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

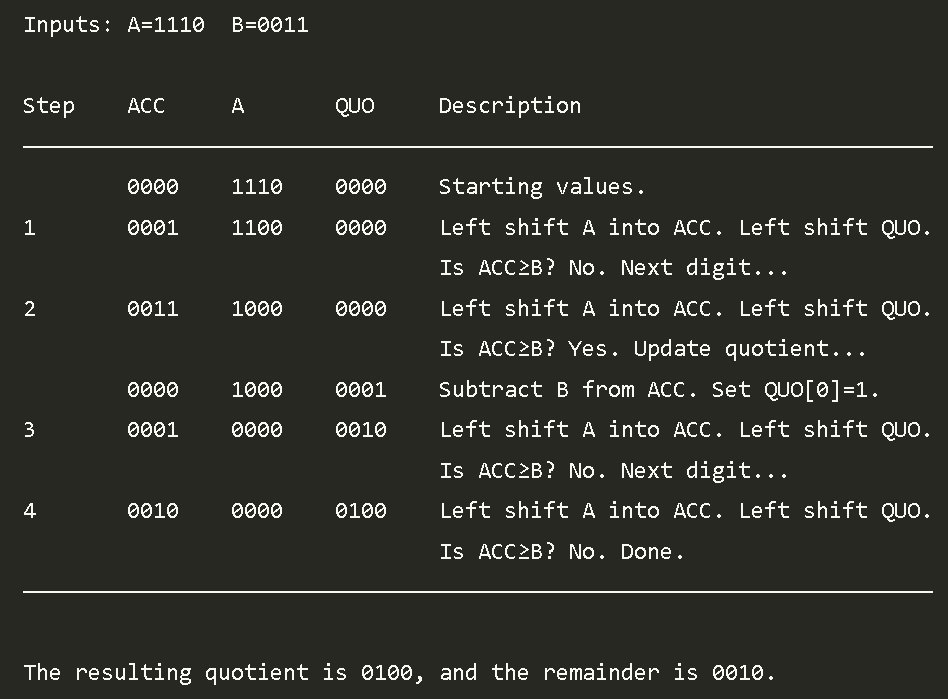
输出：

[31:0]quotient：商

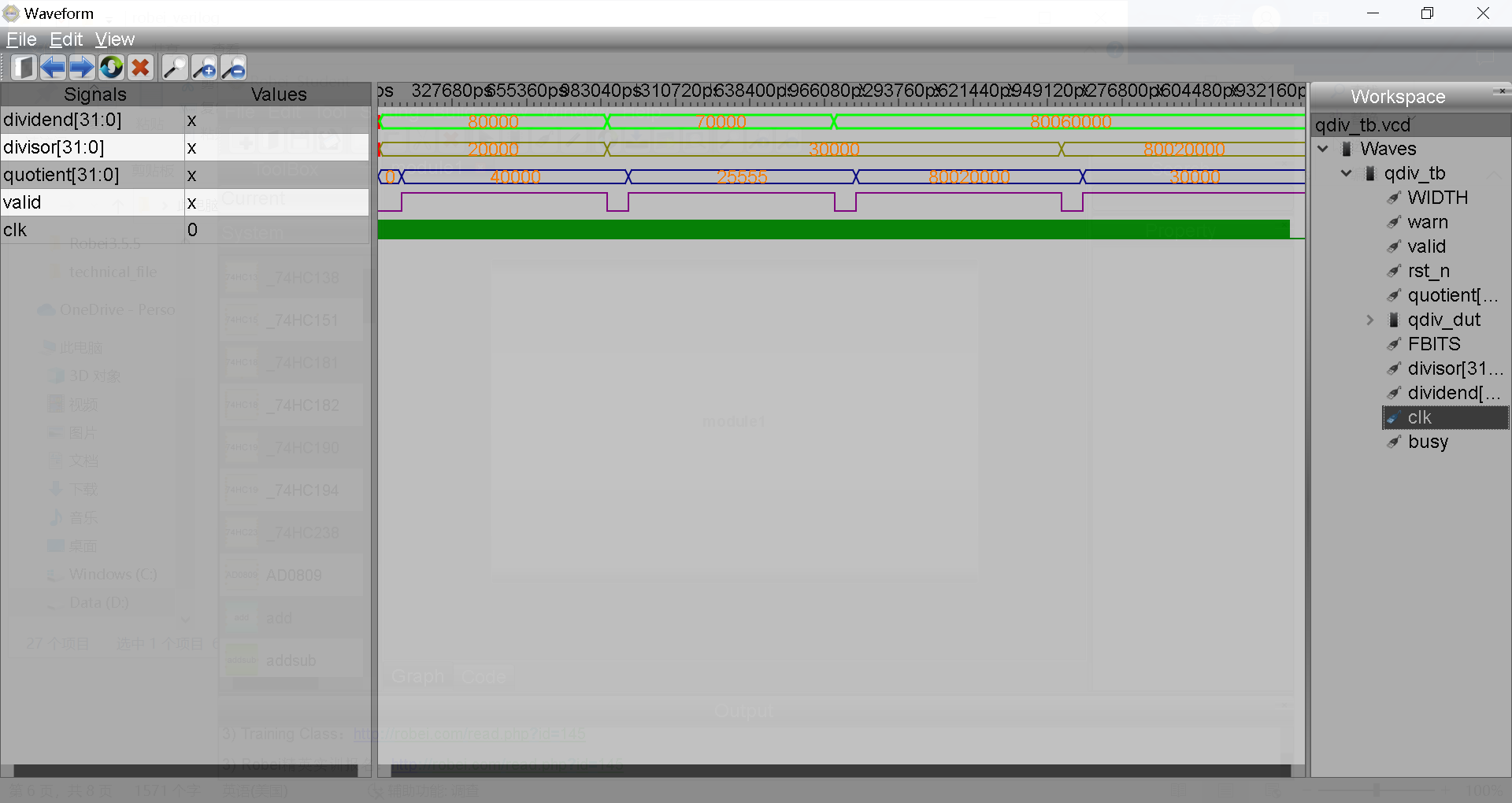
valid：输出有效信号，高电平表示运算完成

    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

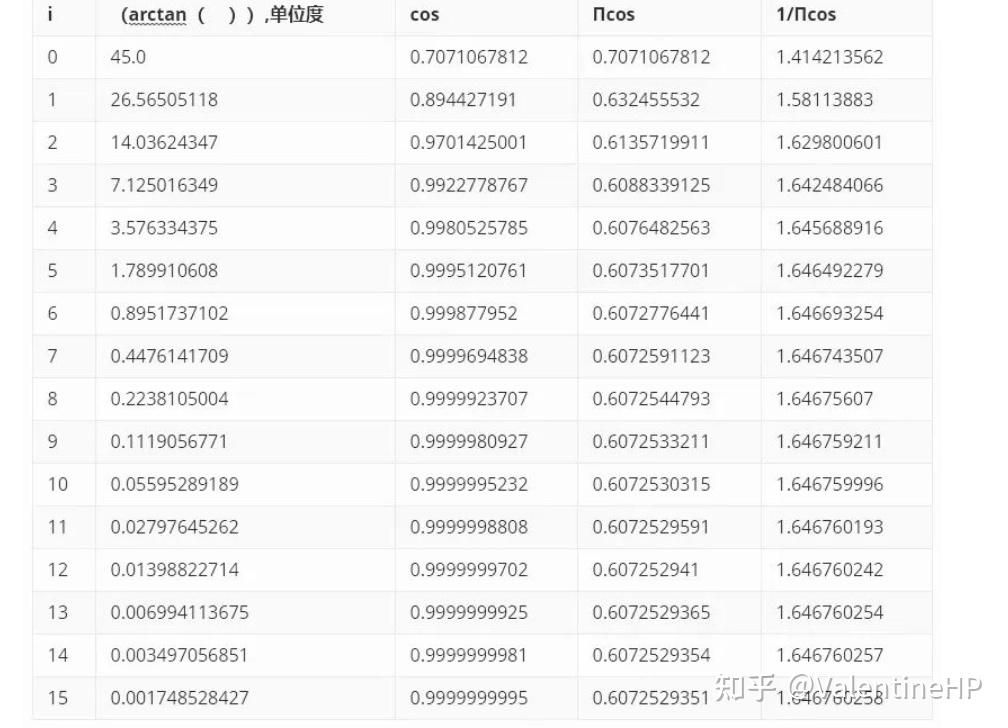
i：表示输入正切值1/2^i

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

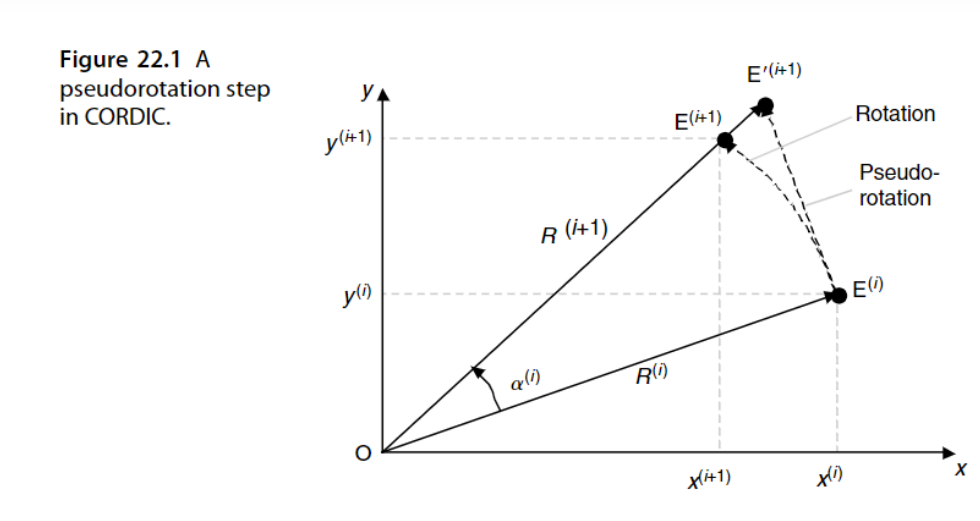
输出：

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

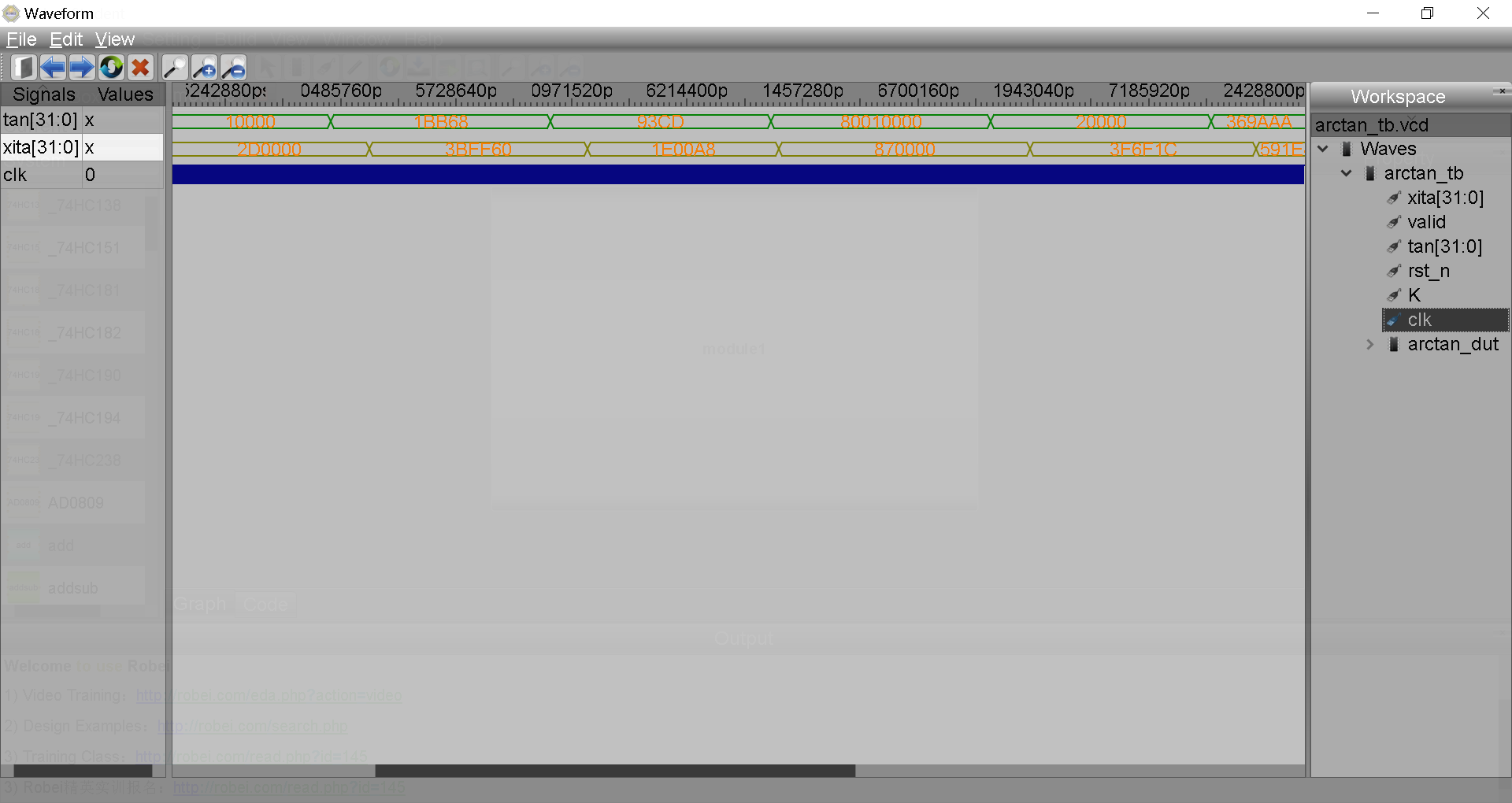
        begin

          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

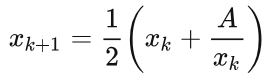
[31:0]sqrter：被开方的数

输出：

valid：输出有效信号（高电平有效）

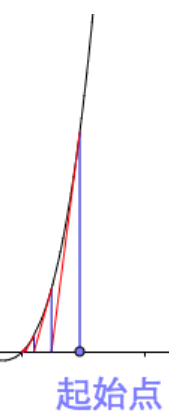
sqrted：运算结果

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)

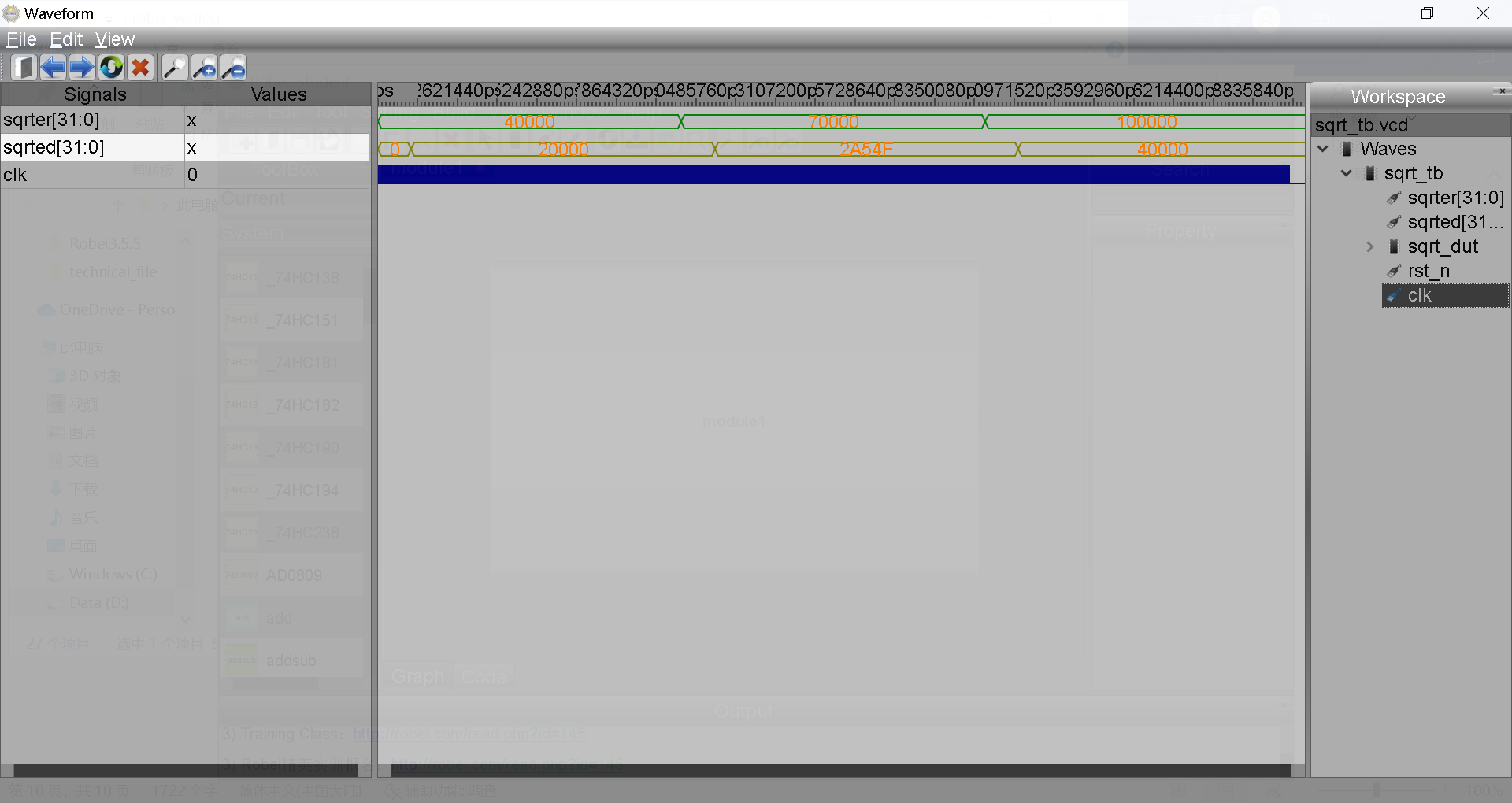
                begin

                    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

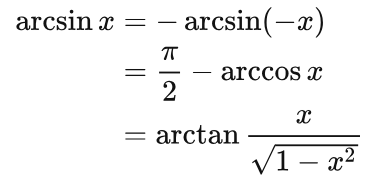
[31:0]value\_sin正弦值

输出：

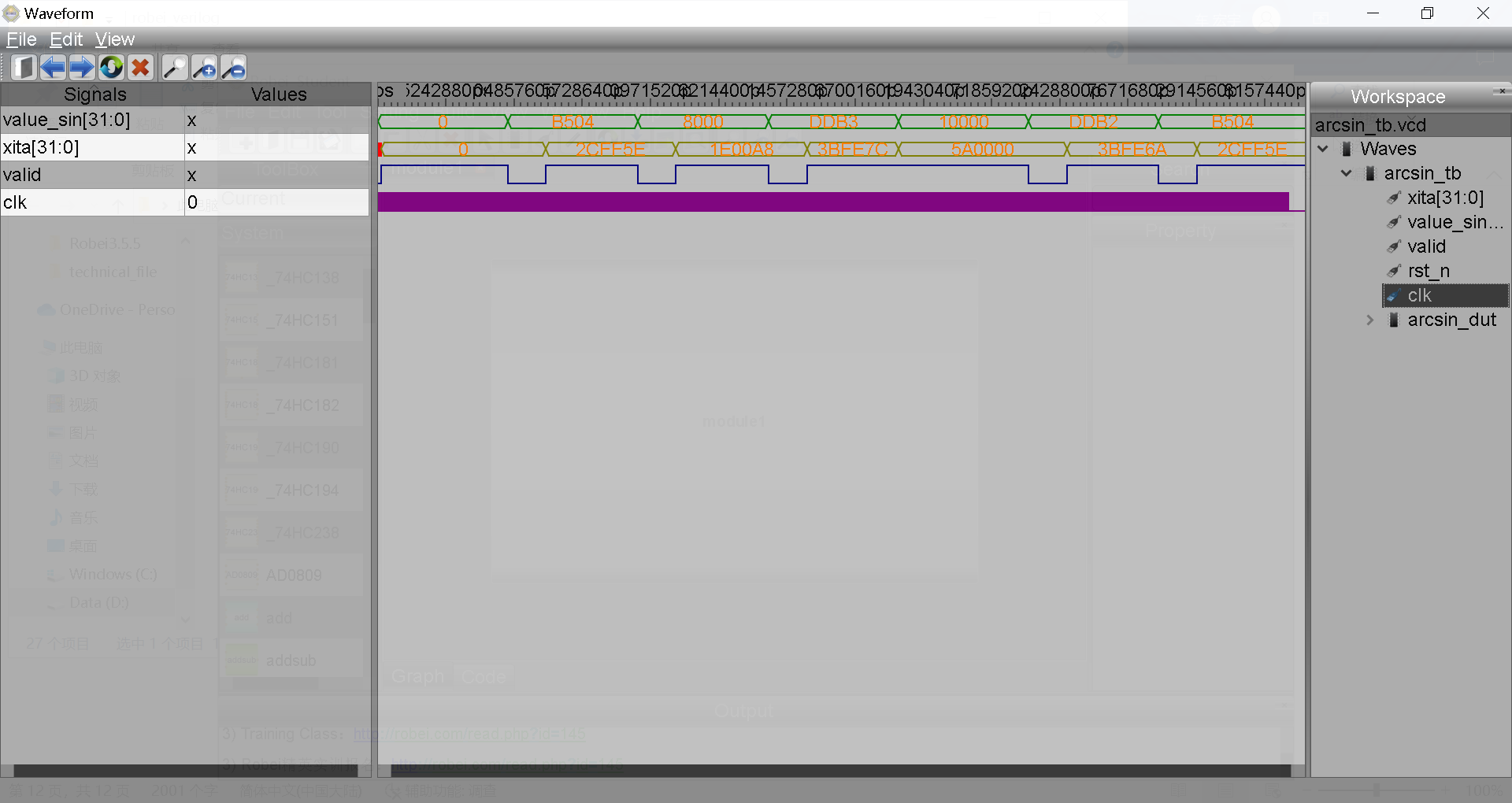
valid：输出有效（高电平）

[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

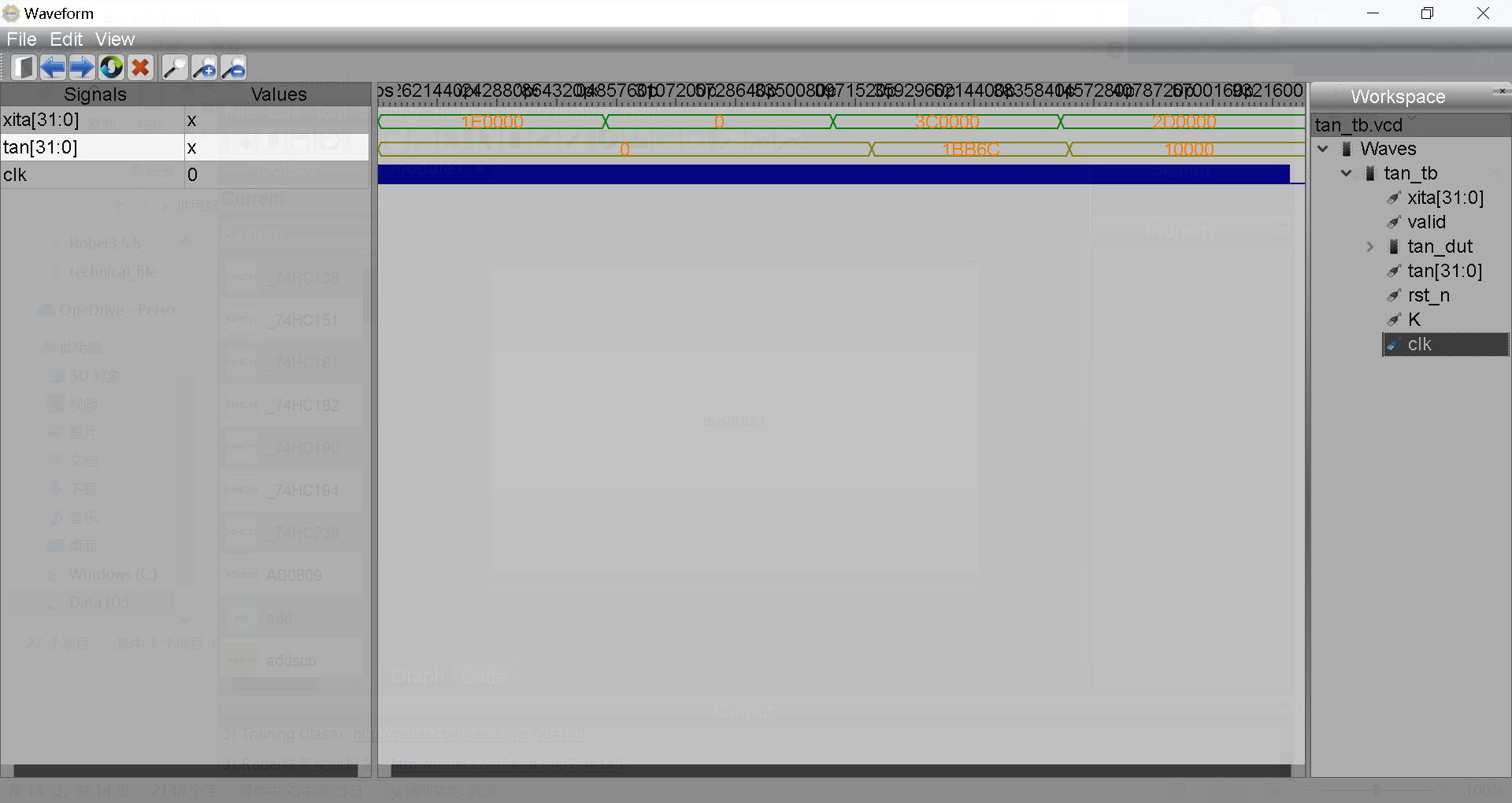
              begin

              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

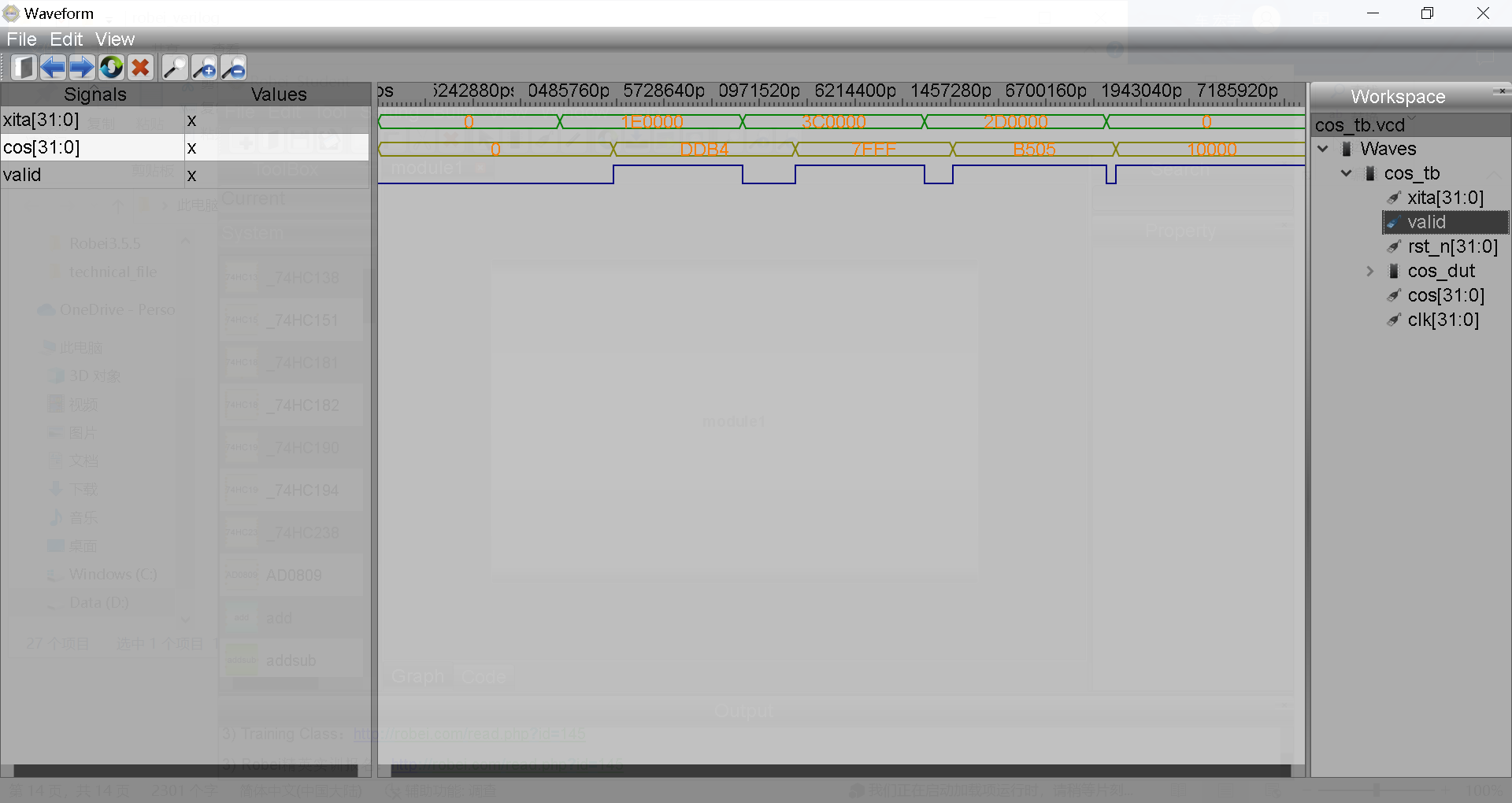
[31:0]xita：32位定点角度值

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）

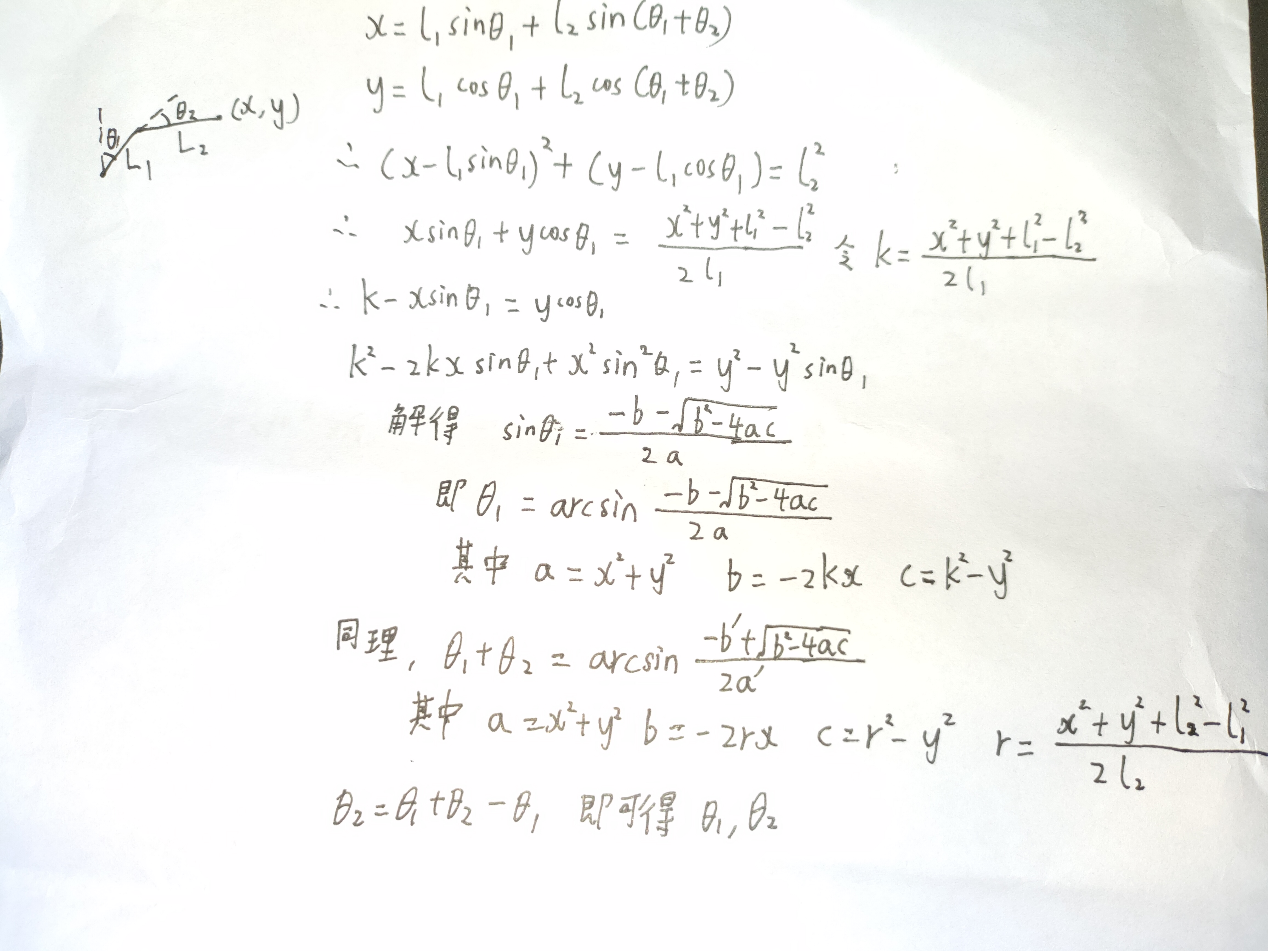
[31:0]y：距关节1垂直距离（mm）

输出：

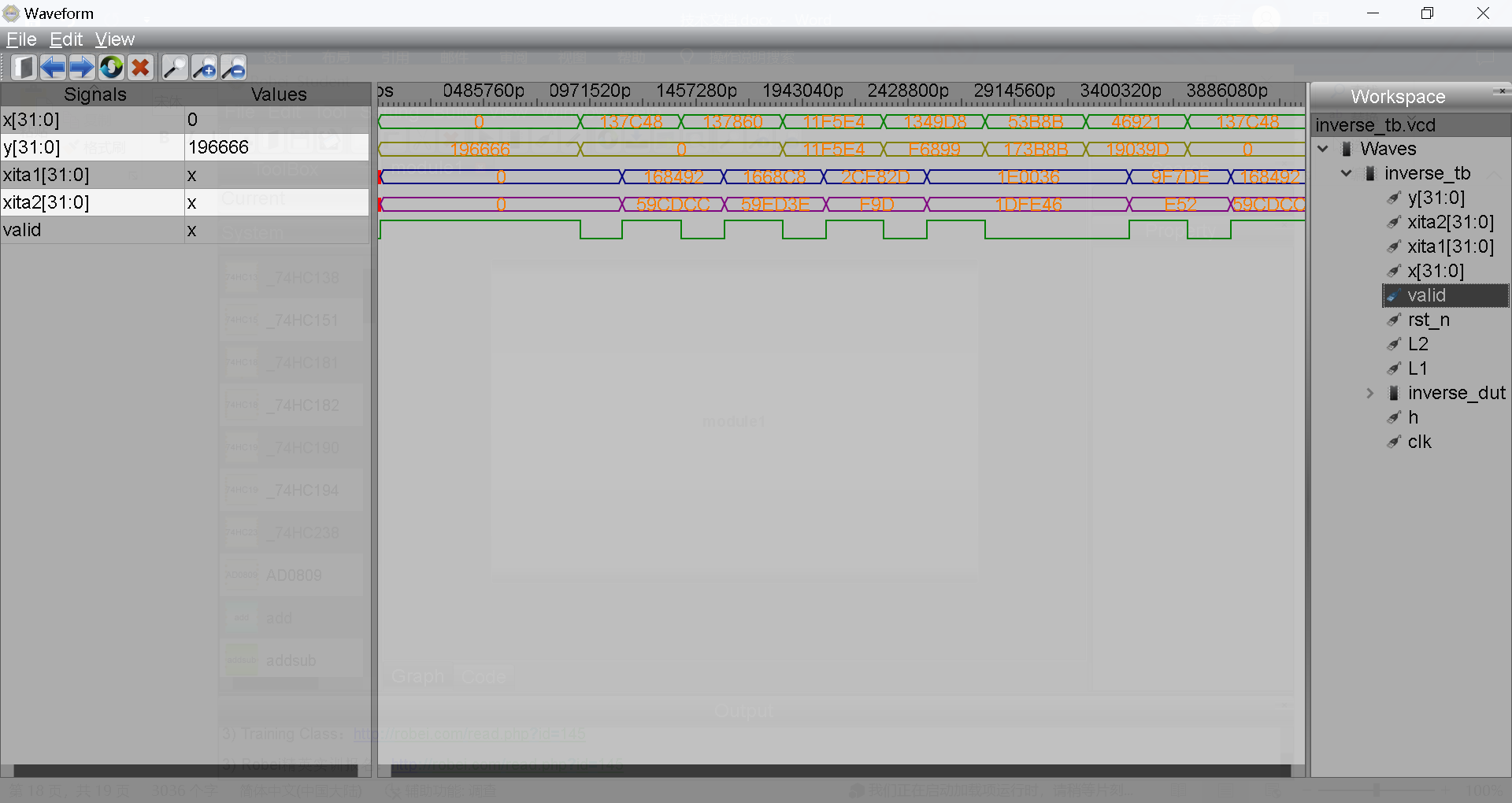
[31:0]xita1：舵机1角度

[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

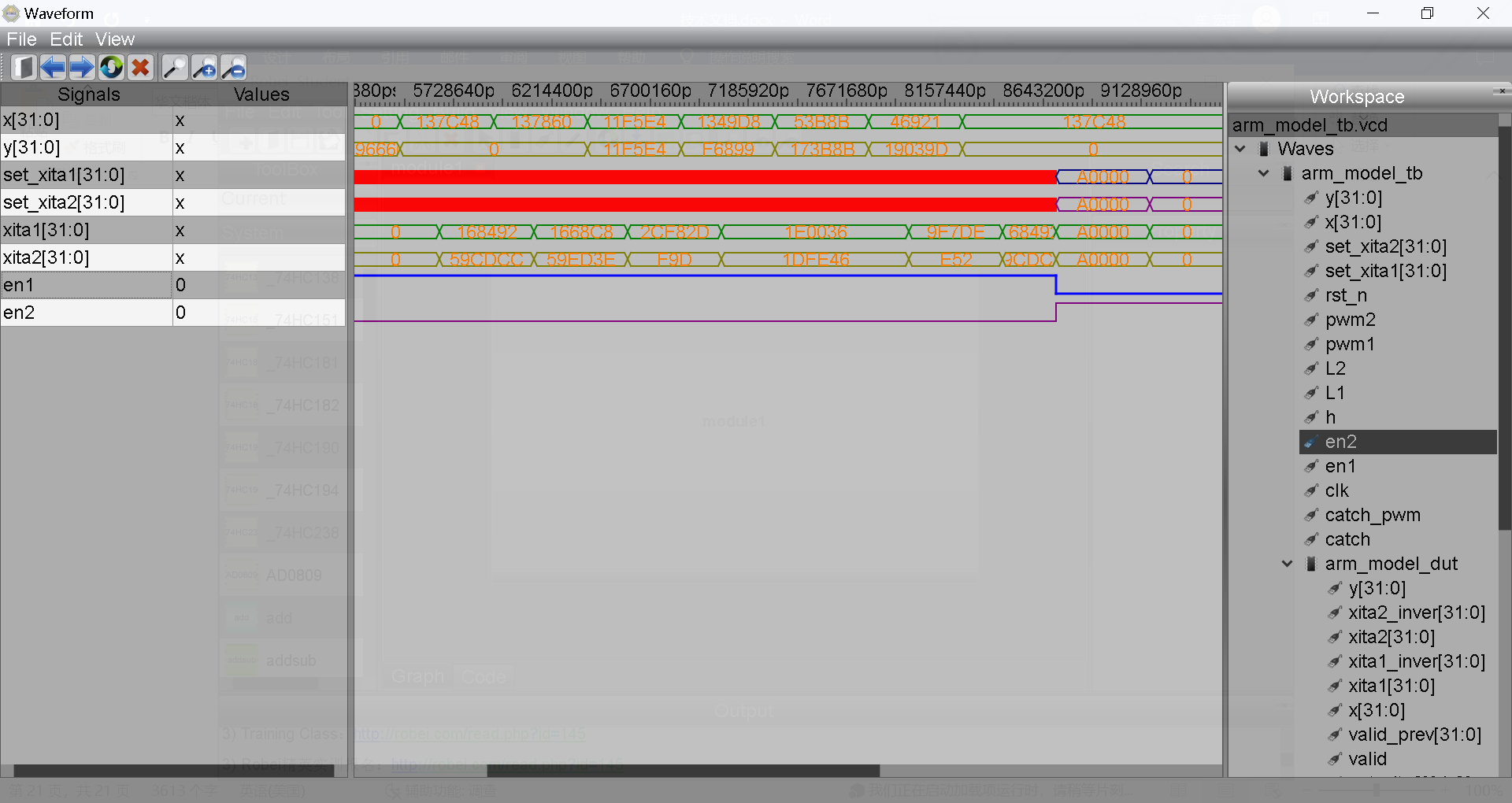
输出：

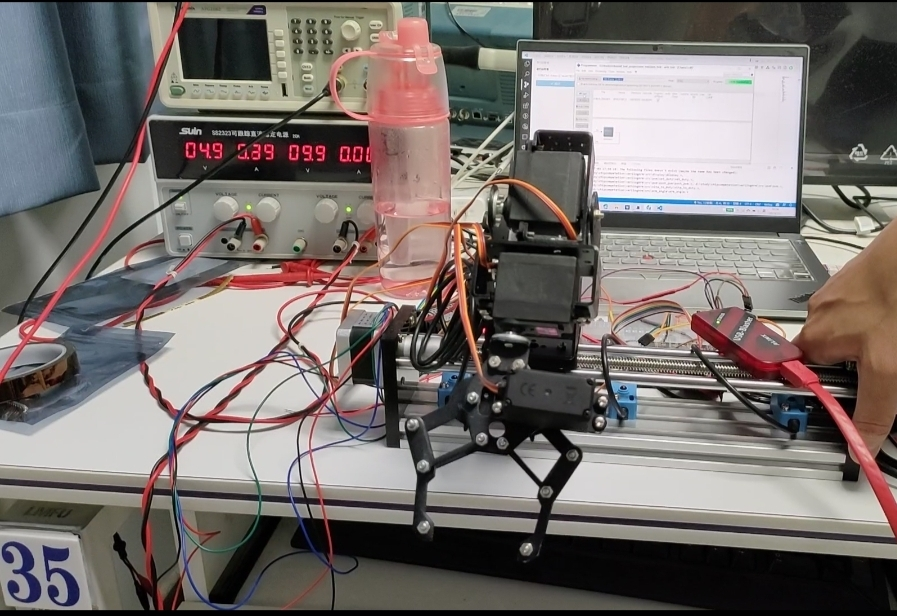
pwm1：控制舵机1的pwm波

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

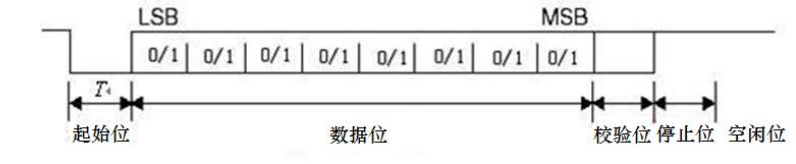
输出：

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;

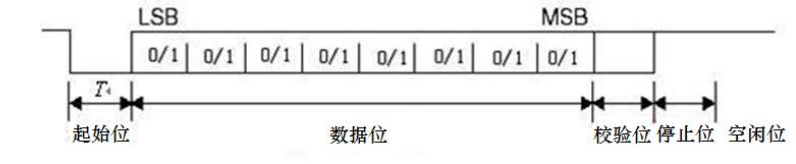
                end

            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：

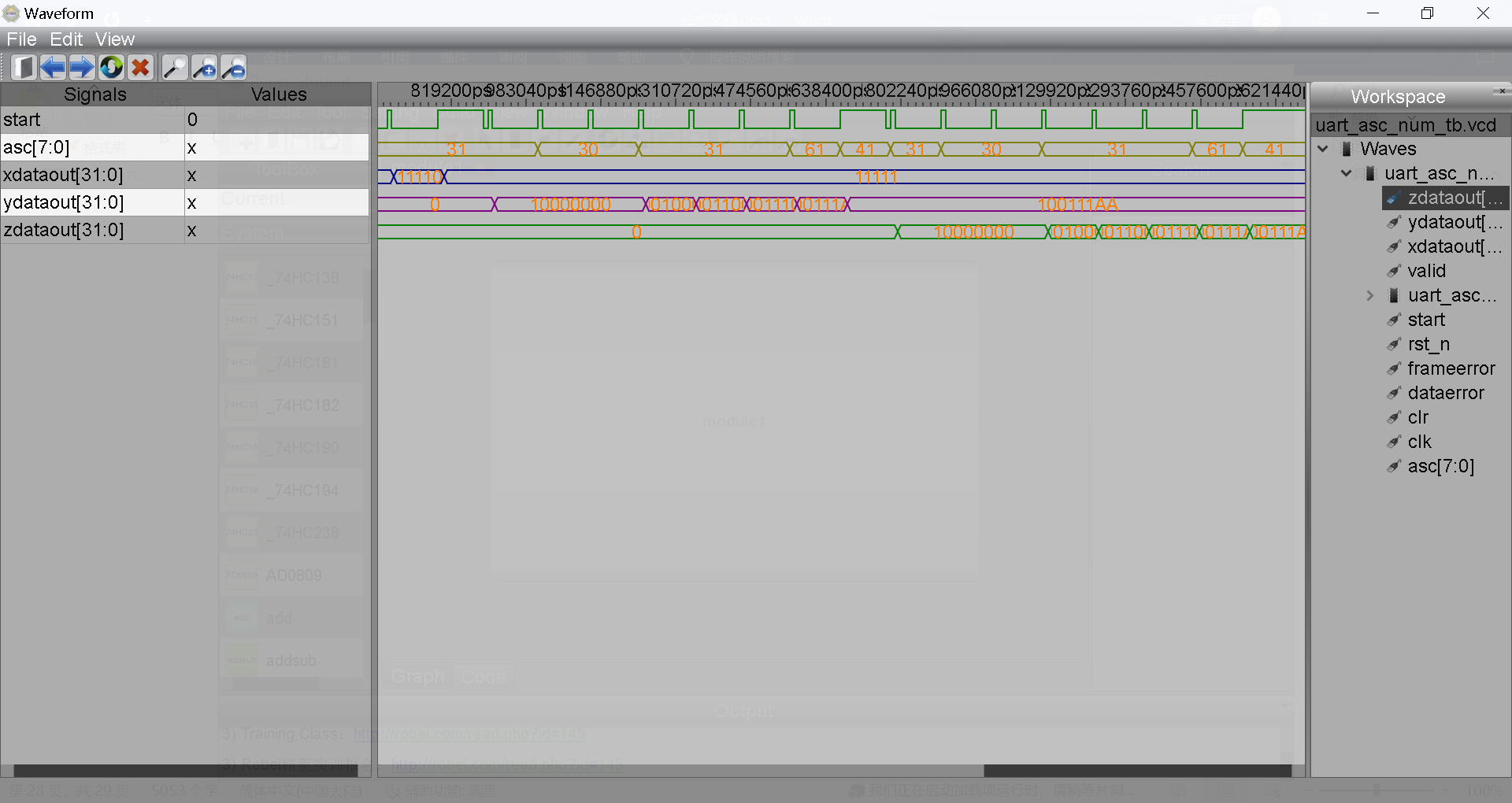
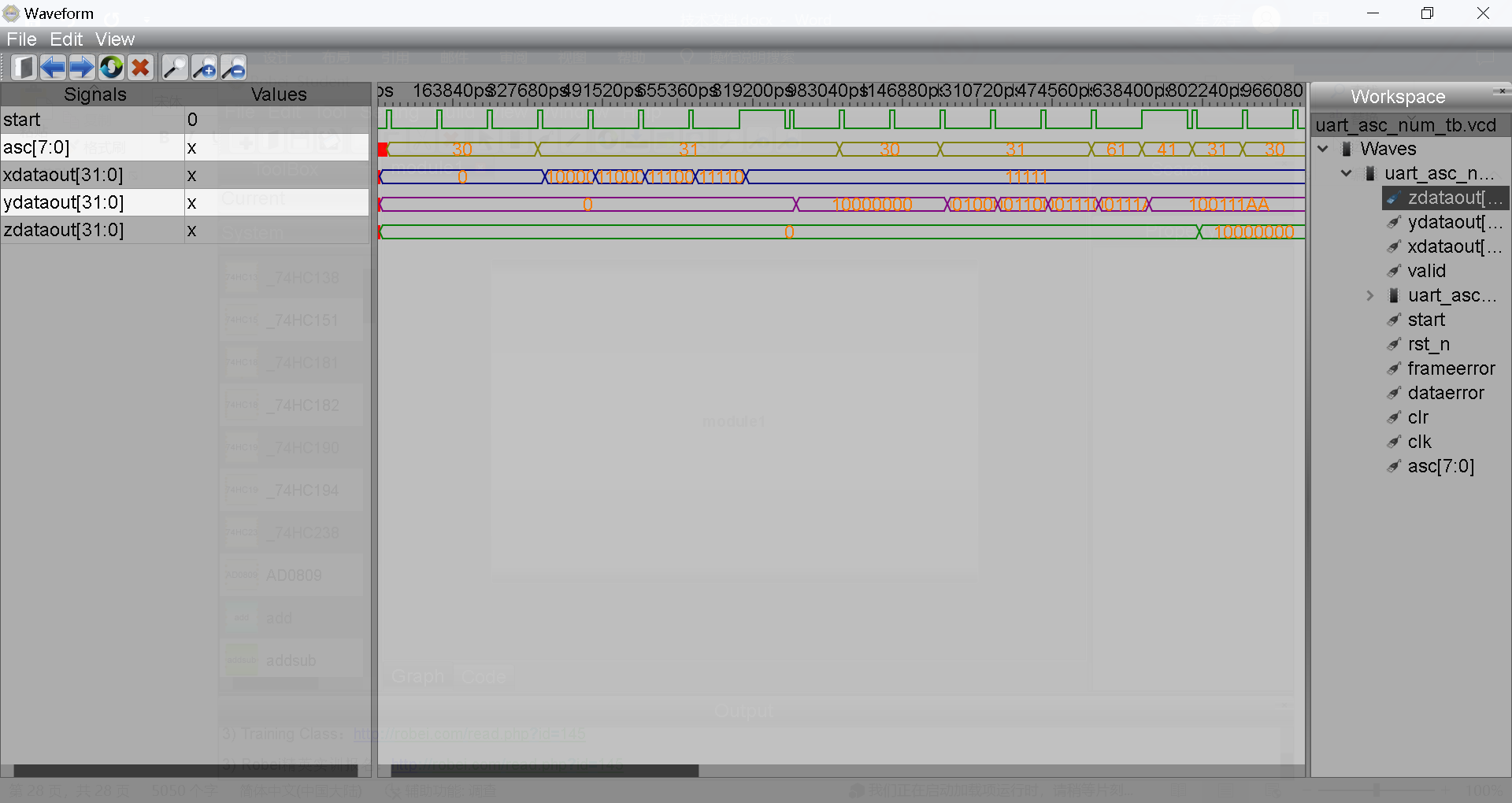
[31:0]xdataout：32位定点数输出x

[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：