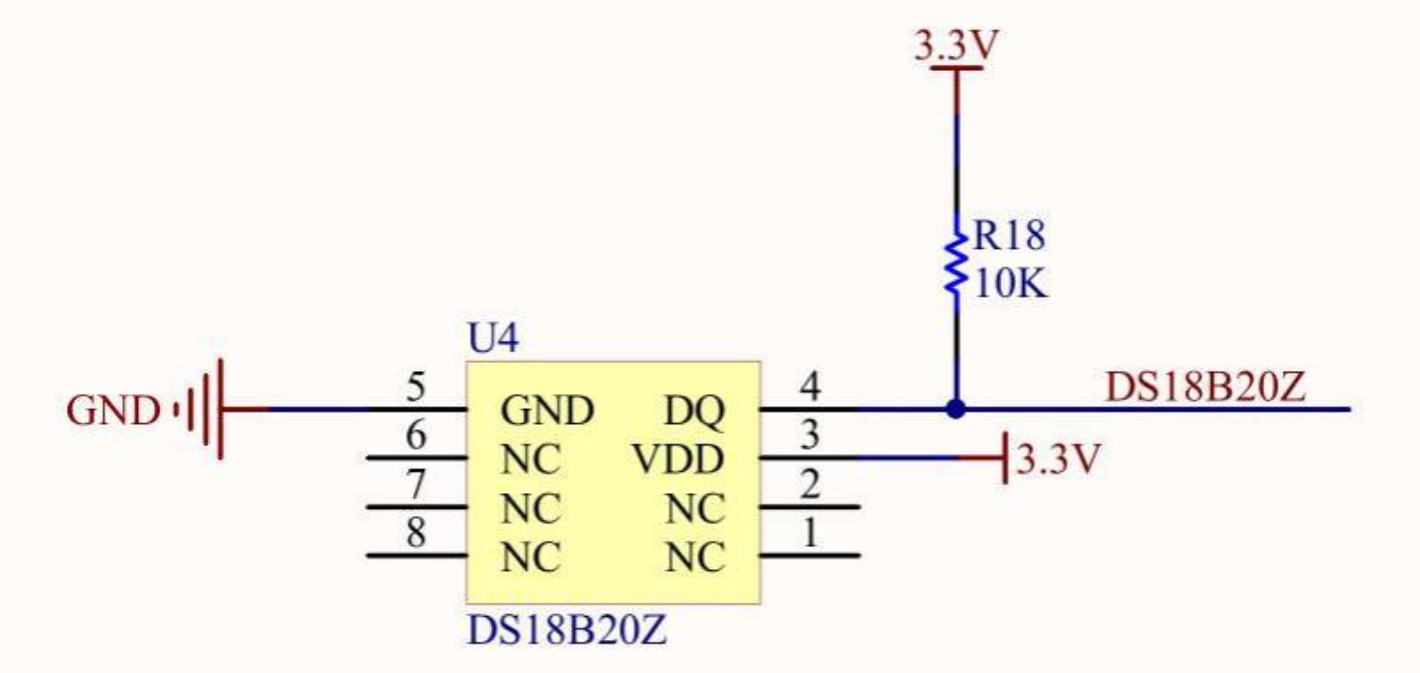
STEP FPGA drives temperature sensor DS18B20Z

In this section, we will use FPGA to drive the DS18B20Z single-bus temperature sensor on the backplane to collect temperature data.

====Hardware description====

DS18B20 is a temperature sensor chip commonly used in our daily design. It only needs a bus to realize communication. It is very convenient. Our STEP-BaseBoard has integrated temperature sensor DS18B20Z. Let's learn about it together. Hardware link and drive method.
DS18B20Z has only one bus, the hardware circuit is very simple, but you must remember that the bus needs to be pulled up, as shown in the figure below, a 10K pull-up resistor (the value of the pull-up resistor can be adjusted within a certain range) is connected to the bus, and we use FPGA To drive, remember to configure the FPGA corresponding pins for the same pull-up configuration. The important thing is said three times, bus pull-up, bus pull-up, bus pull-up. After talking about the hardware connection, let's briefly introduce how to drive (you need to refer to the data manual for more detailed information). Different functional requirements correspond to different register configurations. The operation cases implemented in this design are as follows. The following shows you the timing requirements of each link in the above case:



MASTER MODE	DATA (LSB FIRST)	COMMENTS
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	44h	Master issues Convert T command.
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	BEh	Master issues Read Scratchpad command.
Rx	2 data Byte	Master reads temperature data of scratchpad.

Figure 13. Initialization Timing

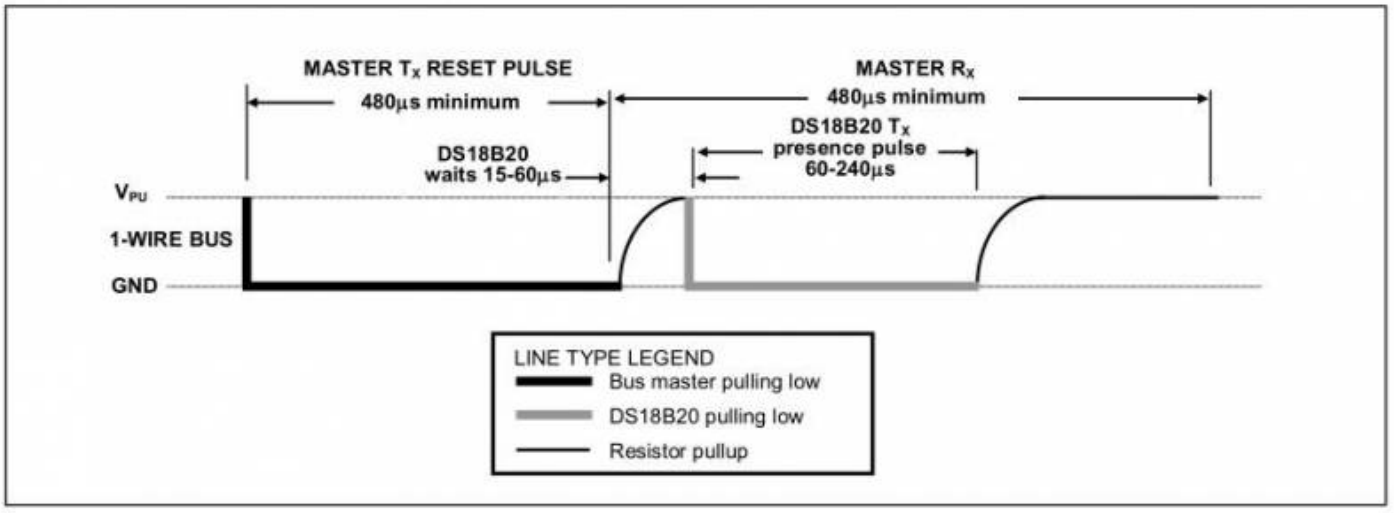
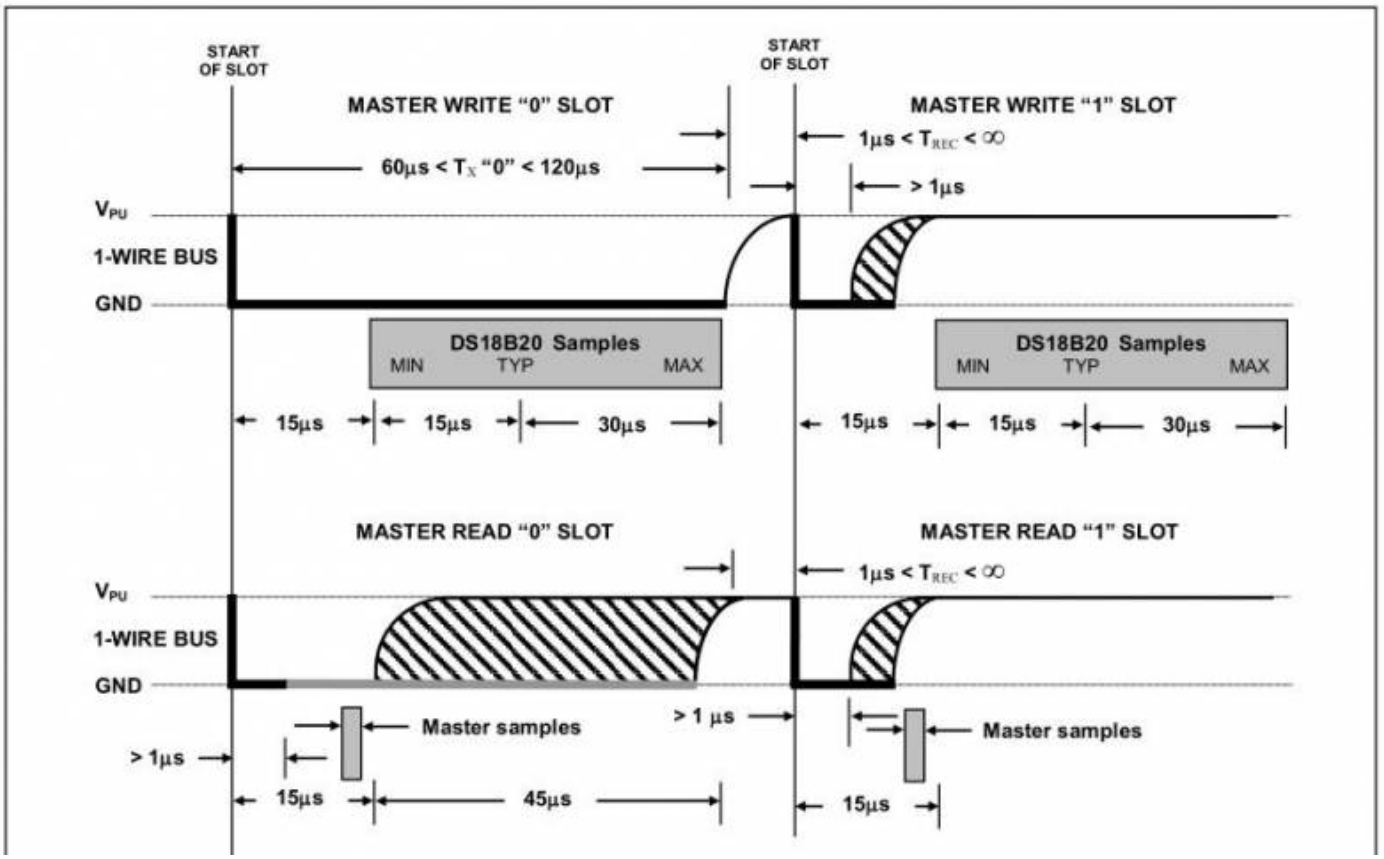


Figure 14. Read/Write Time Slot Timing Diagram



CONFIGURATION REGISTER

Byte 4 of the scratchpad memory contains the configuration register, which is organized as illustrated in Figure 8. The user can set the conversion resolution of the DS18B20 using the R0 and R1 bits in this register as shown in Table 2. The power-up default of these bits is R0 = 1 and R1 = 1 (12-bit resolution). Note that there is a direct tradeoff between resolution and conversion time. Bit 7 and bits 0 to 4 in the configuration register are reserved for internal use by the device and cannot be overwritten.

Figure 8. Configuration Register

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

Table 2. Thermometer Resolution Configuration

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})

上电默认采用12-bit解析，温度转换时间为 750ms

Figure 2. Temperature Register Format

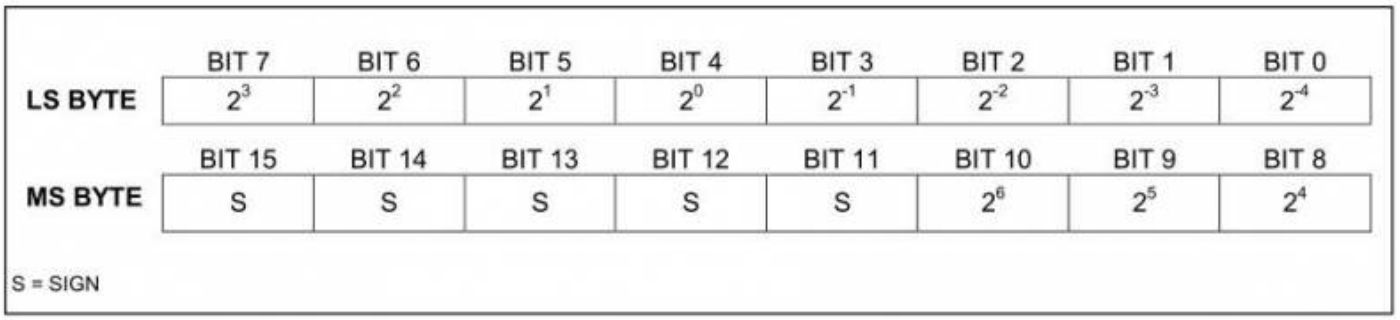


Table 1. Temperature/Data Relationship

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

====Verilog code====

```
// ---------------------------------------------- --------------------
// >>>>>>>>>>>>>>>>>>>>>>> COPYRIGHT NOTICE < <<<<<<<<<<<<<<<<<<<<<<
// ------------------------ ---------------------------------------------
// Module:DS18B20Z
/ /
// Author: Step
//
// Description: Drive DS18B20Z to get temperature code
//
// Web: www.stepfpga.com
//
// ----------------- ---------------------------------------------------- -
// Code Revision History:
// ---------------------------------------- ----------------------------
// Version: |Mod. Date:|Changes Made:
// V1.0 |2015/11/11 |Initial ver
// ---------------------------------------------- --------------------
module DS18B20Z
 (
        input                           clk_in ,                        //system clock
        input                           rst_n_in ,              //system reset, low effective
        inout                           one_wire ,              //DS18B20Z sensor single bus,
 Bidirectional pin
        output  reg     [ 15 : 0 ]      data_out                 //DS18B20Z effective temperat
ure data output
) ;

        /*
        This design obtains temperature data by driving the DS18B20Z chip,
        Need to understand how the inout type interface realizes two-way communication,
        Various delays and register instruction operations are involved in the middle. The com
ment part is for a brief description. For more details, please refer to the data sheet
        */

        localparam      IDLE     =      3'd0 ;
        localparam      MAIN     =      3'd1 ;
        localparam      INIT     =      3'd2 ;
        localparam      WRITE    =      3'd3 ;
        localparam      READ     =      3'd4 ;
        localparam      DELAY    =      3'd5 ;

        //Counter divides to generate 1MHz clock signal
        reg                                     clk_1mhz ;
        reg             [ 2 : 0 ]               cnt_1mhz ;
        always @ ( posedge clk_in or  negedge rst_n_in )  begin
                if ( ! Rst_n_in )  begin
                        cnt_1mhz <=  3'd0 ;
                        clk_1mhz <=  1'b0 ;
                end  else  if ( cnt_1mhz >=  3'd5 )  begin
                        cnt_1mhz <=  3'd0 ;
                        clk_1mhz <=  ~ clk_1mhz ;       //Generate 1MHz frequency divider
                end  else  begin
```

```
                        cnt_1mhz <= cnt_1mhz +  1'b1 ;
                end
        end

        reg             [ 2 : 0 ]               cnt ;
        reg                                     one_wire_buffer ;
        reg             [ 3 : 0 ]               cnt_main ;
        reg             [ 7 : 0 ]               data_wr ;
        reg             [ 7 : 0 ]               data_wr_buffer ;
        reg             [ 2 : 0 ]               cnt_init ;
        reg             [ 19 : 0 ]              cnt_delay ;
        reg             [ 19: 0 ]               num_delay ;
        reg             [ 3 : 0 ]               cnt_write ;
        reg             [ 2 : 0 ]               cnt_read ;
        reg             [ 15 : 0 ]              temperature ;
        reg             [ 7 : 0 ]               temperature_buffer ;
        reg             [ 2 : 0 ]               state = IDLE ;
        reg             [ 2 : 0 ]               state_back= IDLE ;
        //Use a 1MHz clock signal to trigger the function of the following state machine
        always @ ( posedge clk_1mhz or  negedge rst_n_in )  begin
                if ( ! Rst_n_in )  begin
                        state <= IDLE ;
                        state_back <= IDLE ;
                        cnt <=  1'b0 ;
                        cnt_main <=  1'b0 ;
                        cnt_init <=  1'b0 ;
                        cnt_write <=  1'b0 ;
                        cnt_read <=  1'b0 ;
                        cnt_delay <=  1'b0 ;
                        one_wire_buffer <=  1'bz ;
                        temperature <=  1 6'h0 ;
                end  else  begin
                        case ( state )
                                IDLE : begin            //IDLE state, the soft reset function
 of the program design, all state abnormalities will jump At this state
                                                        state <= MAIN ;         //soft reset is comple
ted, the MAIN state that jumped to work
                                                        again state_back <= MAIN ;
                                                        cnt <=  1'b0 ;
                                                        cnt_main <=  1'b0 ;
                                                        cnt_init <=  1'b0 ;
                                                        cnt_write <=  1'b0 ;
                                                        cnt_read <=  1'b0 ;
                                                        cnt_delay <=  1'b0 ;
                                                        one_wire_buffer <=  1'bz ;
                                        end
                                MAIN : begin            //MAIN state control state machine jum
ps between different states to achieve complete temperature Data collection
                                                        if ( cnt_main >=  4'd11 ) cnt_main <=  1'b0 ;
                                                        else cnt_main <= cnt_main +  1'b1 ;
                                                        case ( cnt_main )
                                                                4'd0 :  begin state<= INIT ;  end
//Jump to INIT state to reset and verify the chip
```

```
                                                    4'd1 :  begin data_wr <=  8'hcc ; stat
e <= WRITE ;  end       //The master device issues a jump ROM instruction
                                                    4'd2 :  begin data_wr <=  8'h44 ; stat
e <= WRITE ;  end       //The main device issues a temperature conversion command
                                                    4'd3 :  begin num_delay <=  2 0'd75000
0 ; state <= DELAY ; state_back <= MAIN ;  end  //Delay 750ms for the conversion to complete

                                                    4'd4 :  begin state <= INIT ;  end
//Jump to INIT state to reset and verify the chip
                                                    4'd5 :  begin data_wr <=  8'hcc ; stat
e <= WRITE ;  end       //The master sends a jump ROM Command
                                                    4'd6 :  begin data_wr <=  8'hbe ; stat
e <= WRITE ;  end       //The master device issues a temperature reading command

                                                    4'd7 :  begin state <= READ ;  end
//Jump to READ state for single-bus data reading
                                                    4'd8 :  begin temperature [ 7 : 0 ]  <
= temperature_buffer ;  end     // The lower 8 bits are read first data

                                                    4'd9 :  begin state <= READ ;  end
//Jump to READ state for single-bus data reading
                                                    4'd10 :  begin temperature [ 15 : 8 ]
<= temperature_buffer ;  end    // The next read is high 8 data

                                                    4'd11 :  begin state <= IDLE ; data_ou
t <= temperature ;  end         //Output the complete temperature data and repeat all the abov
e operations
                                                    default : state <= IDLE ;
                                                endcase
                                        end
                                    INIT : begin            //The reset state of DS18B20Z is compl
eted And verification function
                                                if ( cnt_init >=  3'd6 ) cnt_init <=  1'b0 ;
                                                else cnt_init <= cnt_init +  1'b1 ;
                                                case ( cnt_init )
                                                    3'd0:  begin one_wire_buffer <=  1'b0
;  end  //single bus reset pulse pull down
                                                    3'd1 :  begin num_delay <=  2 0'd500 ;
state <= DELAY ; state_back <= INIT ;  end      //reset pulse stay low for 500us Time
                                                    3'd2 :  begin one_wire_buffer <=  1'bz
;  end  //Single bus reset pulse release, automatic pull up
                                                    3'd3 :  begin num_delay <=  2 0'd100 ;
state <= DELAY ;state_back <= INIT ;  end       //Reset pulse keeps releasing for 100us time
                                                    3'd4 :  begin  if ( one_wire ) state <
= IDLE ;  else state <= INIT ;  end     // Determine whether to continue according to the dete
ction result of the existence of single bus
                                                    3'd5 :  begin num_delay <=  2 0'd400 ;
state <= DELAY ; state_back <= INIT ;  end      //If the detection is normal, continue to rele
ase 400us time
                                                    3'd6 :  begin state <= MAIN;  end
//INIT state operation is complete, return to MAIN state
                                                    default : state <= IDLE ;
                                                endcase
```

```
                                        end
                                WRITE : begin           //Write operation according to DS18B20
Z chip single bus timing
                                                if ( cnt <=  3'd6 )  begin      //Total need t
o send 8bit data, here control the number of cycles
                                                        if ( cnt_write >=  4'd6 )  begin cnt_w
rite <=  1'b1 ; cnt <= cnt +  1'b1 ;  end
                                                        else  begin cnt_write <= cnt_write +
1'b1 ; cnt <= cnt ;  end
                                                end  else  begin
                                                        if ( cnt_write >=  4'd8 )  begin cnt_w
rite <=  1'b0 ; cnt <=  1'b0 ;  end     // Both variables are restored to their initial values
                                                        else  begin cnt_write <= cnt_write +
1'b1 ; cnt <= cnt ;  end
                                                end
                                                //For cnt_write in WRITE state, the execution
 process is: 0;[1~6]*8;7;8;
                                                case ( cnt_write )
                                                        //lock data_wr
                                                        4'd0:  begin data_wr_buffer <= data_wr
;  end  //buffer the data that needs to be written
                                                        //send 1bit data in 60~120us, refer to
data manual
                                                        4'd1 :  begin one_wire_buffer <=  1'b0
;  end  //bus
                                                        Pull down 4'd2 :  begin num_delay <=
2 0'd2 ; state <= DELAY ; state_back <= WRITE ;  end    //Delay for 2us time, guarantee within
                                                        15us 4'd3 :  begin one_wire_buffer <=
 data_wr_buffer [ cnt ] ; end   //Send the lowest bit of data first
                                                        4'd4 :  begin num_delay <=  2 0'd80 ;
 state <= DELAY ; state_back <= WRITE ;  end    // Delay 80us time
                                                        4'd5 :  begin one_wire_buffer <=  1'bz
;  end  //Bus release
                                                        4'd6 :  begin num_delay <=  2 0'd2 ; s
tate <= DELAY ; state_back <= WRITE ;  end      //delay 2us time
                                                        //back to main
                                                        4'd7 :  begin num_delay <=  2 0'd80 ;
 state <= DELAY ; state_back <= WRITE ;  end    //delay 80us time
                                                        4'd8 :  begin state <= MAIN ;  end
//return to MAIN state
                                                        default : state < = IDLE ;
                                                endcase
                                        end
                                READ : begin            //Read operation according to DS18B20Z
chip single bus timing
                                                if ( cnt <=  3'd6 )  begin      // A total of
 8bit data needs to be received, here is the number of cycles to control
                                                        if( cnt_read >=  3'd5 )  begin cnt_rea
d <=  1'b0 ; cnt <= cnt +  1'b1 ;  end
                                                        else  begin cnt_read <= cnt_read +
1'b1 ; cnt <= cnt ;  end
                                                end  else  begin
                                                        if ( cnt_read > =  3'd6 )  begin cnt_r
```

```
ead <=  1'b0 ; cnt <=  1'b0 ;  end      //Both variables are restored to their initial values
                                                    else begin cnt_read <= cnt_read +  1'b
1 ; cnt <= cnt ;  end
                                                end
                                                case ( cnt_read )
                                                    //The time to read 1bit data is betwee
n 60~120us, and the data will be read within 15us after the bus is pulled down, refer to the d
ata manual
                                                    3'd0 :  begin one_wire_buffer <=  1'b0
;  end  //Bus pull down
                                                    3'd1 :  begin num_delay <=  2 0'd2 ; s
tate <= DELAY ; state_back <= READ ;  end       //delay 2us time
                                                    3' d2 : begin one_wire_buffer <=  1'bz
;  end  //Bus release
                                                    3'd3 :  begin num_delay <=  2 0'd5 ; s
tate <= DELAY ; state_back <= READ ;  end       //delay 5us time
                                                    3'd4 :  begin temperature_buffer [ cnt
]  <= one_wire ;  end   //Read the bus data returned by DS18B20Z, receive the lowest bit
                                                    3'd5 first :  begin num_delay <=  2
0'd60 ; state <= DELAY; state_back <= READ ;  end       //Delay 60us time
                                                    //back to main
                                                    3'd6 :  begin state <= MAIN ;  end
//Return to MAIN state
                                                    default : state <= IDLE ;
                                                endcase
                                        end
                                DELAY : begin           //Delay Control
                                                if ( cnt_delay >= num_delay )  begin    // Del
ay control, the delay time is specified by num_delay
                                                    cnt_delay <=  1'b0 ;
                                                    state <= state_back ;   //Many states
 need delay, which state to return after delay is specified by state_back
                                                end  else cnt_delay <= cnt_delay +  1'b1 ;
                                        end
                        endcase
                end
        end

        assign  one_wire = one_wire_buffer ;

endmodule
```

====Summary====

This section mainly explains the driving method and software implementation of DS18B20Z for everyone. You need to create a project yourself while mastering it, and generate FPGA configuration file loading test through the entire design process.
If you are not familiar with the use of Diamond software, please refer to here: Use of Diamond .

====Related Information====

Use STEP-MXO2 second generation based DS18B20Z thermometer design process: subsequent download link will be updated
using the STEP-MAX10 based DS18B20Z thermometer design process: subsequent download link will be updated