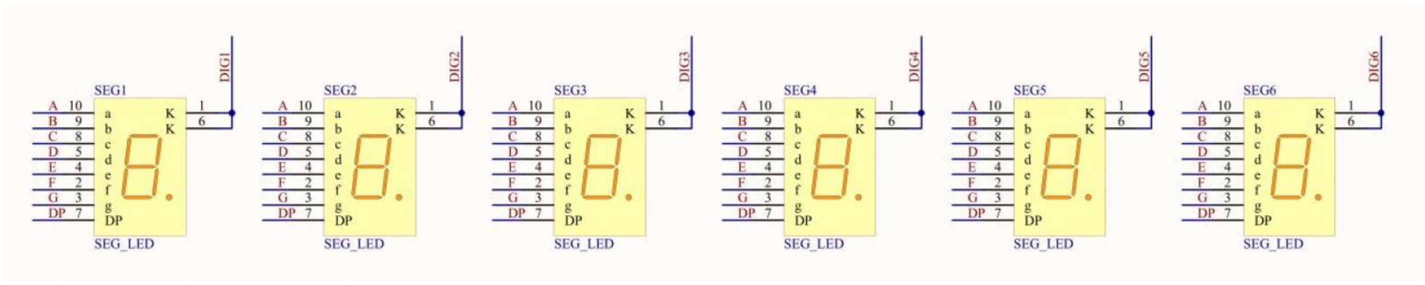
STEP FPGA drives the nixie tube module based on 74HC595

In this section, we will use the 6-digit digital tube on the FPGA driver board to realize dynamic display.

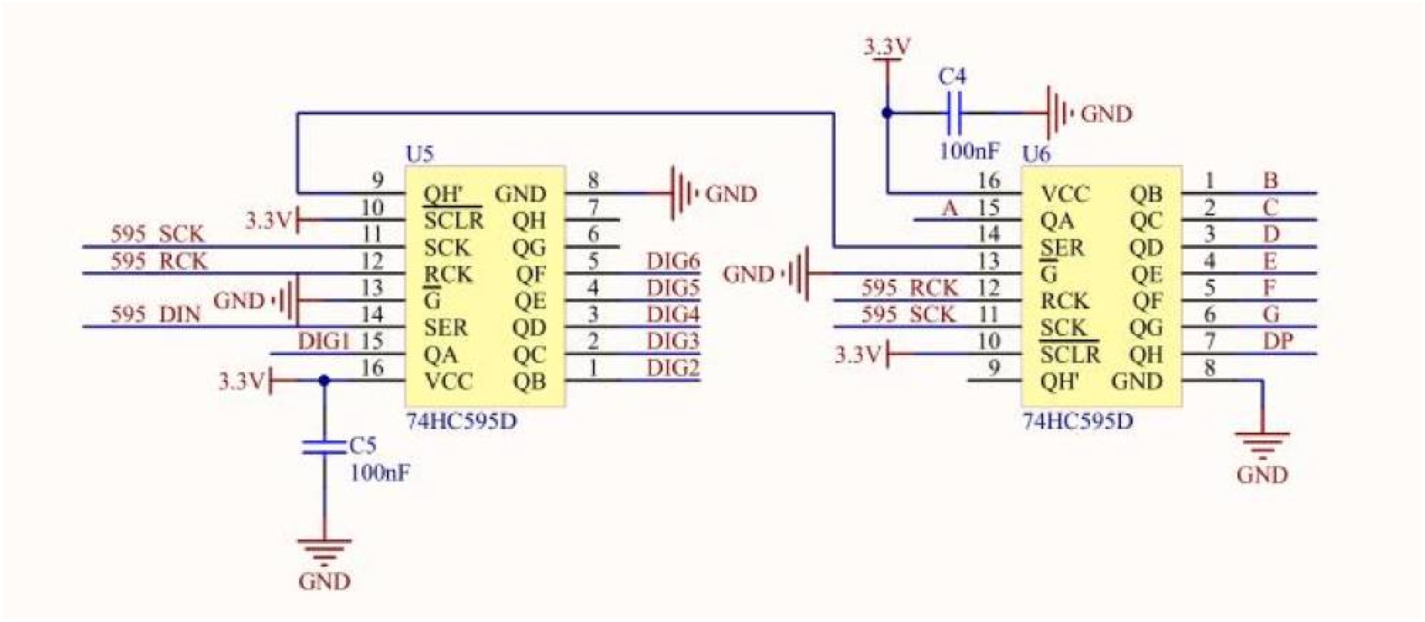
====Hardware description====

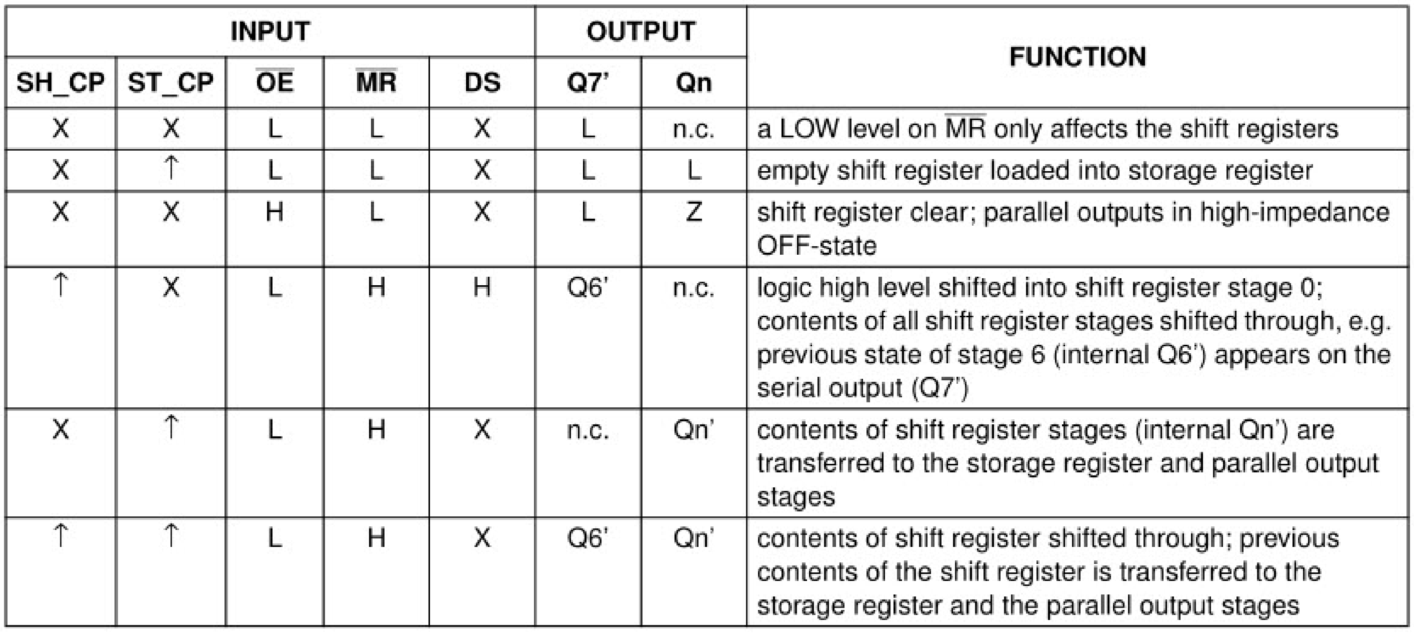
In the introductory tutorial before in front of the digital display separate chapter has introduced the LED independent display relevant content for everyone, on a separate display will not repeat here. There are 6-digit digital tubes on our bottom board. According to the different driving methods, there are the following comparisons:
Independent display: to control each digital tube requires at least 8 I/O ports to control, 6-digit digital tubes need 6*8 = 48 signals The lines can be displayed separately. The independent display is simple to implement, but requires a lot of signal lines.
Scanning display: Connect the same segment selection signal of each digital tube together, so we only need 8 segment selection signals and 6 position selection signals, a total of 14 signals. Scanning display can effectively save I/O port resources, which is slightly more complicated to implement. The 6-digit common cathode digital tube used on the bottom board of our little foot, the principle of analysis and scanning display is as follows: At a certain moment, FPGA controls 8 common segment selection interfaces to output the digital tube font data 8'h06 (DP= 0, G=0, F=0, E=0, D=0, C=1, B=1, A=0), the 6-digit digital tube is controlled at the same time only the first bit is enabled (DIG1=0, DIG2 =1, DIG3=1, DIG4=1, DIG5=1, DIG6=1) In this way, we will see that the first digital tube displays the number 1, and the other 5 digital tubes do not display. If you don't understand, you can refer to the experiment in the introductory tutorial IV: LED independent display section according to the scanning manner, divided into a total of 6 time segments corresponding to selected output ports are six digital font data to be displayed, each time the bit selected from the port remains only an LED is enabled. State, 6 moments in turn, when the scanning frequency is high enough (for example, when the scanning frequency is equal to 100Hz), the display of the digital tube seen by the human eye is continuous, and what we see is 6 different numbers.



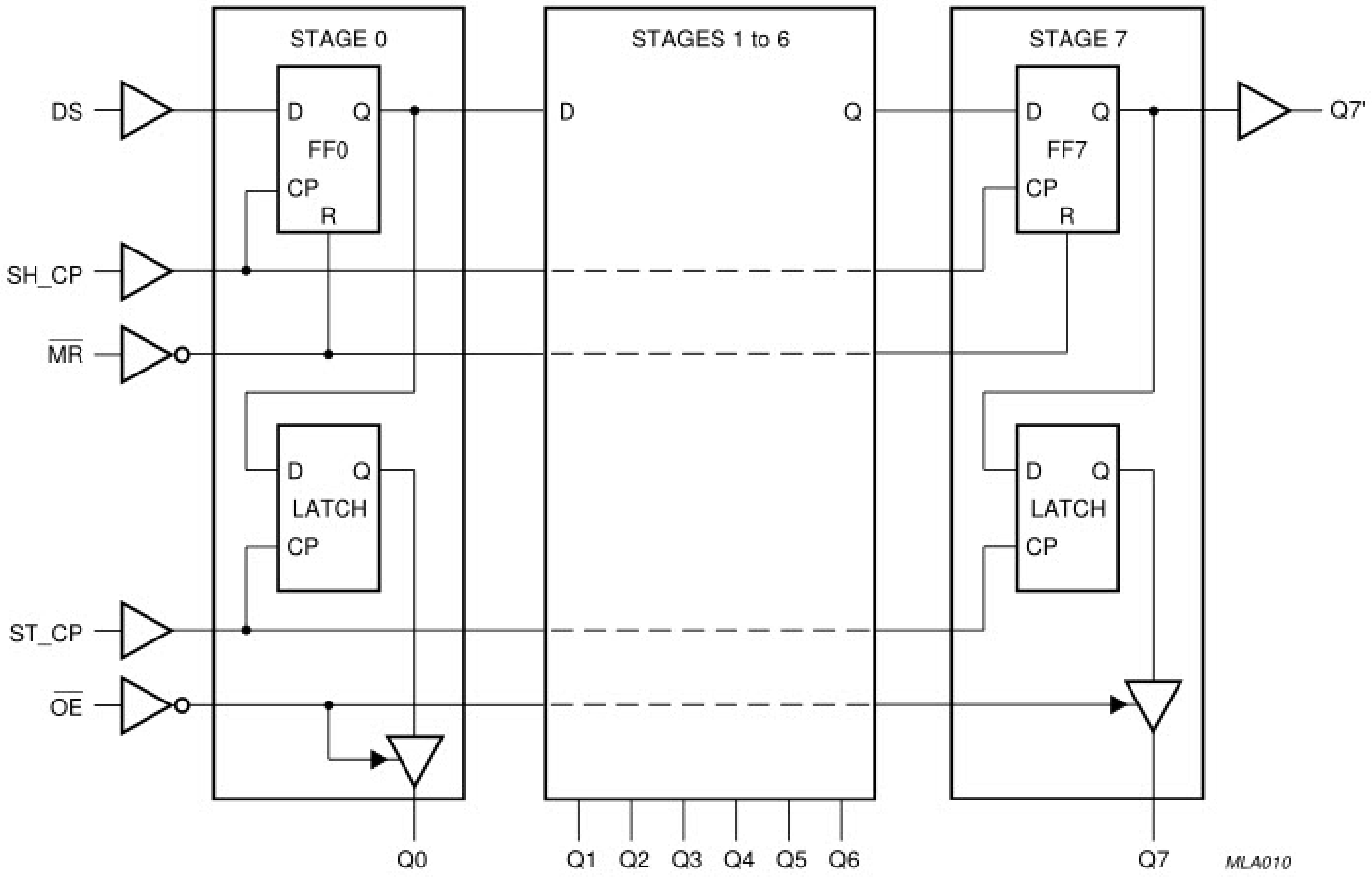
The above has introduced two methods of independent display and scanning display of the digital tube. The scanning display method uses 14 I/O ports to control. Compared with a simple processor, 14 I/O ports are also very much. Here we use a common driver chip 74HC595, let's take a look:
74HC595 is a more commonly used serial to parallel chip, which integrates an 8-bit shift register, a memory and 8 three-state buffer outputs. . In the simplest case, we only need to control 3 pin inputs to get 8 pin parallel output signals, and it can be used in cascade. We use 3 I/O ports to control two cascaded 74HC595 chips to generate 16 channels Parallel output, connected to the 6-digit digital tube scanning display, can easily complete the digital tube driving task. Different IC manufacturers can produce 74HC595 chips with the same functions. However, the chip manuals of different

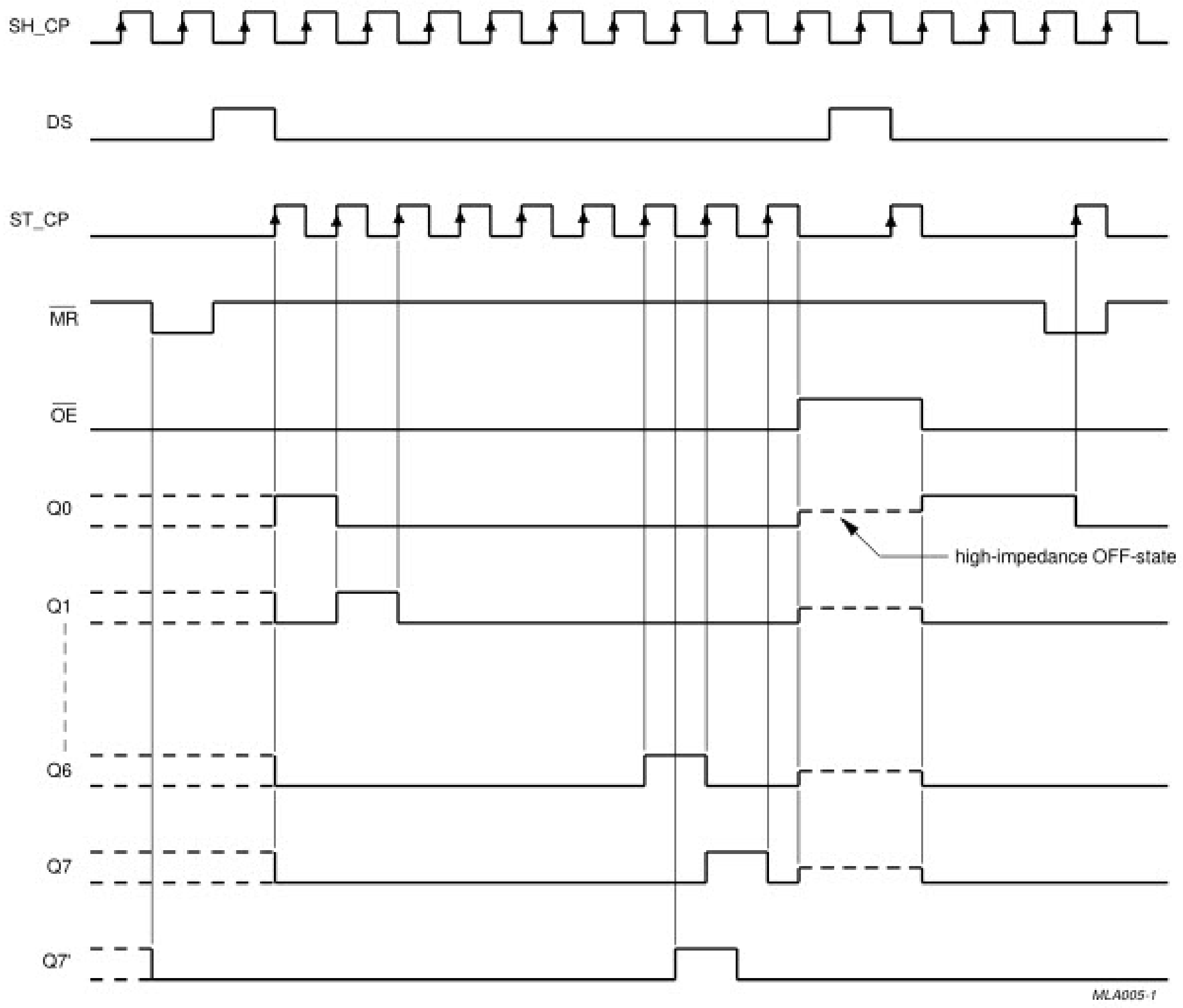
manufacturers have different pin names. The pin order is the same. You can identify the hardware of the 74HC595 chip in this design. For circuit connection, refer to the 74HC595 data manual for its specific usage. In the figure below, we understand that the OE# (G#) and MR# (SCLR#) signals are output enable (low-level output) and reset pins (low-level complex Bit), OE# (G#) we connect GND to enable the chip output, MR# (SCLR#) we connect VCC so that the shift register of the chip will never be reset, so FPGA only needs to control SH *CP (SCK), ST* CP (RCK) and DS (SER).

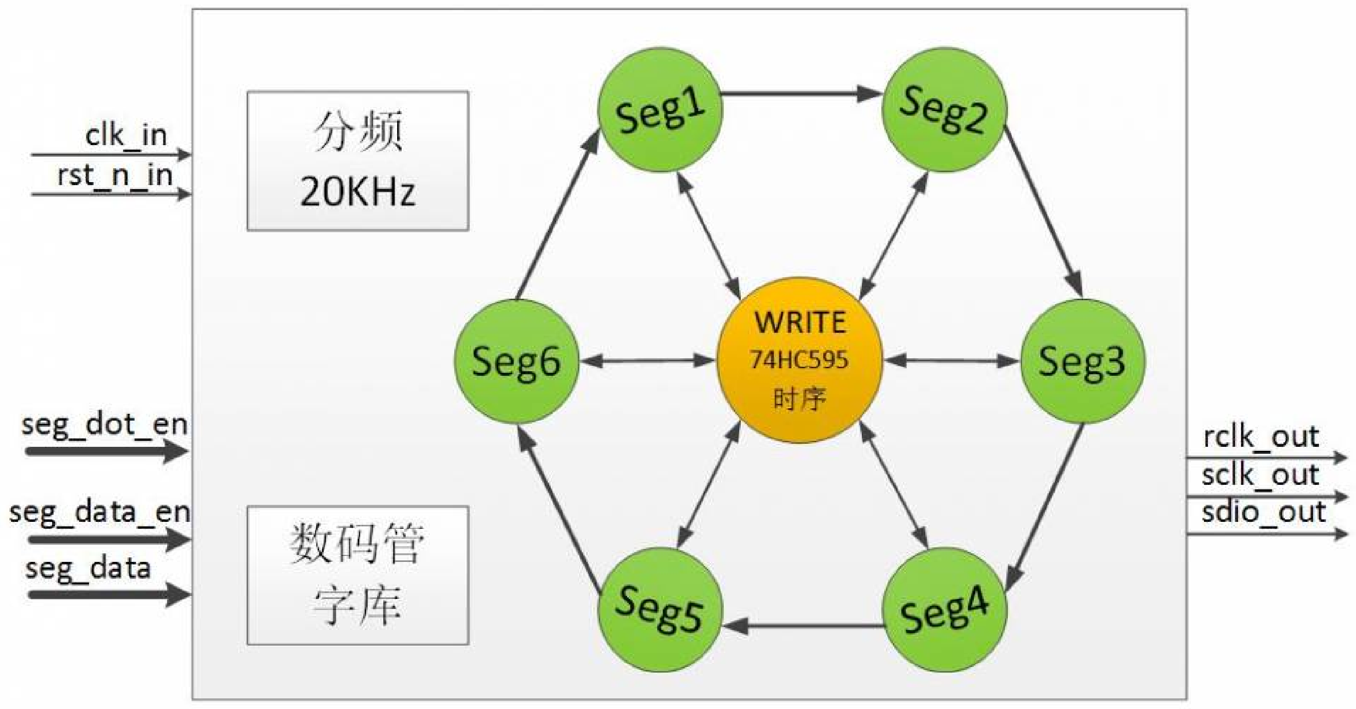




INPUT					OUTPUT		FUNCTION
SH_CP	ST_CP	$\overline{OE}$	$\overline{MR}$	DS	Q7'	Qn	
X	X	L	L	X	L	n.c.	a LOW level on $\overline{MR}$ only affects the shift registers
X	↑	L	L	X	L	L	empty shift register loaded into storage register
X	X	H	L	X	L	Z	shift register clear; parallel outputs in high-impedance OFF-state
↑	X	L	H	H	Q6'	n.c.	logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7')
X	↑	L	H	X	n.c.	Qn'	contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages
↑	↑	L	H	X	Q6'	Qn'	contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages


STAGE 0
STAGES 1 to 6
STAGE 7
DS
SH_CP
$\overline{MR}$
ST_CP
$\overline{OE}$
D Q FF0 CP R
D Q LATCH CP
D Q
D Q FF7 CP R
D Q LATCH CP
Q7'
Q0
Q1 Q2 Q3 Q4 Q5 Q6
Q7
MLA010


SH_CP
DS
ST_CP
MR
OE
Q0
Q1
Q6
Q7
Q7'
high-impedance OFF-state
MLA005-1


clk_in
rst_n_in
分频
20KHz
seg_dot_en
seg_data_en
seg_data
数码管
字库
Seg1
Seg2
Seg3
Seg4
Seg5
Seg6
WRITE
74HC595
时序
rclk_out
sclk_out
sdio_out

====Verilog code====

```
// ------------------------------------------------ --------------------
// >>>>>>>>>>>>>>>>>>>>>>>>> COPYRIGHT NOTICE < <<<<<<<<<<<<<<<<<<<<<<<
// ------------------------ --------------------------------------------
// Module:Segment_scan
/ /
// Author: Step
//
// Description: Display with Segment tube
//
// Web: www.stepfpga.com
//
// -------------------- ------------------------------------------------
/ / Code Revision History:
// ------------------------------------------- ------------------------
// Version: |Mod. Date: |Changes Made:
// V1.0 |2015/11/11 |Initial ver
// ------------------------------------------------ --------------------
module Segment_scan
 (
input                           clk_in ,                        //System clock
input                           rst_n_in ,              //System reset, low active
input           [ 3 : 0 ]       seg_data_1 ,            // SEG1 digital tube to display data
input           [ 3 : 0 ]       seg_data_2 ,            //SEG2 digital tube to display data
input           [ 3 : 0 ]       seg_data_3 ,            //SEG3 digital tube to display data
input           [ 3 : 0 ]       seg_data_4 ,            //SEG4 digital tube to display data
input           [ 3 : 0 ]       seg_data_5 ,            //SEG5 digital tube to display data
input           [ 3 : 0 ]       seg_data_6 ,            //SEG6 digital tube to display data
input           [ 5 : 0 ]       seg_data_en ,   //
 Enable every digital tube data display, [MSB~LSB]=[SEG6~SEG1] input            [ 5 : 0 ]
seg_dot_en ,            //
 Enable every digital tube decimal point display, [MSB~LSB]=[SEG6 ~SEG1] output         reg
rclk_out ,              //74HC595 RCK pin
output  reg                     sclk_out ,              //74HC595 SCK pin
output  reg                     sdio_out                 //74HC595 SER pin
) ;

parameter CLK_DIV_PERIOD =  600 ;  //Division coefficient

localparam      IDLE     =      3'd0 ;
localparam      MAIN     =      3'd1 ;
localparam      WRITE    =      3'd2 ;

localparam      LOW              =      1'b0 ;
localparam      HIGH     =      1'b1 ;

//Create a font library for the digital tube. The font data is related in the order of segment
codes.
// Here the font data [MSB~LSB]={DP,G,F,E,D,C,B,A}
reg [ 7 : 0 ] seg [ 15 : 0 ] ;
initial  begin
    seg [ 0 ]   =       8'h3f ;    // 0
    seg [ 1 ]   =       8'h06 ;    // 1
    seg [ 2 ]   =       8'h5b ;    // 2
    seg [ 3 ]   =       8 'h4f ;    // 3
    seg [4 ]    =       8'h66 ;    // 4
    seg [ 5 ]   =       8'h6d ;    // 5
    seg [ 6 ]   =       8'h7d ;    // 6
    seg [ 7 ]   =       8'h07 ;    // 7
    seg [ 8 ]   =       8'h7f ;    // 8
    seg [ 9 ]   =       8'h6f ;    // 9
        seg [ 10 ]      =       8'h77 ;    // A
    seg [ 11 ]  =       8'h7c ;    // b
    seg [ 12 ]  =       8'h39 ;    // C
    seg [ 13 ]  =       8'h5e ;    // d
    seg [ 14 ]  =       8'h79 ;    // E
    seg [ 15 ]  =       8'h71 ;    // F
end

//The counter counts the system clock signal
reg [ 9 : 0 ] cnt = 0 ;
always @ ( posedge clk_in or  negedge rst_n_in )  begin
        if ( ! Rst_n_in )  begin
                cnt <=  1'b0 ;
        end  else  begin
                if ( cnt >= ( CLK_DIV_PERIOD - 1 ) ) cnt <=  1'b0 ;
                else cnt <= cnt+  1'b1 ;
        end
end

// generates a pulse signal frequency-divided according to the period counted by the counter
REG CLK_DIV ;
Always @ ( posedge CLK_IN or  negedge rst_n_in )  the begin
        IF ( ! Rst_n_in )  the begin
                CLK_DIV <=  1'b0 ;
        End  the else  the begin
                IF ( CNT == ( CLK_DIV_PERIOD - 1 ) ) clk_div <=  1'b1 ;
                else clk_div <=  1'b0 ;
        end
end

//Use the state machine to complete the scanning of the digital tube and the realization of 74
HC595 timing
reg             [ 15 : 0 ]              data_reg ;
reg             [ 2 : 0 ]               cnt_main ;
reg             [ 5 : 0 ]               cnt_write ;
reg             [ 2 : 0 ]               state = IDLE ;
always @ ( posedge clk_in or  negedge rst_n_in )  begin
        if ( ! rst_n_in )  begin        //In the reset state, each register is set to the init
ial value
                state <= IDLE ;
                cnt_main <=  3'd0 ;
                cnt_write <=  6'd0 ;
                sdio_out <=  1'b0 ;
                sclk_out <= LOW ;
                rclk_out <= LOW ;
        end  else  begin
                case ( state )
                        IDLE : begin    //IDLE as the first state is equivalent to soft reset
                                        state <= MAIN ;
                                        cnt_main <=  3'd0 ;
                                        cnt_write <=  6'd0;
                                        sdio_out <=  1'b0 ;
                                        sclk_out <= LOW ;
                                        rclk_out <= LOW ;
                                end
                        MAIN : begin
                                        if ( cnt_main >=  3'd5 ) cnt_main <=  1'b0 ;
                                        else cnt_main <= cnt_main +  1'b1 ;
                                        case ( cnt_main )
                                                //Scan the 6-digit digital tube bit by bit
                                                3'd0 :  begin
                                                                        state <= WRITE ;
//After configuring the data sent to 74HC595, jump to the WRITE state and complete the serial
 sequence
                                                                        data_reg <=  { seg [ s
eg_data_1 ] | ( seg_dot_en [ 0 ] ? 8'h80 : 8'h00 ) , seg_data_en [ 0 ] ? 8'hfe : 8'hff } ;
                                                                        //data_reg[15:8] is se
gment selection, data_reg[7:0] is bit selection
                                                                        //seg[seg_data_1] is t
o obtain corresponding font data according to port input
                                                                        //seg_dot_en[0 ]?8'h8
0:8'h00 is to control the level of the decimal point DP segment of the SEG1 digital tube accor
ding to the decimal point display enable signal
                                                                        //seg_data_en[0]? 8'hf
e:8'hff is to control SEG1 according to the data display enable signal The level of the bit se
lection pin of the digital tube
                                                                end
                                                3'd1 :  begin
                                                                        state<= WRITE ;
                                                                        data_reg <=  { seg [ s
eg_data_2 ] | ( seg_dot_en [ 1 ] ? 8'h80 : 8'h00 ) , seg_data_en [ 1 ] ? 8'hfd : 8'hff } ;
                                                                end
                                                3'd2 :  begin
                                                                        state < = WRITE ;
                                                                        data_reg <=  { seg [ s
eg_data_3 ] | ( seg_dot_en [2 ] ? 8'h80 : 8'h00 ) , seg_data_en [ 2 ] ? 8'hfb : 8'hff } ;
                                                                end
                                                3'd3 :  begin
                                                                        state <= WRITE ;
                                                                        data_reg <=  { seg [ s
eg_data_4 ] | ( seg_dot_en [ 3 ] ? 8'h80 : 8'h00 ) , seg_data_en [ 3 ] ? 8'hf7 :8'hff } ;
                                                                end
                                                3'd4 :  begin
                                                                        state <= WRITE ;
                                                                        data_reg <=  { seg [ s
eg_data_5 ] | ( seg_dot_en [ 4 ] ? 8'h80 : 8'h00 ) , seg_data_en [ 4 ] ? 8'hef : 8 'hff } ;
                                                                end
                                                3'd5 :  begin
```

```
                                                                           state <= WRITE ;
                                                                           data_reg <= { seg [ se
g_data_6 ] | ( seg_dot_en [ 5 ] ? 8'h80 : 8'h00 ) , seg_data_en [ 5 ] ? 8'hdf : 8'hff } ;
                                                                   end
                                                     default : state <= IDLE ;
                                             endcase
                                     end
                             WRITE : begin
                                             if ( clk_div )  begin   //74HC595 serial clock has spe
ed requirements, it needs to follow the divided beat
                                                     if ( cnt_write >=  6'd33 )cnt_write <=  1'b0 ;
                                                     else cnt_write <= cnt_write +  1'b1 ;
                                                     case ( cnt_write )
                                                             //74HC595 is a serial to parallel chi
p, 3 inputs can produce 8 outputs, and can be used in cascade
                                                             //74HC595 For timing implementation, r
efer to 74HC595 chip manual
                                                             6'd0 :   begin sclk_out <= LOW ; sdio_
out <= data_reg [ 15 ] ;  end           //SER update data at the falling edge of SCK
                                                             6'd1 :   begin sclk_out <= HIGH ;  end
//SCK SER data is stable at the rising edge
                                                             6'd2 :   beginsclk_out <= LOW ; sdio_o
ut <= data_reg [ 14 ] ;  end
                                                             6'd3 :   begin sclk_out <= HIGH ;  end
                                                             6'd4 :   begin sclk_out <= LOW ; sdio_
out <= data_reg [ 13 ] ;  end
                                                             6'd5 :   begin sclk_out <= HIGH ;  end
                                                             6'd6 :   begin sclk_out <= LOW ;sdio_o
ut <= data_reg [ 12 ] ;  end
                                                             6'd7 :   begin sclk_out <= HIGH ;  end
                                                             6'd8 :   begin sclk_out <= LOW ; sdio_
out <= data_reg [ 11 ] ;  end
                                                             6'd9 :   begin sclk_out <= HIGH ;  end
                                                             6'd10 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [10 ] ;  end
                                                             6'd11 :  begin sclk_out <= HIGH ;  end
                                                             6'd12 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 9 ] ;  end
                                                             6'd13 :  begin sclk_out <= HIGH ;  end
                                                             6'd14 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 8 ] ;  end
                                                             6'd15:  begin sclk_out <= HIGH ;  end
                                                             6'd16 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 7 ] ;  end
                                                             6'd17 :  begin sclk_out <= HIGH ;  end
                                                             6'd18 :  begin sclk_out <= LOW ; sdio_
out < = data_reg [ 6 ] ;  end
                                                             6'd19 :  begin sclk_out <=HIGH ;  end
                                                             6'd20 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 5 ] ;  end
                                                             6'd21 :  begin sclk_out <= HIGH ;  end
                                                             6'd22 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 4 ] ;  end
```

```
                                                        6'd23 :  begin sclk_out <= HIGH ;  end
                                                        6'd24 : begin sclk_out <= LOW ; sdio_o
ut <= data_reg [ 3 ] ;  end
                                                        6'd25 :  begin sclk_out <= HIGH ;  end
                                                        6'd26 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 2 ] ;  end
                                                        6'd27 :  begin sclk_out <= HIGH ;  end
                                                        6'd28 :  begin sclk_out <= LOW; sdio_o
ut <= data_reg [ 1 ] ;  end
                                                        6'd29 :  begin sclk_out <= HIGH ;  end
                                                        6'd30 :  begin sclk_out <= LOW ; sdio_
out <= data_reg [ 0 ] ;  end
                                                        6'd31 :  begin sclk_out <= HIGH ;  end
                                                        6'd32 :  begin rclk_out <= HIGH ;  end
//When the 16-bit data transmission is completed, RCK is pulled high, and the output is effect
ive
                                                        6'd33:  begin rclk_out <= LOW ; state
<= MAIN ;  end
                                                        default : state <= IDLE ;
                                                endcase
                                        end  else  begin
                                                sclk_out <= sclk_out ;
                                                sdio_out <= sdio_out ;
                                                rclk_out <= rclk_out ;
                                                cnt_write <= cnt_write ;
                                                state <= state ;
                                        end
                                end
                        default : state <=IDLE ;
                endcase
        end
end

endmodule
```

====Summary====

This section mainly explains the related principles and software design of the digital tube display. You need to master it while creating your own project, and generate the 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 digital control Scanner: subsequent download connection will be updated
using the STEP-MAX10 a digital scanner: subsequent download link will be updated