=  6'd0;
               us_cnt <= 22'd0;
            end
            else if(dht11_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
               us_cnt <= 22'd0;
               if(us_cnt <= '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;
               us_cnt <= us_cnt + 1'd1;
            end
         end
         default:;
      endcase
end
//校验读取的数据是否符合校验规则
always @(posedge clk_1us or negedge rst_n)begin
   if(!rst_n)
      temperature_data <= 32'd0;
   else if((data_temp[7:0] == data_temp[39:32] + data_temp[31:24] +
   data_temp[23:16] + data_temp[15:8]))
      temperature_data <= data_temp[39:8];                       //符合规则，则把有
效数据赋值给输出
   else
      temperature_data <= temperature_data;                      //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
always @(posedge clk_1us or negedge rst_n)begin
   if(!rst_n)
      data_vld <= 1'b0;
   else if(dht11_rise && bit_cnt == 'd40)
      data_vld <= 1'b1;
```

```
    else
        data_vld <= 1'b0;
end
```

模块使用一个三段式状态机处理板卡与 DHT11 的通讯逻辑。我们可以通过状态机的跳转过程来理解模块的工作逻辑。这个状态机总共有如下状态：WAIT_1S（上电延迟状态）、WAIT_TRI（等待触发状态）、START（发送触发信号状态）、DELAY_30US（主机释放总线 30us）、BUS_REPLY_LOW（接收 DHT11 低电平响应）、BUS_REPLY_HIGH（接收 DHT11 高电平响应）、REV_data（解析 DHT11 数据）。下面我们挨个解释这些状态:

WAIT_1S：上电后等待 1 秒的时间，确保 DHT11 传感器稳定工作。

WAIT_TRI：等待外部触发信号 cap_start_flag，用于启动温湿度数据采集。cap_start_flag 在顶层模块由按键产生。

START：发送触发信号，拉低总线 18 毫秒，使 DHT11 传感器进入响应模式。

DELAY_30US：主机释放总线 30 微秒，等待 DHT11 传感器的响应信号。

BUS_REPLY_LOW：检测 DHT11 传感器的低电平响应信号，低电平持续时间由 us_cnt 计数，需要满足低电平持续时间在 70~100us。

BUS_REPLY_HIGH：检测 DHT11 传感器的高电平响应信号，低电平持续时间由 us_cnt 计数，需要满足高电平持续时间大于在 70us。

REV_data：当响应信号接收无误之后开始，接收 DHT11 传感器发送的 40 位数据，并根据接收到的数据位更新内部数据存储，直到接收完所有数据。
这里需要注意一下，代码中：assign dht11 = dht11_en ? dht11_out : 1'bz;
通过 dht11_en 信号来控制双向端口 dht11。当 dht11_en 为 1 时双向端口 dht11 为输出，当 dht11_en 为 0 时双向端口 dht11 为输入。这是一种常用的控制双向端口的方式。

串口发送代码如下：

```
//cnt_20ms:如果时钟的上升沿检测到外部按键输入的值为低电平时，计数器开始计数
always@(posedge clk or negedge rst_n)
    if(rst_n == 1'b0)
        cnt_20ms <= 20'b0;
    else    if(key_in == 1'b1)
        cnt_20ms <= 20'b0;
    else    if(cnt_20ms == CNT_MAX && key_in == 1'b0)
        cnt_20ms <= cnt_20ms;
    else
        cnt_20ms <= cnt_20ms + 1'b1;
```

```
always@(posedge clk or negedge rst_n)
   if(rst_n == 1'b0)
      key_flag <= 1'b0;
   else    if(cnt_20ms == CNT_MAX - 1'b1)
      key_flag <= 1'b1;
   else
      key_flag <= 1'b0;
always@(posedge clk or negedge rst_n)
   if(rst_n == 1'b0)
      work_en <= 1'b0;
   else    if(data_vld == 1'b1)
      work_en <= 1'b1;
   else    if(tx_byte_cnt== 7'd23&&uart_tx_done)
      work_en <= 1'b0;
   else
      work_en <= work_en;
//串口发送逻
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      tx_byte_cnt <= 7'b0;
   end else if (work_en) begin
      if (tx_byte_cnt == 7'd23&&uart_tx_done)                    // 总
共发送 24 字节
           tx_byte_cnt <= 7'b0;
      else if(uart_tx_done)
           tx_byte_cnt <= tx_byte_cnt + 1'b1;
   end
end
// 串口发送数据控制
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      uart_tx_req <= 1'b0;
      uart_tx_data <= 8'b0;
      wendu_str <= WENDU_STR;
      shidu_str <= SHIDU_STR;
      ascii_table[0] = 8'd48; //'0'
      ascii_table[1] = 8'd49; //'1'
      ascii_table[2] = 8'd50; //'2'
      ascii_table[3] = 8'd51; //'3'
      ascii_table[4] = 8'd52; //'4'
      ascii_table[5] = 8'd53; //'5'
      ascii_table[6] = 8'd54; //'6'
      ascii_table[7] = 8'd55; //'7'
```

```
      ascii_table[8] = 8'd56; //'8'
      ascii_table[9] = 8'd57; //'9'
   end else if (work_en && !uart_tx_busy) begin
      case (tx_byte_cnt)                                          // 根据
当前字节计数器发送数据
           7'd0: uart_tx_data <= wendu_str[47:40];                 // "
温"
           7'd1: uart_tx_data <= wendu_str[39:32];                 // "
温"
           7'd2: uart_tx_data <= wendu_str[31:24];                 // "
度"
           7'd3: uart_tx_data <= wendu_str[23:16];                 // "
度"
           7'd4: uart_tx_data <= wendu_str[15:8];                  // ":
"
           7'd5: uart_tx_data <= wendu_str[7:0];                   // ":
"
           7'd6: uart_tx_data <=
ascii_table[temperature_data[15:12]];         // 温度高字节的 ASCII
           7'd7: uart_tx_data <=
ascii_table[temperature_data[11:8]];         // 温度低字节的 ASCII
           7'd8: uart_tx_data <= 8'h2e;                           // 小数
点
           7'd9: uart_tx_data <=
ascii_table[temperature_data[7:4]];          // 温度高字节的 ASCII
           7'd10: uart_tx_data <=
ascii_table[temperature_data[3:0]];          // 温度低字节的 ASCII
           7'd11: uart_tx_data <= 8'h20;                           // 空
格
           7'd12: uart_tx_data <= shidu_str[47:40];                // "
湿"
           7'd13: uart_tx_data <= shidu_str[39:32];                // "
湿"
           7'd14: uart_tx_data <= shidu_str[31:24];                // "
度"
           7'd15: uart_tx_data <= shidu_str[23:16];                // "
度"
           7'd16: uart_tx_data <= shidu_str[15:8];                 // ":
"
           7'd17: uart_tx_data <= shidu_str[7:0];                  // ":
"
           7'd18: uart_tx_data <=
ascii_table[temperature_data[31:28]];          // 湿度高字节的 ASCII
```

```
            7'd19: uart_tx_data <=
ascii_table[temperature_data[27:24]];          // 湿度高字节的 ASCII
            7'd20: uart_tx_data <= 8'h2e;                          // 小数
点
            7'd21: uart_tx_data <=
ascii_table[temperature_data[23:20]];          // 湿度低字节的 ASCII
            7'd22: uart_tx_data <=
ascii_table[temperature_data[19:16]];          // 湿度低字节的 ASCII
            7'd23: uart_tx_data <= 8'h0A;                          // 换行
符
            default: uart_tx_data <= 8'b0;
      endcase
      uart_tx_req <= 1'b1;                                         // 拉高
请求信号
   end else begin
      uart_tx_req <= 1'b0;                                         // 请求
信号拉高仅一个时钟周期
   end
end
```

我们在顶层模块接收用户输入的按键信号 key_in 使用一个 20ms 计数器对其进行消抖产生温湿度驱动模块的采样触发信号 cap_start_flag。将 DHT11 得到的数据通过串口发送到上位机进行查看，汉字使用 GB2312 编码，模块中初始化了一个 ascii_table 用来储存阿拉伯数字 0~9 的 ASCII 值，将 DHT11 采样得到的温湿度数据在 ascii_table 索引到对应的 ASCII 值进行发送。其中串口发送部分的知识在之前的章节已有介绍，读者若对这部分不熟悉，可到串口回环章节进行查看。

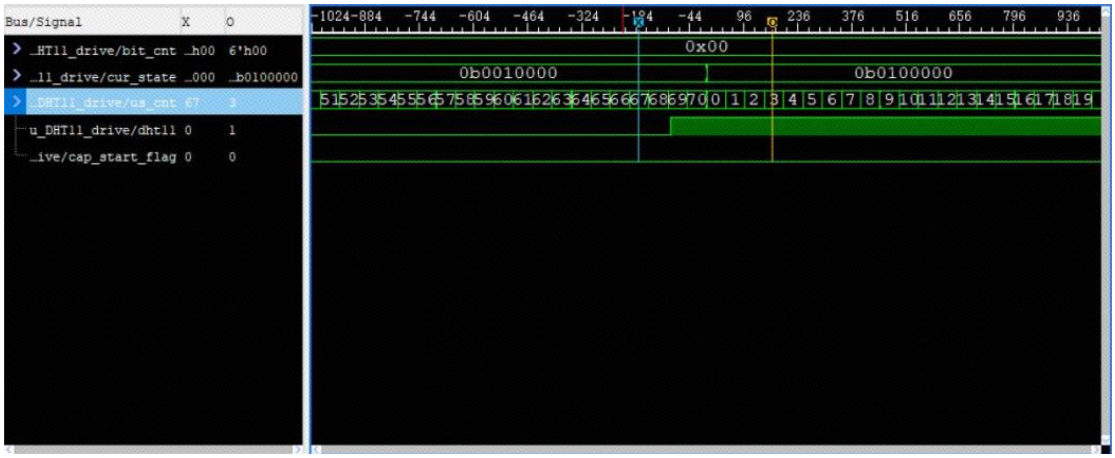
22.4. 代码仿真

由于本实验需要与 DHT11 通过总线交互信息，在测试文件中编写 DHT11 的应答逻辑比较麻烦，我们可以将 DHT11 驱动模块中的关键信号打上 debug 标记，分析 DHT11 驱动模块是否正确工作，同时检查 DHT11 是否正确应答我们通过总线发送的信息。

DHT11 的控制流程大致为主机触发、DHT11 应答、DHT11 发送温度数据主机接收温度数据。我们将 DHT11 驱动代码中的状态机 cur_stste、顶层模块发送的采样触发信号 cap_start_flag、DHT11 总线信号 dht11、微秒计数器 us_cnt、接收数据比特计数器 bit_cnt 添加上 debug 标记。

将触发模式设置为 cap_start_flag 高电平触发，按下板卡按键 k0，捕捉到的波形如下图所示，总线信号 dht11 在空闲状态为高电平，在顶层发送触发信号后状态机由等待触发状态（WAIT_TIR）跳转至 START 状态，此时 FPGA 占用总线并拉低总线 18ms，向 DHT11 发送开始信号。

将触发模式设置为（cur_state = 7’b0010000&&us_cnt = 70）（状态机在接收来自 DHT11 的低电平响应状态），按下按键，观察捕获到的波形图，如果状态机正确跳转，那么说明我们向 DHT11 发送的触发信号被回应.



由上图可知我们状态机在状态机在接收来自 DHT11 的低电平响应状态时，dht11 为低电平，并且 us_cnt 成功计数到 70 以上，满足响应时间在（70~100）us 的要求，说明来自 DHT11 的低电平响应是正确的。

同样的我们将触发条件设置为（cur_state = 7’b0100000&&us_cnt = 70）（状态机在接收来自 DHT11 的高电平响应状态）按下按键，观察捕获到的波形图如下：

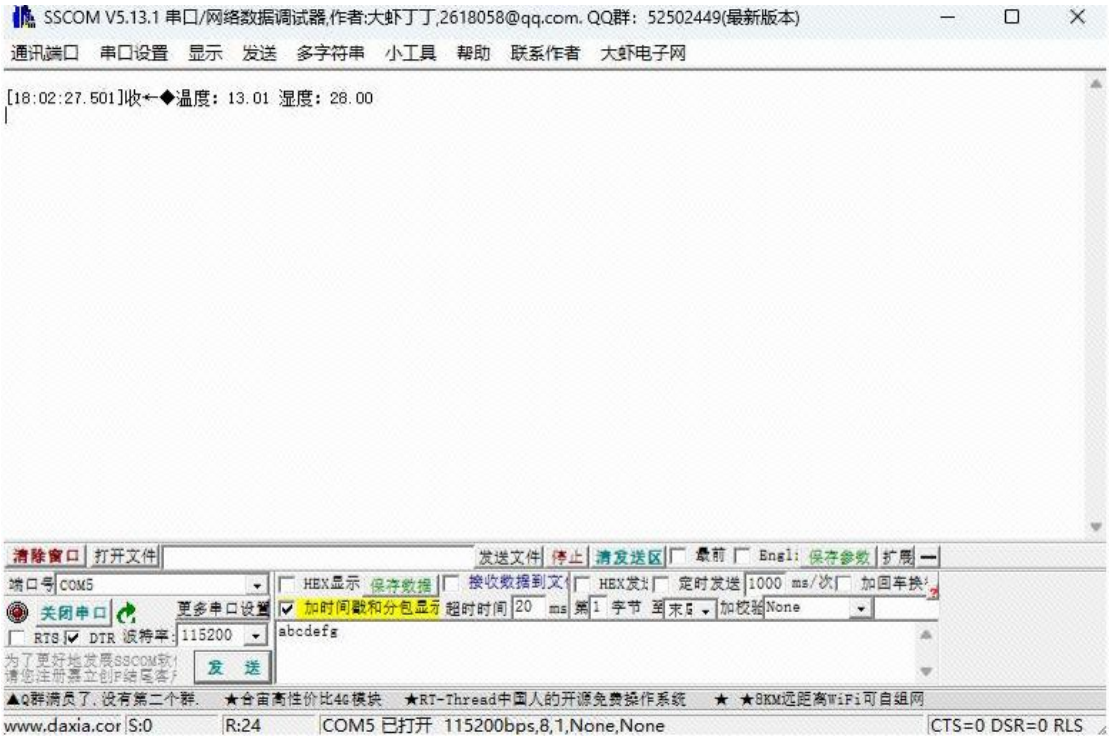
结合波形图可以发现状态机在接收高电平响应的状态时，us_cnt 也成功计数到 70us 以上，说明来自 DHT11 的高电平响应也是正确的。至此 DHT11 应答成功，开始发送温度数据。

我们知道，DHT11 发送的温度数据是 40bit 的 8 位湿度整数数据 +8 位湿度小数数据 +8 位温度整数数据 +8 位温度小数数据 +8 位校验和。我们将触发状态设置为（cur_state = 7'b1000000&&bit_cnt = 40）（状态机在接收来自 DHT11 的数据并且接收完成），再次按下按键，观察捕获到的波形图如下

由捕获到的波形可知，状态机在接收数据状态时，成功接收数据，并且接收 bit 计数器成功计数到 40，说明接收到了来自 DHT11 的完整 40bit 数据，至此，模块功能验证成功。

22.5. 实验现象

将程序下载进入板卡后，将 DHT11 按照约束文件正确连接，打开上位机串口助手，按下板卡按键 S0，可得到如下结果。


SSCOM V5.13.1 串口/网络数据调试器,作者:大虾丁丁,2618058@qq.com. QQ群: 52502449(最新版本)
通讯端口 串口设置 显示 发送 多字符串 小工具 帮助 联系作者 大虾电子网
[18:02:27.501]收←◆温度: 13.01 湿度: 28.00
清除窗口 打开文件 发送文件 停止 清发送区 最前 Engli 保存参数 扩展
端口号 COM5
HEX显示 保存数据 接收数据到文
HEX发送 定时发送 1000 ms/次 加回车换行
关闭串口 更多串口设置
加时间戳和分包显示 超时时间 20 ms 第 1 字节 至 末尾 加校验 None
RTS DTR 波特率: 115200
abcdefg
为了更好地发展SSCOM软
请您注册嘉立创P结尾客户
发 送
▲Q群满员了,没有第二个群. ★合宙高性价比4G模块 ★RT-Thread中国人的开源免费操作系统 ★ ★8KM远距离WiFi可自组网
www.daxia.cor S:0 R:24 COM5 已打开 115200bps,8,1,None,None CTS=0 DSR=0 RLS