基于robei eda的实验室桌面辅助机器人

技术文档

车宏宇 申梦飞 戴喆

总说明：在本项目中，坐标角度等数据均使用32位定点数储存。31位为符号位，0正1负。30-16位为整数位。15-0位为小数位。以原码形式储存。

环境：robei eda+quartus13.1联合开发烧录

目录：

一：机械臂部分

1：运算模块（basic文件夹）

（1）：qadd加法模块

（2）：qmulti乘法模块

（3）：qdiv除法模块

（4）：xita\_tan\_lut 1/2^i部分正切值查找表

（5）：arctan 0-180°反正切求解模块

（6）：sqrt开方模块

（7）：arcsin反正弦模块

（8）：tan正切模块

（9）：cos余弦模块

（10）：xita\_to\_duty角度-占空比转换模块

2：pwm波模块（arm文件夹）

（1）：set\_duty占空比设置模块

（2）：push\_pwm pwm输出模块

（3）：pwm pwm顶层模块

3：机械臂顶层相关模块（arm文件夹）

（1）：inverse机械臂逆解运算模块

（2）：arm\_angle角度控制机械臂模块

（3）：arm\_model坐标逆解控制机械臂模块

二：滑台部分

1：滑台模块（table文件夹）

（1）：push\_pwm\_fre调频pwm波输出模块

（2）：pwm\_fre 滑台顶层模块

三：UART串口通信部分

1：UART发送模块

（1）：clkdiv分频模块

（2）：uarttx发送模块

2：UART接收模块

（1）：uartrx接收模块

（2）：uart\_asc\_num解码模块

3：UART顶层模块

（1）：uart\_top uart顶层模块

四：摄像头与Flash

1：ov7725模块（ov7725文件夹）

（1）：sccb sccb通信模块

（2）：ov7725\_cfg ov7725设置模块

（3）：ov7725\_data ov7725数据缓存模块

（4）：ov7725\_top ov7725顶层模块

2：flash模块（flash文件夹）

（1）：flash\_spi spi通信模块

（2）：rgb\_yuv rgb565转灰度模块

（3）：ov7725\_flash flash图片写入模块

（4）：flash\_top flash顶层模块

五：语音识别部分

1：LD3320模块

（1）：

（2）：

（3）：

（4）：

（5）：

（6）：

（7）：

（8）：

六：温湿度传感器及OLED屏幕

1：温湿度传感器模块（OLED\_sensor文件夹）

（1）：dht11温湿度数据获取模块

2：OLED通信及显示模块（OLED\_sensor文件夹）

（1）：iic iic通信模块

（2）：OLED\_Init OLED初始化模块

（3）：OLED\_Refresh OLED刷新模块

（4）：OLED\_NumData OLED数字字模模块

（5）：OLED\_FrontData OLED字模模块

（6）：OLED\_SelData OLED命令选择模块

（7）：OLED\_ShowData OLED数字显示模块

（8）：OLED\_ShowFront OLED字显示模块

（9）：OLED\_Top OLED显示顶层模块

（10）：OLED\_sensor OLED接收数据和显示顶层模块

七：各类传感器

1：热释电传感器

一：机械臂部分

1：运算模块（basic文件夹）

（1）：qadd加法模块

说明：小数加法器，32位数据，31位为符号位，30-16位为整数位，15-0位为小数位

输入：

[31:0]add1：32位加数1

[31:0]add2：32位加数2

输出：

[31:0]sum：32位的和

功能：根据二者符号位，进行该数据类型的加或者减

算法：单独提取出符号位，判断加减，而后将后30位进行运算，并根据大小和符号位确定结果符号位。

（2）：qmulti乘法模块

说明：乘法器，能分辨正负

输入：

[31:0]multi1：乘数1

[31:0]multi2：乘数2

输出：

[31:0]result：结果

功能：进行乘法运算

算法：同样单独提取出符号位，得出结果符号位。其它位相乘存入64位寄存器，结果取46-16位以实现小数点在16-15位之间。

（3）：qdiv除法模块

说明：定点数除法器，运算消耗约40个时钟

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

[31:0]dividend：被除数

[31:0]divisor：除数

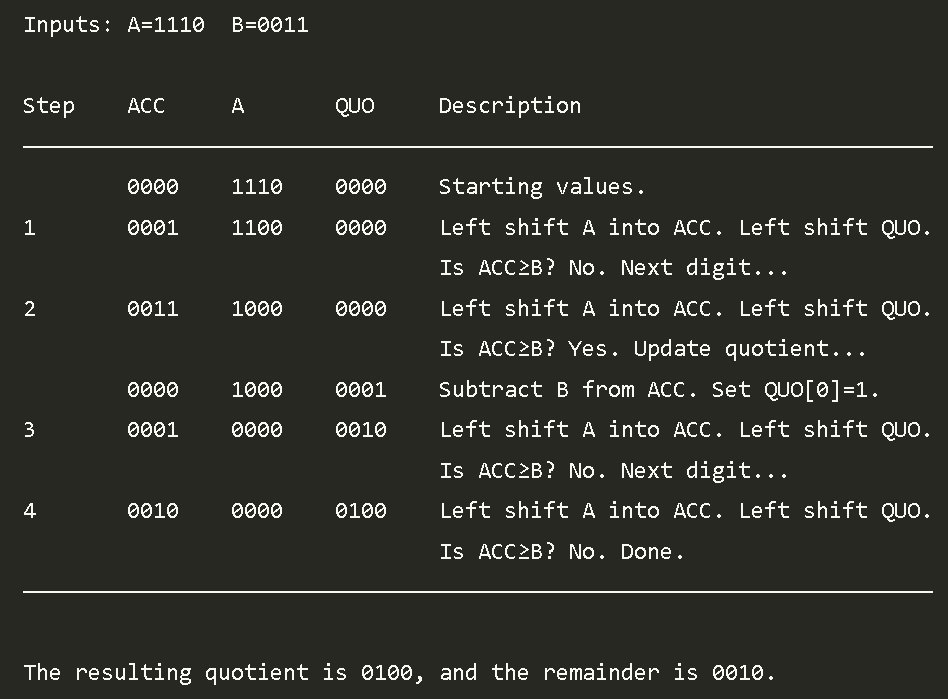
输出：

[31:0]quotient：商

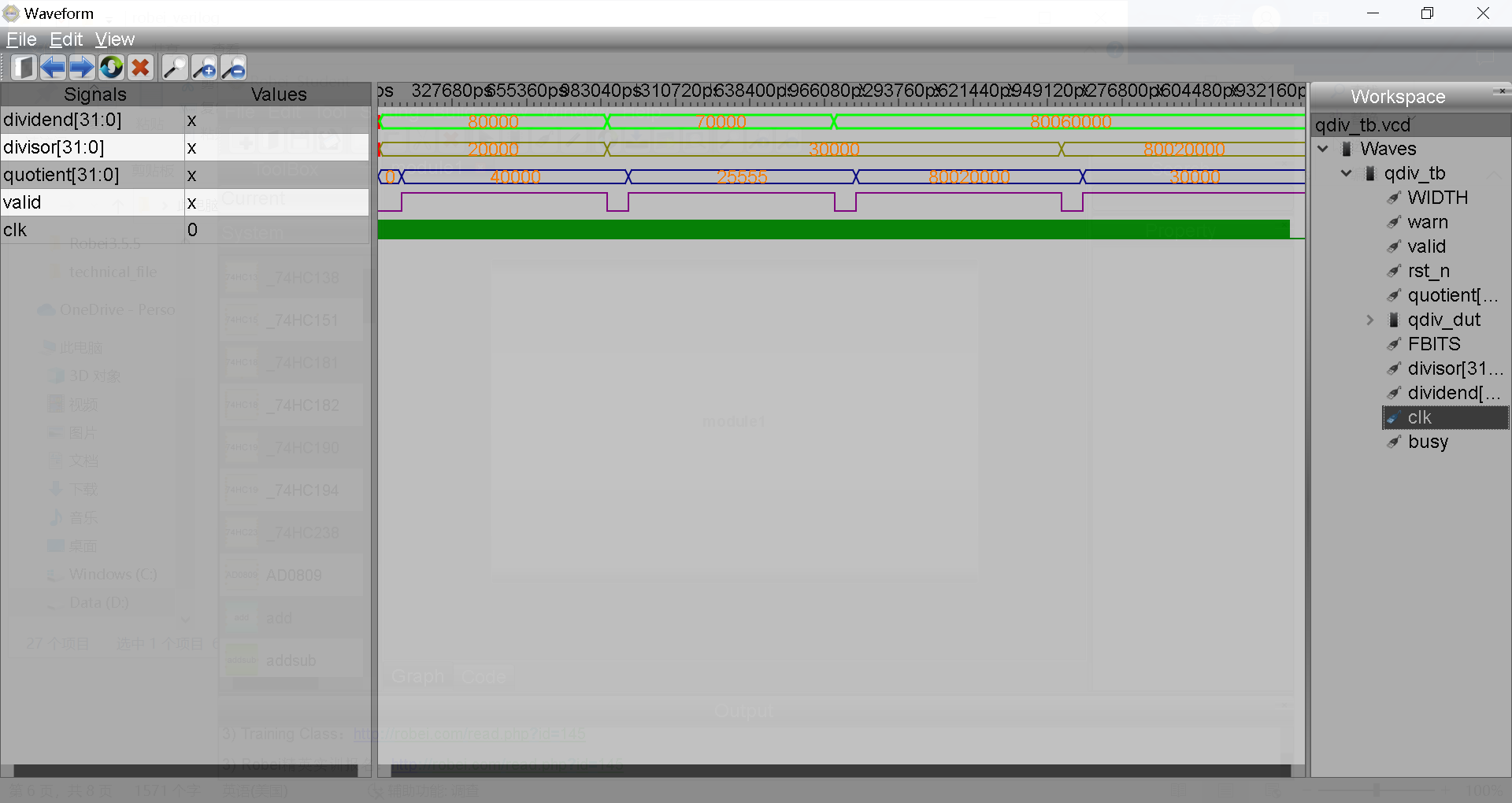
valid：输出有效信号，高电平表示运算完成

    warn：除数为零警告，高电平有效

    busy：运算中，高电平有效

算法：仍然是将符号位提取后单独判断。无符号数使用移位相减，将被除数一位一位移入累加器acc，商同步移位，当acc内数字大于除数时减去除数，商末尾置1，移被除数位数次，得出无符号数的商，再与符号位连接即可。运算完成则valid置1，若时钟沿读取到的被除数和除数与上一个时钟沿不同，则valid置0，busy置1，重新运算。运算过程见图。

仿真图：



（4）：xita\_tan\_lut 1/2^i部分正切值查找表

说明：输入1/2^i即tanθ，输出θ（角度值）

输入：

clk：50MHz系统时钟

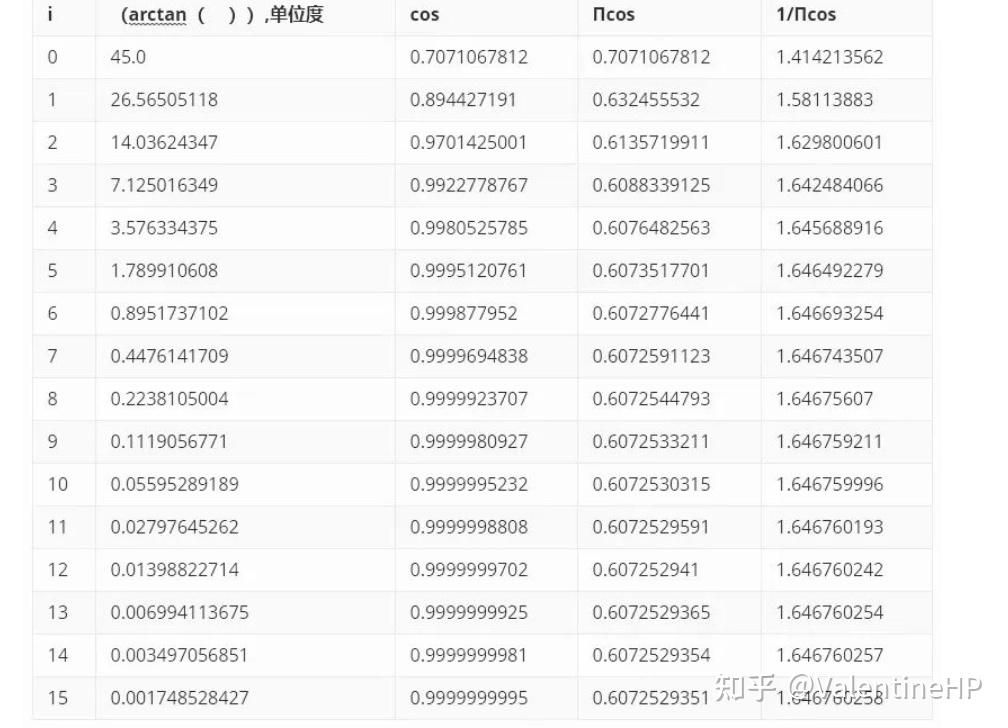
i：表示输入正切值1/2^i

输出：

[31:0]xita：正切值1/2^i对应的角度

算法：case(i)即可

查找表：



（5）：arctan 0-180°反正切求解模块

说明：可求0-180°反正切，输入变化时开始计算。运算消耗约900个时钟

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

[31:0]tan：正切值

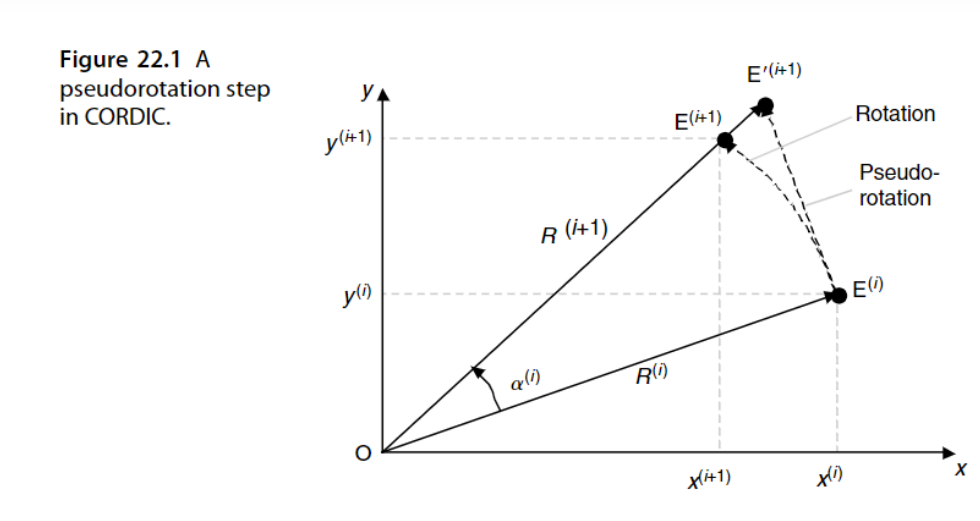
输出：

valid：输出有效（高电平）

[31:0]xita：角度

算法：使用cordic算法进行伪旋转迭代，每次旋转arctan(1/2^i)角度，x(n+1)=x(n)-d\*(y(n)>>i)，y(n+1)=y(n)+d\*(x(n)>>i)每次比较当前正切和输入正切，当前正切大则d=-1，反之d=1。由于伪旋转存在增益，旋转次数增加，增益趋近常数K=1.64676，故只需令初始x=1/K=0.60725，y=0即可，转化为32位定点数的格式再用十进制表示，也就是初始x=39796，y=0。本模块迭代了20次，精度符合要求。

伪旋转：



关键运算代码：

if(tan\_c > tan\_abs\_reg)

        begin

        x <= x+(y>>i);

          y <= y-(x>>i);

          z <= z-xita\_c;

        end

        else if(tan\_c < tan\_abs\_reg)

        begin

          x <= x-(y>>i);

          y <= y+(x>>i);

          z <= z+xita\_c;

        end

        else

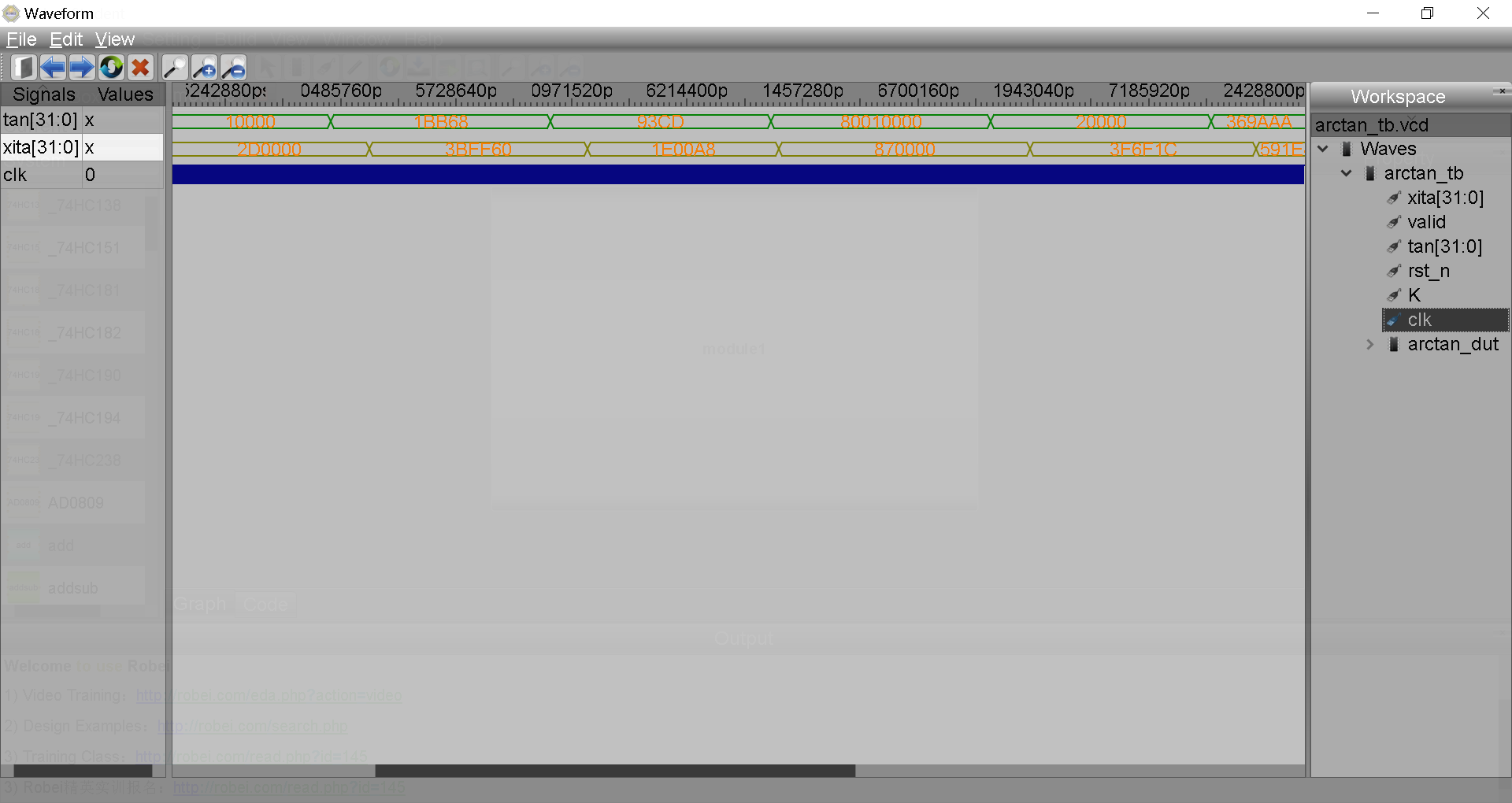
        begin

          x <= x;

          y <= y;

          z <= z;

        end

仿真：

（6）：sqrt开方模块

说明：逻辑类似，每次输入变化自动开始计算，算完给valid拉高，约消耗500个时钟

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

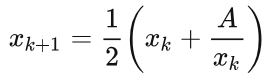
[31:0]sqrter：被开方的数

输出：

valid：输出有效信号（高电平有效）

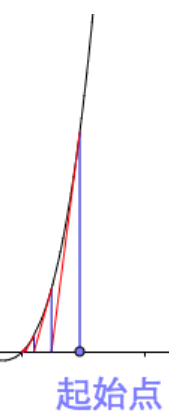
sqrted：运算结果

算法：使用牛顿迭代法：



A为被开方的数，起始x设为被开方数，而后迭代20次，得到精度足够的运算结果。

牛顿迭代法图示：



关键代码：

单次迭代：

qdiv

qdiv\_dut1 (

        .clk (clk ),

        .rst\_n (rst\_n ),

        .valid(valid\_dived),

        .busy(),

        .dividend (sqrter ),

        .divisor (iter ),

        .quotient (dived ),

        .warn  ( )

    );

    qadd

    qadd\_dut (

        .add1 (dived ),

        .add2 (iter ),

        .sum (sum )

);

迭代20次：

sqrter\_reg <= sqrter;

            if(sqrter\_reg != sqrter)

            begin

                i <= 0;

                iter\_reg <= sqrter;

                valid <= 0;

            end

            else if(i == 21)

            begin

                sqrted <= iter;

                valid <= 1;

            end

            else

            begin

                if(valid\_dived)

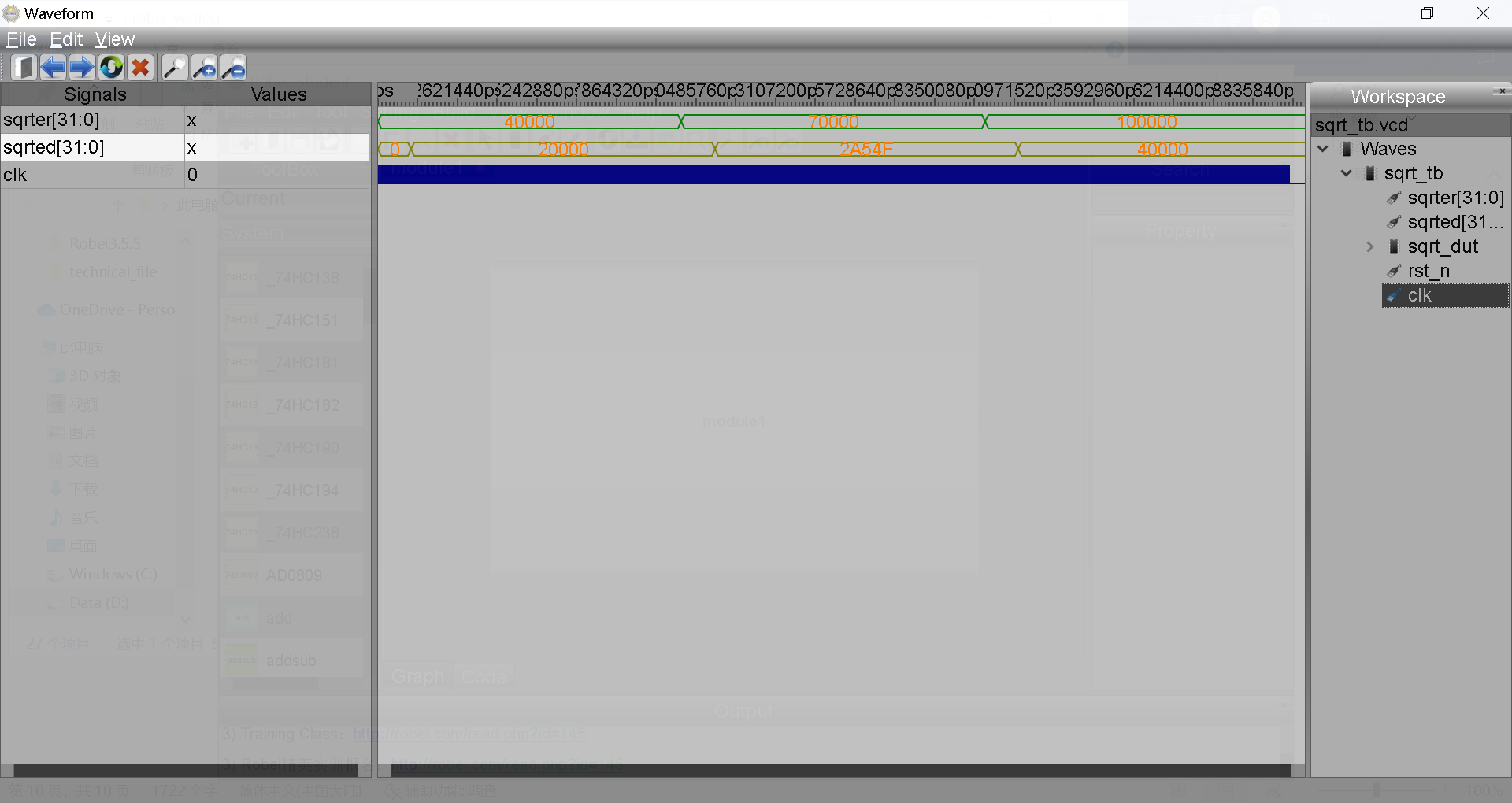
                begin

                    iter\_reg <= iter\_next;

                    i <= i+1;

                end

            end

仿真：

（7）：arcsin反正弦模块

说明：求反正弦，输入变化时开始运算，运算完成valid拉高。消耗约1400个时钟。

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

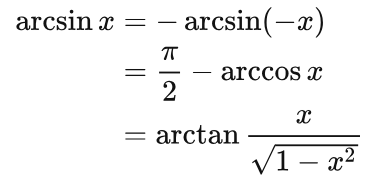
[31:0]value\_sin正弦值

输出：

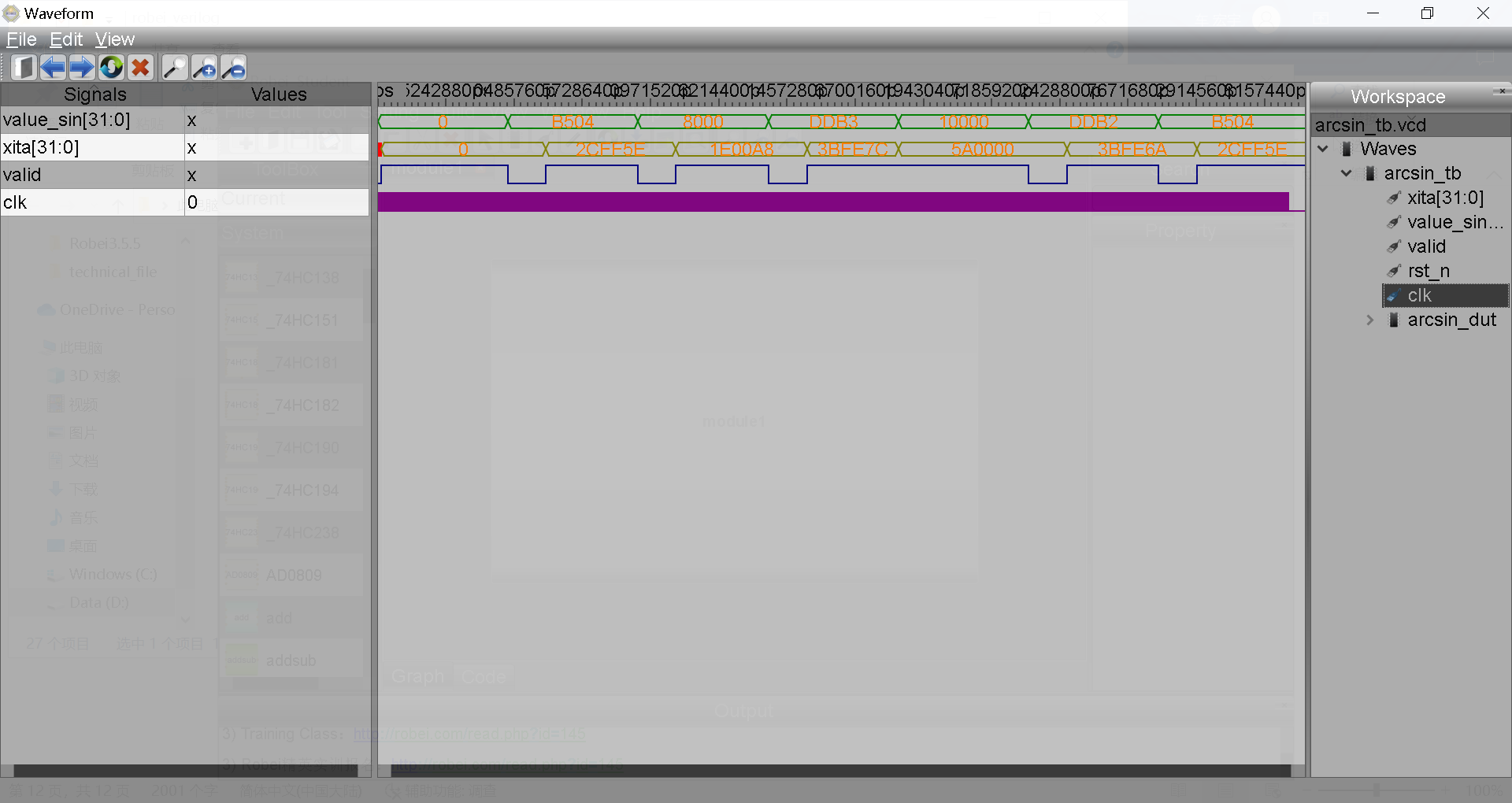
valid：输出有效（高电平）

[31:0]xita：角度

算法：



使用这个公式，运用前面的反正切、除法、乘法、开方、加法等模块，每次前一模块valid拉高时后一模块输入赋值为前一模块输出，算出反正弦。由于90°角对应的分母为0，故正弦值为1时单独处理，直接将结果置为 90°。

仿真：

（8）：tan正切模块

说明：求正切，输入角度变化时开始运算，运算完成拉高valid。消耗约800个时钟

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

[31:0]xita：角度

输出：

valid：输出有效（高电平）

tan：求得的正切值

算法：同样是使用cordic算法伪旋转，只不过比的是角度，输出的是y/x正切。

关键代码：

if(z<xita)

              begin

              z<=z+xita\_c;

                x<=x-(y>>i);

                y<=y+(x>>i);

              end

              else if(z>xita)

              begin

                z<=z-xita\_c;

                x<=x+(y>>i);

                 y<=y-(x>>i);

              end

              else

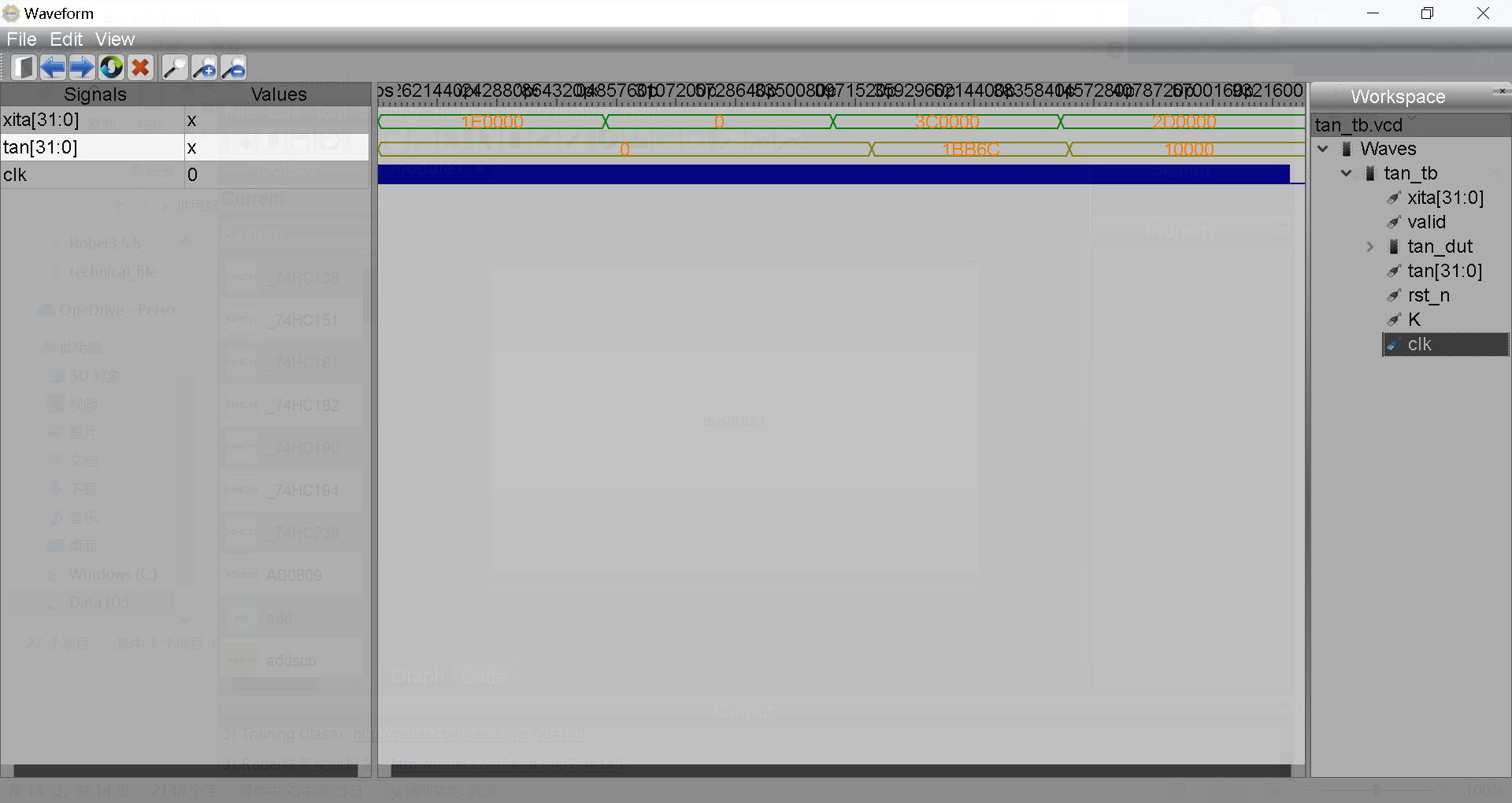
              begin

              z<=z;

                 x<=x;

                y<=y;

              end

仿真：

（9）：cos余弦模块

说明：输入32位角度值表示的角，运算完成后valid拉高，输出余弦值。消耗约1400时钟。

输入：

clk：50MHz系统时钟

rst\_n：低电平复位信号

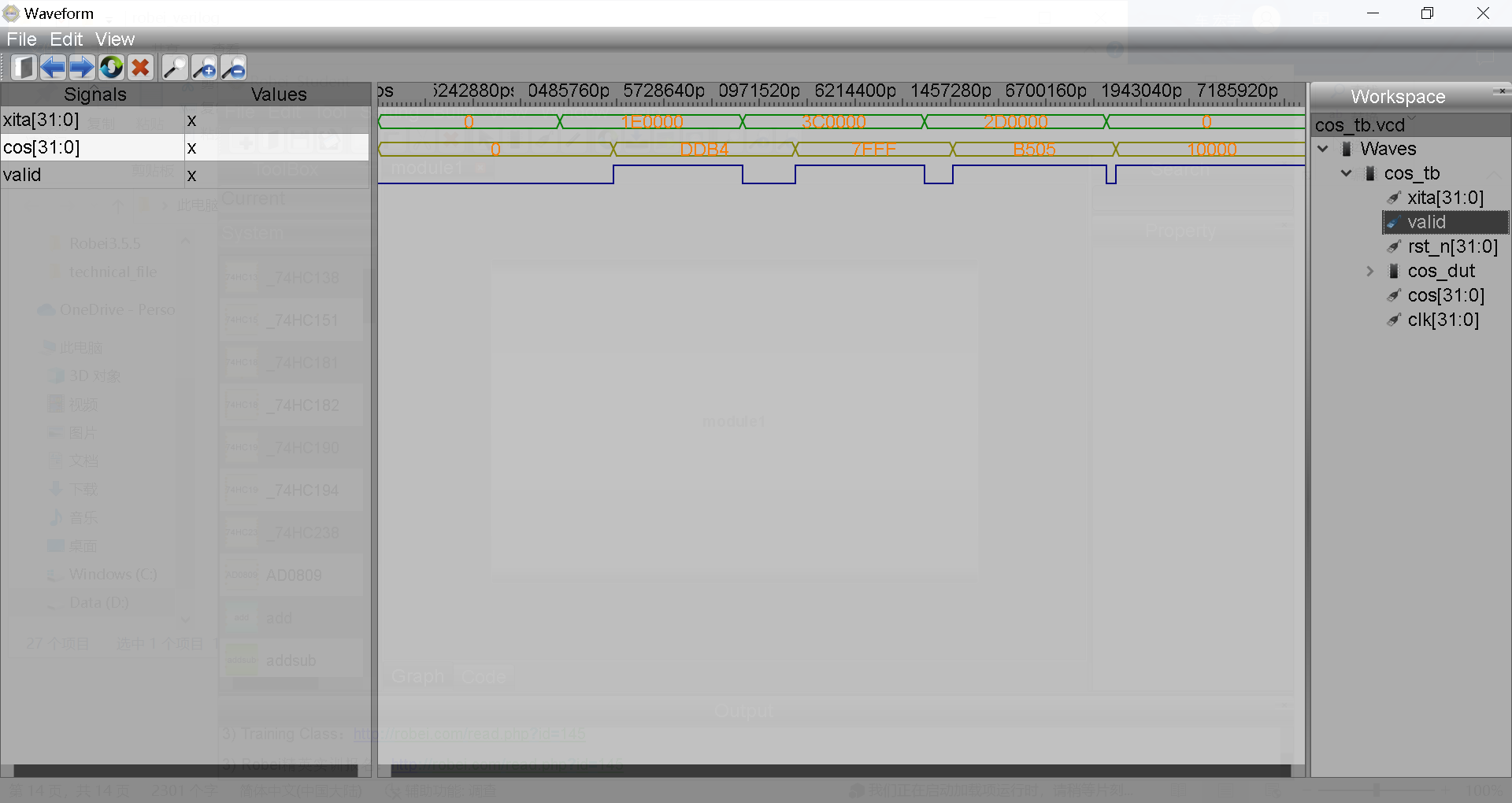
[31:0]xita：32位定点角度值

输出：

[31:0]cos：余弦值

valid：输出有效信号（高电平）

算法：，使用此前的乘法、开方、除法、正切模块进行计算，前一模块运算valid上升沿使得下一模块输入变为上一模块输出。

仿真：

（10）：xita\_to\_duty角度-占空比转换模块

说明：将角度转换为舵机pwm的占空比值

输入：

[31:0]xita：32位定点数角度

输出：

[19:0]duty：20位占空比duty

算法：简单的运算和上下限判断

2：pwm波模块（arm文件夹）

（1）：set\_duty占空比设置模块

说明：占空比输入缓冲器，设置占空比时能够让输出占空比缓慢变化至所需值

输入：

clk：输入时钟50MHz

rst\_n：复位信号，低电平有效

[19:0]duty\_need：需要达到的高电平时长（50\_000代表1ms）（上限1\_000\_000即20ms）

[11:0]duty\_gap：占空比变化1所用的时钟数（1000代表转180度用2秒）

输出：

[19:0]duty\_out：缓慢变化的占空比值

算法：每过duty\_gap次时钟，duty向需求值加或减1

关键代码：

if(duty\_out<duty\_need) begin

                if(count==duty\_gap-1) begin

                    count<=0;

                    duty\_out<=duty\_out+1;

                end

                else begin

                    count<=count+1;

                end

            end

            else if (duty\_out>duty\_need) begin

                if(count==duty\_gap-1) begin

                    count<=0;

                    duty\_out<=duty\_out-1;

                end

                else begin

                    count<=count+1;

                end

            end

（2）：push\_pwm pwm输出模块

说明：pwm波输出器，根据给定的duty立刻输出pwm波

输入：

clk：50MHz时钟

rst\_n：复位信号，低电平有效

duty：高电平持续时间（50\_000代表1ms）（上限1\_000\_000即20ms）

输出：

pwm\_wave：周期20ms的pwm波

关键代码：

//if(count==19)//仅用于仿真

            if(count==999\_999)//20ms

            begin

                count<=0;

            end

            else begin

                count<=count+1;

            end

            if(count<duty)//duty输入50\_000时，高电平1ms

            begin

                pwm\_wave<=1'b1;

            end

            else begin

                pwm\_wave<=1'b0;

            end

（3）：pwm pwm顶层模块

说明：根据给定的duty\_need输出逐渐变化到需求值的pwm波

输入：

clk：时钟50MHz

rst\_n：复位信号，低电平有效

[19:0]duty\_need：需要的高电平时长（50\_000代表1ms）（上限1\_000\_000即20ms），

[11:0]duty\_gap：加一的间隔时钟沿次数（1000代表转180度用2秒）

输出：

pwm\_out：逐渐变化到需求的高电平时长的pwm波

算法：将前两个模块连起来即可

3：机械臂顶层相关模块（arm文件夹）

（1）：inverse机械臂逆解运算模块

说明：逆运动学求解舵机角度，根据输入的x，y坐标，计算出舵机1和舵机2的角度。约消耗2200个时钟

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

[31:0]x：距机械臂水平距离（mm）

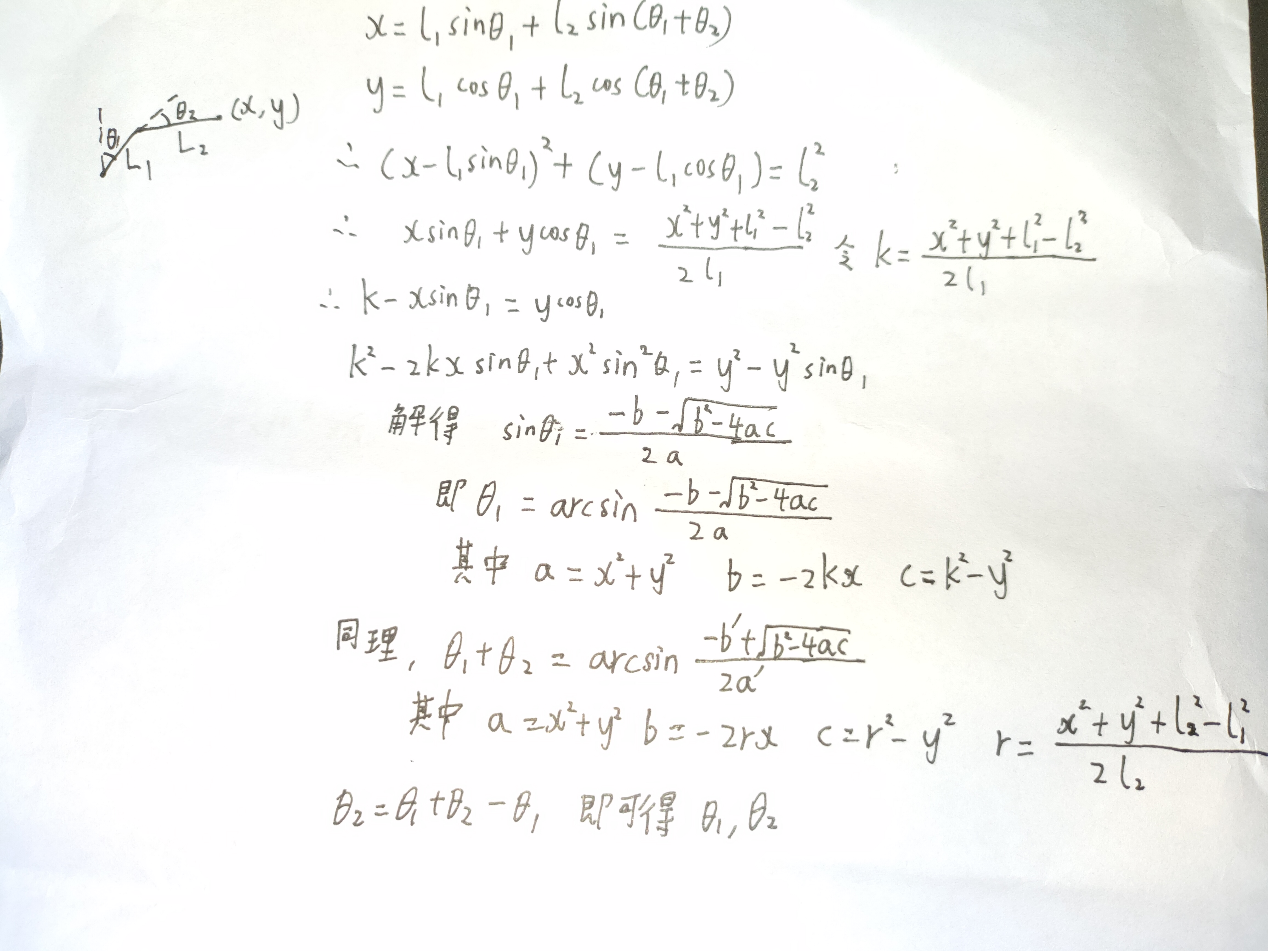
[31:0]y：距关节1垂直距离（mm）

输出：

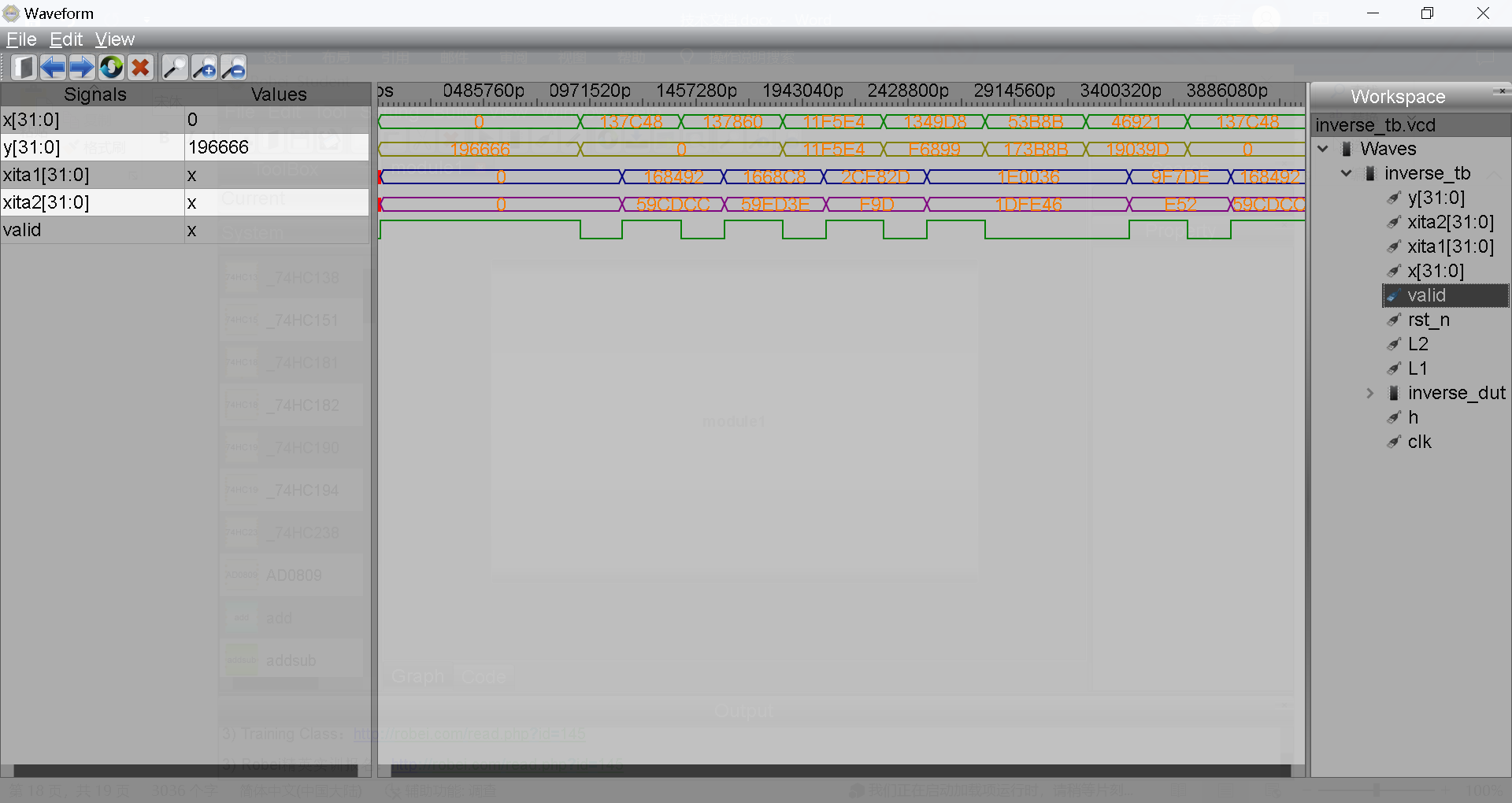
[31:0]xita1：舵机1角度

[31:0]xita2：舵机2角度

valid：输出有效

算法：逆解三自由度机械臂，本项目使用的机械臂L1=7.4cm,L2=18cm，关节1距地面h=5.2cm

解出算式后，使用前面逆解模块的运算模块，每次前一模块valid上升时下一模块输入变化开始运算，后一模块输入为前一模块输出。

仿真：

（2）：arm\_angle角度控制机械臂模块

说明：根据输入的角度控制舵机转动到相应角度，夹取信号为0时夹手，为1时松手

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

[31:0] xita1：舵机1角度

[31:0] xita2：舵机2角度

catch：夹手信号

输出：

pwm1：舵机1pwm波

pwm2：舵机2pwm波

pwm\_catch：夹手pwm波

算法：调用前面所写的pwm模块和xita\_duty模块，将输入转换为duty后驱动输出占空比逐渐变化到需求值的pwm波。夹取信号为1时夹取舵机pwm占空比设为25\_000，否则设为125\_000

（3）：arm\_model坐标逆解控制机械臂模块

说明：机械臂顶层模块，将坐标和夹取信号转换为输出的pwm波。en1为1时，根据输入的x，y坐标，计算出舵机1和舵机2的角度，输出pwm给舵机模块，若en2为1，则舵机转到置位角度。若catch为0，则夹取，否则放开。

输入：

clk：50MHz系统时钟

rst\_n：低电平复位

[31:0]x：相对于关节1x坐标

[31:0]y：相对于关节1y坐标

en1：坐标逆解使能

en2：角度置位使能

[31:0]set\_xita1：舵机1角度置位值

[31:0]set\_xita2：舵机1角度置位值

catch：夹取信号，0夹1松

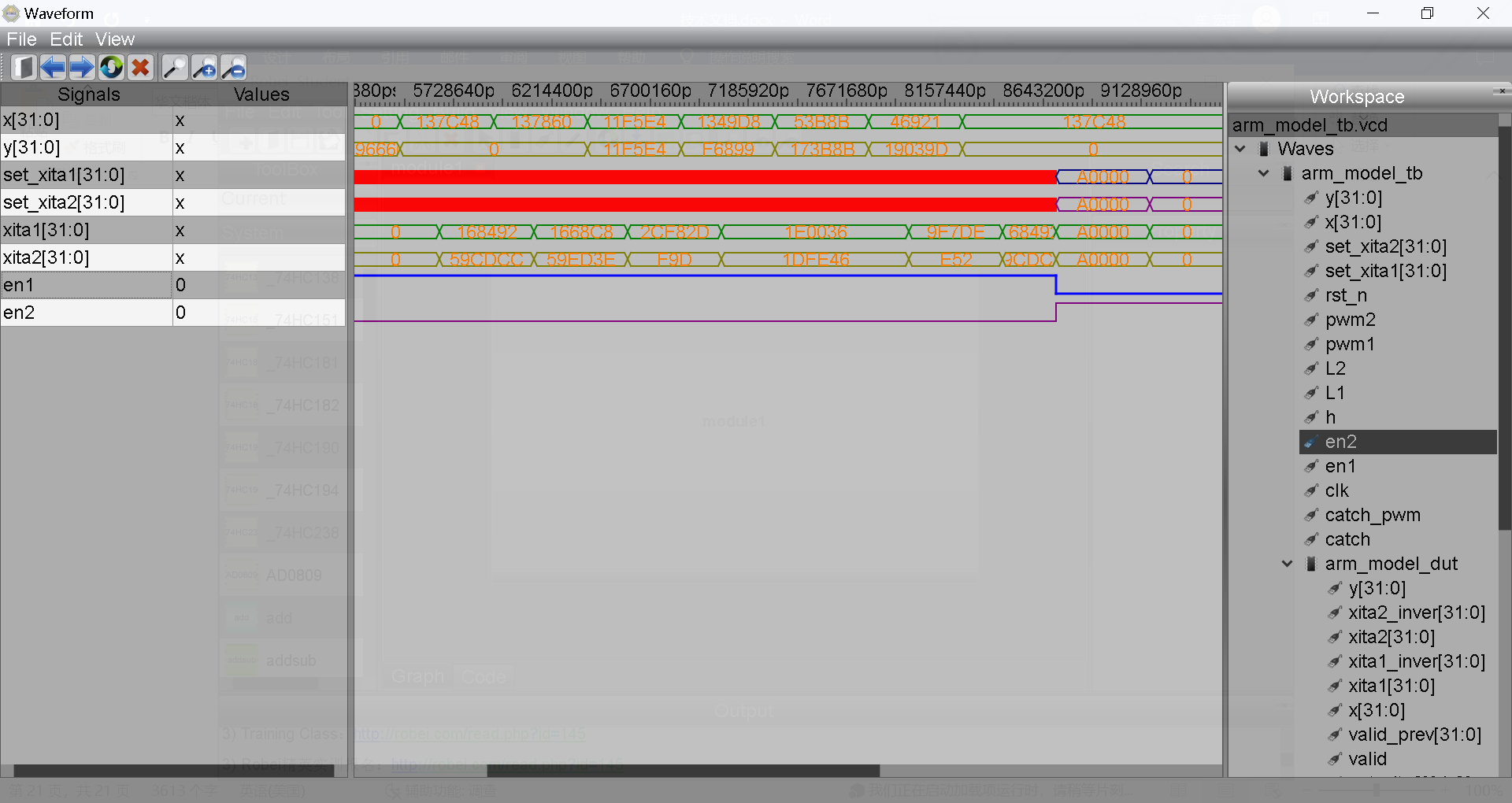
输出：

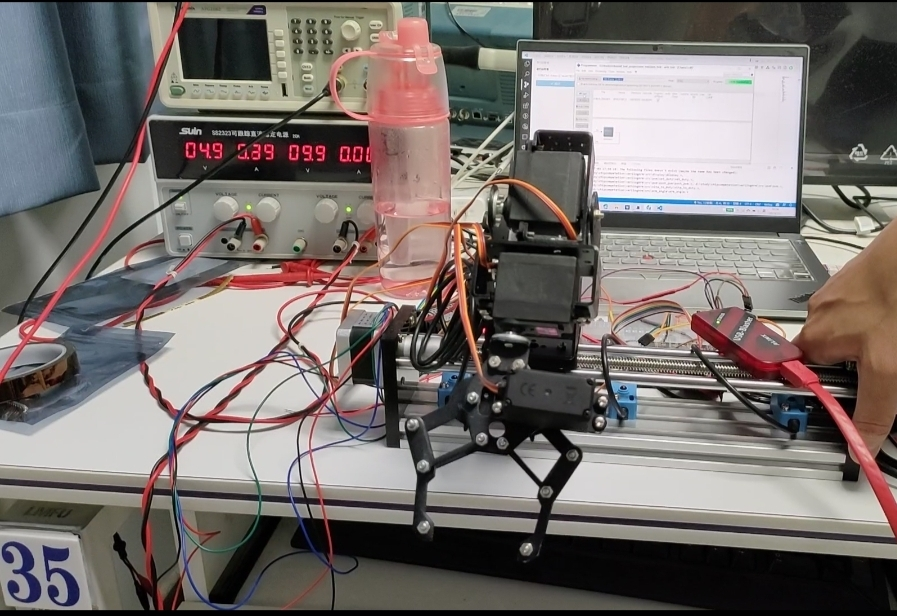
pwm1：控制舵机1的pwm波

pwm2：控制舵机2的pwm波

catch\_pwm：控制夹取舵机的pwm波

算法：将逆解模块与角度控制机械臂模块相连，en1高电平时坐标输入逆解模块，输出逆解角度输入角度控制机械臂模块，en2高电平时set\_xita1和set\_xita2输入角度控制机械臂模块。

仿真：

实物测试下，输入x=1277000，y=0（32位定点数表示下十进制的值，即x=19.49cm），机械臂运动正确：

二：滑台部分

1：滑台模块（table文件夹）

（1）：push\_pwm\_fre调频pwm波输出模块

（2）：pwm\_fre 滑台顶层模块

三：UART串口通信部分

1：UART发送模块

（1）：clkdiv分频模块

说明：分频模块将50MHz的系统时钟326分频以实现uart通信（波特率9600，每个比特16个时钟，故需要时钟频率9600\*16）

输入：

clk50：50MHz系统时钟

rst\_n：低电平复位信号

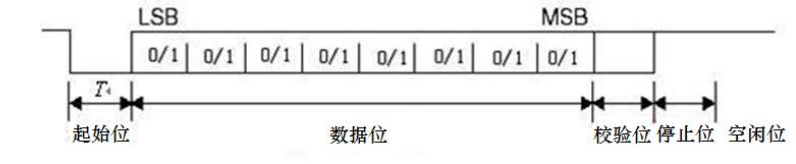
输出：

clkout：分频后的uart时钟

算法：偶数分频，简单的计数器分频即可

（2）：uarttx发送模块

说明：平时tx高电平，每16个clock发送一个 bit。每个发送命令发送一个起始位（tx拉低）,8 个数据位,一个校验位（奇偶校验，自动生成）,一个停止位（tx拉高）。



输入：

clk：uart时钟

rst\_n：低电平复位

[7:0]datain：需要发送的8位数据

wrsig：上升沿触发的发送起始信号

输出：

idle：发送状态，高电平为发送中

tx：数据发送线

算法：发送端，数据应当在每个16位的数据位开始时写入，即0，16，32等时刻，改变tx的值。平时tx高电平，拉低起始，拉高结束。

关键代码：

case(cnt) //产生起始位

                8'd0: begin

                    tx <= 1'b0;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd16: begin

                    tx <= datain[0]; //发送数据 0 位

                    presult <= datain[0]^paritymode;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd32: begin

                    tx <= datain[1]; //发送数据 1 位

                    presult <= datain[1]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd48: begin

                    tx <= datain[2]; //发送数据 2 位

                    presult <= datain[2]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd64: begin

                    tx <= datain[3]; //发送数据 3 位

                    presult <= datain[3]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd80: begin

                    tx <= datain[4]; //发送数据 4 位

                    presult <= datain[4]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd96: begin

                    tx <= datain[5]; //发送数据 5 位

                    presult <= datain[5]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd112: begin

                    tx <= datain[6]; //发送数据 6 位

                    presult <= datain[6]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd128: begin

                    tx <= datain[7]; //发送数据 7 位

                    presult <= datain[7]^presult;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd144: begin

                    tx <= presult; //发送奇偶校验位

                    presult <= datain[0]^paritymode;

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd160: begin

                    tx <= 1'b1; //发送停止位

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                end

                8'd168: begin

                    tx <= 1'b1;

                    idle <= 1'b0; //一帧数据发送结束

                    cnt <= cnt + 8'd1;

                end

                default: begin

                    cnt <= cnt + 8'd1;

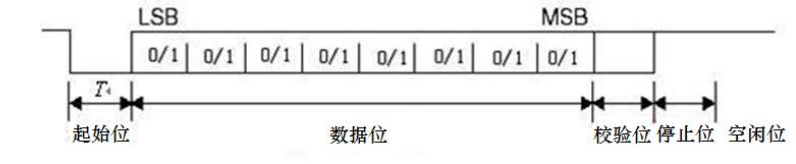
                end

            endcase

2：UART接收模块

（1）：uartrx接收模块

说明：接收uart模式发送的数据



输入：

clk：uart时钟

rst\_n：低电平复位

rx：接收数据线

输出：

[7:0]dataout：接收到的数据

rdsig：接收到8位数据标志输出（上升沿）

dataerror：数据出错指示（高电平）

frameerror：帧出错指示（高电平）

算法：根据uart通信协议，接收端应当在每个数据位的第8个时钟采集，即第24，40，56等时钟位记录数据。

关键代码：

case (cnt)

                8'd0: begin

                    idle <= 1'b1;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd24: begin //接收第 0 位数据

                    idle <= 1'b1;

                    dataout[0] <= rx;

                    presult <= paritymode^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd40: begin //接收第 1 位数据

                    idle <= 1'b1;

                    dataout[1] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd56: begin //接收第 2 位数据

                    idle <= 1'b1;

                    dataout[2] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd72: begin //接收第 3 位数据

                    idle <= 1'b1;

                    dataout[3] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd88: begin //接收第 4 位数据

                    idle <= 1'b1;

                    dataout[4] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd104: begin //接收第 5 位数据

                    idle <= 1'b1;

                    dataout[5] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd120: begin //接收第 6 位数据

                    idle <= 1'b1;

                    dataout[6] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b0;

                end

                8'd136: begin //接收第 7 位数据

                    idle <= 1'b1;

                    dataout[7] <= rx;

                    presult <= presult^rx;

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b1;

                end

                8'd152: begin //接收奇偶校验位

                    idle <= 1'b1;

                    if(presult == rx)

                        dataerror <= 1'b0;

                    else

                        dataerror <= 1'b1; //如果奇偶校验位不对，表示数据出错

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b1;

                end

                8'd168: begin

                    idle <= 1'b1;

                    if(1'b1 == rx)

                        frameerror <= 1'b0;

                    else

                        frameerror <= 1'b1; //如果没有接收到停止位，表示帧出错

                    cnt <= cnt + 8'd1;

                    rdsig <= 1'b1;

                end

                default: begin

                    cnt <= cnt + 8'd1;

                end

            endcase

（2）：uart\_asc\_num解码模块

说明：将uart接收到的ascii码转换为32位定点数，接收8次asc码，输出32位定点数，共接收24次，输出x，y，z

输入：

clk：uart时钟

rst\_n：低电平复位

[7:0]asc：8位ascii码

start：上升沿启动信号

dataerror：数据错误信号

frameerror：帧错误信号

clr：清零（高电平）

输出：

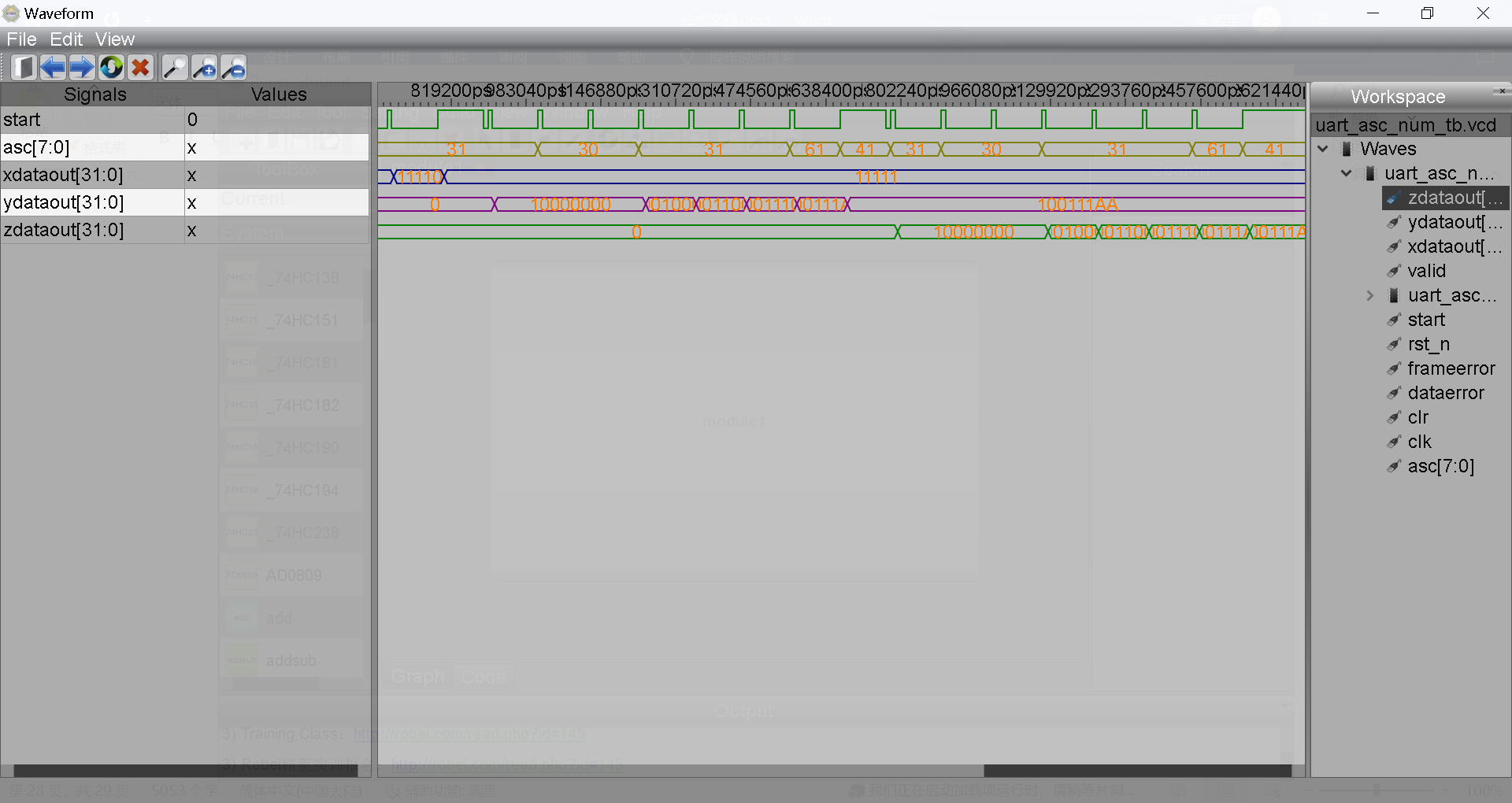
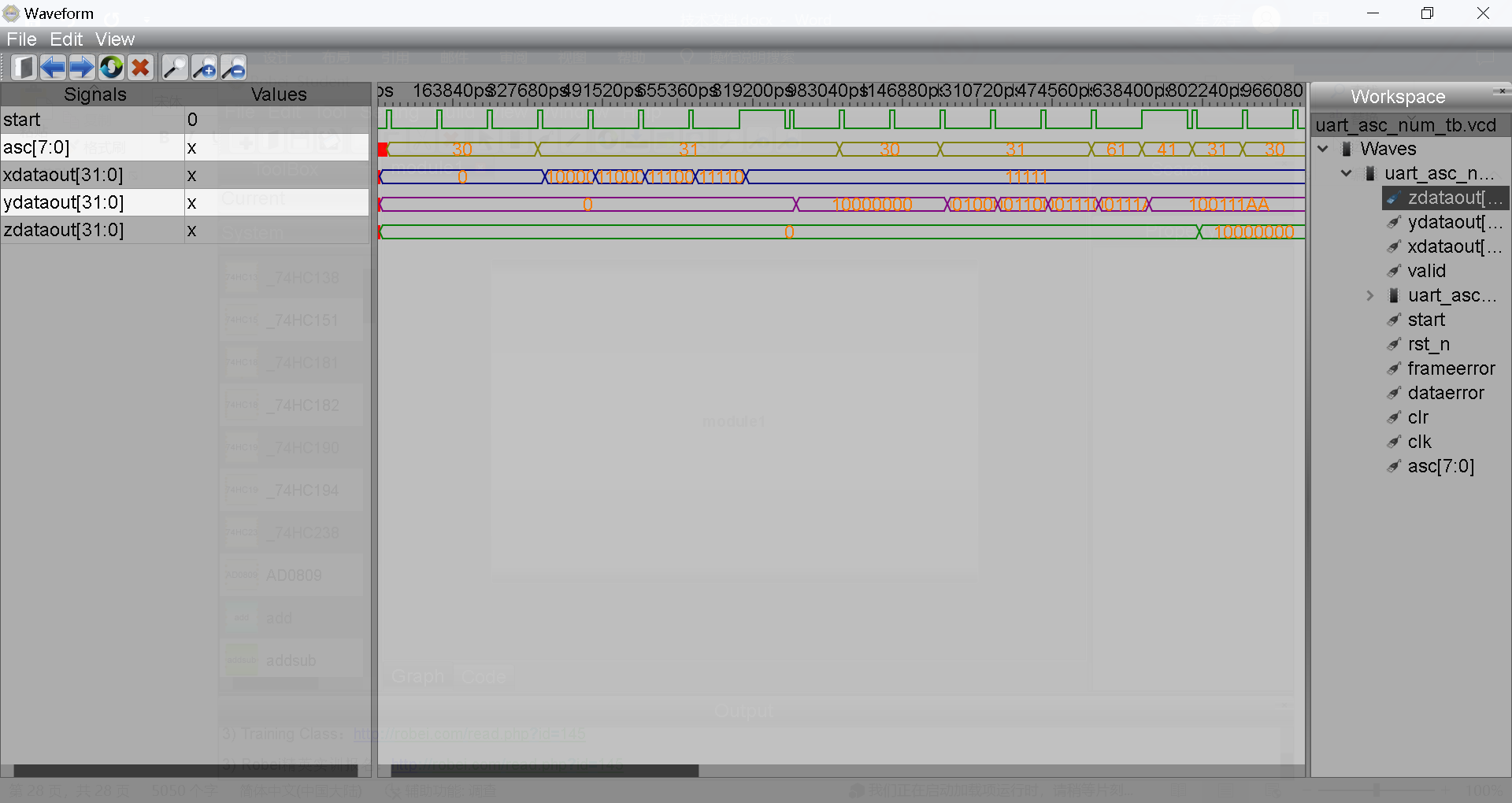
[31:0]xdataout：32位定点数输出x

[31:0]ydataout：32位定点数输出y

[31:0]zdataout：32位定点数输出z

valid：完成信号（高电平）

算法：使用接收模块的接收完成信号，检测其上升沿，每次上升沿填充4位，逐次填充直至xyz全被填满，valid拉高。

仿真：

3：UART顶层模块

（1）：uart\_top uart顶层模块

说明：将串口uart输入ascii码的数据转换为x,y,z分别32位坐标输出,并通过串口uart发送回去作为调试信息

输入：

clk50：50MHz系统时钟

rst\_n：低电平复位

rx：接收数据线

clr：清零位（高电平）

输出：

[31:0]x：uart接收到的x坐标

[31:0]y：uart接收到的y坐标

[31:0]z：uart接收到的z坐标

valid：完成信号（高电平）

算法：当接收模块转换全部完成，valid高电平时，不断拉高拉低发送模块的发送信号，逐8位发送接收存储的xyz，从而实现反馈，方便调试。

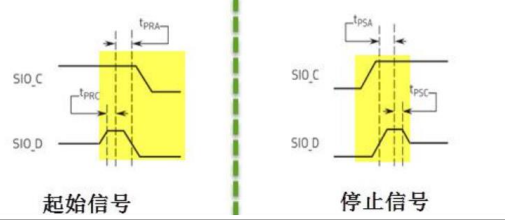
实际运行：

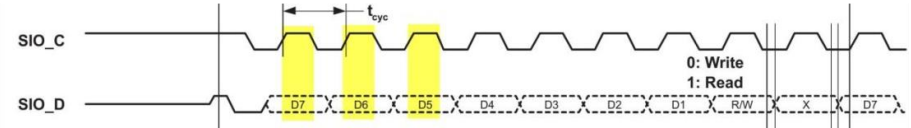
四：摄像头与Flash

1：ov7725模块（ov7725文件夹）

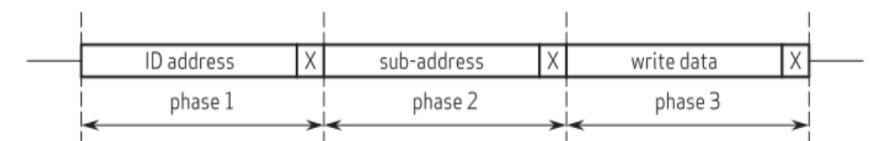
（1）：sccb sccb通信模块

说明：向摄像头发送配置寄存器信号, sccb和i2c相似，此处使用的特性均适合i2c。写数据传输三个数据，分别为主设备id、地址+方向和数据，中间以X分隔（0或1）。读数据写入两个数据，主设备id和地址+方向，再读取两个数据，从设备id和数据，同样X（0或1）分隔。

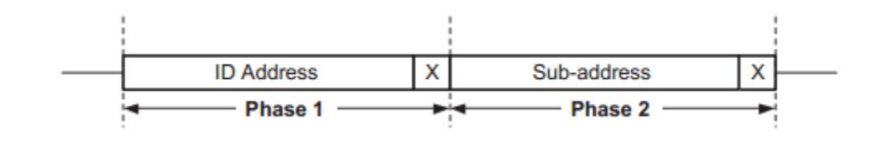


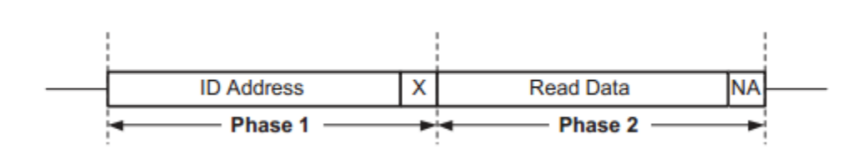


发送数据，写入三次：



接收数据，先写入两次，后接收两次：





输入：

sys\_clk：系统时钟50MHz

sys\_rst\_n：低电平复位

wr\_en：输入写使能信号

rd\_en：输入读使能信号

i2c\_start：输入触发信号

addr\_num：输入地址字节数

[15:0]byte\_addr：字节地址

[7:0]wr\_data：输入设备数据

输出：

i2c\_clk：工作时钟

i2c\_end：一次读写完成的信号

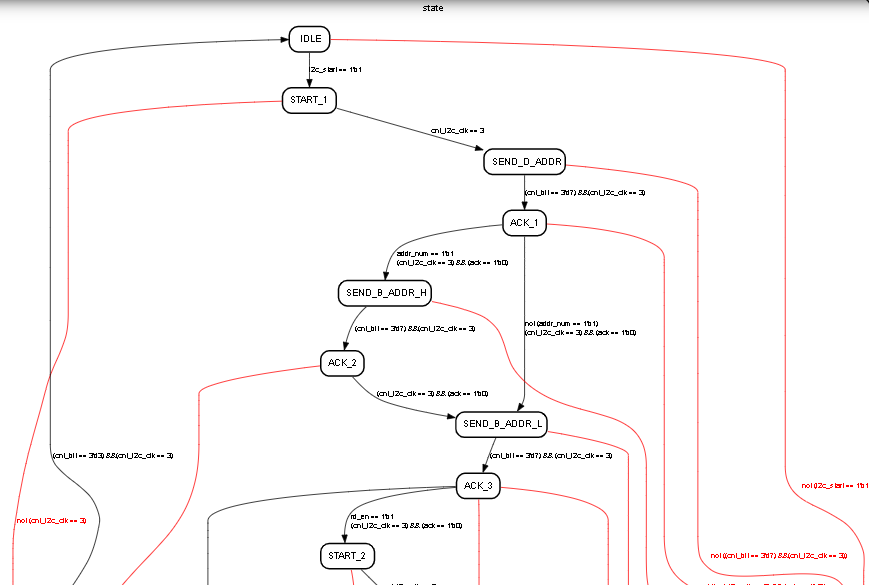
[7:0]rd\_data：读取到的数据

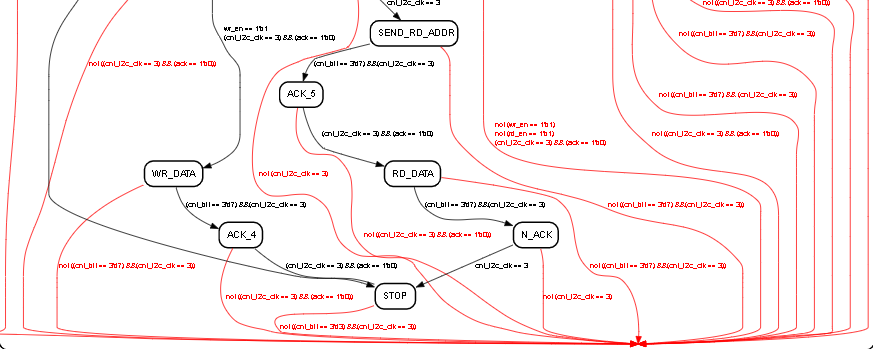
i2c\_scl：输出至设备的串行时钟线

inout：

i2c\_sda：串行数据线

算法：有限状态机





使用状态机，控制时序，较为麻烦。

（2）：ov7725\_cfg ov7725设置模块

说明：内部储存了ov7725内部寄存器配置相关数据，通过sccb模块发给ov7725配置摄像头。

输入：

sys\_clk：50MHz系统时钟

sys\_rst\_n：低电平系统复位

cfg\_end：单个寄存器配置完成信号（ov7725反馈回来的）

输出：

cfg\_start：单个寄存器配置触发信号

[15:0]cfg\_data：寄存器配置数据

cfg\_done：寄存器配置完成信号

算法：即每次接收到上单个寄存器配置完成的信号，触发下单个寄存器配置信号，通过计数器每次发送对应位寄存器应设为的值

关键代码：

//cfg\_start:单个寄存器配置触发信号

    always@(posedge sys\_clk or negedge sys\_rst\_n) begin

        if(sys\_rst\_n == 1'b0)

            cfg\_start <= 1'b0;

        else if(cnt\_wait == (CNT\_WAIT\_MAX - 1'b1))

            cfg\_start <= 1'b1;

        else if((cfg\_end == 1'b1) && (reg\_num < REG\_NUM))

            cfg\_start <= 1'b1;

        else

            cfg\_start <= 1'b0;

end

//cfg\_data\_reg：寄存器配置数据暂存 ID REG\_ADDR REG\_VAL

assign cfg\_data\_reg[00] = {8'h3d, 8'h03};

    assign cfg\_data\_reg[01] = {8'h15, 8'h00};

    assign cfg\_data\_reg[02] = {8'h17, 8'h23};

    assign cfg\_data\_reg[03] = {8'h18, 8'ha0};

    assign cfg\_data\_reg[04] = {8'h19, 8'h07};

    assign cfg\_data\_reg[05] = {8'h1a, 8'hf0};

    assign cfg\_data\_reg[06] = {8'h32, 8'h00};

    assign cfg\_data\_reg[07] = {8'h29, 8'ha0};

    assign cfg\_data\_reg[08] = {8'h2a, 8'h00};

    assign cfg\_data\_reg[09] = {8'h2b, 8'h00};

    assign cfg\_data\_reg[10] = {8'h2c, 8'hf0};

    assign cfg\_data\_reg[11] = {8'h0d, 8'h41};

    assign cfg\_data\_reg[12] = {8'h11, 8'h00};

    assign cfg\_data\_reg[13] = {8'h12, 8'h06};

    assign cfg\_data\_reg[14] = {8'h0c, 8'hd0};

    assign cfg\_data\_reg[15] = {8'h42, 8'h7f};

    assign cfg\_data\_reg[16] = {8'h4d, 8'h09};

    assign cfg\_data\_reg[17] = {8'h63, 8'hf0};

    assign cfg\_data\_reg[18] = {8'h64, 8'hff};

    assign cfg\_data\_reg[19] = {8'h65, 8'h00};

    assign cfg\_data\_reg[20] = {8'h66, 8'h00};

    assign cfg\_data\_reg[21] = {8'h67, 8'h00};

    assign cfg\_data\_reg[22] = {8'h13, 8'hff};

    assign cfg\_data\_reg[23] = {8'h0f, 8'hc5};

    assign cfg\_data\_reg[24] = {8'h14, 8'h11};

    assign cfg\_data\_reg[25] = {8'h22, 8'h98};

    assign cfg\_data\_reg[26] = {8'h23, 8'h03};

    assign cfg\_data\_reg[27] = {8'h24, 8'h40};

    assign cfg\_data\_reg[28] = {8'h25, 8'h30};

    assign cfg\_data\_reg[29] = {8'h26, 8'ha1};

    assign cfg\_data\_reg[30] = {8'h6b, 8'haa};

    assign cfg\_data\_reg[31] = {8'h13, 8'hff};

    assign cfg\_data\_reg[32] = {8'h90, 8'h0a};

    assign cfg\_data\_reg[33] = {8'h91, 8'h01};

    assign cfg\_data\_reg[34] = {8'h92, 8'h01};

    assign cfg\_data\_reg[35] = {8'h93, 8'h01};

    assign cfg\_data\_reg[36] = {8'h94, 8'h5f};

    assign cfg\_data\_reg[37] = {8'h95, 8'h53};

    assign cfg\_data\_reg[38] = {8'h96, 8'h11};

    assign cfg\_data\_reg[39] = {8'h97, 8'h1a};

    assign cfg\_data\_reg[40] = {8'h98, 8'h3d};

    assign cfg\_data\_reg[41] = {8'h99, 8'h5a};

    assign cfg\_data\_reg[42] = {8'h9a, 8'h1e};

    assign cfg\_data\_reg[43] = {8'h9b, 8'h3f};

    assign cfg\_data\_reg[44] = {8'h9c, 8'h25};

    assign cfg\_data\_reg[45] = {8'h9e, 8'h81};

    assign cfg\_data\_reg[46] = {8'ha6, 8'h06};

    assign cfg\_data\_reg[47] = {8'ha7, 8'h65};

    assign cfg\_data\_reg[48] = {8'ha8, 8'h65};

    assign cfg\_data\_reg[49] = {8'ha9, 8'h80};

    assign cfg\_data\_reg[50] = {8'haa, 8'h80};

    assign cfg\_data\_reg[51] = {8'h7e, 8'h0c};

    assign cfg\_data\_reg[52] = {8'h7f, 8'h16};

    assign cfg\_data\_reg[53] = {8'h80, 8'h2a};

    assign cfg\_data\_reg[54] = {8'h81, 8'h4e};

    assign cfg\_data\_reg[55] = {8'h82, 8'h61};

    assign cfg\_data\_reg[56] = {8'h83, 8'h6f};

    assign cfg\_data\_reg[57] = {8'h84, 8'h7b};

    assign cfg\_data\_reg[58] = {8'h85, 8'h86};

    assign cfg\_data\_reg[59] = {8'h86, 8'h8e};

    assign cfg\_data\_reg[60] = {8'h87, 8'h97};

    assign cfg\_data\_reg[61] = {8'h88, 8'ha4};

    assign cfg\_data\_reg[62] = {8'h89, 8'haf};

    assign cfg\_data\_reg[63] = {8'h8a, 8'hc5};

    assign cfg\_data\_reg[64] = {8'h8b, 8'hd7};

    assign cfg\_data\_reg[65] = {8'h8c, 8'he8};

    assign cfg\_data\_reg[66] = {8'h8d, 8'h20};

    assign cfg\_data\_reg[67] = {8'h0e, 8'h65};

    assign cfg\_data\_reg[68] = {8'h09, 8'h00};

（3）：ov7725\_data ov7725数据缓存模块

说明：将摄像头输入的图像数据进行缓存,每次输出一个像素和使能信号

输入：

sys\_rst\_n：复位信号

ov7725\_pclk：摄像头像素时钟24MHz

ov7725\_href：摄像头行同步信号，为高表示该帧在有效行范围

ov7725\_vsync：摄像头场同步信号，用于帧的计数

[7:0]ov7725\_data：摄像头图像数据

输出：

ov7725\_wr\_en：图像数据有效使能信号

[15:0]ov7725\_data\_out：图像数据，RGB565类型

算法：即不断缓存数据后根据场同步信号，将两次8位缓存数据拼接并输出，并拉高图像使能信号。不够稳定的前10帧不会输出。

关键代码：

//data\_out\_reg,pic\_data\_reg,data\_flag:输出 16 位图像数据缓冲,输入 8 位图像数据缓存输入 8 位,图像数据缓存

    always@(posedge ov7725\_pclk or negedge sys\_rst\_n) begin

        if(sys\_rst\_n == 1'b0) begin

            data\_out\_reg <= 16'd0;

            pic\_data\_reg <= 8'd0;

            data\_flag <= 1'b0;

        end

        else if(ov7725\_href == 1'b1) begin

            data\_flag <= ~data\_flag;

            pic\_data\_reg <= ov7725\_data;

            data\_out\_reg <= data\_out\_reg;

            if(data\_flag == 1'b1)

                data\_out\_reg <= {pic\_data\_reg,ov7725\_data};

            else

                data\_out\_reg <= data\_out\_reg;

        end

        else begin

            data\_flag <= 1'b0;

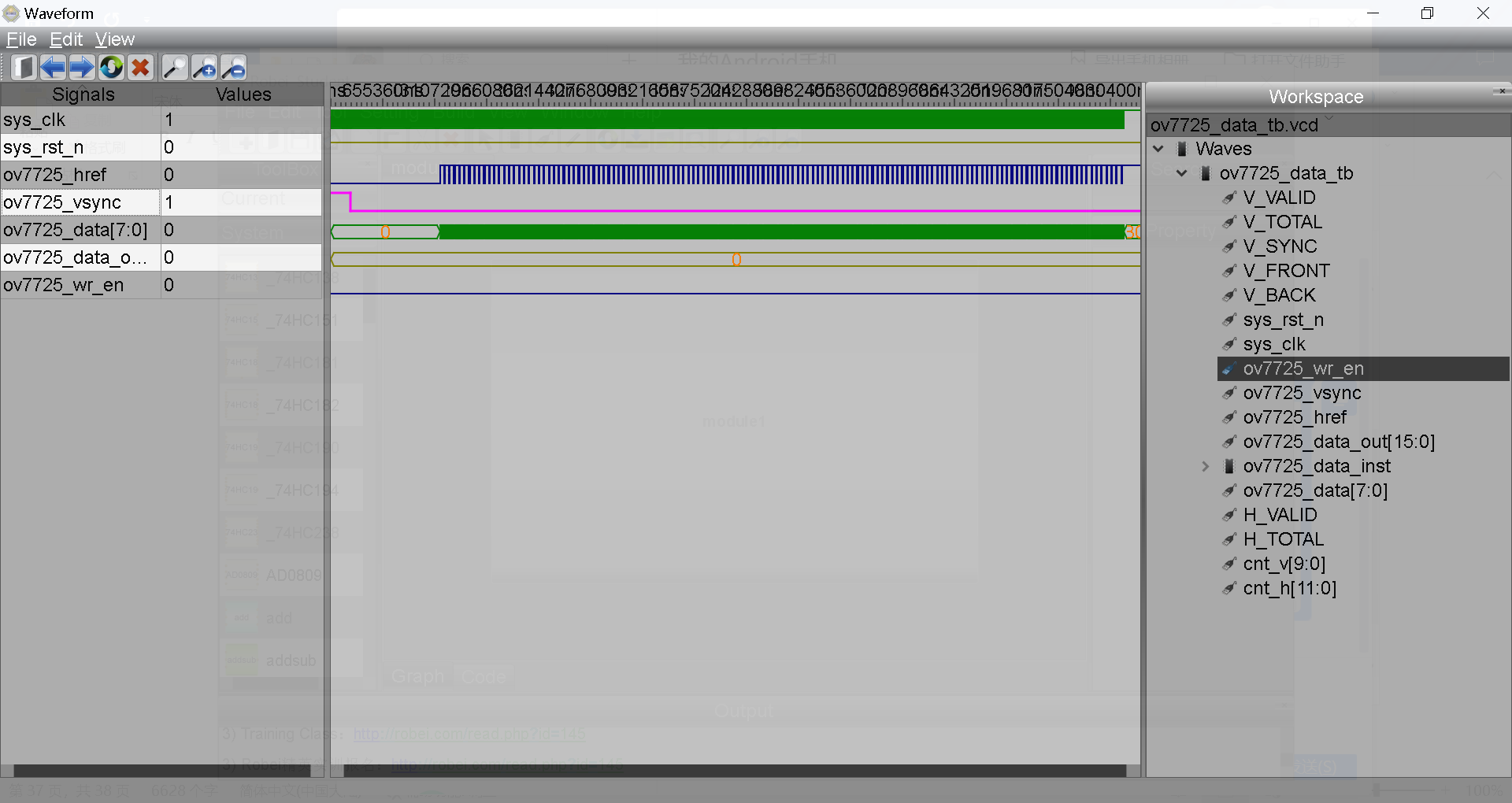
            pic\_data\_reg <= 8'd0;

            data\_out\_reg <= data\_out\_reg;

        end

    end

仿真：



（4）：ov7725\_top ov7725顶层模块

说明：ov7725初始化配置，接收ov7725传输的各信号，输出图像数据和图像有效信号

输入：

sys\_clk：系统时钟50MHz

sys\_rst\_n：系统低电平复位

sys\_init\_done：初始化完成信号

ov7725\_pclk：ov7725时钟24MHz

ov7725\_href：ov7725行同步信号

ov7725\_vsync：ov7725场同步信号

[7:0]ov7725\_data：ov7725传输的数据

输出：

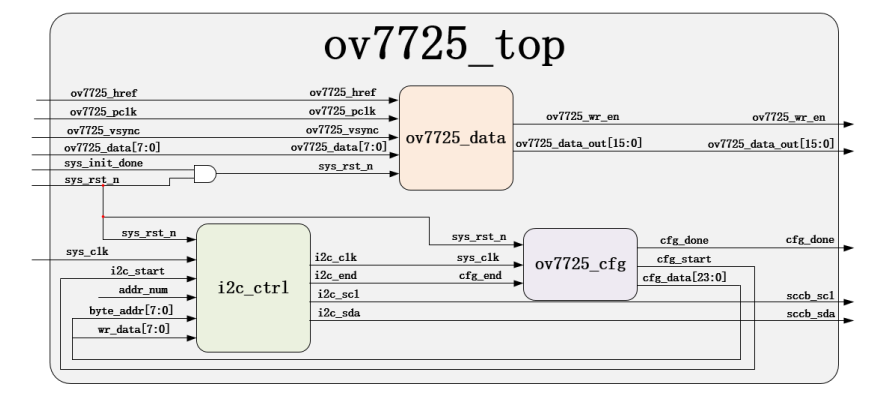
cfg\_done：ov7725寄存器配置完成信号

sccb\_scl：sccb的SIO\_C线，作为通信同步时钟

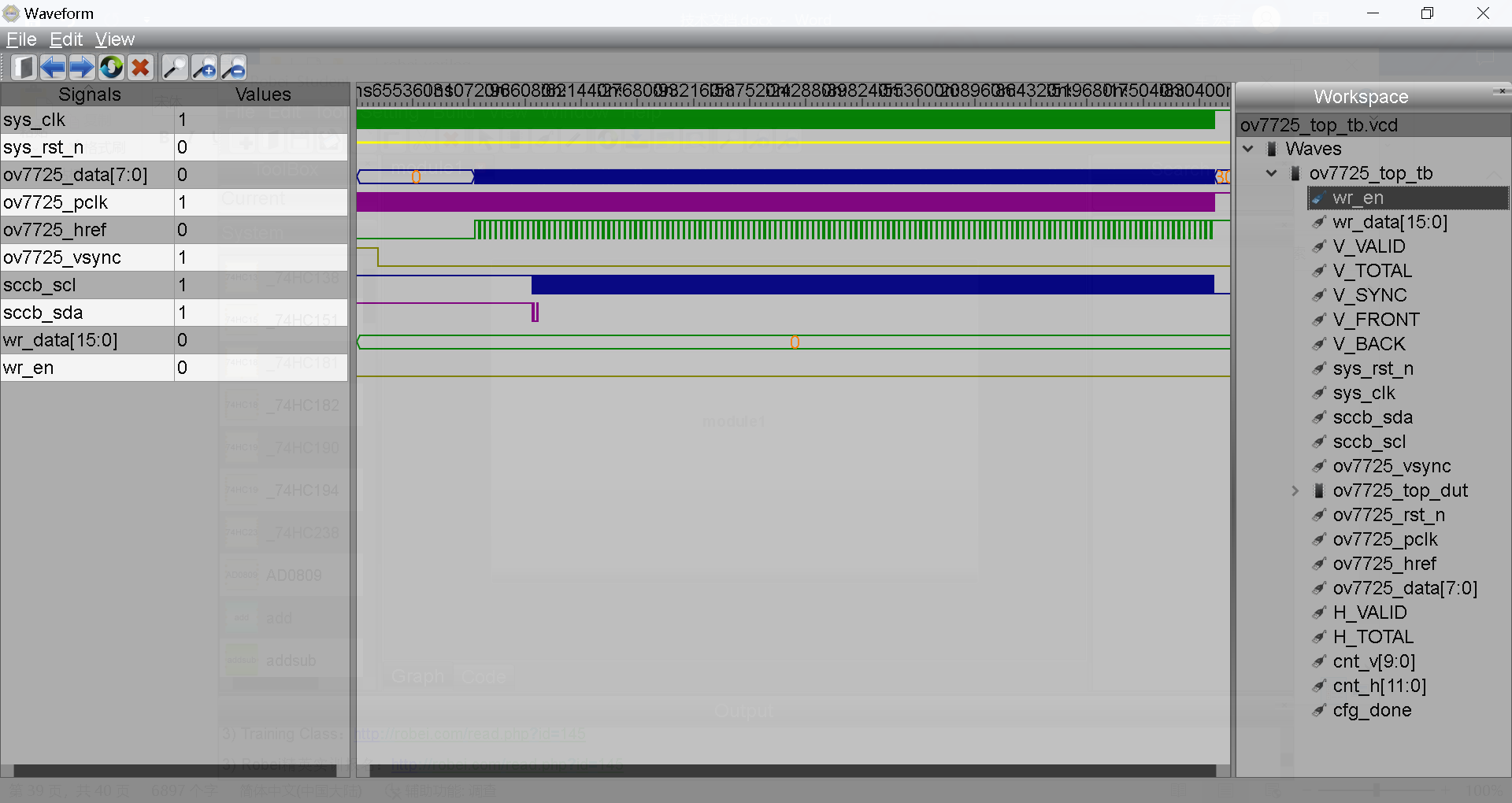
sccb\_sda：sccb的SIO\_D线，发送ov7725配置信号

ov7725\_wr\_en：图像数据有效

[15:0]ov7725\_data\_out：图像一帧的数据，RGB565格式

算法：

将各模块联合起来即可。

仿真：

2：flash模块（flash文件夹）

（1）：flash\_spi spi通信模块

说明：使用spi通信协议与flash通信的模块，可以发送命令、地址，读写flash数据

输入：

[7:0]flash\_cmd：命令内容，由spi通信协议内容确定

[23:0]flash\_addr：读写地址

[3:0]cmd\_type：命令类型，据此判断下一次读写的内容，例如地址

flash\_rstn：复位

clock24M：24MHz时钟

flash\_dataout：flash输出数据线，即miso

输出：

flash\_clk：spi时钟

flash\_cs：spi片选信号

flash\_datain：flash输入数据线，即mosi

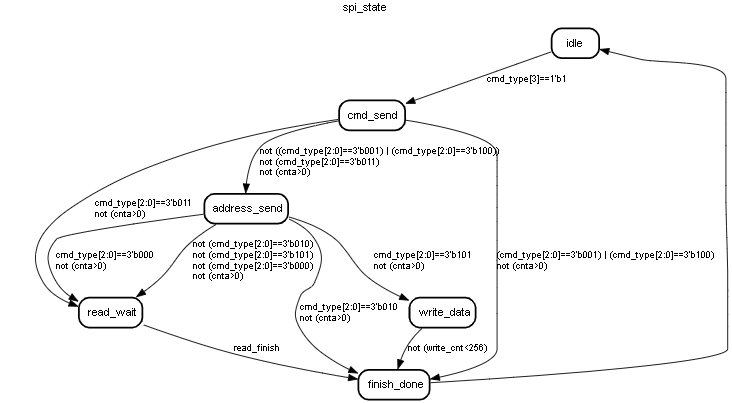
[7:0]mydata\_o：读取到的数据

myvalid\_o：一个字节读完成

Done\_Sig：发送完成信号

spi\_state：spi状态机变量

算法：有限状态机，根据cmd\_type进行跳转，发送或接收数据。



（2）：rgb\_yuv rgb565转灰度模块

说明：由Y = 0.257\*R + 0.504\*G + 0.098\*B + 16可以将RGB转化为灰度

输入：

[15:0]rgb\_in：16位rgb565数据

输出：

    [7:0]y\_out：8位灰度值

算法：由于图像传输效率要求较高，此处用移位进行近似改写，可得Y = 16 + (((R<<6) + (R<<1) + (G<<7) + G + (B<<4) + (B<<3) + B)>>8)，将乘除运算转化为移位运算

（3）：ov7725\_flash flash图片写入模块

说明：本模块将ov7725采集到的图像数据经灰度转化之后，一列一列写入flash

输入：

clk：50MHz时钟

rst\_n：复位

clk24M：24MHz时钟

miso：flash spi数据输出线

[7:0]pixel：输入像素(灰度)

start：上升沿触发起始信号

matrix：上升沿触发帧有效信号

输出：

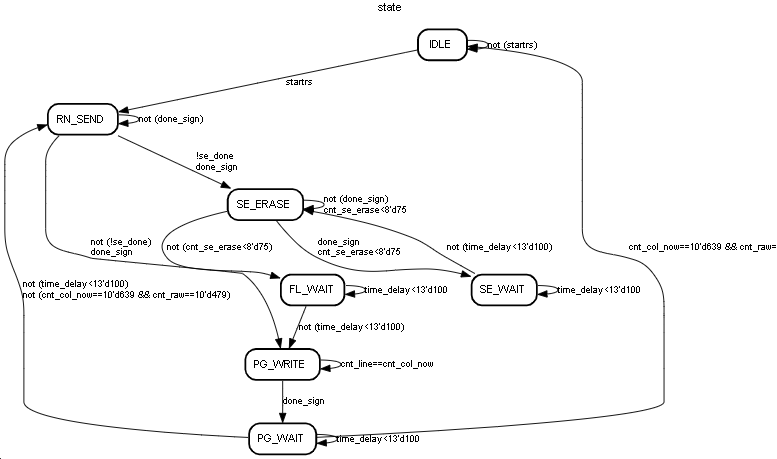
sck：spi时钟

cs：spi片选信号

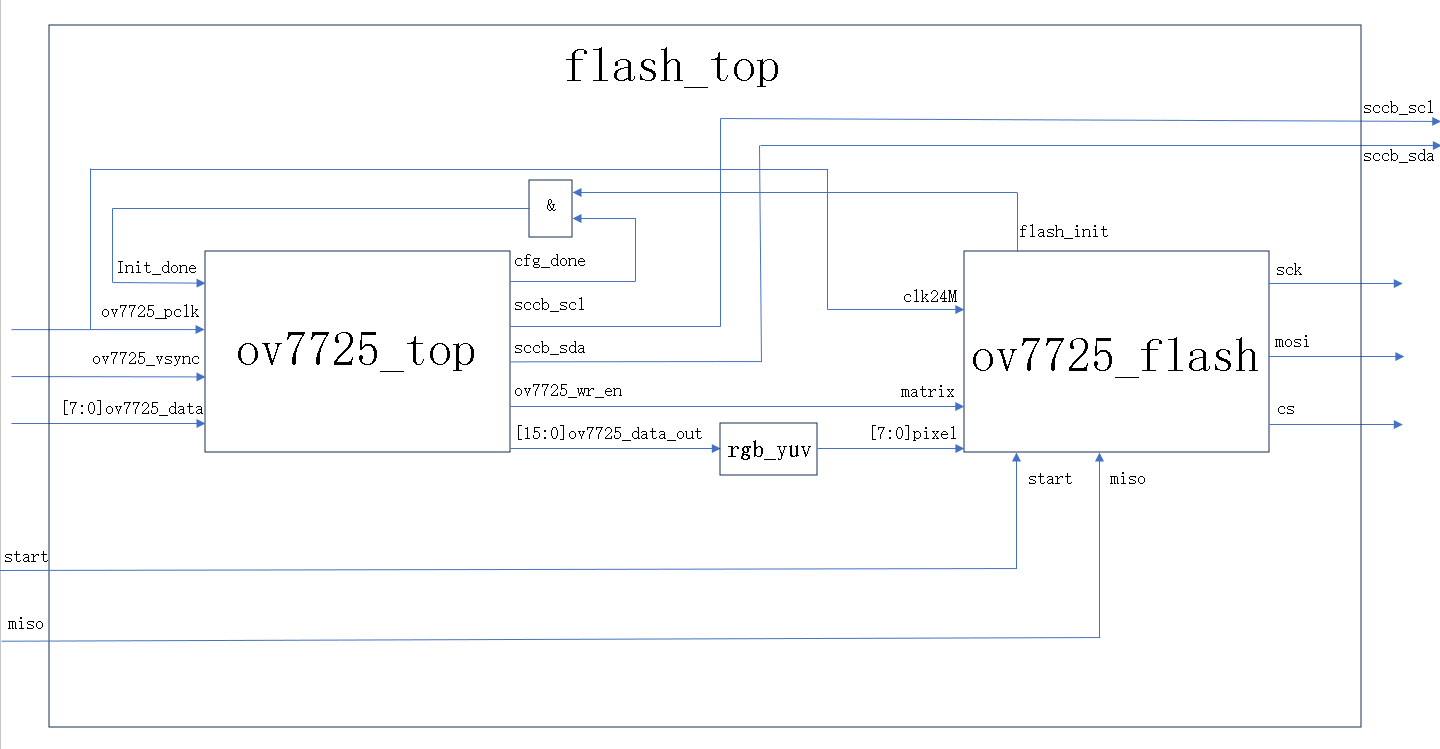
    mosi： spi数据输给flash线

    flash\_init：初始化完成,需和ov7725\_top.v中的cfg\_done相与,后连接sys\_init\_done

算法：由于ov7725输出一个像素仅耗两个ov7725时钟，而写入flash需要写入写使能、地址、页写入内容，速率远不能满足ov7725传输的需求，故每行仅写入一个像素，一张图写入一列。图像为640x480，故每接收640帧写入一帧。每个像素有640个时钟可以使用。使用有限状态机来完成。



（4）：flash\_top flash顶层模块



说明：模块连接如图所示

输入：

sys\_clk：系统时钟

sys\_rst\_n：复位信号

ov7725\_pclk：摄像头像素时钟

ov7725\_href：摄像头行同步信号

ov7725\_vsync：摄像头场同步信号

[7:0]ov7725\_data：摄像头图像数据

miso：flash spi数据输出

start：上升沿触发起始信号

输出：

sccb\_scl：SCL

wire sccb\_sda：SDA

sck：spi时钟

cs：spi片选信号

mosi：spi数据输给flash

五：语音识别部分

1：LD3320模块

（1）：

（2）：

（3）：

（4）：

（5）：

（6）：

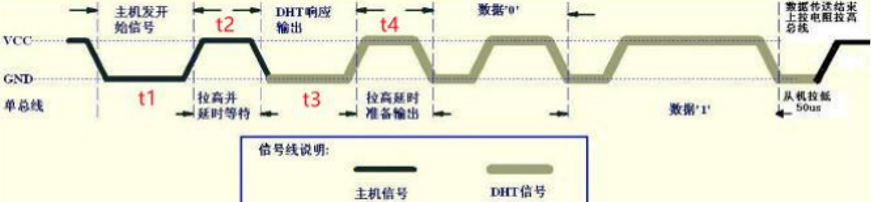
（7）：

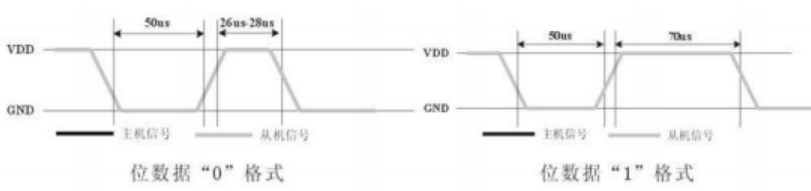
（8）：

六：温湿度传感器及OLED屏幕

1：温湿度传感器模块（OLED\_sensor文件夹）

（1）：dht11温湿度数据获取模块

说明：采用单总线通信

先拉低dht11线表示开始，而后接收一次低电平表明连接成功，之后接收数据，其中01如图定义

从而实现单总线通信。

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

dht11\_req：dht11数据采集请求

输出：

dht11\_done：dht11数据采集结束

dht11\_error：dht11数据采集正确与否判断，1为错误

[7:0]tempH：温度数据整数

[7:0]tempL：温度数据小数

[7:0]humidityH：温度数据整数

[7:0]humidityL：温度数据小数

inout：

dht11：通信总线

算法：有限状态机实现发送接收停止等状态的转化。接收时使用计数器在总线高电平时计数，若计数多于阈值则判定为1，反之判定为0

2：OLED通信及显示模块（OLED\_sensor文件夹）

（1）：iic iic通信模块

说明：控制oled的iic通信模块，8bit的数据 + 8bit的寄存器地址 + 8bit的从机地址。时钟分频100

输入：

sys\_clk：系统时钟

rst\_n：系统复位

[15:0]IICSlave：从机 8bit的寄存器地址 + 8bit的从机地址

IICWriteReq：IIC写寄存器请求

[7:0]IICWriteData：IIC发送数据8bit的数据

IICReadReq：IIC读寄存器请求

输出：

IICSCL：IIC时钟输出

IICWriteDone：IIC写寄存器完成

IICReadDone：IIC读寄存器完成

[7:0]IICReadData：IIC读取数据

inout：

IICSDA：IIC 数据线

算法：普通的有限状态机，包括：

localparam IIC\_IDLE       =   6'b000\_001;  /\*空闲态\*/

localparam IIC\_START      =   6'b000\_010;  /\*起始态\*/

localparam IIC\_WRDATA     =   6'b000\_100;  /\*写数据态\*/

localparam IIC\_RDDATA     =   6'b001\_000;  /\*读数据态\*/

localparam IIC\_ACK        =   6'b010\_000;  /\*应答态\*/

localparam IIC\_STOP       =   6'b100\_000;  /\*停止态\*/

而后根据i2c通信协议进行状态之间转换与SDA线拉高/低，实现i2c通信。

（2）：OLED\_Init OLED初始化模块

说明：oled初始化

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

init\_req：初始化请求

write\_done：一组初始化数据完成信号

输出：

init\_finish：初始化完成输出

[23:0]Init\_data：初始化的数据

算法：reg储存各寄存器应当配置的变量，再使用计数器一个一个输出。

关键代码：

case(Init\_index)

        'd0: Init\_data\_reg <= {8'h78,8'h00,8'hAE};

        'd1:        Init\_data\_reg <= {8'h78,8'h00,8'h00};

        'd2:        Init\_data\_reg <= {8'h78,8'h00,8'h10};

        'd3:        Init\_data\_reg <= {8'h78,8'h00,8'h40};

        'd4:        Init\_data\_reg <= {8'h78,8'h00,8'hB0};

        'd5:        Init\_data\_reg <= {8'h78,8'h00,8'h81};

        'd6:        Init\_data\_reg <= {8'h78,8'h00,8'hFF};

        'd7:        Init\_data\_reg <= {8'h78,8'h00,8'hA1};

        'd8:        Init\_data\_reg <= {8'h78,8'h00,8'hA6};

        'd9:        Init\_data\_reg <= {8'h78,8'h00,8'hA8};

        'd10:   Init\_data\_reg <= {8'h78,8'h00,8'h3F};

        'd11:   Init\_data\_reg <= {8'h78,8'h00,8'hC8};

        'd12:   Init\_data\_reg <= {8'h78,8'h00,8'hD3};

        'd13:   Init\_data\_reg <= {8'h78,8'h00,8'h00};

        'd14:   Init\_data\_reg <= {8'h78,8'h00,8'hD5};

        'd15:   Init\_data\_reg <= {8'h78,8'h00,8'h80};

        'd16:   Init\_data\_reg <= {8'h78,8'h00,8'hD8};

        'd17:   Init\_data\_reg <= {8'h78,8'h00,8'h05};

        'd18:   Init\_data\_reg <= {8'h78,8'h00,8'hD9};

        'd19:   Init\_data\_reg <= {8'h78,8'h00,8'hF1};

        'd20:   Init\_data\_reg <= {8'h78,8'h00,8'hDA};

        'd21:   Init\_data\_reg <= {8'h78,8'h00,8'h12};

        'd22:   Init\_data\_reg <= {8'h78,8'h00,8'hDB};

        'd23:   Init\_data\_reg <= {8'h78,8'h00,8'h30};

        'd24:   Init\_data\_reg <= {8'h78,8'h00,8'h8D};

        'd25:   Init\_data\_reg <= {8'h78,8'h00,8'h14};

        'd26:   Init\_data\_reg <= {8'h78,8'h00,8'hAF};

        default:

            Init\_data\_reg <= {8'h78,8'h00,8'hAE};

        endcase

（3）：OLED\_Refresh OLED刷新模块

说明：将oled屏幕全部显示为黑色，刷新模块

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

refresh\_req：初始化请求

write\_done：一组初始化数据完成信号

输出：

refresh\_finish：初始化完成输出

[23:0]refresh\_data：初始化的数据

算法：通过计数器，逐页向oled写入清零命令

关键代码：

case(refresh\_index)

    'd0:  refresh\_data\_reg <= {8'h78,8'h00,8'hB0 + page};

    'd1:  refresh\_data\_reg <= {8'h78,8'h00,8'h00};

    'd2:  refresh\_data\_reg <= {8'h78,8'h00,8'h10};

    default: refresh\_data\_reg <= {8'h78,8'h40,8'h00};

    endcase

（4）：OLED\_NumData OLED数字字模模块

说明：将各数字字模保存，根据输入数字在给定区域信号时控制像素亮，从而实现显示数字的功能

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

font\_row：字体行（每个page两行）

[4:0]font\_sel：选择输出哪个数字

[4:0]index：字体列

输出：

[7:0]data：像素数据

算法：用数字储存，用索引读取。

关键代码：

if(rst\_n == 1'b0)

        data <= 'd0;

    else if(font\_sel == 'd0)

        data <= data0[index + 'd8 \* font\_row];

    else if(font\_sel == 'd1)

        data <= data1[index + 'd8 \* font\_row];

    else if(font\_sel == 'd2)

        data <= data2[index + 'd8 \* font\_row];

    else if(font\_sel == 'd3)

        data <= data3[index + 'd8 \* font\_row];

    else if(font\_sel == 'd4)

        data <= data4[index + 'd8 \* font\_row];

    else if(font\_sel == 'd5)

        data <= data5[index + 'd8 \* font\_row];

    else if(font\_sel == 'd6)

        data <= data6[index + 'd8 \* font\_row];

    else if(font\_sel == 'd7)

        data <= data7[index + 'd8 \* font\_row];

    else if(font\_sel == 'd8)

        data <= data8[index + 'd8 \* font\_row];

    else if(font\_sel == 'd9)

        data <= data9[index + 'd8 \* font\_row];

（5）：OLED\_FrontData OLED字模模块

说明：类似数字字模，文字自定义

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

font\_row：行选择（一个page两行）

[5:0]font\_sel：选择自定义的第几个字

[8:0]index：列

输出：

[7:0]data：输出的像素数据

（6）：OLED\_SelData OLED命令选择模块

说明：根据输入命令进行数据选择，输出有效命令或者像素数据

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

init\_req：初始化命令

[23:0]init\_data：初始化命令内容

refresh\_req：刷新命令

[23:0] refresh\_data：刷新命令内容

showfont\_req：显示文字命令

[23:0] showfont\_data：显示文字命令内容

showdata\_req：显示数字命令

[23:0]showdata\_data显示数字命令内容

输出：

IICWriteReq：需要写入标志位

[23:0]IICWriteData：写入内容

关键代码：

assign IICWriteReq  = init\_req | showfont\_req | refresh\_req | showdata\_req;

assign IICWriteData = (init\_req == 1'b1) ? init\_data : (refresh\_req == 1'b1) ? refresh\_data : (showfont\_req == 1'b1) ? showfont\_data : showdata\_data;

（7）：OLED\_ShowData OLED数字显示模块

说明：在OLED显示屏上显示DHT11温湿度传感器采集到的数据（仅数字）

输入：

sys\_clk：系统时钟信号

rst\_n：复位信号（低电平有效）

dht11\_done：DHT11传感器采集完成信号

tempH：温度的高8位数据

tempL：温度的低8位数据

humidityH：湿度的高8位数据

humidityL：湿度的低8位数据

ShowData\_req：字符显示请求信号

write\_done：IIC（I2C）一组数据写完成信号

输出：

ShowData\_Data：用于OLED屏幕显示的数据，为24位二进制数，包含了控制显示位置和显示内容的信息

ShowData\_finish：字符显示完成信号

算法：将温湿度传感器采集到的数据存入相应的寄存器中。当字符显示请求信号ShowData\_req拉高时，根据当前需要显示的字符选择信号和数字选择信号，计算出需要显示的字符在OLED屏幕上的位置和需要显示的字体数据的列数和行数。然后，将需要显示的字体数据的列数和行数传递给OLED\_NumData模块，获取用于OLED屏幕显示的数据。接着，将获取到的用于OLED屏幕显示的数据与显示位置的信息组合，得到最终的用于OLED屏幕显示的数据。最后，发送24位用于OLED屏幕显示的数据。

在字符显示完成后，根据当前需要显示的字符选择信号和数字选择信号的值更新下一个需要显示的字符选择信号的值，并根据当前需要显示的字符选择信号和数字选择信号的值计算出下一个需要显示的字符在OLED屏幕上的位置

（8）：OLED\_ShowFront OLED字显示模块

说明：在OLED屏幕上显示自定义的文字

输入：

sys\_clk：系统时钟

rst\_n：低电平复位

ShowFont\_req：字符显示请求

write\_done：iic一组数据写完成

输出：

[23:0]ShowFont\_Data：字符显示数据

ShowFont\_finish：字符显示完成

算法：使用前面OLED\_FontData模块来读取自定字体数据。字体数据生成ShowFont\_Data输出。当所有字符都已显示，ShowFont\_finish拉高

（9）：OLED\_Top OLED显示顶层模块

说明：OLED显示部分的顶层，将前面模块生成的传输数据用i2c协议发送，实现OLED字符和数字显示

输入：

sys\_clk：系统时钟

rst\_n：低电平复位信号

dht11\_done：DHT11传感器数据读取完成信号

tempH：温度数据整数部分

tempL：温度数据小数部分

humidityH：湿度数据整数部分

humidityL：湿度数据小数部分

输出：

OLED\_SCL：OLED显示屏IIC时钟输出

OLED\_SDA：OLED显示屏IIC数据线

算法：将前面的模块串联，先调用初始化模块，而后将前面生成的文字数字信息转化为i2c通信内容，调用刷新模块，不断进行刷新，实现实时显示文字和温湿度

（10）：OLED\_sensor OLED接收数据和显示顶层模块

说明：在OLED\_Top模块的温湿度输入端连接DHT11模块，实现温湿度的采集与OLED屏幕显示

输入：

sys\_clk：系统时钟

rst\_n低电平复位

输出：

OLED\_SCL：i2c时钟线

inout：

dht11：温湿度传感器单总线通信线

OLED\_SDA：i2c数据线

算法：计数器延时，每55\_000\_000个时钟周期更新一次，防止数字变化过快