附件一 verilog 硬件底层

I2c_config.v:

```
module I2C_Config (     //      Host Side
                        input sys_clk,
                        input sys_rst_n,
                        input[23:0]     I2C_DATA,
                        input GO,
                        //      I2C Side
                        output I2C_SCLK,
                        inout I2C_SDAT
                        );

reg   [15:0]     mI2C_CLK_DIV;
reg              mI2C_CTRL_CLK;


parameterCLK_Freq     =     50000000;     //    50    MHz
parameterI2C_Freq =     20000;          //    20    KHz


always@(posedge sys_clk or negedge sys_rst_n)
begin
      if(!sys_rst_n)
      begin
            mI2C_CTRL_CLK <=   0;
            mI2C_CLK_DIV   <=   0;
      end
      else
      begin
            if( mI2C_CLK_DIV      < (CLK_Freq/I2C_Freq) )
            mI2C_CLK_DIV   <=   mI2C_CLK_DIV+1;
            else
            begin
                  mI2C_CLK_DIV   <=   0;
                  mI2C_CTRL_CLK <=   ~mI2C_CTRL_CLK;
            end
      end
end

I2C_Controller      u0    (     .CLOCK(mI2C_CTRL_CLK),          //      Controller Work Clock
                                .I2C_SCLK(I2C_SCLK),          //      I2C CLOCK
                                .I2C_SDAT(I2C_SDAT),          //      I2C DATA
```

```
                                .I2C_DATA(I2C_DATA),            //
        DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
                                .GO(GO),                        //      GO transfor
                                .RESET(sys_rst_n)
);
Endmodule
```

Iic_controller.v:

```
module I2C_Controller (
        CLOCK,
        I2C_SCLK,//I2C CLOCK
        I2C_SDAT,//I2C DATA
        I2C_DATA,//DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
        GO,             //GO transfor
        END,            //END transfor
        W_R,            //W_R
        ACK,              //ACK
        RESET,
        //TEST
        SD_COUNTER,
        SDO
);
        input    CLOCK;
        input    [23:0]I2C_DATA;
        input    GO;
        input    RESET;
        input    W_R;
        inout    I2C_SDAT;
        output I2C_SCLK;
        output END;
        output ACK;

//TEST
        output [5:0] SD_COUNTER;
        output SDO;



reg SDO;
reg SCLK;
reg END;
reg [23:0]SD;
reg [5:0]SD_COUNTER;

wire I2C_SCLK=SCLK | ( ((SD_COUNTER >= 4) & (SD_COUNTER <=30))? ~CLOCK :0 );
```

```
wire I2C_SDAT=SDO?1'bz:0 ;

reg ACK1,ACK2,ACK3;
wire ACK=ACK1 | ACK2 |ACK3;

//--I2C COUNTER
always @(negedge RESET or posedge CLOCK ) begin
if (!RESET) SD_COUNTER=6'b111111;
else begin
if (GO==0)
      SD_COUNTER=0;
      else
      if (SD_COUNTER < 6'b111111) SD_COUNTER=SD_COUNTER+1;
end
end
//----

always @(negedge RESET or    posedge CLOCK ) begin
if (!RESET) begin SCLK=1;SDO=1; ACK1=0;ACK2=0;ACK3=0; END=1; end
else
case (SD_COUNTER)
      6'd0    : begin ACK1=0 ;ACK2=0 ;ACK3=0 ; END=0; SDO=1; SCLK=1;end
      //start
      6'd1    : begin SD=I2C_DATA;SDO=0;end
      6'd2    : SCLK=0;
      //SLAVE ADDR
      6'd3    : SDO=SD[23];
      6'd4    : SDO=SD[22];
      6'd5    : SDO=SD[21];
      6'd6    : SDO=SD[20];
      6'd7    : SDO=SD[19];
      6'd8    : SDO=SD[18];
      6'd9    : SDO=SD[17];
      6'd10 : SDO=SD[16];
      6'd11 : SDO=1'b1;//ACK

      //SUB ADDR
      6'd12    : begin SDO=SD[15]; ACK1=I2C_SDAT; end
      6'd13    : SDO=SD[14];
      6'd14    : SDO=SD[13];
      6'd15    : SDO=SD[12];
      6'd16    : SDO=SD[11];
      6'd17    : SDO=SD[10];
      6'd18    : SDO=SD[9];
```

```
6'd19    : SDO=SD[8];
6'd20    : SDO=1'b1;//ACK

//DATA
6'd21    : begin SDO=SD[7]; ACK2=I2C_SDAT; end
6'd22    : SDO=SD[6];
6'd23    : SDO=SD[5];
6'd24    : SDO=SD[4];
6'd25    : SDO=SD[3];
6'd26    : SDO=SD[2];
6'd27    : SDO=SD[1];
6'd28    : SDO=SD[0];
6'd29    : SDO=1'b1;//ACK


//stop
6'd30 : begin SDO=1'b0;SCLK=1'b0; ACK3=I2C_SDAT; end
6'd31 : SCLK=1'b1;
6'd32 : begin SDO=1'b1; END=1; end

endcase
end
endmodule
```

qrs_curriculum_design.v:

```
module qrs_curriculum_design(
input sys_clk,
input sys_rst_n,
inout I2C_SDAT,
input KEY0,
input KEY1,
output I2C_SCLK
);

reg  [23:0]     mI2C_DATA;
reg             mI2C_GO;

wire        mI2C_END;
wire        mI2C_ACK;

    system_qsy u0 (
        .clk_clk              (sys_clk),              //         clk.clk
        .reset_reset_n     (sys_rst_n),     //       reset.reset_n
        .iic_data_export (mI2C_DATA), // iic_data.export
```

```
        .iic_go_export     (mI2C_GO),     //     iic_go.export
        .key1_export       (KEY1),        //       key1.export
        .key0_export       (KEY0)         //       key0.export
    );

    I2C_Config (  //     Host Side
                        .sys_clk(sys_clk),
                        .sys_rst_n(sys_rst_n),
                        //     I2C Side
                        .I2C_SCLK(I2C_SCLK),
                        .I2C_SDAT(I2C_SDAT),
                        .I2C_DATA(mI2C_DATA),
                        .GO(mI2C_GO)
    );
Endmodule
```

附件二：计算 PI 等 c 语言代码

Main.v:

```
#include <stdio.h>
#include "system.h" //系统头文件
#include "alt_types.h" //数据类型头文件
#include "altera_avalon_pio_regs.h"//pio 寄存器头文件
#include "oled.h"

int main()
{
    int a=10000, b=0, c=400, d=0, e=0, f[401],
g=0,num[100]={0},x=0;
    u8 i=0;
    u32 key0,key1;
    OLED_Init();
    OLED_ShowString(45,3, (u8 *)"Hello from Nios II!",1);
    delay_ms(100);
    while(1)
    {
        if(num[99]==0)
        {
        for(;b-c;)
        f[b++]=a/5;
        for(;d=0,g=c*2;c-
=14,num[x]=e+d/a,printf("%.4d",num[x]),delay_ms(1),x++,e=d%a)
        for(b=c; d+=f[b]*a, f[b]=d%--g, d/=g--, --b; d*=b);
        }
```

```
        else
        {
            show_pi_ten(num,i);
            key0 = IORD_ALTERA_AVALON_PIO_DATA(KEY0_BASE);
            key1 = IORD_ALTERA_AVALON_PIO_DATA(KEY1_BASE);
            if(key0 == 0)
            {
                delay_ms(10);
                if(key0 == 0)
                {
                    i++;
                    if(i>=10)i = 10;
                }
            }
            if(key1 == 0)
            {
                delay_ms(10);
                if(key1 == 0)
                {
                    i--;
                    if(i<=0)i = 0;
                }
            }
        }
        delay_ms(5);
    }
  return 0;
}
```

Oled.c:

```
#include "oled.h"
#include "oledfont.h"
#include "system.h" //系统头文件
#include "alt_types.h" //数据类型头文件
#include "altera_avalon_pio_regs.h"//pio 寄存器头文件

void delay_us(u16 time_us)
{
   u16 i=0;
   while(time_us--)
   {
      i=50;  //自己定义
      while(i--) ;
   }
```

```
}

void delay_ms(u16 time_ms)
{
   u16 i=0;
   while(time_ms--)
   {
      i=50000;  //自己定义
      while(i--) ;
   }
}
void I2C_WriteByte(u8 addr, u8 data)
{
    u32 iic_data;
    u32 go;
    iic_data = (OLED_ADDRESS<<16) + ((u32)addr<<8) + ((u32)data);
    go = 0;
    IOWR_ALTERA_AVALON_PIO_DATA(IIC_GO_BASE,go);
    delay_us(100);

    IOWR_ALTERA_AVALON_PIO_DATA(IIC_DATA_BASE,iic_data);

    go = 1;
    IOWR_ALTERA_AVALON_PIO_DATA(IIC_GO_BASE,go);
    delay_ms(1);
}
void OLED_WR_Byte(u8 dat, u8 cmd, u8 type)
{
    if(!type)
    {
        dat = ~dat;
    }
    if(cmd)
        I2C_WriteByte(0x40, dat);
    else
        I2C_WriteByte(0x00, dat);
}
void OLED_Set_Pos(unsigned char x, unsigned char y)
{
    OLED_WR_Byte(0xb0 + y, OLED_CMD, 1);
    OLED_WR_Byte(((x & 0xf0) >> 4) | 0x10, OLED_CMD, 1);
    OLED_WR_Byte((x & 0x0f) | 0x01, OLED_CMD, 1);
}
//开启屏幕显示
```

```
void OLED_Display_On(void)
{
    OLED_WR_Byte(0X8D, OLED_CMD, 1); //SET DCDC命令
    OLED_WR_Byte(0X14, OLED_CMD, 1); //DCDC ON
    OLED_WR_Byte(0XAF, OLED_CMD, 1); //DISPLAY ON
}
//关闭OLED显示
void OLED_Display_Off(void)
{
    OLED_WR_Byte(0X8D, OLED_CMD, 1); //SET DCDC命令
    OLED_WR_Byte(0X10, OLED_CMD, 1); //DCDC OFF
    OLED_WR_Byte(0XAE, OLED_CMD, 1); //DISPLAY OFF
}
//清屏函数，清完屏，整个屏幕是黑色的和没点一样。
void OLED_Clear(void)
{
    u8 i, n;
    for(i = 0; i < 8; i++)
    {
        OLED_WR_Byte (0xb0 + i, OLED_CMD, 1); //设置页地址
        OLED_WR_Byte (0x00, OLED_CMD, 1);    //设置显示位置-列低地址·
        OLED_WR_Byte (0x10, OLED_CMD, 1);    //设置显示位置-列高位置·
        for(n = 0; n < 128; n++)OLED_WR_Byte(0, OLED_DATA, 1);
    } //更新显示
}
//在指定位置显示一个字符，包括部分字符
//x 0~127
//y 0~63
//mode：0，反白显示，1，正常显示
//size：选择字体 16/12
void OLED_ShowChar(u8 x, u8 y, u8 chr, u8 type)
{
    unsigned char c = 0, i = 0;
    c = chr - ' ';
    if(x > Max_Column - 1) {
        x = 0;
        y = y + 2;
    }
    if(SIZE == 16)
    {
        OLED_Set_Pos(x, y);
        for(i = 0; i < 8; i++)
            OLED_WR_Byte(F8X16[c * 16 + i], OLED_DATA, type);
        OLED_Set_Pos(x, y + 1);
```

```
        for(i = 0; i < 8; i++)
            OLED_WR_Byte(F8X16[c * 16 + i + 8], OLED_DATA, type);
    }
    else
    {
        OLED_Set_Pos(x, y + 1);
        for(i = 0; i < 6; i++)
            OLED_WR_Byte(F6x8[c][i], OLED_DATA, type);
    }
}
//m^n函数
u32 oled_pow(u8 m, u8 n)
{
    u32 result = 1;
    while(n--)result *= m;
    return result;
}
//显示2个数字
//x,y:起点坐标
//len:数字的位数
//size:字体的大小
//mode:模式 0,填充模式;1叠加模式
//num:数值(0~4294967295);
void OLED_ShowNum(u8 x, u8 y, u32 num, u8 len, u8 size, u8 type)
{
    u8 t, temp;
    u8 enshow = 0;
    for(t = 0; t < len; t++)
    {
        temp = (num / oled_pow(10, len - t - 1)) % 10;
        if(enshow == 0 && t < (len - 1))
        {
            if(temp == 0)
            {
                OLED_ShowChar(x + (size / 2)*t, y, ' ', type);
                continue;
            } else enshow = 1;

        }
        OLED_ShowChar(x + (size / 2)*t, y, temp + '0', type);
    }
}
//显示一个字符号串
void OLED_ShowString(u8 x, u8 y, u8 *chr, u8 type)
```

```
{
    unsigned char j = 0;
    while (chr[j] != '\0')
    {
        if(!type)
            OLED_ShowChar(x, y, chr[j], 0);
        else
            OLED_ShowChar(x, y, chr[j], 1);
        x += 6;
        if(x > 122) {
            x = 0;
            y += 2;
        }
        j++;
    }
}
//显示汉字
void OLED_ShowCHinese(u8 x, u8 y, u8 no, u8 type)
{
}
//显示BMP图片
void OLED_DrawBMP(unsigned char x0, unsigned char y0, unsigned
char x1, unsigned char y1, unsigned char BMP[], u8 type)
{
    unsigned int j = 0;
    unsigned char x, y;

    if(y1 % 8 == 0) y = y1 / 8;
    else y = y1 / 8 + 1;
    for(y = y0; y < y1; y++)
    {
        OLED_Set_Pos(x0, y);
        for(x = x0; x < x1; x++)
        {
            OLED_WR_Byte(BMP[j++], OLED_DATA, type);
        }
    }
}
void OLED_Init(void)
{
    delay_ms(100);

    OLED_WR_Byte(0xAE, OLED_CMD, 1);
    OLED_WR_Byte(0x20, OLED_CMD, 1);
```

```
    OLED_WR_Byte(0x10, OLED_CMD, 1);
    OLED_WR_Byte(0xb0, OLED_CMD, 1);
    OLED_WR_Byte(0xc8, OLED_CMD, 1);
    OLED_WR_Byte(0x00, OLED_CMD, 1);
    OLED_WR_Byte(0x10, OLED_CMD, 1);
    OLED_WR_Byte(0x40, OLED_CMD, 1);
    OLED_WR_Byte(0x81, OLED_CMD, 1);
    OLED_WR_Byte(0xff, OLED_CMD, 1);
    OLED_WR_Byte(0xa1, OLED_CMD, 1);
    OLED_WR_Byte(0xa6, OLED_CMD, 1);
    OLED_WR_Byte(0xa8, OLED_CMD, 1);
    OLED_WR_Byte(0x3f, OLED_CMD, 1);
    OLED_WR_Byte(0xa4, OLED_CMD, 1);
    OLED_WR_Byte(0xd3, OLED_CMD, 1);
    OLED_WR_Byte(0x00, OLED_CMD, 1);
    OLED_WR_Byte(0xd5, OLED_CMD, 1);
    OLED_WR_Byte(0xf0, OLED_CMD, 1);
    OLED_WR_Byte(0xd9, OLED_CMD, 1);
    OLED_WR_Byte(0x22, OLED_CMD, 1);
    OLED_WR_Byte(0xda, OLED_CMD, 1);
    OLED_WR_Byte(0x12, OLED_CMD, 1);
    OLED_WR_Byte(0xdb, OLED_CMD, 1);
    OLED_WR_Byte(0x20, OLED_CMD, 1);
    OLED_WR_Byte(0x8d, OLED_CMD, 1);
    OLED_WR_Byte(0x14, OLED_CMD, 1);
    OLED_WR_Byte(0xaf, OLED_CMD, 1);

    OLED_Clear();
    OLED_Set_Pos(0,0);
}
void show_pi_ten(int c[100],int i)
{
    OLED_ShowNum(10,10,c[i],1,16,1);
    OLED_ShowNum(14,10,c[i+1],1,16,1);
    OLED_ShowNum(18,10,c[i+2],1,16,1);
    OLED_ShowNum(22,10,c[i+3],1,16,1);
    OLED_ShowNum(26,10,c[i+4],1,16,1);
    OLED_ShowNum(30,10,c[i+5],1,16,1);
    OLED_ShowNum(34,10,c[i+6],1,16,1);
    OLED_ShowNum(38,10,c[i+7],1,16,1);
    OLED_ShowNum(42,10,c[i+8],1,16,1);
    OLED_ShowNum(46,10,c[i+9],1,16,1);

}
```