# FPGA算法设计考核报告

本设计旨在设计、仿真硬件计算复数模值及幅角。

## 1 算法原理

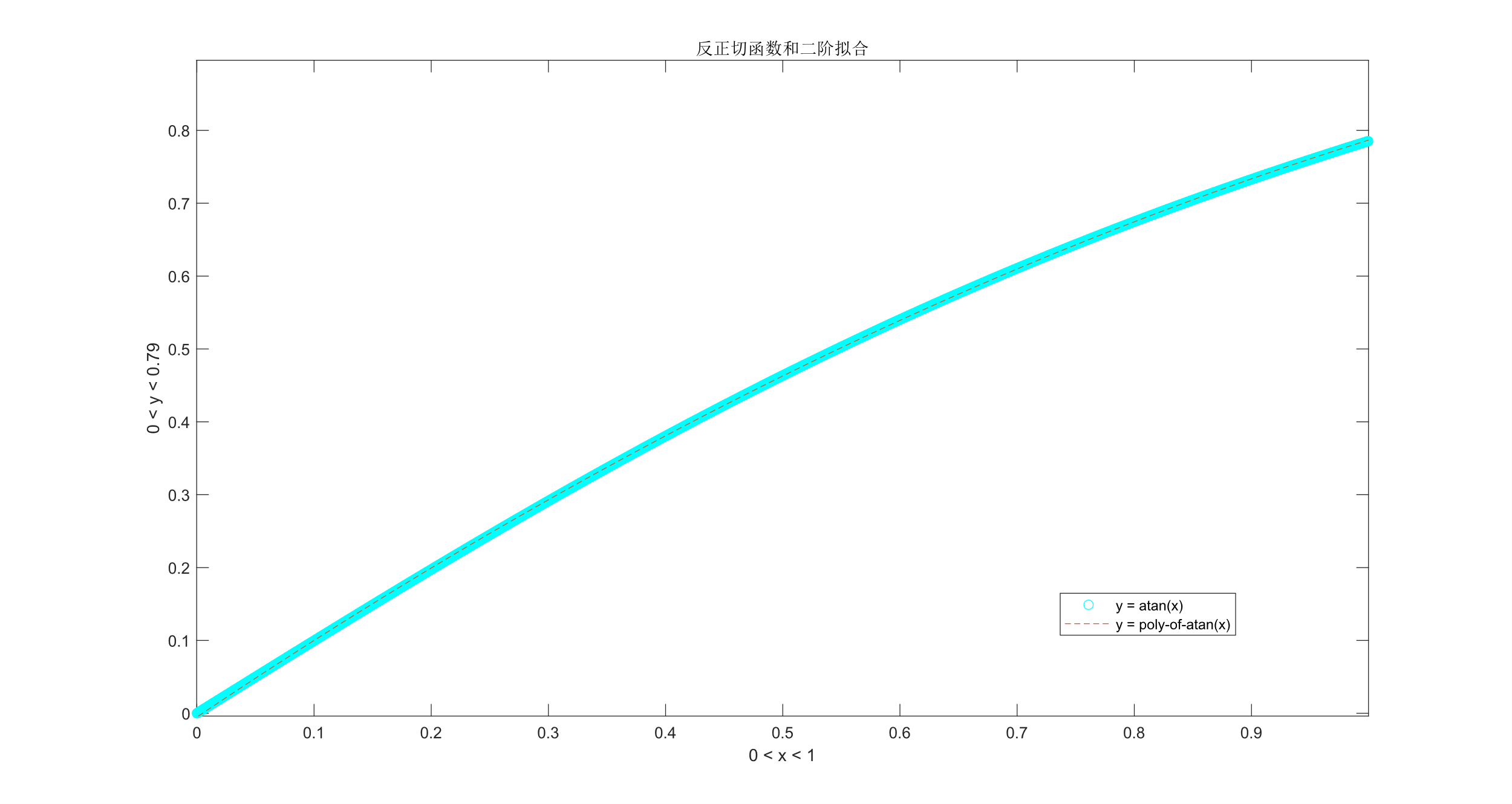
计算复数模值及幅角会涉及到复杂函数计算，在模值计算中对应平方根函数，在幅角计算中对应反正切函数:

在FPGA中实现此类复杂函数计算，有几种不同方法，一是近似计算，二是拟合法，三是线性插值法，四是迭代法。本设计中，线性插值法和拟合法分别实现平方根函数及反正切函数。

### 1.1 反正切函数之拟合实现原理

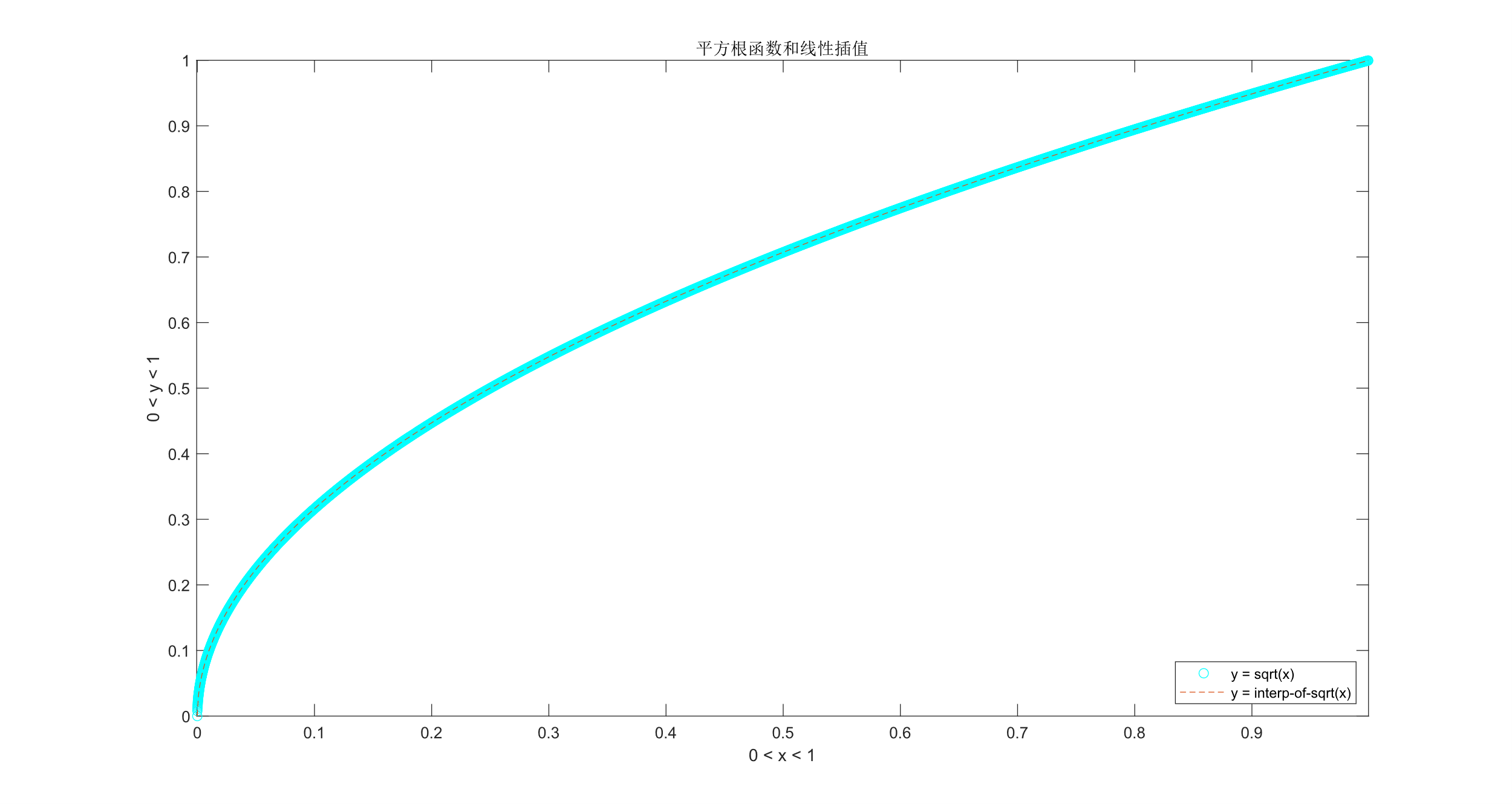
拟合法基于泰勒展开原理:

即用多项式函数叠加接近复杂函数，将复杂函数计算转化为乘法、加法运算。多项式次数越高，误差越小，但乘法次数也会增加，带来资源上的压力。由于[0,1]范围内反正切函数斜率变化不大，较为适合用此二阶拟合方法。



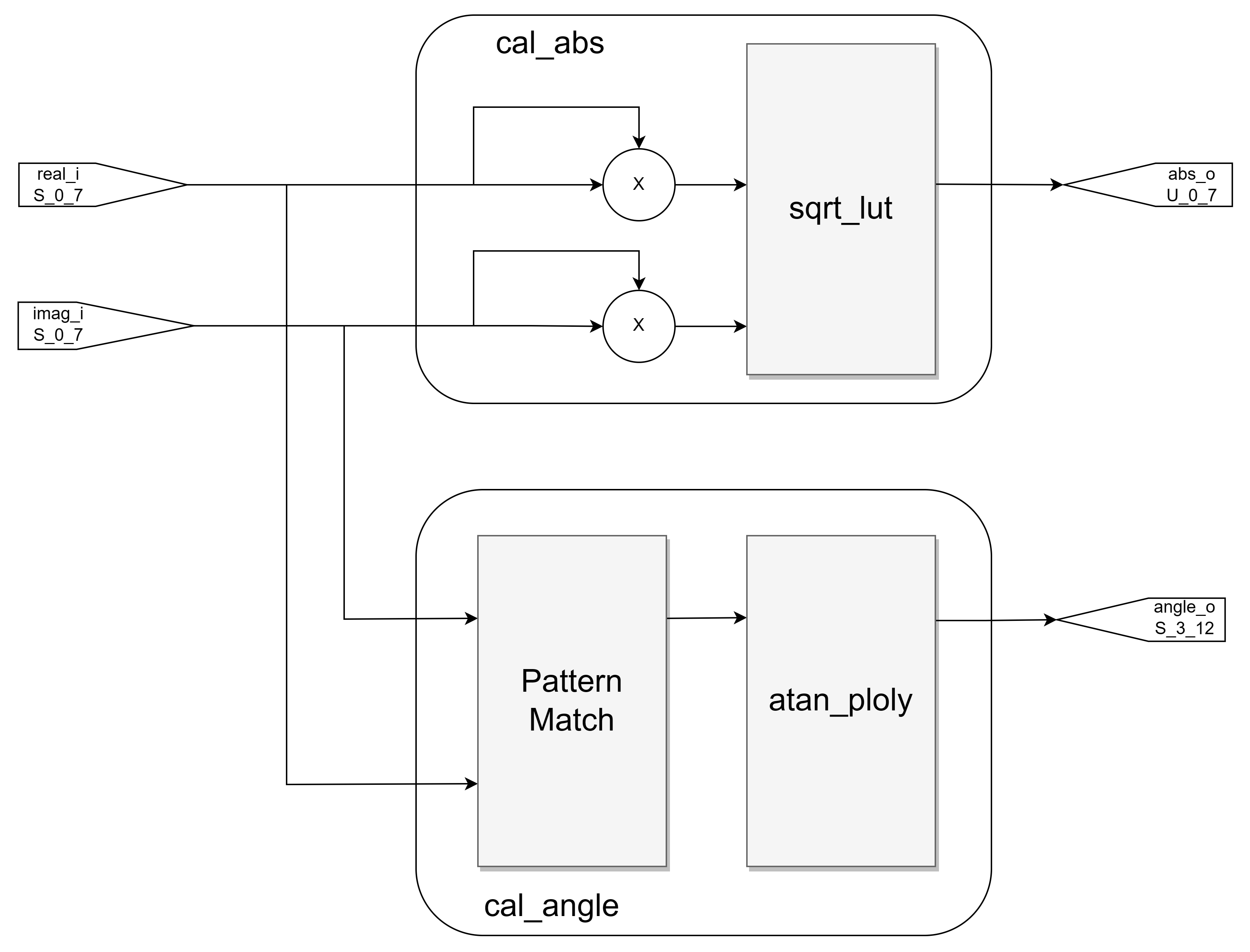
### 1.2 平方根函数之线性插值实现原理

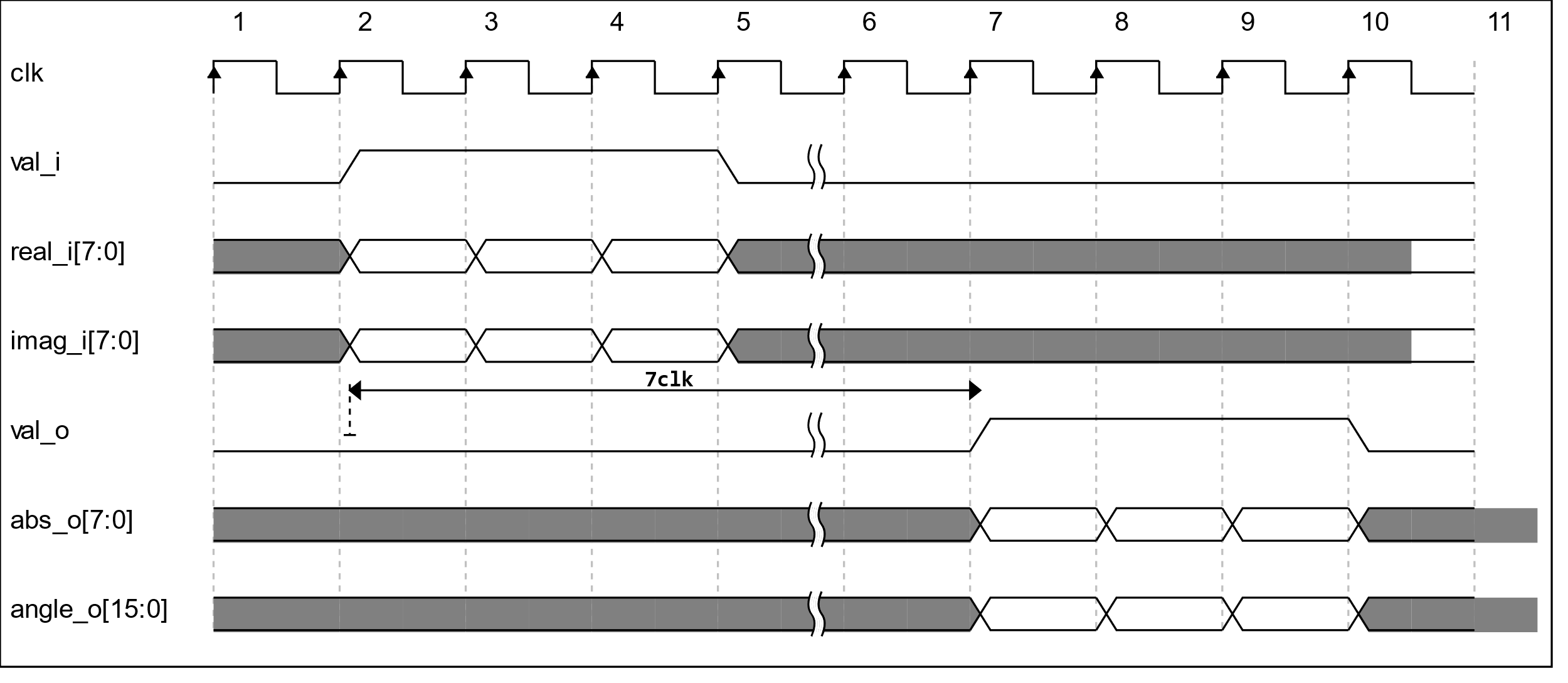
线性插值法即用若干段直线接近复杂曲线，如此可用乘法、加法运算计算复杂函数值。一般而言，直线段越多误差越小，但段数太多会给计算资源带来压力。故在设计硬件前需仿真确定段数，平衡性能与资源。



## 2 模块说明

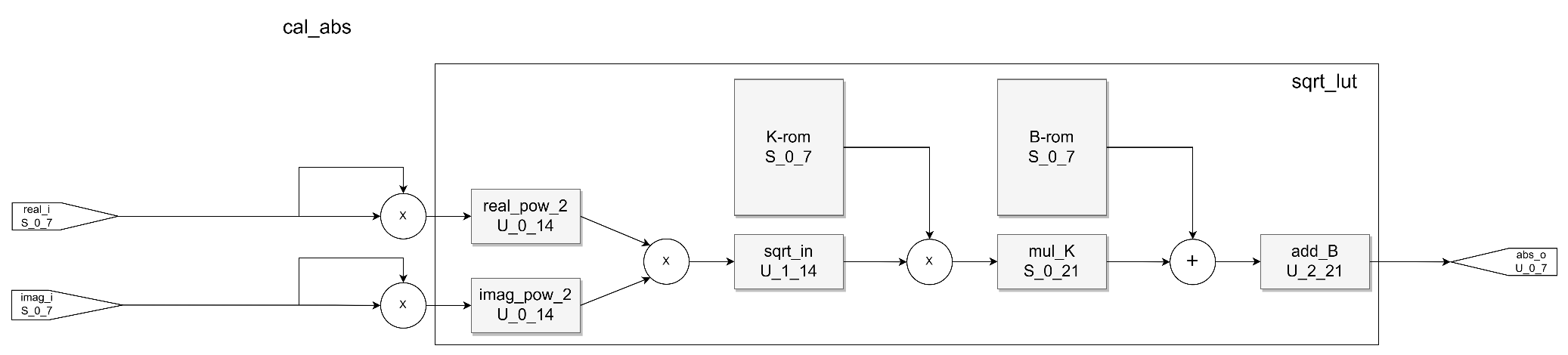
### 2.1 系统整体结构

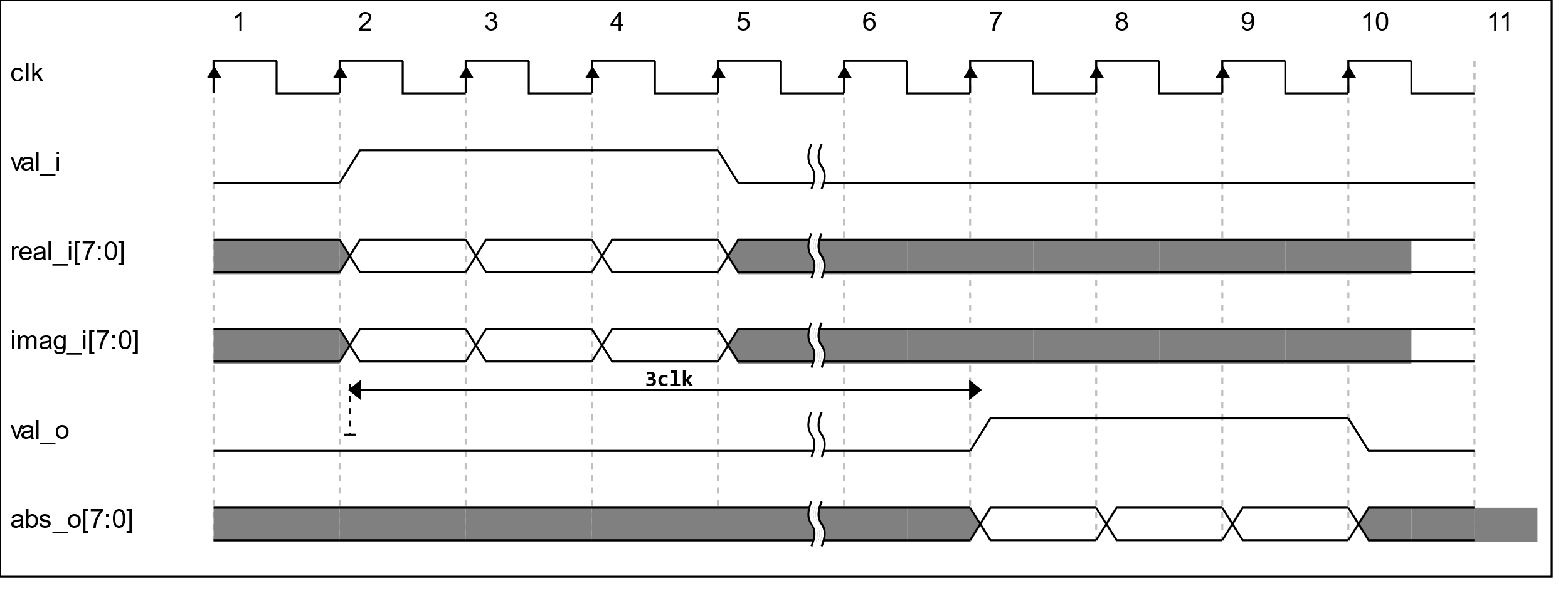




如图所示整个系统由两大模块组成，一是计算幅角的cal\_angle模块，二是计算模值的cal\_abs模块。输入数据是复数的实部和虚部，为8位有符号数，小数位为7位。二者并行运算同时输出16位有符号幅角值和8位无符号模值。系统整体延迟为7个周期。

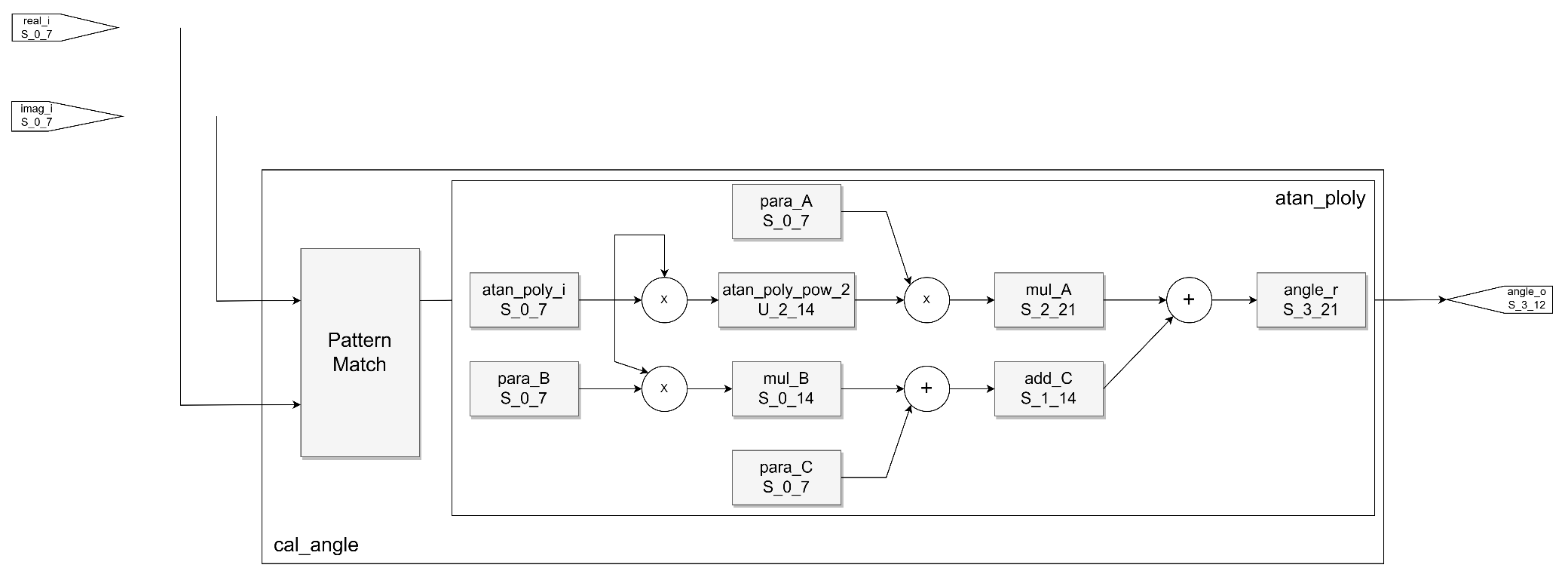
### 2.2 模值计算模块

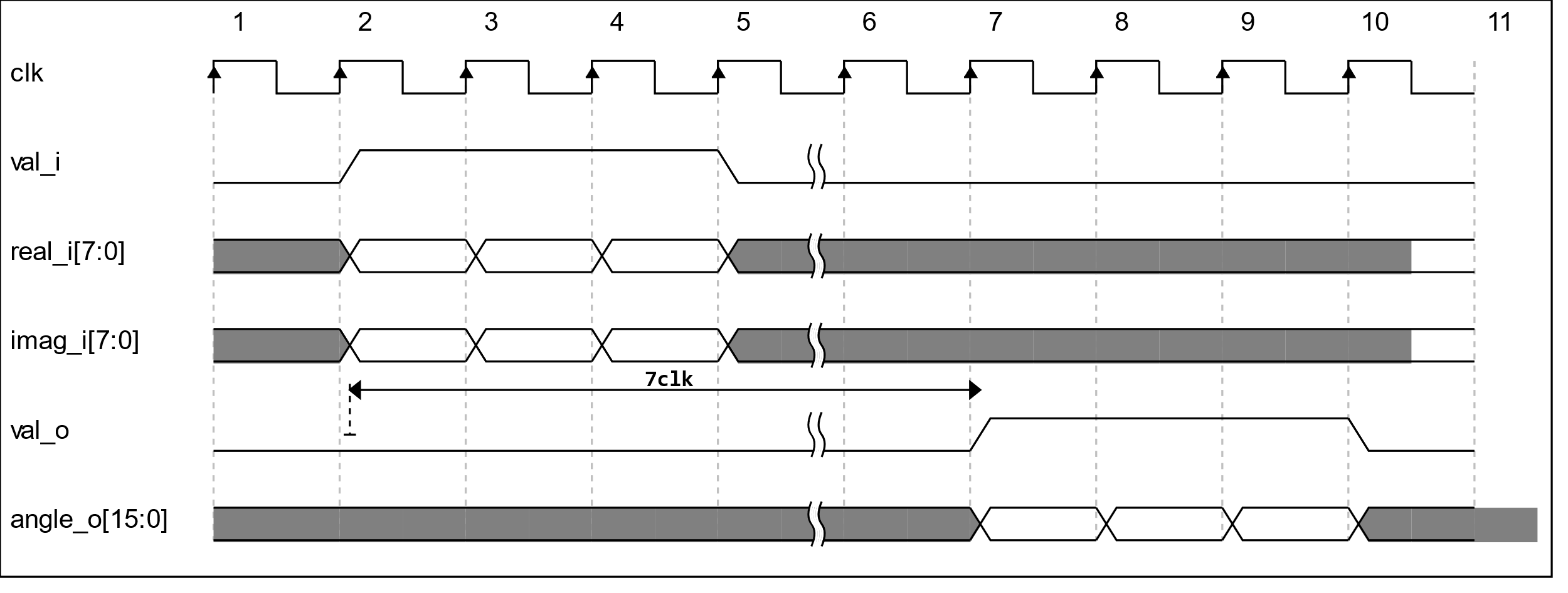




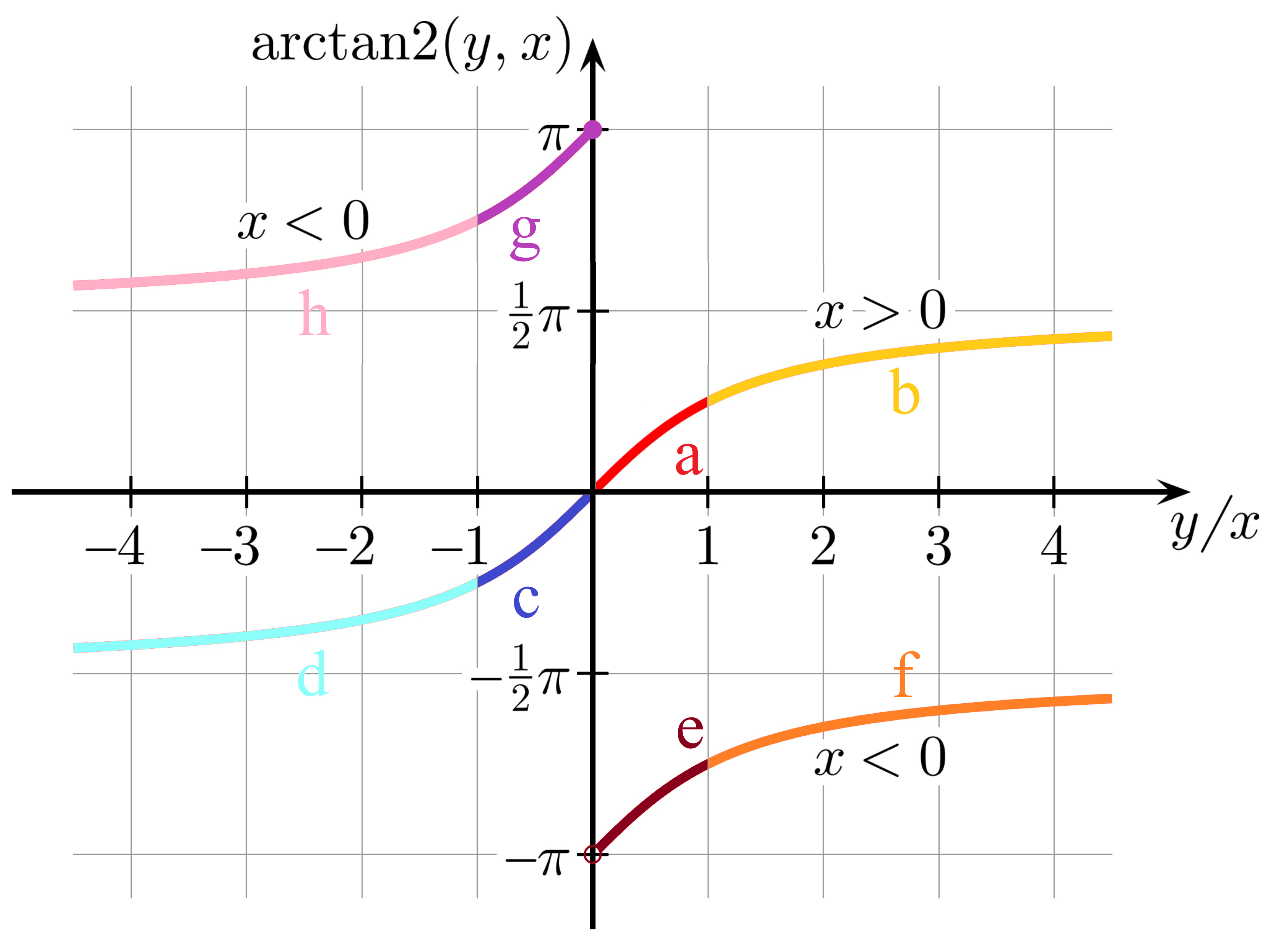
如图所示，cal\_abs模块（模值计算模块）中嵌套了sqrt\_lut模块，后者即通过线性插值计算平方根函数，其中包含提前计算得到的1024段直线斜率和截距，按升序存入ROM ip:k\_rom和b\_rom。输入实部和虚部分别平方运算再相加得到平方和，输入至sqrt\_lut模块，其高10位作为地址输入两ROM模块，得到相应直线段的斜率和截距，再乘加计算得到8bit开方值。整体延迟为3个周期。

### 2.3 幅角计算模块

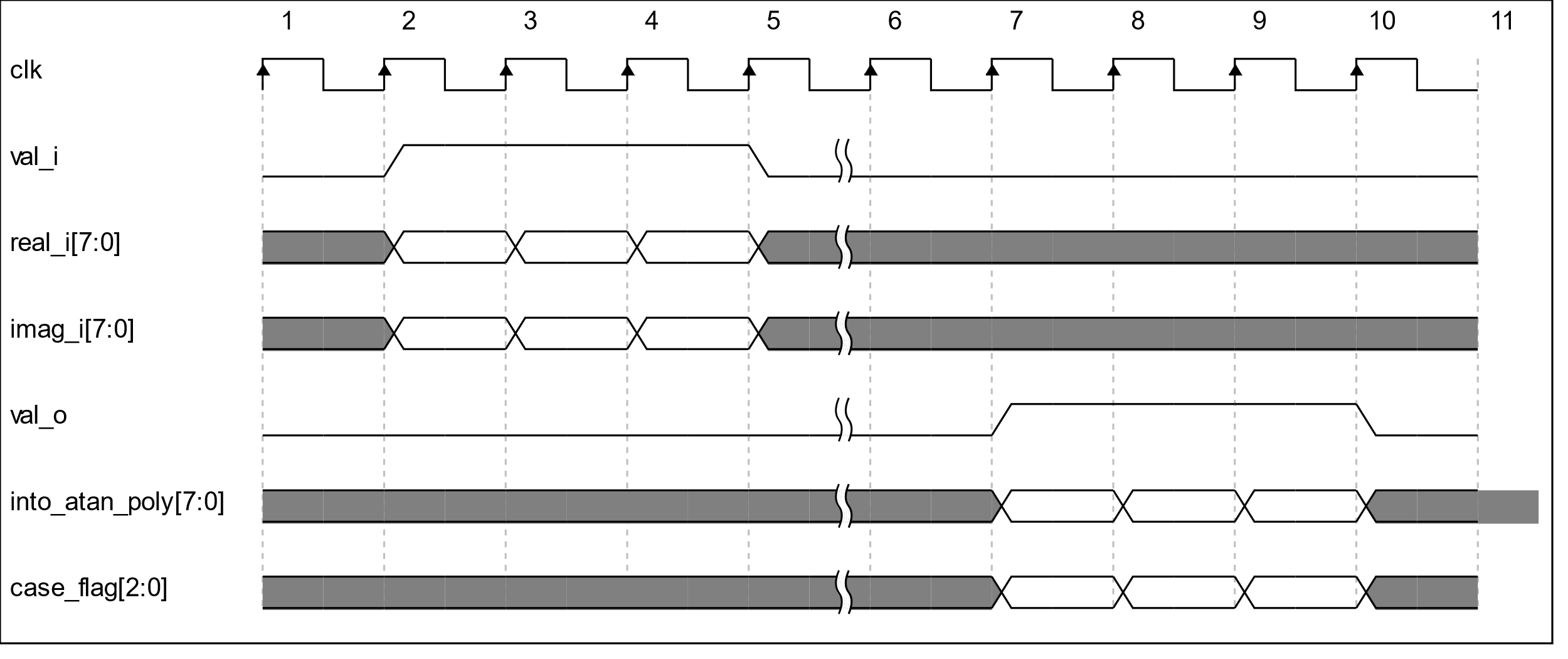




如图所示，cal\_angle模块（幅角计算模块）划为了两部分：pattern\_match和atan\_poly，延迟为7个周期。后者完成二阶拟合计算，前者负责检测输入实部虚部正负、比较两者绝对值大小，保证cal\_angle的输入数据在[0,1]范围内。具体而言，硬件实现反正切函数有两主要问题。一是虚部除以实部后，cal\_angle的输入数据理论上可接近无穷，位宽不得不随值增大；二是反正切函数自变量对应多值问题，需根据实部虚部正负值判断函数值在哪段曲线上。



利用三角函数互余特性，结合实部正负判断，可解决上述两问题。如上图所示，将反正切函数分为a、b、c、d、e、f、g六段：  
其中x为实部，y为虚部。可知，a、b、c、d段只在实部大于零时成立，e、f、g、h段在实部小于零时成立。实部等于零时，再根据虚部正负情况判断。除此之外，易得a段与剩余段之间的关系，故可只拟合自变量在[0,1]的a段，用其表示剩下五段。

故设计了pattern\_match模块，其根据实部虚部正负及绝对值大小配置后续计算模式，时序如下图所示，其中into\_atan\_poly[7:0]为输入atan\_poly信号，case\_flag指示上述八种情况中的一种：  


## 3 性能指标

### 3.1 精度及资源指标

#### 3.1.1计算误差

由于本题并未给出具体误差指标，本设计自行设计模值、幅角最大相对误差为1%。首先在MATLAB设计拟合、插值阶段即划定了误差上限。如表所示

|  | 模值计算误差 | 幅角计算误差 |
| --- | --- | --- |
| MATLAB设计 | 0.78% | 0.67% |
| C++软件仿真 | 0.78% | 0.36% |
| Verilog仿真 | 0.78% | 0.36% |

#### 3.1.2 资源占用

| Site Type | Used | Util% |
| --- | --- | --- |
| Slice LUTs | 143 | 1.79 |
| Slice Registers | 123 | 0.77 |
| MUL IP | 6 | NULL |
| BRAM IP | 1 | NULL |
| DIV IP | 1 | NULL |

### 3.2 速度指标

结合上述，模值计算延迟为3个周期，幅角运算延迟为7个周期，经时序对齐，系统整体延迟为7个周期。

结合综合工具给出的时序综合结果：

| Clock | Edges (WNS) | WNS (ns) | WHS (ns) | WPWS (ns) |
| --- | --- | --- | --- | --- |
| clk | rise - rise | 0.042 | 0.065 | 4.416 |

由  
且  
可求  
最终系统速度指标如下表所示：

| 最高工作频率 | 吞吐量 | 延迟周期数 |
| --- | --- | --- |
| 100.42Mhz | 24bit/clk | 7clk |

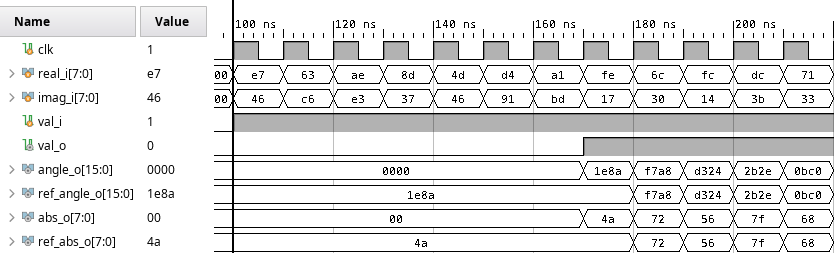
## 4 验证

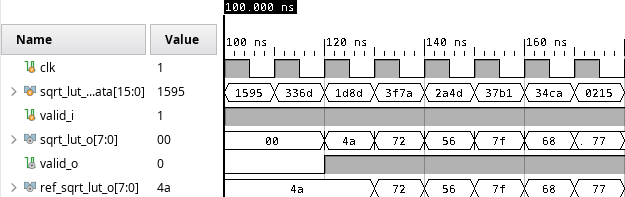
### 4.1 验证方法

为保证软硬件验证的连续性，在设计过程中先用C++模拟出了定点数计算，将函数与硬件模块对应后导出了相应输入输出值，如此在编写testbench时可直接读取这些文件，分别作为激励和参照结果。并在testbench中一一比对仿真输出与软件输出结果，所有测试结果一致或误差咋在允许范围内者即打印输出“Test passed”。如此测试方法比看波形更高效，更完整。

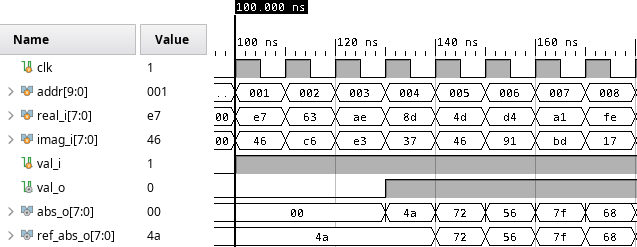
### 4.2 验证结果

cal\_abs\_angle验证结果

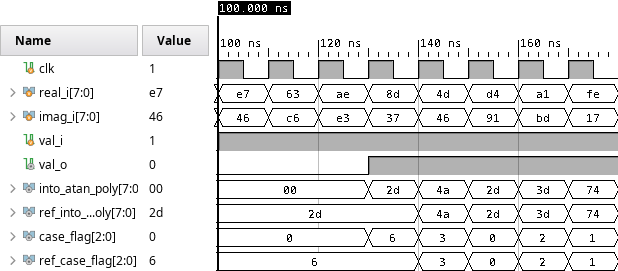


sqrt\_lut验证结果  


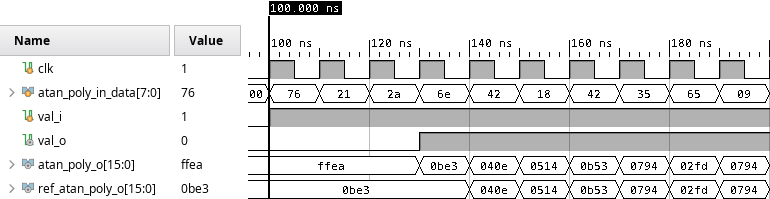
cal\_abs验证结果



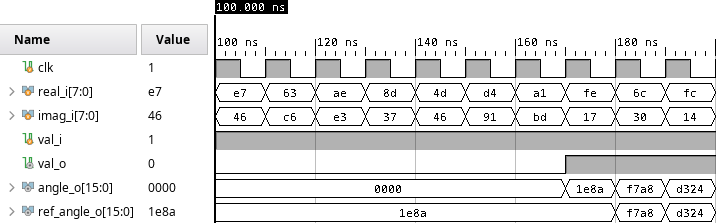
pattern\_match验证结果



atan\_poly验证结果



cal\_angle验证结果



且在命令行内，上述所有测试均打印输出“Test passed”字样。

## 5 附件

### 5.1 模块代码

module cal\_abs\_angle(

    input           clk     ,

    input [7:0]     real\_i  ,

    input [7:0]     imag\_i  ,

    input           val\_i   ,

    output [7:0]    abs\_o   ,

    output [15:0]   angle\_o ,

    output          val\_o

    );

    wire        cal\_abs\_val\_o;

    wire[7:0]   cal\_abs\_o;

    reg [7:0]   abs\_o\_r[4:0];

    cal\_abs cal\_abs\_inst(

        .clk   (clk),

        .val\_i (val\_i),

        .real\_i(real\_i),

        .imag\_i(imag\_i),

        .abs\_o (cal\_abs\_o),

        .val\_o (cal\_abs\_val\_o)

    );

    assign abs\_o = abs\_o\_r[3];

    always@(posedge clk) begin

        abs\_o\_r[0] <= cal\_abs\_o;

        abs\_o\_r[1] <= abs\_o\_r[0];

        abs\_o\_r[2] <= abs\_o\_r[1];

        abs\_o\_r[3] <= abs\_o\_r[2];

        abs\_o\_r[4] <= abs\_o\_r[3];

    end

    wire cal\_angle\_val\_o;

    cal\_angle cal\_angle\_inst(

        .clk    (clk),

        .val\_i  (val\_i),

        .real\_i (real\_i),

        .imag\_i (imag\_i),

        .angle\_o(angle\_o),

        .val\_o  (cal\_angle\_val\_o)

    );

    assign val\_o = cal\_angle\_val\_o;

endmodule

module cal\_abs(

    input           clk     ,

    input           val\_i   ,

    input [7:0]     real\_i  ,

    input [7:0]     imag\_i  ,

    output [7:0]    abs\_o   ,

    output          val\_o

    );

    wire [15:0] real\_pow\_2\_w;

    wire [15:0] imag\_pow\_2\_w;

    pow\_2\_mul\_wrapper real\_pow\_2\_mul(

        .data\_i(real\_i),

        .data\_o(real\_pow\_2\_w),

        .clk(clk)

    );

    pow\_2\_mul\_wrapper imag\_pow\_2\_mul(

        .data\_i(imag\_i),

        .data\_o(imag\_pow\_2\_w),

        .clk(clk)

    );

    reg [15:0] real\_pow\_2\_r = 0;

    reg [15:0] imag\_pow\_2\_r = 0;

    reg        val\_r = 0;

    always@(posedge clk) begin

        real\_pow\_2\_r <= real\_pow\_2\_w;

        imag\_pow\_2\_r <= imag\_pow\_2\_w;

        val\_r <= val\_i;

    end

    wire [15:0] sqrt\_lut\_in = real\_pow\_2\_w + imag\_pow\_2\_w;

    sqrt\_lut sqrt\_lut\_inst(

        .clk(clk),

        .sqrt\_lut\_i(sqrt\_lut\_in),

        .val\_i(val\_r),

        .sqrt\_lut\_o(abs\_o),

        .val\_o(val\_o)

    );

endmodule

module cal\_angle(

    input           clk     ,

    input           val\_i   ,

    input  [7: 0]   real\_i  ,

    input  [7: 0]   imag\_i  ,

    output [15: 0]  angle\_o ,

    output          val\_o

    );

    wire [2: 0] case\_flag;

    wire [7: 0] into\_atan\_poly;

    wire        pattern\_match\_val\_o;

    pattern\_match pattern\_match\_inst(

        .clk            (clk),

        .val\_i          (val\_i),

        .real\_i         (real\_i),

        .imag\_i         (imag\_i),

        .into\_atan\_poly (into\_atan\_poly),

        .case\_flag      (case\_flag),

        .val\_o          (pattern\_match\_val\_o)

    );

    wire [15: 0] atan\_poly\_o;

    wire         atan\_poly\_val\_o;

    atan\_poly atan\_poly\_inst(

        .clk        (clk) ,

        .val\_i      (pattern\_match\_val\_o) ,

        .val\_o      (atan\_poly\_val\_o) ,

        .atan\_poly\_i(into\_atan\_poly) ,

        .atan\_poly\_o(atan\_poly\_o)

    );

    reg   [1:0] val\_r = 0;

    assign      val\_o = val\_r[0];

    always@(posedge clk) begin

        val\_r[0] <= atan\_poly\_val\_o;

        val\_r[1] <= val\_r[0];

    end

    reg [15:0] pi\_div\_2\_fix  = 16'd6434;

    reg [15:0] pi\_fix        = 16'd12868;

    reg [15:0] angle\_r       = 0;

    reg [15:0] atan\_poly\_o\_r = 0;

    reg [15:0] pi\_para\_r     = 0;

    reg [8:0]  case\_flag\_r   = 0;

    always@(posedge clk) begin

        case\_flag\_r[2:0] <= case\_flag;

        case\_flag\_r[5:3] <= case\_flag\_r[2:0];

        case\_flag\_r[8:6] <= case\_flag\_r[5:3];

    end

    always@(\*) begin

        case(case\_flag\_r[8:6])

            0: begin

                atan\_poly\_o\_r =  atan\_poly\_o;

                pi\_para\_r     =  -pi\_fix;

            end

            1 : begin

                atan\_poly\_o\_r =  atan\_poly\_o;

                pi\_para\_r     =  0;

            end

            2 : begin

                atan\_poly\_o\_r =  -atan\_poly\_o;

                pi\_para\_r     =  pi\_fix;

            end

            3 : begin

                atan\_poly\_o\_r =  -atan\_poly\_o;

                pi\_para\_r     =  0;

            end

            4 : begin

                atan\_poly\_o\_r =  -atan\_poly\_o;

                pi\_para\_r     =  -pi\_div\_2\_fix;

            end

            5 : begin

                atan\_poly\_o\_r =  -atan\_poly\_o;

                pi\_para\_r     =  pi\_div\_2\_fix;

            end

            6 : begin

                atan\_poly\_o\_r =  atan\_poly\_o;

                pi\_para\_r     =  pi\_div\_2\_fix;

            end

            7 : begin

                atan\_poly\_o\_r =  atan\_poly\_o;

                pi\_para\_r     =  -pi\_div\_2\_fix;

            end

            default : begin

                atan\_poly\_o\_r =  atan\_poly\_o;

                pi\_para\_r     =  0;

            end

        endcase

    end

    assign      angle\_o = angle\_r;

    always@(posedge clk) begin

        angle\_r <= (atan\_poly\_val\_o)?(atan\_poly\_o\_r + pi\_para\_r):0;

    end

endmodule

module sqrt\_lut(

    input        clk,

    input [15:0] sqrt\_lut\_i,

    input        val\_i,

    output [7:0] sqrt\_lut\_o,

    output       val\_o

    );

    reg  [2:0]  val\_r = 0;

    assign      val\_o = val\_r[2];

    always@(posedge clk) begin

        val\_r[0] <= val\_i;

        val\_r[1] <= val\_r[0];

        val\_r[2] <= val\_r[1];

    end

    wire [19:0] co\_K;

    wire [15:0] co\_B;

    co\_K\_rom\_wrapper co\_K\_rom\_inst(

        .clk(clk),

        .val\_i(val\_r[0]),

        .read\_addr(sqrt\_lut\_i[13:4]),

        .para\_out(co\_K)

    );

    co\_B\_rom\_wrapper co\_B\_rom\_inst(

        .clk(clk),

        .val\_i(val\_r[1]),

        .read\_addr(sqrt\_lut\_i\_r[13:4]),

        .para\_out(co\_B)

    );

    reg [15:0] sqrt\_lut\_i\_r = 0;

    reg [15:0] sqrt\_lut\_i\_r\_1 = 0;

    always@(posedge clk) begin

        sqrt\_lut\_i\_r <= sqrt\_lut\_i;

        sqrt\_lut\_i\_r\_1 <= sqrt\_lut\_i\_r;

    end

    wire [29:0] mid\_mul\_K;

    mul\_k\_gen\_0\_wrapper mul\_k(

        .clk(clk),

        .co\_K(co\_K),

        .sqrt\_poly\_in(sqrt\_lut\_i\_r),

        .mid\_mul\_K(mid\_mul\_K)

    );

    wire [29:0] mid\_add\_B = mid\_mul\_K + {co\_B, 14'b0};

    reg  [29:0] mid\_add\_B\_r = 0;

    always@(posedge clk) begin

        if(val\_r[1])

            mid\_add\_B\_r <= mid\_add\_B;

        else

            mid\_add\_B\_r <= 0;

    end

    assign      sqrt\_lut\_o = mid\_add\_B[28:21];

endmodule

module pattern\_match(

    input           clk         ,

    input           val\_i       ,

    input [7:0]     real\_i      ,

    input [7:0]     imag\_i      ,

    output [7:0]    into\_atan\_poly,

    output [2:0]    case\_flag   ,

    output          val\_o

    );

    reg [3:0] val\_r = 0;

    assign    val\_o = val\_r[3];

    always@(posedge clk) begin

        val\_r[0] <= val\_i;

        val\_r[1] <= val\_r[0];

        val\_r[2] <= val\_r[1];

        val\_r[3] <= val\_r[2];

    end

    wire [7:0] abs\_real\_w = (real\_i[7]? ~real\_i + 1'b1 : real\_i) ;

    wire [7:0] abs\_imag\_w = (imag\_i[7]? ~imag\_i + 1'b1 : imag\_i) ;

    reg  [7:0] abs\_real\_r = 0;

    reg  [7:0] abs\_imag\_r = 0;

    reg        real\_sign\_r = 0;

    reg        imag\_sign\_r = 0;

    always@(posedge clk) begin

        real\_sign\_r <= real\_i[7];

        imag\_sign\_r <= imag\_i[7];

        abs\_real\_r <= abs\_real\_w;

        abs\_imag\_r <= abs\_imag\_w;

    end

    wire real\_sign = real\_sign\_r;

    wire imag\_sign = imag\_sign\_r;

    wire imag\_eq\_zero = (abs\_imag\_r == 0)?1:0;

    wire real\_eq\_zero = (abs\_real\_r == 0)?1:0;

    wire div\_sign = (real\_sign ^ imag\_sign) & (~imag\_eq\_zero) & (~real\_eq\_zero);

    wire imag\_eq\_real = (((abs\_real\_r == abs\_imag\_r)&&(~imag\_eq\_zero))?1'b1 : 1'b0);

    wire imag\_ls\_real = ((abs\_real\_r > abs\_imag\_r)?1'b1 : 1'b0);

    wire imag\_bg\_real = ((abs\_real\_r < abs\_imag\_r)?1'b1 : 1'b0);

    wire all\_zero = (~(abs\_real\_r && abs\_imag\_r));

    wire abs\_div\_within = ~div\_sign;

    wire abs\_div\_without = ((~div\_sign) && ~(real\_eq\_zero && (imag\_sign)));

    reg [2:0]   case\_flag\_r = 0;

    reg [2:0]   case\_flag\_r\_1 = 0;

    reg [15:0]  dividened\_r = 0;

    reg [7:0]   divisor\_r = 0;

    assign      case\_flag = case\_flag\_r\_1;

    always@(posedge clk) begin

        case\_flag\_r\_1 <= case\_flag\_r;

    end

    always@(posedge clk) begin

        if(all\_zero && val\_r[1]) begin

            case\_flag\_r <= 0;

            dividened\_r <= 0;

            divisor\_r <= 1;

        end

        if((imag\_ls\_real || imag\_eq\_real) && val\_r[1]) begin

            dividened\_r <= {abs\_imag\_r, 7'b0};

            divisor\_r <= abs\_real\_r;

            if(abs\_div\_within) begin

                if(real\_sign)

                    case\_flag\_r <= 0;

                else

                    case\_flag\_r <= 1;

            end

            else begin

                if(real\_sign)

                    case\_flag\_r <= 2;

                else

                    case\_flag\_r <= 3;

            end

        end

        if(imag\_bg\_real && val\_r[1]) begin

            dividened\_r <= {abs\_real\_r, 7'b0};

            divisor\_r <= abs\_imag\_r;

            if(abs\_div\_without) begin

                if(real\_sign)

                    case\_flag\_r <= 4;

                else

                    case\_flag\_r <= 5;

            end

            else begin

                if(real\_sign)

                    case\_flag\_r <= 6;

                else

                    case\_flag\_r <= 7;

            end

        end

    end

    wire    [23:0]  div\_res;

    wire            div\_val\_o;

    assign          into\_atan\_poly = div\_res[15:8];

    cal\_angle\_div\_wrapper cal\_angle\_div(

        .clk      (clk) ,

        .dividened(dividened\_r) ,  // [15:0]

        .divisor  (divisor\_r) ,  // [7:0]

        .val\_i    (val\_r[2]) ,

        .val\_o    (div\_val\_o) ,

        .div\_res  (div\_res)    // [23:0]

    );

endmodule

### 5.2 验证代码

module cal\_abs\_angle\_tb();

    localparam  N = 1024;

    reg        clk = 1;

    localparam period = 10;

    initial begin

       forever begin

         #(period/2) clk = ~clk;

       end

    end

    reg [7:0]   real\_mem [0:N-1];

    reg [7:0]   imag\_mem [0:N-1];

    reg [7:0]   res\_sqrt\_out\_mem [0:N-1];

    reg [15:0]  res\_angle\_cal\_mem [0:N-1];

    integer     file\_real\_value; // file\_real\_value handle

    integer     file\_imag\_value; // file\_real\_value handle

    integer     file\_res\_sqrt\_out; // file\_res\_sqrt\_out handle

    integer     file\_res\_angle\_cal\_res; // file\_res\_sqrt\_out handle

    integer     cnt; // counter

    // read stimulus and result from file\_real\_value

    initial begin

        file\_real\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_real\_value.txt", "r");

        file\_imag\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_imag\_value.txt", "r");

        file\_res\_sqrt\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/res\_sqrt\_out.txt", "r");

        file\_res\_angle\_cal\_res = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/res\_angle\_cal\_res.txt", "r");

        cnt = 0;

        while (!$feof(file\_real\_value)) begin

            $fscanf(file\_real\_value, "%h", real\_mem[cnt]);

            $fscanf(file\_imag\_value, "%h", imag\_mem[cnt]);

            $fscanf(file\_res\_sqrt\_out, "%h", res\_sqrt\_out\_mem[cnt]);

            $fscanf(file\_res\_angle\_cal\_res, "%h", res\_angle\_cal\_mem[cnt]);

            cnt = cnt + 1;

        end

        $fclose(file\_real\_value);

        $fclose(file\_imag\_value);

        $fclose(file\_res\_sqrt\_out);

        $fclose(file\_res\_angle\_cal\_res);

    end

    // output stimulus on rising edge of clock

    reg [9:0]   addr = 0; // address of current stimulus real\_i

    reg [7:0]   real\_i = 0; // stimulus real\_i

    reg [7:0]   imag\_i = 0; // stimulus imag\_i

    reg         val\_i   = 0;

    initial begin

        # 100 val\_i = 1;

        forever@(posedge clk) begin

            addr <= addr + 1;

            real\_i <= real\_mem[addr];

            imag\_i <= imag\_mem[addr];

        end

    end

    // Instantiation

    wire [15:0]  angle\_o;

    wire [7:0]  abs\_o;

    wire        val\_o;

    cal\_abs\_angle cal\_abs\_angle\_inst(

        .clk    (clk) ,

        .real\_i (real\_i) ,

        .imag\_i (imag\_i) ,

        .val\_i  (val\_i) ,

        .abs\_o  (abs\_o) ,

        .angle\_o(angle\_o) ,

        .val\_o  (val\_o)

    );

    // Check the result

    integer        i\_check = 0;

    wire    [7:0]  ref\_abs\_o = res\_sqrt\_out\_mem[i\_check];

    wire    [15:0] ref\_angle\_o = res\_angle\_cal\_mem[i\_check];

    wire    [15:0] angle\_o\_sim\_err = (ref\_angle\_o > angle\_o) ? ref\_angle\_o - angle\_o: angle\_o - ref\_angle\_o;

    wire    [7:0]  abs\_o\_sim\_err = (ref\_abs\_o > abs\_o) ? ref\_abs\_o - abs\_o: abs\_o - ref\_abs\_o;

    reg     [15:0] max\_angle\_err = 0;

    reg     [7:0]  max\_abs\_err = 0;

    initial begin

        forever@(posedge clk) begin

            if(val\_o) begin

                i\_check <= i\_check + 1;

                if(max\_angle\_err < angle\_o\_sim\_err)

                    max\_angle\_err <= angle\_o\_sim\_err;

                if(max\_abs\_err < abs\_o\_sim\_err)

                    max\_abs\_err <= abs\_o\_sim\_err;

                if((angle\_o\_sim\_err > 1)|| (ref\_abs\_o != abs\_o)) begin

                    $display("There is a problem at %d. Simulation stopped.", i\_check + 1);

                    $stop( 0 ) ;

                end

                else if(i\_check == N-1) begin

                    $display("Test passed.");

                    $finish( 0 ) ;

                end

            end

        end

    end

endmodule

module cal\_abs\_tb();

    localparam  N = 1024;

    reg        clk = 1;

    localparam period = 10;

    initial begin

       forever begin

         #(period/2) clk = ~clk;

       end

    end

    reg [7:0]   real\_mem [0:N-1];

    reg [7:0]   imag\_mem [0:N-1];

    reg [7:0]   abs\_out\_mem [0:N-1];

    integer     file\_real\_value; // file\_real\_value handle

    integer     file\_imag\_value; // file\_real\_value handle

    integer     file\_abs\_out; // file\_abs\_out handle

    integer     cnt; // counter

    // read stimulus and result from file\_real\_value

    initial begin

        file\_real\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_real\_value.txt", "r");

        file\_imag\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_imag\_value.txt", "r");

        file\_abs\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/res\_sqrt\_out.txt", "r");

        cnt = 0;

        while (!$feof(file\_real\_value)) begin

            $fscanf(file\_real\_value, "%h", real\_mem[cnt]);

            $fscanf(file\_imag\_value, "%h", imag\_mem[cnt]);

            $fscanf(file\_abs\_out, "%h", abs\_out\_mem[cnt]);

            cnt = cnt + 1;

        end

        $fclose(file\_real\_value);

        $fclose(file\_imag\_value);

        $fclose(file\_abs\_out);

    end

    // output stimulus on rising edge of clock

    reg [9:0]   addr = 0; // address of current stimulus real\_i

    reg [7:0]   real\_i = 0; // stimulus real\_i

    reg [7:0]   imag\_i = 0; // stimulus imag\_i

    reg         val\_i   = 0;

    initial begin

        # 100 val\_i = 1;

        forever@(posedge clk) begin

            addr <= addr + 1;

            real\_i <= real\_mem[addr];

            imag\_i <= imag\_mem[addr];

        end

    end

    // Instantiation

    wire [7:0]  abs\_o;

    wire        val\_o;

    cal\_abs cal\_abs\_inst(

        .clk   (clk),

        .val\_i (val\_i),

        .real\_i(real\_i),

        .imag\_i(imag\_i),

        .abs\_o (abs\_o),

        .val\_o (val\_o)

    );

    // Check the result

    integer i\_check = 0;

    wire [7:0] ref\_abs\_o = abs\_out\_mem[i\_check];

    initial begin

        forever@(posedge clk) begin

            if(val\_o) begin

                i\_check <= i\_check + 1;

                if(ref\_abs\_o !=  abs\_o) begin

                    $display("There is a problem at %d. Simulation stopped.", cnt);

                    $stop( 0 ) ;

                end

                else if(i\_check == N-1) begin

                    $display("Test passed.");

                    $finish( 0 ) ;

                end

            end

        end

    end

endmodule

module cal\_angle\_tb();

    localparam  N = 1024;

    reg        clk = 1;

    localparam period = 10;

    initial begin

       forever begin

         #(period/2) clk = ~clk;

       end

    end

    reg [7:0]   real\_mem [0:N-1];

    reg [7:0]   imag\_mem [0:N-1];

    reg [15:0]   res\_angle\_cal\_out\_mem [0:N-1];

    integer     file\_real\_value; // file\_real\_value handle

    integer     file\_imag\_value; // file\_real\_value handle

    integer     file\_res\_angle\_cal\_out; // file\_res\_angle\_cal\_out handle

    integer     cnt; // counter

    // read stimulus and result from file\_real\_value

    initial begin

        file\_real\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_real\_value.txt", "r");

        file\_imag\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_imag\_value.txt", "r");

        file\_res\_angle\_cal\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/res\_angle\_cal\_res.txt", "r");

        cnt = 0;

        while (!$feof(file\_real\_value)) begin

            $fscanf(file\_real\_value, "%h", real\_mem[cnt]);

            $fscanf(file\_imag\_value, "%h", imag\_mem[cnt]);

            $fscanf(file\_res\_angle\_cal\_out, "%h", res\_angle\_cal\_out\_mem[cnt]);

            cnt = cnt + 1;

        end

        $fclose(file\_real\_value);

        $fclose(file\_imag\_value);

        $fclose(file\_res\_angle\_cal\_out);

    end

    // output stimulus on rising edge of clock

    reg [9:0]   addr = 0; // address of current stimulus real\_i

    reg [7:0]   real\_i = 0; // stimulus real\_i

    reg [7:0]   imag\_i = 0; // stimulus imag\_i

    reg         val\_i   = 0;

    initial begin

        # 100 val\_i = 1;

        forever@(posedge clk) begin

            addr <= addr + 1;

            real\_i <= real\_mem[addr];

            imag\_i <= imag\_mem[addr];

        end

    end

    // Instantiation

    wire [15:0] angle\_o;

    wire        val\_o;

    cal\_angle cal\_angle\_inst(

        .clk    (clk),

        .val\_i  (val\_i),

        .real\_i (real\_i),

        .imag\_i (imag\_i),

        .angle\_o(angle\_o),

        .val\_o  (val\_o)

    );

    // Check the result

    integer        i\_check = 0;

    wire    [15:0] ref\_angle\_o = res\_angle\_cal\_out\_mem[i\_check];

    wire    [15:0] sim\_err = (ref\_angle\_o > angle\_o) ? ref\_angle\_o - angle\_o: angle\_o - ref\_angle\_o;

    initial begin

        forever@(posedge clk) begin

            if(val\_o) begin

                i\_check <= i\_check + 1;

                if(sim\_err > 1) begin

                    $display("There is a problem at %d. Simulation stopped.", i\_check + 1);

                    $stop( 0 ) ;

                end

                else if(i\_check == N-1) begin

                    $display("Test passed.");

                    $finish( 0 ) ;

                end

            end

        end

    end

endmodule

module atan\_poly\_tb();

    localparam  N = 1024;

    reg                     clk = 1;

    localparam period = 10;

    initial begin

       forever begin

         #(period/2) clk = ~clk;

       end

    end

    reg [7:0]  atan\_in\_mem [0:N-1];

    reg [15:0]   atan\_out\_mem [0:N-1];

    integer     file\_atan\_in; // file\_atan\_in handle

    integer     file\_atan\_out; // file\_atan\_out handle

    integer     cnt; // counter

    // read stimulus and result from file\_atan\_in

    initial begin

        file\_atan\_in = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/mid\_atan\_in.txt", "r");

        file\_atan\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/res\_atan\_out.txt", "r");

        cnt = 0;

        while (!$feof(file\_atan\_in)) begin

            $fscanf(file\_atan\_in, "%h", atan\_in\_mem[cnt]);

            $fscanf(file\_atan\_out, "%h", atan\_out\_mem[cnt]);

            cnt = cnt + 1;

        end

        $fclose(file\_atan\_in);

        $fclose(file\_atan\_out);

    end

    // output stimulus on rising edge of clock

    reg [9:0]   addr = 0; // address of current stimulus atan\_poly\_in\_data

    reg [7:0]  atan\_poly\_in\_data = 0; // stimulus atan\_poly\_in\_data

    reg         val\_i   = 0;

    initial begin

        # 100 val\_i = 1;

        forever@(posedge clk) begin

            addr <= addr + 1;

            atan\_poly\_in\_data <= atan\_in\_mem[addr];

        end

    end

    // Instantiation

    wire [15:0] atan\_poly\_o;

    wire        val\_o;

    atan\_poly atan\_poly\_inst(

        .clk        (clk) ,

        .val\_i      (val\_i) ,

        .val\_o      (val\_o) ,

        .atan\_poly\_i(atan\_poly\_in\_data) ,

        .atan\_poly\_o(atan\_poly\_o)

    );

    // Check the result

    integer i\_check = 0;

    wire [15:0] ref\_atan\_poly\_o = atan\_out\_mem[i\_check];

    initial begin

        forever@(posedge clk) begin

            if(val\_o) begin

                i\_check <= i\_check + 1;

                if(ref\_atan\_poly\_o !=  atan\_poly\_o) begin

                    $display("There is a problem at index %d. Simulation stopped.", i\_check);

                    $stop( 0 ) ;

                end

                else if(i\_check == N-1) begin

                    $display("Test passed.");

                    $finish( 0 ) ;

                end

            end

        end

    end

endmodule

module pattern\_match\_tb();

    localparam  N = 1024;

    reg        clk = 1;

    localparam period = 10;

    initial begin

       forever begin

         #(period/2) clk = ~clk;

       end

    end

    reg [7:0]   real\_mem [0:N-1];

    reg [7:0]   imag\_mem [0:N-1];

    reg [7:0]   mid\_atan\_out\_mem [0:N-1];

    reg [2:0]   case\_flag\_out\_mem [0:N-1];

    integer     file\_real\_value; // file\_real\_value handle

    integer     file\_imag\_value; // file\_real\_value handle

    integer     file\_mid\_atan\_out; // file\_mid\_atan\_out handle

    integer     file\_case\_flag\_out; // file\_mid\_atan\_out handle

    integer     cnt; // counter

    // read stimulus and result from file\_real\_value

    initial begin

        file\_real\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_real\_value.txt", "r");

        file\_imag\_value = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/input\_imag\_value.txt", "r");

        file\_mid\_atan\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/mid\_atan\_out.txt", "r");

        file\_case\_flag\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/mid\_case\_flag\_out.txt", "r");

        cnt = 0;

        while (!$feof(file\_real\_value)) begin

            $fscanf(file\_real\_value, "%h", real\_mem[cnt]);

            $fscanf(file\_imag\_value, "%h", imag\_mem[cnt]);

            $fscanf(file\_mid\_atan\_out, "%h", mid\_atan\_out\_mem[cnt]);

            $fscanf(file\_case\_flag\_out, "%h", case\_flag\_out\_mem[cnt]);

            cnt = cnt + 1;

        end

        $fclose(file\_real\_value);

        $fclose(file\_imag\_value);

        $fclose(file\_mid\_atan\_out);

        $fclose(file\_case\_flag\_out);

    end

    // output stimulus on rising edge of clock

    reg [9:0]   addr = 0; // address of current stimulus real\_i

    reg [7:0]   real\_i = 0; // stimulus real\_i

    reg [7:0]   imag\_i = 0; // stimulus imag\_i

    reg         val\_i   = 0;

    initial begin

        # 100 val\_i = 1;

        forever@(posedge clk) begin

            addr <= addr + 1;

            real\_i <= real\_mem[addr];

            imag\_i <= imag\_mem[addr];

        end

    end

    // Instantiation

    wire [2:0]  case\_flag;

    wire [7:0]  into\_atan\_poly;

    wire        val\_o;

    pattern\_match pattern\_match\_inst(

        .clk         (clk),

        .val\_i       (val\_i),

        .real\_i      (real\_i),

        .imag\_i      (imag\_i),

        .into\_atan\_poly(into\_atan\_poly),

        .case\_flag   (case\_flag),

        .val\_o       (val\_o)

    );

    // Check the result

    integer        i\_check = 0;

    wire    [7:0]  ref\_into\_atan\_poly = mid\_atan\_out\_mem[i\_check];

    wire    [2:0]  ref\_case\_flag = case\_flag\_out\_mem[i\_check];

    initial begin

        forever@(posedge clk) begin

            if(val\_o) begin

                i\_check <= i\_check + 1;

                if((ref\_into\_atan\_poly !=  into\_atan\_poly)|| (ref\_case\_flag != case\_flag)) begin

                    $display("There is a problem at %d. Simulation stopped.", i\_check + 1);

                    $stop( 0 ) ;

                end

                else if(i\_check == N-1) begin

                    $display("Test passed.");

                    $finish( 0 ) ;

                end

            end

        end

    end

endmodule

module sqrt\_lut\_tb();

    localparam  N = 1024;

    reg                     clk = 1;

    localparam period = 10;

    initial begin

       forever begin

         #(period/2) clk = ~clk;

       end

    end

    reg [15:0]  sqrt\_in\_mem [0:N-1];

    reg [7:0]   sqrt\_out\_mem [0:N-1];

    integer     file\_sqrt\_in; // file\_sqrt\_in handle

    integer     file\_sqrt\_out; // file\_sqrt\_out handle

    integer     cnt; // counter

    // read stimulus and result from file\_sqrt\_in

    initial begin

        file\_sqrt\_in = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/mid\_sqrt\_in.txt", "r");

        file\_sqrt\_out = $fopen("/media/aaa/I/\_ELSE/fpga\_prj/fpga\_design\_course/task\_3/design\_src/src.c/para\_and\_res/res\_sqrt\_out.txt", "r");

        cnt = 0;

        while (!$feof(file\_sqrt\_in)) begin

            $fscanf(file\_sqrt\_in, "%h", sqrt\_in\_mem[cnt]);

            $fscanf(file\_sqrt\_out, "%h", sqrt\_out\_mem[cnt]);

            cnt = cnt + 1;

        end

        $fclose(file\_sqrt\_in);

        $fclose(file\_sqrt\_out);

    end

    // output stimulus on rising edge of clock

    reg [9:0]   addr = 0; // address of current stimulus sqrt\_lut\_in\_data

    reg [15:0]  sqrt\_lut\_in\_data = 0; // stimulus sqrt\_lut\_in\_data

    reg         valid\_i = 0;

    initial begin

        # 100 valid\_i   = 1;

        forever@(posedge clk) begin

            addr <= addr + 1;

            sqrt\_lut\_in\_data <= sqrt\_in\_mem[addr];

        end

    end

    // Instantiation

    wire [7:0]  sqrt\_lut\_o;

    wire        valid\_o;

    sqrt\_lut sqrt\_lut\_inst(

        .clk(clk),

        .sqrt\_lut\_i(sqrt\_lut\_in\_data),

        .val\_i(valid\_i),

        .sqrt\_lut\_o(sqrt\_lut\_o),

        .val\_o(valid\_o)

        );

    // Check the result

    integer i\_check = 0;

    wire [7:0] ref\_sqrt\_lut\_o = sqrt\_out\_mem[i\_check];

    initial begin

        forever@(posedge clk) begin

            if(valid\_o) begin

                i\_check <= i\_check + 1;

                if(ref\_sqrt\_lut\_o !=  sqrt\_lut\_o) begin

                    $display("There is a problem at %d. Simulation stopped.", cnt);

                    $stop( 0 ) ;

                end

                else if(i\_check == N-1) begin

                    $display("Test passed.");

                    $finish( 0 ) ;

                end

            end

        end

    end

endmodule