```
module top(
    input clk,
    input rstn,
    output reg [3:0] led,
    input [3:0] key,
    output [5:0] seg_sel,
    output [7:0] seg_dig
);

    reg pause;
    reg calculate_enable;
    reg [19:0] delay_prime;
    reg [19:0] display_prime;
    wire [19:0] current_prime;
    wire display_enable;
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            delay_prime <= 2;
            display_prime <= 2;
        end else begin
            delay_prime <= current_prime;
            if (display_enable) display_prime <= current_prime;
        end
    end

    assign prime_change = (delay_prime != current_prime) ? 1 : 0;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            calculate_enable <= 0;
        end else begin
            if(display_enable) calculate_enable <= 1;
            else if(prime_change)    calculate_enable <= 0;
        end
    end

    wire [3:0] key_press;
    wire [3:0] key_down;
    key_filter #(
        .KEY_W(4),
        .DELAY_TIME(1000000)
    ) key_filter_instance (
        .clk(clk),
        .rst_n(rstn),
```

```
        .key_in(key),
        .key_down(key_down)
);

display_enable display_enable_instance(
        .clk(clk),
        .rstn(rstn),
        .display_enable(display_enable)
);

always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
                pause <= 1'b1;
        end else begin
                pause <= key_down[0] | calculate_enable;
        end
end

my_prime_checker checker_instance (
        .sclk(clk),
        .rst_n(rstn),
        .pause(pause),
        .prime_data(current_prime)
);

wire [23:0] dec_value;
bin_to_dec b2d_instance (
        .bin(display_prime),
        .dec(dec_value)
);

reg [3:0] digits [5:0];
integer i;

always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
                for (i = 0; i < 6; i = i + 1) begin
                        digits[i] <= 0;
                end
        end else begin
                digits[5] <= dec_value[23:20];
                digits[4] <= dec_value[19:16];
                digits[3] <= dec_value[15:12];
                digits[2] <= dec_value[11:8];
```

```
                digits[1] <= dec_value[7:4];
                digits[0] <= dec_value[3:0];
            end
        end

        wire [7:0] seg_out [5:0];
        generate
            genvar j;
            for (j = 0; j < 6; j = j + 1) begin : seg_decoder
                led7seg_decode d(.digit(digits[j]), .valid(1'b1), .seg(seg_out[j]));
            end
        endgenerate

        reg [47:0] prime_digits;
        always @(*) begin
            for (i = 0; i < 6; i = i + 1) begin
                prime_digits[i*8 +: 8] = seg_out[i];
            end
        end

        seg_driver #(6) driver(clk, rstn, 6'b111111, prime_digits, seg_sel, seg_dig);

endmodule
```

```
module bin_to_dec(
    input [19:0] bin,     // 20 位二进制输入
    output reg [23:0] dec   // 24 位十进制输出 (BCD 编码：最高支持 6 位十进制)
);
    integer i, j;
    reg [43:0] shift_reg;   // 44 位移位寄存器，包含原始二进制和 BCD 结果

    always @(*) begin
        // 初始化移位寄存器
        shift_reg = {24'b0, bin};   // 高 24 位为 BCD 结果，低 20 位为二进制输入

        // Double Dabble 算法
        for (i = 0; i < 20; i = i + 1) begin
            // 检查高位是否需要加 3 修正
```

```
            for (j = 43; j >= 20; j = j - 4) begin
                if (shift_reg[j-:4] >= 5) begin
                    shift_reg[j-:4] = shift_reg[j-:4] + 3;    // 修正加 3
                end
            end
            // 左移一位
            shift_reg = shift_reg << 1;
        end

        // 提取 BCD 结果
        dec = shift_reg[43:20];
    end
endmodule
```

```
module display_enable(
    input clk,
    input rstn,
    output reg display_enable
);

    reg [31:0] display_cnt;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            display_cnt <= 0;
            display_enable <= 0;
        end else if (display_cnt == 32'd49999999) begin
            display_cnt <= 0;
            //display_enable <= ~display_enable;
            display_enable <= 1;
        end else begin
            display_cnt <= display_cnt + 1;
            display_enable <= 0;
        end
    end
endmodule
```

```
module key_filter #(parameter KEY_W = 4,DELAY_TIME = 1_000_000)(
    input                           clk         ,
    input                           rst_n       ,
    input           [KEY_W-1:0]     key_in      ,
    output  reg     [KEY_W-1:0]     key_down
);

//计数器
reg     [19:0]      cnt;
wire                add_cnt;
wire                end_cnt;

// 标志信号
reg                 filter_flag;

reg     [KEY_W-1:0] key_r0;
reg     [KEY_W-1:0] key_r1;
reg     [KEY_W-1:0] key_r2;

wire                n_edge;
wire                p_edge;


// 对输入按键进行打拍，异步信号同步并检测边沿
// 打几拍就是延时几个周期
always @(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        key_r0 <= -1;//负数以补码的方式存放，对原码取反加 1
        key_r1 <= -1;
        key_r2 <= -1;
    end
    else begin
        key_r0 <= key_in;
        key_r1 <= key_r0;
        key_r2 <= key_r1;
    end
end

// assign   n_edge = {!key_r1[1] && key_r2[1],!key_r1[0] && key_r2[0]};//第一种检测边沿
//三目运算符
assign   n_edge = ~key_r1 & key_r2?1'b1:1'b0;//第二种检测边沿
```

```
// assign    p_edge = {key_r1[1] && !key_r2[1],key_r1[0] && !key_r2[1]};
assign    p_edge = key_r1 & ~key_r2?1'b1:1'b0;


//当检测到下降沿，filter_flag 为 1
always @(posedge clk or negedge rst_n)begin
      if(!rst_n)begin
            filter_flag <= 1'b0;
      end
      //检测到下降沿
      else if(n_edge)begin
            filter_flag <= 1'b1;
      end
      else if(end_cnt)begin
            filter_flag <= 1'b0;
      end
      else begin
            filter_flag <= filter_flag;
      end
end

//当检测到 filter_flga 为 1 的时候开始计数
always @(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
            cnt <= 0;
      end
      else if(add_cnt)begin
                  if(end_cnt || p_edge)begin
                        cnt <= 0;
                  end
                  else begin
                        cnt <= cnt + 1;
                  end
      end
    else    begin
          cnt <= cnt;
      end
end

assign add_cnt = filter_flag;
assign end_cnt = add_cnt && cnt == DELAY_TIME-1;

//key_down 取的值是最后当前周期的 key_r2 值，是个稳定的值
always @(posedge clk or negedge rst_n)begin
```

```
        if(!rst_n)begin
                key_down <= 0;
        end
        else if(end_cnt)begin
                key_down <= ~key_r2;
        end
        else begin
                key_down <= 0;
        end
end
endmodule




module my_prime_checker(
        input                           sclk                    ,
        input                           rst_n                   ,
        input                           pause                   ,
        output reg [19:0]       prime_data
        );

wire clk;
assign clk = sclk & pause;

reg                     [19:0]  temp_prime_data;
reg                     [19:0]  cnt_data                ;
reg                     [9:0]   cnt_cnt_data        ;

reg                     flag;
reg [19:0] D;
reg [9:0] mod;
reg [19:0]  temp_A;
reg [19:0]  temp_B;
reg [19:0]  temp_D;
reg [4:0]   i_cnt;
reg     start;
reg   done;

reg [2:0]  pstate, nstate;
localparam              idle    =   3'b000,
                        div_init    =   3'b001,
                div_shift       =   3'b010,
```

```
                    div_sub    =     3'b011,
                    div_end    =     3'b100;
always@(posedge clk, negedge rst_n)
  if (!rst_n)
      pstate<=     idle;
  else
      pstate<=    nstate;

always@(*)
    case (pstate)
idle: begin
        temp_B     =    20'd0;
        temp_A     =    20'd0;
            mod     =    10'd0;
              D     =    20'd0;
        done        =     1'b0;
        if (cnt_data <= 20'd3)
           nstate        =    div_end;
        else if( start    ==    1'b0 )
            nstate        =    div_init;
        else
            nstate        =    idle;
      end
div_init: begin
      temp_B     =     {cnt_cnt_data,10'b0};
      temp_A     =     cnt_data ;
        i_cnt    =     5'd11;
      temp_D     =    20'd0;
  nstate        =    div_sub;
         end
div_shift:begin
            if (i_cnt > 5'd0)
            begin
                temp_B    =     temp_B    >> 1;    //{1'b0,temp_B[9:1]};
                temp_D    =     temp_D    << 1;
                nstate       =    div_sub;
            end
            else
                 nstate       =    div_end;
            end
div_sub: begin
            if ((i_cnt > 5'd0))// && (temp_A >= temp_B))//
            begin
                            i_cnt        =    i_cnt-1'b1;
```

```
                    nstate      =   div_shift;
                if (temp_A >= temp_B)
                begin
                    temp_A     =     temp_A -   temp_B;
                    temp_D[0]       =     1'b1;

                end
                else
                begin
                     temp_A   =   temp_A;
                     temp_D[0]       =     1'b0;
                end
            end
            else
            begin
                    i_cnt       =     5'd11;
                    nstate      =   div_end;

            end
          end
div_end: begin
             D   =   temp_D;
        done   =   1'b1;
        nstate =   idle;
      if (cnt_data <= 20'd3)
          mod = 10'd1;
      else
         mod = temp_A[9:0];
         end
default: begin
        nstate = idle;
          end
endcase

always @(posedge clk or negedge rst_n)
    if(rst_n == 1'b0)
    begin
        cnt_data                <=                  20'd2;
     end
    else //if (pause)
          if(cnt_data ==20'd999999 && (cnt_cnt_data == 10'd1000))
          begin
                cnt_data                <=                  20'd2;
          end
```

```
            else if (cnt_data    ==    20'd2)
            begin
                    cnt_data                <=                    cnt_data + 1'b1;
            end
           else  if(((  cnt_cnt_data  >=  cnt_data-2'd2  )  &&  (done  ==  1'b1))  ||  (flag==1'b1))
//cnt_data >>1
            begin
                     cnt_data               <=                    cnt_data + 2'b10;
            end
      else
            cnt_data                  <=                   cnt_data;

always @(posedge clk or negedge rst_n)
      if(rst_n == 1'b0)
      begin
           cnt_cnt_data             <=                   14'd3;
      end
      else if (pause)
           begin
           if((done == 1'b1) && ((cnt_cnt_data >= cnt_data    -    2'd2 )    ||    (cnt_cnt_data ==
10'd1000))|| (flag==1'b1))     //cnt_data >> 1
           begin
                cnt_cnt_data            <=                   14'd3;
           end
           else if (done == 1'b1)
           begin
                cnt_cnt_data            <=                   cnt_cnt_data + 2'd2;
           end
       else
                cnt_cnt_data            <=                   cnt_cnt_data;
      end

 reg cnt_data_1,cnt_cnt_data_1;
 wire cnt_change,cnt_cnt_change;
 assign cnt_change = cnt_data[1] ^ cnt_data_1;
 assign cnt_cnt_change = cnt_cnt_data[1] ^ cnt_cnt_data_1;
 always @(posedge clk or negedge rst_n)
       if(rst_n == 1'b0)
       begin
          cnt_data_1 <=    1'b0;
          cnt_cnt_data_1 <=    1'b0;
       end
       else
       begin
```

```
            cnt_data_1 <=    cnt_data[1];
            cnt_cnt_data_1 <=    cnt_cnt_data[1];
        end

    always @(posedge sclk or negedge rst_n)
                if(rst_n == 1'b0)
                    start <=    1'b1;
                else if (cnt_change == 1'b1 || cnt_cnt_change == 1'b1)
                    start <=      1'b0;
                else
                    start <=    1'b1;

always @ * //(mod or cnt_data)
    if (mod==20'd0 && done == 1'b1)//((cnt_data <= 20'd3)||)
                flag                        =                       1'b1;
    else
                flag                        =                       1'b0;

        always @(posedge clk or negedge rst_n)
        begin
            if (!rst_n)
            begin
                prime_data <= 20'd2;
            end
            else if (prime_data == 20'd2) begin
                prime_data <= 20'd3;
            end
            else if ((cnt_cnt_data == cnt_data - 2'd2)    &&      (mod != 10'd0))
                            prime_data <= cnt_data;
        end

endmodule




// seg_driver.v
module seg_driver #(parameter NPorts=6) (
        input clk, rstn,
        input [NPorts-1:0] valid_i,
        input [NPorts*8-1:0] seg_i,
        output reg [NPorts-1:0] valid_o,
        output [7:0] seg_o
```

```
);

    reg [14:0] cnt;
    always @(posedge clk or negedge rstn)
    if (~rstn)
        cnt <= 0;
    else
        cnt <= cnt + 1;

    reg [NPorts-1:0] sel;
    always @(posedge clk or negedge rstn)
    if (~rstn)
        sel <= 0;
    else if (cnt == 0)
        sel <= sel == NPorts - 1 ? 0 : sel + 1;

    always @(sel, valid_i) begin
        valid_o = {NPorts{1'b1}};
        valid_o[sel] = ~valid_i[sel];
    end

    assign seg_o = ~seg_i[sel*8+:8];

endmodule




// led7seg_decode.v
module led7seg_decode(input [3:0] digit, input valid, output reg [7:0] seg);

    always @(digit)
    if(valid)
        case(digit)
            0: seg = 8'b00111111;
            1: seg = 8'b00000110;
            2: seg = 8'b01011011;
            3: seg = 8'b01001111;
            4: seg = 8'b01100110;
            5: seg = 8'b01101101;
            6: seg = 8'b01111101;
            7: seg = 8'b00000111;
            8: seg = 8'b01111111;
```

```
                9: seg = 8'b01101111;
                10: seg = 8'b01110111;
                11: seg = 8'b01111100;
                12: seg = 8'b00111001;
                13: seg = 8'b01011110;
                14: seg = 8'b01111011;
                15: seg = 8'b01110001;
                default: seg = 0;
            endcase
        else seg = 8'd0;

endmodule
```