一、IP 核设计:

1.axi_rw_v1_0.v

```
`timescale 1 ns / 1 ps

    module axi_rw_v1_0 #
    (
        parameter C_M00_AXI_register_address = 32'h00FFFFF8,
        parameter C_M00_AXI_TARGET_SLAVE_BASE_ADDR      = 32'h01000000,
        parameter integer C_M00_AXI_BURST_LEN      = 16,
        parameter integer C_M00_AXI_ID_WIDTH  = 1,
        parameter integer C_M00_AXI_ADDR_WIDTH   = 32,
        parameter integer C_M00_AXI_DATA_WIDTH    = 64,
        parameter integer C_M00_AXI_AWUSER_WIDTH      = 0,
        parameter integer C_M00_AXI_ARUSER_WIDTH      = 0,
        parameter integer C_M00_AXI_WUSER_WIDTH = 0,
        parameter integer C_M00_AXI_RUSER_WIDTH  = 0,
        parameter integer C_M00_AXI_BUSER_WIDTH  = 0
    )
    (
        input   wire m00_axi_init_axi_txn,
        output wire   m00_axi_txn_done,
        input wire   m00_axi_aclk,
        input wire   m00_axi_aresetn,
        output wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_awid,
        output wire [C_M00_AXI_ADDR_WIDTH-1 : 0] m00_axi_awaddr,
        output wire [7 : 0] m00_axi_awlen,
        output wire [2 : 0] m00_axi_awsize,
        output wire [1 : 0] m00_axi_awburst,
        output wire   m00_axi_awlock,
        output wire [3 : 0] m00_axi_awcache,
        output wire [2 : 0] m00_axi_awprot,
        output wire [3 : 0] m00_axi_awqos,
        output wire [C_M00_AXI_AWUSER_WIDTH-1 : 0] m00_axi_awuser,
        output wire   m00_axi_awvalid,
        input wire   m00_axi_awready,
        output wire [C_M00_AXI_DATA_WIDTH-1 : 0] m00_axi_wdata,
        output wire [C_M00_AXI_DATA_WIDTH/8-1 : 0] m00_axi_wstrb,
        output wire   m00_axi_wlast,
        output wire [C_M00_AXI_WUSER_WIDTH-1 : 0] m00_axi_wuser,
        output wire   m00_axi_wvalid,
        input wire   m00_axi_wready,
        input wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_bid,
```

```
        input wire [1 : 0] m00_axi_bresp,
        input wire [C_M00_AXI_BUSER_WIDTH-1 : 0] m00_axi_buser,
        input wire    m00_axi_bvalid,
        output wire    m00_axi_bready,
        output wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_arid,
        output wire [C_M00_AXI_ADDR_WIDTH-1 : 0] m00_axi_araddr,
        output wire [7 : 0] m00_axi_arlen,
        output wire [2 : 0] m00_axi_arsize,
        output wire [1 : 0] m00_axi_arburst,
        output wire    m00_axi_arlock,
        output wire [3 : 0] m00_axi_arcache,
        output wire [2 : 0] m00_axi_arprot,
        output wire [3 : 0] m00_axi_arqos,
        output wire [C_M00_AXI_ARUSER_WIDTH-1 : 0] m00_axi_aruser,
        output wire    m00_axi_arvalid,
        input wire    m00_axi_arready,
        input wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_rid,
        input wire [C_M00_AXI_DATA_WIDTH-1 : 0] m00_axi_rdata,
        input wire [1 : 0] m00_axi_rresp,
        input wire    m00_axi_rlast,
        input wire [C_M00_AXI_RUSER_WIDTH-1 : 0] m00_axi_ruser,
        input wire    m00_axi_rvalid,
        output wire    m00_axi_rready
    );


    axi_rw_v1_0_M00_AXI # (
         .register_address(C_M00_AXI_register_address),
         .C_M_TARGET_SLAVE_BASE_ADDR(C_M00_AXI_TARGET_SLAVE_BASE_ADDR),
         .C_M_AXI_BURST_LEN(C_M00_AXI_BURST_LEN),
         .C_M_AXI_ID_WIDTH(C_M00_AXI_ID_WIDTH),
         .C_M_AXI_ADDR_WIDTH(C_M00_AXI_ADDR_WIDTH),
         .C_M_AXI_DATA_WIDTH(C_M00_AXI_DATA_WIDTH),
         .C_M_AXI_AWUSER_WIDTH(C_M00_AXI_AWUSER_WIDTH),
         .C_M_AXI_ARUSER_WIDTH(C_M00_AXI_ARUSER_WIDTH),
         .C_M_AXI_WUSER_WIDTH(C_M00_AXI_WUSER_WIDTH),
         .C_M_AXI_RUSER_WIDTH(C_M00_AXI_RUSER_WIDTH),
         .C_M_AXI_BUSER_WIDTH(C_M00_AXI_BUSER_WIDTH)
    ) axi_rw_v1_0_M00_AXI_inst (
         .INIT_AXI_TXN(m00_axi_init_axi_txn),
         .TXN_DONE(m00_axi_txn_done),
         .M_AXI_ACLK(m00_axi_aclk),
         .M_AXI_ARESETN(m00_axi_aresetn),
         .M_AXI_AWID(m00_axi_awid),
```

```
        .M_AXI_AWADDR(m00_axi_awaddr),
        .M_AXI_AWLEN(m00_axi_awlen),
        .M_AXI_AWSIZE(m00_axi_awsize),
        .M_AXI_AWBURST(m00_axi_awburst),
        .M_AXI_AWLOCK(m00_axi_awlock),
        .M_AXI_AWCACHE(m00_axi_awcache),
        .M_AXI_AWPROT(m00_axi_awprot),
        .M_AXI_AWQOS(m00_axi_awqos),
        .M_AXI_AWUSER(m00_axi_awuser),
        .M_AXI_AWVALID(m00_axi_awvalid),
        .M_AXI_AWREADY(m00_axi_awready),
        .M_AXI_WDATA(m00_axi_wdata),
        .M_AXI_WSTRB(m00_axi_wstrb),
        .M_AXI_WLAST(m00_axi_wlast),
        .M_AXI_WUSER(m00_axi_wuser),
        .M_AXI_WVALID(m00_axi_wvalid),
        .M_AXI_WREADY(m00_axi_wready),
        .M_AXI_BID(m00_axi_bid),
        .M_AXI_BRESP(m00_axi_bresp),
        .M_AXI_BUSER(m00_axi_buser),
        .M_AXI_BVALID(m00_axi_bvalid),
        .M_AXI_BREADY(m00_axi_bready),
        .M_AXI_ARID(m00_axi_arid),
        .M_AXI_ARADDR(m00_axi_araddr),
        .M_AXI_ARLEN(m00_axi_arlen),
        .M_AXI_ARSIZE(m00_axi_arsize),
        .M_AXI_ARBURST(m00_axi_arburst),
        .M_AXI_ARLOCK(m00_axi_arlock),
        .M_AXI_ARCACHE(m00_axi_arcache),
        .M_AXI_ARPROT(m00_axi_arprot),
        .M_AXI_ARQOS(m00_axi_arqos),
        .M_AXI_ARUSER(m00_axi_aruser),
        .M_AXI_ARVALID(m00_axi_arvalid),
        .M_AXI_ARREADY(m00_axi_arready),
        .M_AXI_RID(m00_axi_rid),
        .M_AXI_RDATA(m00_axi_rdata),
        .M_AXI_RRESP(m00_axi_rresp),
        .M_AXI_RLAST(m00_axi_rlast),
        .M_AXI_RUSER(m00_axi_ruser),
        .M_AXI_RVALID(m00_axi_rvalid),
        .M_AXI_RREADY(m00_axi_rready)
    );
    endmodule
```

2.axi_rw_v1_0_M00_AXI.v

```
`timescale 1 ns / 1 ps

	module axi_rw_v1_0_M00_AXI #
	(
		parameter	register_address = 32'h00FFFFF8,
		parameter	C_M_TARGET_SLAVE_BASE_ADDR = 32'h01000000,
		parameter integer C_M_AXI_BURST_LEN	= 16,
		parameter integer C_M_AXI_ID_WIDTH	= 1,
		parameter integer C_M_AXI_ADDR_WIDTH	= 32,
		parameter integer C_M_AXI_DATA_WIDTH = 64,
		parameter integer C_M_AXI_AWUSER_WIDTH	= 0,
		parameter integer C_M_AXI_ARUSER_WIDTH	= 0,
		parameter integer C_M_AXI_WUSER_WIDTH	= 0,
		parameter integer C_M_AXI_RUSER_WIDTH	= 0,
		parameter integer C_M_AXI_BUSER_WIDTH	= 0
	)
	(
		input wire	INIT_AXI_TXN,
		output reg	TXN_DONE,
		input wire	M_AXI_ACLK,
		input wire	M_AXI_ARESETN,
		output wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_AWID,
		output wire [C_M_AXI_ADDR_WIDTH-1 : 0] M_AXI_AWADDR,
		output wire [7 : 0] M_AXI_AWLEN,
		output wire [2 : 0] M_AXI_AWSIZE,
		output wire [1 : 0] M_AXI_AWBURST,
		output wire	M_AXI_AWLOCK,
		output wire [3 : 0] M_AXI_AWCACHE,
		output wire [2 : 0] M_AXI_AWPROT,
		output wire [3 : 0] M_AXI_AWQOS,
		output wire [C_M_AXI_AWUSER_WIDTH-1 : 0] M_AXI_AWUSER,
		output wire	M_AXI_AWVALID,
		input wire	M_AXI_AWREADY,
		output wire [C_M_AXI_DATA_WIDTH-1 : 0] M_AXI_WDATA,
		output wire [C_M_AXI_DATA_WIDTH/8-1 : 0] M_AXI_WSTRB,
		output wire	M_AXI_WLAST,
		output wire [C_M_AXI_WUSER_WIDTH-1 : 0] M_AXI_WUSER,
		output wire	M_AXI_WVALID,
		input wire	M_AXI_WREADY,
		input wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_BID,
		input wire [1 : 0] M_AXI_BRESP,
		input wire [C_M_AXI_BUSER_WIDTH-1 : 0] M_AXI_BUSER,

`
```

```
            input wire    M_AXI_BVALID,
            output wire    M_AXI_BREADY,
            output wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_ARID,
            output wire [C_M_AXI_ADDR_WIDTH-1 : 0] M_AXI_ARADDR,
            output wire [7 : 0] M_AXI_ARLEN,
            output wire [2 : 0] M_AXI_ARSIZE,
            output wire [1 : 0] M_AXI_ARBURST,
            output wire    M_AXI_ARLOCK,
            output wire [3 : 0] M_AXI_ARCACHE,
            output wire [2 : 0] M_AXI_ARPROT,
            output wire [3 : 0] M_AXI_ARQOS,
            output wire [C_M_AXI_ARUSER_WIDTH-1 : 0] M_AXI_ARUSER,
            output wire    M_AXI_ARVALID, //
            input wire    M_AXI_ARREADY,
            input wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_RID,
            input wire [C_M_AXI_DATA_WIDTH-1 : 0] M_AXI_RDATA,
            input wire [1 : 0] M_AXI_RRESP,
            input wire    M_AXI_RLAST,
            input wire [C_M_AXI_RUSER_WIDTH-1 : 0] M_AXI_RUSER,
            input wire    M_AXI_RVALID,
            output wire    M_AXI_RREADY
        );

//log2 操作，用来计算数据位宽
         function integer clogb2 (input integer bit_depth);
         begin
           for(clogb2=0; bit_depth>0; clogb2=clogb2+1)
             bit_depth = bit_depth >> 1;
           end
         endfunction

//单次突发的长度
        localparam integer C_TRANSACTIONS_NUM = clogb2(C_M_AXI_BURST_LEN-1);

//一般情况,处理 16 位数据总个数为  2^(C_MASTER_LENGTH-1),非一般情况,通过  fixed_shift  和
register_number  在  always  判断语句中进行改变
        localparam integer C_MASTER_LENGTH   = 13;

//通过除以单次突发处理的数据,得到一般情况下,16 位定点方式所需要的突发次数是
2^(C_NO_BURSTS_REQ-1)
        localparam integer C_NO_BURSTS_REQ
= C_MASTER_LENGTH-clogb2((C_M_AXI_BURST_LEN*C_M_AXI_DATA_WIDTH/8)-1);
```

`

```
 //以下声明用于数据处理
        wire clk2; //e 指数运算模块时钟
        wire read_enable; //读数据总线使能
        reg second_enable; //指数运算模块使能
        reg long_en; //用于延长 clk2 信号,此时全读状态机已经完成,使得仍处于数据流水线中没有输出的
数据顺利输出
        wire [63:0] data_64;    //同步 fifo 数据输出
        wire fifo_rd_enable;    //同步 fifo 数据读使能
        wire fifo_wr_enable;    //同步 fifo 数据写使能
        reg data_valid;    //表明数据经过流水线处理,在有限的延迟后,当前的数据有效,可以存入 fifo 中

//多通道拟合器输入数据
        wire [15:0] data_in_0;
        wire [15:0] data_in_1;
        wire [15:0] data_in_2;
        wire [15:0] data_in_3;
        wire [7:0] data_in_4;
        wire [7:0] data_in_5;
        wire [7:0] data_in_6;
        wire [7:0] data_in_7;

//多通道除法器输入数据
        wire [15:0] data_in_0_d;
        wire [15:0] data_in_1_d;
        wire [15:0] data_in_2_d;
        wire [15:0] data_in_3_d;
        wire [7:0] data_in_4_d;
        wire [7:0] data_in_5_d;
        wire [7:0] data_in_6_d;
        wire [7:0] data_in_7_d;

//多通道拟合器输出数据
        wire [15:0] data_out_0;
        wire [15:0] data_out_1;
        wire [15:0] data_out_2;
        wire [15:0] data_out_3;
        wire [7:0] data_out_4;
        wire [7:0] data_out_5;
        wire [7:0] data_out_6;
        wire [7:0] data_out_7;

//输出 64 位数据
        reg [63:0] data_out_64;
```

```
//8 定点饱和近似
        wire [7:0] data_out3_buf_8fix;
        wire [7:0] data_out2_buf_8fix;
        wire [7:0] data_out1_buf_8fix;
        wire [7:0] data_out0_buf_8fix;

//多通道除法器输出数据
        wire [15:0] data_out_0_d;
        wire [15:0] data_out_1_d;
        wire [15:0] data_out_2_d;
        wire [15:0] data_out_3_d;
        wire [7:0] data_out_4_d;
        wire [7:0] data_out_5_d;
        wire [7:0] data_out_6_d;
        wire [7:0] data_out_7_d;

//单通道累加值
        reg [31:0] accumulator_0;
        reg [31:0] accumulator_1;
        reg [31:0] accumulator_2;
        reg [31:0] accumulator_3;
        reg [25:0] accumulator_4;
        reg [25:0] accumulator_5;
        reg [25:0] accumulator_6;
        reg [25:0] accumulator_7;
        wire [33:0] accumulator_result; //最终累加结果

        wire clk3;     //除法器模块时钟
        reg third_enable; //触发器模块使能

        reg [15:0] reg_max;    //最大值寄存器,用于存储数据的最大值
        reg [7:0] fixed_q; //定点方案,表示数据移位的 range

        wire full_signal,empty_signal;    //同步 fifo 的满信号和空信号
        wire almost_full_signal,almost_empty_signal; //同步 fifo 的将满信号和将空信号

        reg [63:0]data_out_64_buffer; //用于给  data_out_64  寄存

        reg wvalid_enable;     //用于表明在 fifo 中的数据可以写回 ddr

        reg fifo_data_control; //切换从 e 指数运算模块和除法模块到同步 fifo 输入端的数据通路
        reg fifo_reset;      //同步 fifo 复位信号
```

```
        reg [63:0] register_save_configuration; //控制寄存器(配置寄存器),存放控制位,用于控制 IP 核的处理模式

        reg [3:0] register_number; //用于控制所需处理的数据个数

        reg fixed_shift;    //16 定点和 8 定点方式切换

        reg init_start_accumulator;//启动累加寄存器
        wire accumulator_done;    //累加寄存器完成信号

        reg first_enable;  //寻找最大值模块使能
        wire find_max_enable;       //first_enable 和 rnext 组成寻找最大值模块有效信号
        reg init_start_find_max;    //用于启动寻找最大值状态机
        wire max_result_done;       //最大值状态机完成信号
        wire [15:0] max_result;     //最大值状态机最终比较结果

        reg skip_dividen; //使能跳过除法阶段

//以下声明用于 axi 读写控制
      //顶层以及子状态机
        reg [1:0] mst_exec_state;
        reg [1:0] mst_exec_state_read;
        reg [4:0] control_state;
        reg [3:0] burst_single_read_state;

        reg [C_M_AXI_ADDR_WIDTH-1 : 0]    axi_awaddr;//写地址
        reg       axi_awvalid;//写地址握手单边
        reg       axi_wlast;//写最后信号
        reg       axi_wvalid;//写握手单边
        reg       axi_bready;//写响应握手单边
        reg [C_M_AXI_ADDR_WIDTH-1 : 0]    axi_araddr;读地址
        reg       axi_arvalid;//读地址握手单边
        reg       axi_rready;//读握手单边
        reg [C_TRANSACTIONS_NUM : 0]      write_index;//单次突发中写握手计数
        reg [C_TRANSACTIONS_NUM : 0]      read_index;//单次突发中读握手计数
        wire [C_TRANSACTIONS_NUM+3 : 0]       burst_size_bytes;//写突发偏移地址
        reg [C_NO_BURSTS_REQ+8 : 0]       write_burst_counter;//写突发计数
        reg [C_NO_BURSTS_REQ+8 : 0]       read_burst_counter;//读突发计数
        reg       start_single_burst_write;//单次突发写开始信号
        reg       start_single_burst_read;//单次突发读开始信号
        reg       writes_done;//全写状态机完成信号
        reg       reads_done;//全读状态机完成信号
        reg       burst_write_active;//单次突发写活跃信号
        reg       burst_read_active;//单次突发读活跃信号
```

```
        wire      wnext;//写握手有效信号
        wire      rnext;//读握手有效信号

//检测启动上边沿
        reg       init_txn_ff;
        reg       init_txn_ff2;
        reg       init_txn_edge;

      reg   init_txn_pulse_read;//启动全读状态机
      reg   init_txn_pulse_write;//启动全写状态机

      wire init_txn_pulse_state;//启动顶层状态机

      reg [6:0] cnt; //顶层状态机中用于简化状态的计数器
      wire [C_TRANSACTIONS_NUM+3 : 0]  burst_size_bytes_read; //读突发偏移地址
      reg control_burst_single_read;//读控制寄存器使能
      reg burst_single_read_arvalid;//读控制寄存器时,读地址握手单边信号
      reg burst_single_read_rready;//读控制寄存器时,读握手单边信号
      reg burst_single_read_pulse;//读控制寄存器状态机启动信号
      reg burst_single_read_done;//读控制寄存器状态机完成信号

      assign M_AXI_AWID      = 'b0;
      assign M_AXI_AWADDR= C_M_TARGET_SLAVE_BASE_ADDR + axi_awaddr;
      assign M_AXI_AWLEN    = C_M_AXI_BURST_LEN - 1; //对应单次写突发中 16 次写数据握手
      assign M_AXI_AWSIZE   = 3'b011; //写数据握手位宽 8,16,32,64....,这里对应 64 位宽度
      assign M_AXI_AWBURST      = 2'b01; //默认
      assign M_AXI_AWLOCK = 1'b0; //默认
      assign M_AXI_AWCACHE      = 4'b0010; //默认
      assign M_AXI_AWPROT = 3'h0; //默认
      assign M_AXI_AWQOS   = 4'h0; //默认
      assign M_AXI_AWUSER = 'b1; //默认
      assign M_AXI_AWVALID=axi_awvalid;

      assign M_AXI_WDATA    = data_64; //写数据

      assign M_AXI_WSTRB    ={(C_M_AXI_DATA_WIDTH/8){1'b1}};//64 位数据总线全写
      assign M_AXI_WLAST    = axi_wlast;
      assign M_AXI_WUSER    = 'b0;//默认
      assign M_AXI_WVALID   =    wvalid_enable? axi_wvalid:0;
      assign M_AXI_BREADY   = axi_bready;
      assign M_AXI_ARID      = 'b0;
      assign M_AXI_ARADDR
      = control_burst_single_read?register_address:(C_M_TARGET_SLAVE_BASE_ADDR
      + axi_araddr);
```

```
    assign M_AXI_ARLEN    = control_burst_single_read?0:(C_M_AXI_BURST_LEN - 1);
    assign M_AXI_ARSIZE   = 3'b011; //读数据握手位宽 64 位

    assign M_AXI_ARBURST      = 2'b01;//默认
    assign M_AXI_ARLOCK  = 1'b0;//默认
    assign M_AXI_ARCACHE      = 4'b0010;//默认
    assign M_AXI_ARPROT  = 3'h0; //默认
    assign M_AXI_ARQOS   = 4'h0; //默认
    assign M_AXI_ARUSER  = 'b1; //默认
    assign M_AXI_ARVALID
= control_burst_single_read?burst_single_read_arvalid:axi_arvalid;
    assign M_AXI_RREADY
= control_burst_single_read?burst_single_read_rready:axi_rready;
    assign burst_size_bytes      = C_M_AXI_BURST_LEN * C_M_AXI_DATA_WIDTH/8;
    assign burst_size_bytes_read = C_M_AXI_BURST_LEN * C_M_AXI_DATA_WIDTH/8;

    assign init_txn_pulse_state  = (!init_txn_ff2) && init_txn_ff;

//检测启动边沿信号
        always @(posedge M_AXI_ACLK)
        begin
            if (M_AXI_ARESETN == 0 )
            begin
                init_txn_ff <= 1'b0;
                init_txn_ff2 <= 1'b0;
            end
            else
            begin
                init_txn_ff <= INIT_AXI_TXN;
                init_txn_ff2 <= init_txn_ff;
            end
        end

//写地址握手单边信号控制
        always @(posedge M_AXI_ACLK)
        begin
            if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
                axi_awvalid <= 1'b0;
            end
            else if (~axi_awvalid && start_single_burst_write)
            begin
                axi_awvalid <= 1'b1;
            end
```

```
                else if (M_AXI_AWREADY && axi_awvalid)
                begin
                      axi_awvalid <= 1'b0;
                end
                else
                begin
                      axi_awvalid <= axi_awvalid;
                end
           end

//写地址控制
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
            begin
              axi_awaddr <= 'b0;
            end
          else if (M_AXI_AWREADY && axi_awvalid)
            begin
              axi_awaddr <= axi_awaddr + burst_size_bytes;
            end
          else
            axi_awaddr <= axi_awaddr;
          end

//写数据握手有效信号
        assign wnext = M_AXI_WREADY & axi_wvalid;

//写握手单边信号控制
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
              axi_wvalid <= 1'b0;
            end
          else if(empty_signal || (almost_empty_signal && wnext )) //fifo 空将抑制写
              begin
                axi_wvalid<=1'b0;
              end
          else if (~axi_wvalid && start_single_burst_write)
            begin
              axi_wvalid <= 1'b1;
            end
          else if(axi_wlast && !axi_wvalid)
```

```
                axi_wvalid <= 1'b1;
            else if (wnext && axi_wlast)
                axi_wvalid <= 1'b0;
             else  if((write_index<C_M_AXI_BURST_LEN-1)  &&  (!axi_wvalid)  &&  wvalid_enable
&& !empty_signal) //重启写
                begin
                   axi_wvalid<=1'b1;
                end
            else
              axi_wvalid <= axi_wvalid;
        end

//单次突发中最后一个数据信号控制
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
              axi_wlast <= 1'b0;
            end
          else if (((write_index == C_M_AXI_BURST_LEN-2 && C_M_AXI_BURST_LEN >= 2) &&
wnext) || (C_M_AXI_BURST_LEN == 1 )) //满足单次写突发中握手计数即将满时,表明最后一个数据到
来
            begin
              axi_wlast <= 1'b1;
            end
          else if (wnext)
            axi_wlast <= 1'b0;
          else if (axi_wlast && C_M_AXI_BURST_LEN == 1)
            axi_wlast <= 1'b0;
          else
            axi_wlast <= axi_wlast;
        end

//单次写突发中数据握手计数
        always @(posedge M_AXI_ACLK)
        begin
           if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 || start_single_burst_write
== 1'b1)
            begin
              write_index <= 0;
            end
          else if (wnext && (write_index != C_M_AXI_BURST_LEN-1))
            begin
              write_index <= write_index + 1;
```

```
        end
      else
        write_index <= write_index;
    end

//写响应握手单边信号控制
    always @(posedge M_AXI_ACLK)
    begin
      if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
        begin
          axi_bready <= 1'b0;
        end
      else if (M_AXI_BVALID && ~axi_bready)
        begin
          axi_bready <= 1'b1;
        end
      else if (axi_bready)
        begin
          axi_bready <= 1'b0;
        end
           else
        axi_bready <= axi_bready;
    end

//读地址握手单边信号控制
    always @(posedge M_AXI_ACLK)
    begin

      if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1 )
        begin
          axi_arvalid <= 1'b0;
        end
      else if (~axi_arvalid && start_single_burst_read)
        begin
          axi_arvalid <= 1'b1;
        end
      else if (M_AXI_ARREADY && axi_arvalid)
        begin
          axi_arvalid <= 1'b0;
        end
      else
        axi_arvalid <= axi_arvalid;
    end
```

```
//读地址控制
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1)
            begin
              axi_araddr <= 'b0;
            end
          else if (M_AXI_ARREADY && axi_arvalid)
            begin
              axi_araddr <= axi_araddr + burst_size_bytes_read;
            end
          else
            axi_araddr <= axi_araddr;
        end

//读数据握手有效信号
        assign rnext = M_AXI_RVALID && axi_rready;

//单次读突发中数据握手计数
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1 || start_single_burst_read)
            begin
              read_index <= 0;
            end
          else if (rnext && (read_index != C_M_AXI_BURST_LEN-1))
            begin
              read_index <= read_index + 1;
            end
          else
            read_index <= read_index;
        end

//读数据握手单边信号控制
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1 )
            begin
              axi_rready <= 1'b0;
            end
          else if (M_AXI_RVALID)
            begin
               if (M_AXI_RLAST && axi_rready)
                 begin
```

```
                    axi_rready <= 1'b0;
                end
                else if((almost_full_signal&&rnext)|| full_signal ) //满和将满抑制读
                  begin
                     axi_rready<=1'b0;
                  end
                else
                  begin
                    axi_rready <=1'b1;
                  end
            end
        end

//写突发次数计数
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
              write_burst_counter <= 'b0;
            end
          else if (M_AXI_AWREADY && axi_awvalid)
            begin
                 if (write_burst_counter[C_NO_BURSTS_REQ+register_number-fixed_shift] ==
1'b0)
                begin
                  write_burst_counter <= write_burst_counter + 1'b1;
                end
            end
          else
            write_burst_counter <= write_burst_counter;
        end

//读突发次数计数
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1)
            begin
              read_burst_counter <= 'b0;
            end
          else if (M_AXI_ARREADY && axi_arvalid)
            begin
                 if (read_burst_counter[C_NO_BURSTS_REQ+register_number-fixed_shift] ==
1'b0)
                begin
```

```
                read_burst_counter <= read_burst_counter + 1'b1;
              end
          end
        else
          read_burst_counter <= read_burst_counter;
      end

//全写状态机
      always @ ( posedge M_AXI_ACLK)
      begin
        if (M_AXI_ARESETN == 1'b0 )
          begin
            mst_exec_state        <= 0;
            start_single_burst_write <= 1'b0;
          end
        else
          begin
            case (mst_exec_state)

              0:
                if ( init_txn_pulse_write == 1'b1)
                  begin
                    mst_exec_state   <= 1;
                  end
                else
                  begin
                    mst_exec_state   <= 0;
                  end

              1:
                if (writes_done )
                  begin
                    mst_exec_state <= 2;
                  end
                else
                  begin
                    mst_exec_state   <= 1;

                      if (~axi_awvalid && ~start_single_burst_write && ~burst_write_active
&& !empty_signal)
                       begin
                         start_single_burst_write <= 1'b1;
                       end
                      else
```

```
                    begin
                      start_single_burst_write <= 1'b0;
                    end
                  end

              2:
                begin
                  mst_exec_state <= 0;
                end
              default :
                begin
                  mst_exec_state  <= 0;
                end
            endcase
          end
      end

//全读状态机
    always @ ( posedge M_AXI_ACLK)
      begin
        if (M_AXI_ARESETN == 1'b0 )
          begin
            mst_exec_state_read       <= 0;
            start_single_burst_read   <= 1'b0;
          end
        else
          begin
            case (mst_exec_state_read)

              0:
                if ( init_txn_pulse_read == 1'b1)
                  begin
                    mst_exec_state_read   <= 1;
                  end
                else
                  begin
                    mst_exec_state_read   <= 0;
                  end

              1:
                  begin
                       if (reads_done )
                        begin
                          mst_exec_state_read <= 2;
```

```
                        end
                       else
                        begin
                         mst_exec_state_read <= 1;

                        if (~axi_arvalid && ~burst_read_active && ~start_single_burst_read
&& !full_signal)
                            begin
                              start_single_burst_read <= 1'b1;
                            end
                       else
                           begin
                             start_single_burst_read <= 1'b0;
                           end
                       end
                   end


                2:
                  begin
                    mst_exec_state_read <= 0;
                  end
                default :
                  begin
                    mst_exec_state_read   <= 0;
                  end
              endcase
            end
        end

//读控制寄存器状态机
          always @ (posedge M_AXI_ACLK)
          begin
               if(M_AXI_ARESETN==1'b0 || init_txn_pulse_state)
                 begin
                   burst_single_read_rready<=1'b0;
                   burst_single_read_arvalid<=1'b0;
                   burst_single_read_done<=1'b0;
                  burst_single_read_state<=1’b0;
                   register_save_configuration<=1’b0;
                 end
               else
                 begin
                 case (burst_single_read_state)
```

```
0:
begin
if(burst_single_read_pulse)
  begin
  burst_single_read_state<=burst_single_read_state+1;
  end
else
  burst_single_read_state<=burst_single_read_state;
end

1:
begin
 burst_single_read_arvalid<=1'b1;
 burst_single_read_state<=burst_single_read_state+1;
end

2:
begin
  if(M_AXI_ARREADY && burst_single_read_arvalid)
  begin
    burst_single_read_arvalid<=1'b0;
    burst_single_read_state<=burst_single_read_state+1;
  end
  else
  begin
    burst_single_read_arvalid<=burst_single_read_arvalid;
    burst_single_read_state<=burst_single_read_state;
  end
end

3:
begin
  if(M_AXI_RVALID)
  begin
    if(M_AXI_RLAST && burst_single_read_rready)
      begin
      burst_single_read_rready<=1'b0;
      burst_single_read_state<=0;
      burst_single_read_done<=1'b1;
        register_save_configuration<=M_AXI_RDATA;
      end
    else
      begin
        burst_single_read_rready<=1'b1;
```

```
                    end
                  end
                end

                default: burst_single_read_state<=0;
              endcase
              end
          end


        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
            burst_write_active <= 1'b0;
          else if (start_single_burst_write)
            burst_write_active <= 1'b1;
          else if (M_AXI_BVALID && axi_bready)
            burst_write_active <= 0;
        end

//全写完成信号
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
            writes_done <= 1'b0;
       else if (M_AXI_BVALID
&& (write_burst_counter[C_NO_BURSTS_REQ+register_number-fixed_shift]) && axi_bready)
            writes_done <= 1'b1;
          else
            writes_done <= writes_done;
        end


        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1)
            burst_read_active <= 1'b0;
          else if (start_single_burst_read)
            burst_read_active <= 1'b1;
          else if (M_AXI_RVALID && axi_rready && M_AXI_RLAST)
            burst_read_active <= 0;
        end
```

```
//全读完成信号
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_read == 1'b1)
            reads_done <= 1'b0;
          else if (M_AXI_RVALID && axi_rready && (read_index == C_M_AXI_BURST_LEN-1) &&
(read_burst_counter[C_NO_BURSTS_REQ+register_number-fixed_shift]))
            reads_done <= 1'b1;
          else
            reads_done <= reads_done;
          end

//顶层控制状态机
      always @ ( posedge M_AXI_ACLK)
      begin
           if (M_AXI_ARESETN == 1'b0 )
                begin
                     control_state <= 0;
                     init_txn_pulse_read<=1'b0;
                     init_txn_pulse_write<=1'b0;
                     control_burst_single_read<=1'b0;
                     burst_single_read_pulse<=1'b0;
                     TXN_DONE<=1'b0;

                     second_enable<=0;
                     reg_max<=0;
                     fixed_q<=0;
                     data_valid<=0;
                     long_en<=1'b0;
                     cnt<=0;
                     wvalid_enable<=0;
                     fifo_data_control<=0;
                     third_enable<=0;
                     fifo_reset<=1;
                     register_number<=0;
                     fixed_shift<=0;
                     init_start_accumulator<=0;
                     first_enable<=0;
                     init_start_find_max<=0;
                     skip_dividen<=0;
                end
           else
                begin
                     case (control_state)
```

```
        0:
        begin
        if(init_txn_pulse_state)
            begin
                init_txn_pulse_read<=1'b0;
                init_txn_pulse_write<=1'b0;
                control_burst_single_read<=1'b0;
                burst_single_read_pulse<=1'b0;
                TXN_DONE<=1'b0;
                second_enable<=0;
                reg_max<=0;
                fixed_q<=0;
                data_valid<=0;
                long_en<=1'b0;
                cnt<=0;
                wvalid_enable<=0;
                fifo_data_control<=0;
                third_enable<=0;
                fifo_reset<=1;
                register_number<=0;
                fixed_shift<=0;
                init_start_accumulator<=0;
                first_enable<=0;
                init_start_find_max<=0;
                skip_dividen<=0;
                control_state<=control_state+1;
            end
end

        1:
        begin
            fifo_reset<=0;
            control_burst_single_read<=1;
            burst_single_read_pulse<=1;
            control_state<=control_state+1;
    end

    2:
    begin
            if(burst_single_read_pulse)
                burst_single_read_pulse<=0;

            if(!fifo_reset)
```

```
                fifo_reset<=1;

            if(burst_single_read_done)
                begin

                    control_burst_single_read<=0;

                    fixed_q<=register_save_configuration[23:16];
                    register_number<=register_save_configuration[27:24];
                    fixed_shift<=register_save_configuration[32];
                    skip_dividen<=register_save_configuration[48];

                    if(register_save_configuration[40])
                      begin
                        control_state<=control_state+1;
                        first_enable<=1;
                        init_txn_pulse_read<=1'b1;
                      end
                    else
                      begin
                      control_state<=6;
                      reg_max<=register_save_configuration[15:0];
                      end

                end
  end

3:
begin
     if(init_txn_pulse_read)
          init_txn_pulse_read<=1'b0;
      control_state<=control_state+1;

end
4:
begin
 if(reads_done)
 begin
     control_state<=control_state+1;
     init_start_find_max<=1;
     first_enable<=0;
 end
end
```

```
5:
begin
    if(init_start_find_max)
        init_start_find_max<=0;
    control_state<=control_state+1;
end
6:
begin
    if(max_result_done)
        begin
        reg_max<=fixed_shift ? {2'b00,max_result[6:0],7'd0} : max_result;
        control_state<=control_state+1;
        end
end

7:
begin
    second_enable<=1;
    init_txn_pulse_read<=1'b1;
    control_state<=control_state+1;
end

  8:
  begin
    if(init_txn_pulse_read)
    init_txn_pulse_read<=1'b0;

      if(rnext)
          cnt<=cnt+1;

      if(cnt==9)
            begin
            data_valid<=1;
            reg_max<=reg_max;
            control_state<=control_state+1;
            cnt<=0;
            end
  end

  9:
  begin

  if(rnext)
        cnt<=cnt+1;
```

```
    if(cnt==8)
    begin
        init_txn_pulse_write<=1'b1;
        control_state<=control_state+1;
        cnt<=0;
    end

    end


10:
begin
     if(init_txn_pulse_write)
          init_txn_pulse_write<=1'b0;
        control_state<=control_state+1;
end

11:
begin
     wvalid_enable<=1;
     control_state<=control_state+1;
end

    12:
    begin

    if(reads_done)
        begin
     control_state<=control_state+1;
     cnt<=0;
     end
    end

    13:
    begin
if(cnt==11)
     begin
          control_state<=control_state+1;
          data_valid<=0;
     end
else
     begin
          control_state<=control_state;
```

```
            data_valid<=data_valid;
        end

    if(cnt==11)
            begin
            cnt<=0;
            end
    else if(fifo_wr_enable)
            begin
            cnt<=cnt+1;
            end
    else
            begin
            cnt<=cnt;
            end

if(cnt==11)
        begin
            long_en<=0;
        end
else if((almost_full_signal&&long_en) || full_signal)
        begin
            long_en<=0;
        end
else
        begin
            long_en<=1;
        end

    end

    14:
    begin

    if(writes_done)
    begin
            wvalid_enable<=0;

            if(skip_dividen)
            begin
                control_state<=0;
                TXN_DONE<=1;
                second_enable<=0;
            end
```

```
            else
                control_state<=control_state+1;
        end
        else
            begin
            control_state<=control_state;
            end

        end


        15:
        begin
        second_enable<=0;
        reg_max<=0;
        fixed_q<=0;
        control_state<=control_state+1;
        init_start_accumulator<=1;
        end


    16:
        begin
        if(init_start_accumulator)
            init_start_accumulator<=0;

         if(accumulator_done)
          begin
          fifo_reset<=0;
          init_txn_pulse_read<=1'b1;
          third_enable<=1;
          fifo_data_control<=1;
          control_state<=control_state+1;
          end
    end


        17:
        begin
            if(init_txn_pulse_read)
                init_txn_pulse_read<=1'b0;

            if(!fifo_reset)
             fifo_reset<=1'b1;
```

```
        if(rnext)
            cnt<=cnt+1;

        if(cnt==16)
            begin
            control_state<=control_state+1;
            cnt<=0;
            end
    end

    18:
    begin
    if(rnext)
        cnt<=cnt+1;

    if(rnext)
        data_valid<=1;

     if(cnt==8)
      begin
        init_txn_pulse_write<=1'b1;
        control_state<=control_state+1;
        cnt<=0;
      end

end


19:
begin
     if(init_txn_pulse_write)
          init_txn_pulse_write<=1'b0;
    control_state<=control_state+1;
end
20:
begin
     wvalid_enable<=1;
     control_state<=control_state+1;
end

    21:
    begin
```

```
    if(reads_done)
     begin
      control_state<=control_state+1;
      cnt<=0;
      end
    end

    22:
    begin

    if(cnt==18)
      begin
            control_state<=control_state+1;
            data_valid<=0;
      end
     else
      begin
            control_state<=control_state;
            data_valid<=data_valid;
      end

    if(cnt==18)
           begin
           cnt<=0;
           end
    else if(fifo_wr_enable)
           begin
           cnt<=cnt+1;
           end
    else
          begin
          cnt<=cnt;
          end

if(cnt==18)
      begin
            long_en<=0;
      end
else if((almost_full_signal&&long_en) || full_signal)
      begin
            long_en<=0;
      end
else
      begin
```

```
                    long_en<=1;
                  end

                end

                23:
                begin

                if(writes_done)
                    begin
                    control_state<=control_state+1;
                    wvalid_enable<=0;
                    end
                else
                    begin
                    control_state<=control_state;
                    end

                end

                24:
                begin
                third_enable<=0;
                fifo_data_control<=0;
                control_state<=control_state+1;
                end

              25:
              begin
              TXN_DONE<=1;
                 control_state<=0;
              end

              default:control_state <= 0;
              endcase
         end
end

always @(posedge M_AXI_ACLK)
begin
    if( M_AXI_ARESETN==0 || init_txn_pulse_state==1'b1)
        accumulator_0<=0;
    else if(fifo_rd_enable && second_enable)
        accumulator_0<=accumulator_0+(fixed_shift?data_64[7:0]:data_64[15:0]);
```

```
        else
            accumulator_0<=accumulator_0;
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_1<=0;
        else if(fifo_rd_enable && second_enable)
            accumulator_1<=accumulator_1+(fixed_shift?data_64[15:8]:data_64[31:16]);
        else
            accumulator_1<=accumulator_1;
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_2<=0;
        else if(fifo_rd_enable && second_enable)

    accumulator_2<=accumulator_2+(fixed_shift?data_64[23:16]:data_64[47:32]);
        else
            accumulator_2<=accumulator_2;
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_3<=0;
        else if(fifo_rd_enable && second_enable)

    accumulator_3<=accumulator_3+(fixed_shift?data_64[31:24]:data_64[63:48]);
        else
            accumulator_3<=accumulator_3;
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_4<=0;
        else if(fifo_rd_enable && second_enable)
            accumulator_4<=accumulator_4+data_64[39:32];
        else
            accumulator_4<=accumulator_4;
```

```
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_5<=0;
        else if(fifo_rd_enable && second_enable)
            accumulator_5<=accumulator_5+data_64[47:40];
        else
            accumulator_5<=accumulator_5;
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_6<=0;
        else if(fifo_rd_enable && second_enable)
            accumulator_6<=accumulator_6+data_64[55:48];
        else
            accumulator_6<=accumulator_6;
    end

    always @(posedge M_AXI_ACLK)
    begin
        if( M_AXI_ARESETN==0 ||init_txn_pulse_state==1'b1)
            accumulator_7<=0;
        else if(fifo_rd_enable && second_enable)
            accumulator_7<=accumulator_7+data_64[63:56];
        else
            accumulator_7<=accumulator_7;
    end

    assign read_enable = long_en ? 1'b1 : axi_rready && M_AXI_RVALID;

    assign clk2 = second_enable && read_enable ? ~M_AXI_ACLK :0;

    assign clk3 = third_enable && read_enable ? ~M_AXI_ACLK :0;

    assign
data_in_0=fixed_shift?{M_AXI_RDATA[7],1'b0,M_AXI_RDATA[6:0],7'd0}:M_AXI_RDATA[15:0];
    assign
data_in_1=fixed_shift?{M_AXI_RDATA[15],1'b0,M_AXI_RDATA[14:8],7'd0}:M_AXI_RDATA[31:
16];
    assign
```

```
data_in_2=fixed_shift?{M_AXI_RDATA[23],1'b0,M_AXI_RDATA[22:16],7'd0}:M_AXI_RDATA[47:
32];
      assign
data_in_3=fixed_shift?{M_AXI_RDATA[31],1'b0,M_AXI_RDATA[30:24],7'd0}:M_AXI_RDATA[63:
48];

     assign data_in_4=M_AXI_RDATA[39:32];
     assign data_in_5=M_AXI_RDATA[47:40];
     assign data_in_6=M_AXI_RDATA[55:48];
     assign data_in_7=M_AXI_RDATA[63:56];

      operation_fixed_all operation_fixed_all_u1(clk2,data_in_0,reg_max,fixed_q,data_out_0);
      operation_fixed_all operation_fixed_all_u2(clk2,data_in_1,reg_max,fixed_q,data_out_1);
      operation_fixed_all operation_fixed_all_u3(clk2,data_in_2,reg_max,fixed_q,data_out_2);
      operation_fixed_all operation_fixed_all_u4(clk2,data_in_3,reg_max,fixed_q,data_out_3);

     operation_fixed_8
operation_fixed_8_u1(clk2,data_in_4,reg_max[13:7],fixed_q,data_out_4);
     operation_fixed_8
operation_fixed_8_u2(clk2,data_in_5,reg_max[13:7],fixed_q,data_out_5);
     operation_fixed_8
operation_fixed_8_u3(clk2,data_in_6,reg_max[13:7],fixed_q,data_out_6);
     operation_fixed_8
operation_fixed_8_u4(clk2,data_in_7,reg_max[13:7],fixed_q,data_out_7);

      assign data_in_0_d=fixed_shift?{8'd0,M_AXI_RDATA[7:0]}:M_AXI_RDATA[15:0];
      assign data_in_1_d=fixed_shift?{8'd0,M_AXI_RDATA[15:8]}:M_AXI_RDATA[31:16];
      assign data_in_2_d=fixed_shift?{8'd0,M_AXI_RDATA[23:16]}:M_AXI_RDATA[47:32];
      assign data_in_3_d=fixed_shift?{8'd0,M_AXI_RDATA[31:24]}:M_AXI_RDATA[63:48];


      divider_fixed_all
divider_fixed_all_u1(clk3,data_in_0_d,accumulator_result,data_out_0_d);
      divider_fixed_all
divider_fixed_all_u2(clk3,data_in_1_d,accumulator_result,data_out_1_d);
      divider_fixed_all
divider_fixed_all_u3(clk3,data_in_2_d,accumulator_result,data_out_2_d);
      divider_fixed_all
divider_fixed_all_u4(clk3,data_in_3_d,accumulator_result,data_out_3_d);

     divider_fixed_8
divider_fixed_8_u1(clk3,data_in_4,accumulator_result[28:0],data_out_4_d);
     divider_fixed_8
divider_fixed_8_u2(clk3,data_in_5,accumulator_result[28:0],data_out_5_d);
```

```
    divider_fixed_8
divider_fixed_8_u3(clk3,data_in_6,accumulator_result[28:0],data_out_6_d);
    divider_fixed_8
divider_fixed_8_u4(clk3,data_in_7,accumulator_result[28:0],data_out_7_d);

assign      data_out3_buf_8fix      =      data_out_3[12]?      8'b1111_1111      :
{data_out_3[11:5],data_out_3[4]|data_out_3[3]};
assign      data_out2_buf_8fix      =      data_out_2[12]?      8'b1111_1111      :
{data_out_2[11:5],data_out_2[4]|data_out_2[3]};
assign      data_out1_buf_8fix      =      data_out_1[12]?      8'b1111_1111      :
{data_out_1[11:5],data_out_1[4]|data_out_1[3]};
assign      data_out0_buf_8fix      =      data_out_0[12]?      8'b1111_1111      :
{data_out_0[11:5],data_out_0[4]|data_out_0[3]};

assign find_max_enable= first_enable && rnext;

find_max find_max
(
  M_AXI_ACLK,
  M_AXI_ARESETN,
  init_txn_pulse_state,
  M_AXI_RDATA,
  find_max_enable,
  fixed_shift,
  init_start_find_max,
  max_result_done,
  max_result
);

accumulator_final_result accumulator_final_result_u0
(
  M_AXI_ACLK,
  M_AXI_ARESETN,
  init_txn_pulse_state,
  fixed_shift,
  init_start_accumulator,
  accumulator_0,
  accumulator_1,
  accumulator_2,
  accumulator_3,
  accumulator_4,
  accumulator_5,
  accumulator_6,
  accumulator_7,

`
```

```
  accumulator_result,
  accumulator_done
);

always @ (negedge M_AXI_ACLK)
begin
    if(read_enable&&(third_enable||second_enable))

    begin
        if(fifo_data_control)
            begin
            if(fixed_shift)
data_out_64<={data_out_7_d,data_out_6_d,data_out_5_d,data_out_4_d,data_out_3_d[15:9],
data_out_3_d[8]|data_out_3_d[7],data_out_2_d[15:9],data_out_2_d[8]|data_out_2_d[7],data
_out_1_d[15:9],data_out_1_d[8]|data_out_1_d[7],data_out_0_d[15:9],data_out_0_d[8]|data_
out_0_d[7]};
            else
                data_out_64<={data_out_3_d,data_out_2_d,data_out_1_d,data_out_0_d};
            end
        else
             begin
              if(fixed_shift)

     data_out_64<={data_out_7,data_out_6,data_out_5,data_out_4,data_out3_buf_8fix,dat
a_out2_buf_8fix,data_out1_buf_8fix,data_out0_buf_8fix};
              else
                  data_out_64<={data_out_3,data_out_2,data_out_1,data_out_0};
             end
    end
end

    always @(posedge M_AXI_ACLK)
    begin
        if(!M_AXI_ARESETN)
            data_out_64_buffer<=0;
        else
            data_out_64_buffer<=data_out_64;
    end

     assign fifo_wr_enable= data_valid && read_enable && !full_signal;

     assign fifo_rd_enable= M_AXI_WREADY & axi_wvalid && (!empty_signal) &&
wvalid_enable;
```

```
      fifo
fifo_u0(fifo_reset,M_AXI_ACLK,fifo_wr_enable,fifo_rd_enable,data_out_64_buffer,data_64,full
_signal,empty_signal,almost_full_signal,almost_empty_signal);

endmodule
```

3.operation_fixed_all.v

```
module operation_fixed_all
(
  input clk;//时钟信号
  input [15:0] data_in;//数据输入
  input [15:0] reg_max; //输入最大值
  input [7:0] fixed_q;//输入定点方案
  output wire [15:0] data_out;//输出 e 指数运算结果
);

reg [15:0] sub_data_result; //存放定点数据减法结果
reg [7:0] sub_fixed_result; //存放定点方案减法结果
reg fixed_signed; //存放定点减法符号，为 0 则输入定点方案大与等于 131，为 1 则输入定点方案小于
131
reg [31:0] fixed_shift_buffer;  //存放定点转换中间值，高十六位随后相或作为溢出位
reg [15:0] fixed_shift_result;  //存放定点转换结果，最高位为溢出位，后 15 位为数据
wire [15:0] fitting_result; //存放拟合结果

//最大值减去输入数据,得到 sub_data_result
always @(posedge clk)
begin
    if(!data_in[15])  //最高位为符号位,根据符号选择使用减法还是加法
            sub_data_result<=reg_max[15:0]-{1'b0,data_in[14:0]};
        else
            sub_data_result<=reg_max[15:0]+{1'b0,data_in[14:0]};
end

//定点方案减去 131 的绝对值为 sub_fixed_result,fixed_signed 存放符号
always @(posedge clk)
begin
    if(fixed_q>=8'd131)
        begin
            sub_fixed_result<=fixed_q-8'd131;
            fixed_signed<=1'b0;
```

```
            end
        else if(fixed_q<8'd131)
            begin
                sub_fixed_result<=8'd131-fixed_q;
                fixed_signed<=1'b1;
            end
        else
            begin
                sub_fixed_result<=sub_fixed_result;
                fixed_signed<=fixed_signed;
            end
end

//根据定点方案减法结果进行移位,化为 3+12 定点方案
always @(posedge clk)
begin
    if(sub_fixed_result>8'd16 && (!fixed_signed) )
            fixed_shift_buffer<=32'hffff_0000;   //左移数据 16 位以上,溢出则将高 16 位置 1
        else if(sub_fixed_result>8'd16 &&   fixed_signed    )
            fixed_shift_buffer<=32'h0000_0000;   //右移数据 16 位以上,直接舍去为 0
        else if(sub_fixed_result<=8'd16 && (!fixed_signed))
            fixed_shift_buffer<=sub_data_result<<sub_fixed_result; //左移  sub_fixed_result
        else if(sub_fixed_result<=8'd16 &&   fixed_signed   )
            fixed_shift_buffer<=sub_data_result>>sub_fixed_result; //右移  sub_fixed_result
        else
            fixed_shift_buffer<=fixed_shift_buffer;
end

//定点转换结果输出
always @(posedge clk)
begin
            begin
                fixed_shift_result[15]<=|fixed_shift_buffer[31:15];   //高 16 位相或作为溢出位
                fixed_shift_result[14:0]<=fixed_shift_buffer[14:0];
            end
end

//二阶函数拟合器
funtion_filter filter0(clk,fixed_shift_result,fitting_result);

assign data_out=fitting_result;


endmodule

`
```

(1)funtion_filter.v

```
module funtion_filter
(
  input clk;  //时钟信号
  input [15:0] fixed_shift_result;  //输入定点转换结果
  output reg [15:0] fitting_result; //输出拟合值
);

reg [3:0] address;  //用于查询 rom 表地址
wire [4:0] fixed_part;  //输入定点高位，用于判断所需查找表的地址
wire overflow_bit;  //溢出位,表明输入值溢出
reg [14:0] x;    //缓存输入的 fixed_shift_result 的低 15 位,作为二阶方程输入值
reg [14:0] x_buffer;  //输入拟合器缓存一拍的 x 值
reg [29:0] x_2;    //x^2
reg [26:0] b_x;    //b*x

reg [40:0] a_x_2; //a*x^2

reg [40:0] b_x_c; //b*x+c
reg [40:0] sum; //sum 为最终和

wire [10:0] a_p;  //rom 输出未缓存的系数 A
reg [10:0] a; //a_p 打上一拍,缓存为 a
reg [10:0] a_buffer;  //为了等待 x^2 的运算结果,多打一拍

wire [11:0] b_p;  //rom 输出未缓存的系数 B
reg [11:0] b; //b_p 打上一拍,缓存为 b

wire [12:0] c_p;  //rom 输出未缓存的系数 C
reg [12:0] c; //c_p 打上一拍,缓存为 c
reg [12:0] c_buffer;  //为了等待 b*x 的运算结果,多打一拍

//取高五位用于划分地址
assign fixed_part=fixed_shift_result[14:10];

// 取最高位为溢出位
assign overflow_bit=fixed_shift_result[15];
```

`

```
//为了等待查表的地址信号 address，多打一拍
always @(posedge clk)
begin
    x<=fixed_shift_result[14:0];
end

//产生地址信号
always @(posedge clk)
begin
  if(overflow_bit)
    address<=4'b0000;
  else if(fixed_part==5'b00000)
    address<=4'b0001;
  else if(fixed_part==5'b00001)
    address<=4'b0010;
  else if(fixed_part==5'b00010)
    address<=4'b0011;
  else if(fixed_part==5'b00011)
    address<=4'b0100;
  else if(fixed_part==5'b00100)
    address<=4'b0101;
  else if(fixed_part==5'b00101)
    address<=4'b0110;
  else if(fixed_part==5'b00110)
    address<=4'b0111;
  else if(fixed_part==5'b00111)
    address<=4'b1000;
  else if(fixed_part[4:1]==4'b0100)
    address<=4'b1001;
  else if(fixed_part[4:1]==4'b0101)
    address<=4'b1010;
  else if(fixed_part[4:1]==4'b0110)
    address<=4'b1011;
  else if(fixed_part[4:1]==4'b0111)
    address<=4'b1100;
  else if(fixed_part[4:2]==3'b100)
    address<=4'b1101;
  else if(fixed_part[4:2]==3'b101)
    address<=4'b1110;
  else if(fixed_part[4:3]==2'b11)
    address<=4'b1111;
  else
    address<=address;
end
```

`

```
list1 list1_u0(address,a_p);//rom,用于查询系数 a
list2 list2_u0(address,b_p);//rom,用于查询系数 b
list3 list3_u0(address,c_p);//rom，用于查询系数 c

//将查表所得系数和输入 x,缓冲一拍
always @(posedge clk)
begin
    a<=a_p;
    b<=b_p;
    c<=c_p;
    x_buffer<=x;
end

//使用 dsp 运算得到 x^2,b*x
always @(posedge clk)
begin
    x_2<=x_buffer*x_buffer;
    b_x<=x_buffer*b;
    a_buffer<=a;//等待,多打一拍
    c_buffer<=c;//等待,多打一拍
end

//使用 dsp 运算得到 a*x^2,b*x+c
always @(posedge clk)
begin
    a_x_2<=x_2*a_buffer;
    b_x_c<=(c_buffer<<12)-b_x;   //c_buffer 左移 12 位用于定点方案匹配
end

always @(posedge clk)
begin
    sum<=(b_x_c<<12)+a_x_2; //b_x_c 左移 12 位用于定点方案匹配
end

//最终结果输出
always @(posedge clk)
begin
    fitting_result<={sum[39:25],sum[24]|sum[23]}; //最后一位和舍去的第一位相或,在占用尽量
少的资源的同时减少误差
end

endmodule
```

(2)list1.v

```
module list1
(
  input [3:0] address, //输入地址
  output reg [10:0] coefficient //输出系数
);

always @(address)
begin
  case(address)
  4'd0:coefficient<=11'd0;
  4'd1:coefficient<=11'd1809;
  4'd2:coefficient<=11'd1409;
  4'd3:coefficient<=11'd1097;
  4'd4:coefficient<=11'd855;
  4'd5:coefficient<=11'd666;
  4'd6:coefficient<=11'd518;
  4'd7:coefficient<=11'd404;
  4'd8:coefficient<=11'd314;
  4'd9:coefficient<=11'd217;
  4'd10:coefficient<=11'd132;
  4'd11:coefficient<=11'd80;
  4'd12:coefficient<=11'd48;
  4'd13:coefficient<=11'd23;
  4'd14:coefficient<=11'd9;
  4'd15:coefficient<=11'd2;
  default:coefficient<=11'd0;
  endcase
end

endmodule
```

(3)list2.v

```
module list2
(
  input [3:0] address,   //输入地址
  output reg [11:0] coefficient //输出系数

`
```

```
);

always @(address)
begin
  case(address)
  4'd0:coefficient<=12'd0;
  4'd1:coefficient<=12'd4073;
  4'd2:coefficient<=12'd3876;
  4'd3:coefficient<=12'd3568;
  4'd4:coefficient<=12'd3206;
  4'd5:coefficient<=12'd2830;
  4'd6:coefficient<=12'd2463;
  4'd7:coefficient<=12'd2120;
  4'd8:coefficient<=12'd1808;
  4'd9:coefficient<=12'd1410;
  4'd10:coefficient<=12'd987;
  4'd11:coefficient<=12'd678;
  4'd12:coefficient<=12'd460;
  4'd13:coefficient<=12'd255;
  4'd14:coefficient<=12'd111;
  4'd15:coefficient<=12'd32;
  default:coefficient<=12'd0;
  endcase
end

endmodule
```

(4)list3.v

```
module list3
(
  input [3:0] address,   //输入地址
  output reg [12:0] coefficient //输出系数
);

always @(address)
begin
  case(address)
  4'd0:coefficient<=13'd0;
  4'd1:coefficient<=13'd4096;
  4'd2:coefficient<=13'd4070;
  4'd3:coefficient<=13'd3994;
```

`

```
  4'd4:coefficient<=13'd3858;
  4'd5:coefficient<=13'd3671;
  4'd6:coefficient<=13'd3442;
  4'd7:coefficient<=13'd3185;
  4'd8:coefficient<=13'd2913;
  4'd9:coefficient<=13'd2507;
  4'd10:coefficient<=13'd1981;
  4'd11:coefficient<=13'd1521;
  4'd12:coefficient<=13'd1140;
  4'd13:coefficient<=13'd724;
  4'd14:coefficient<=13'd369;
  4'd15:coefficient<=13'd131;
  default:coefficient<=13'd0;
  endcase
end

endmodule
```

4.operation_fixed_8.v

```
module operation_fixed_8
(
  input clk,  //时钟信号
  input [7:0] data_in,  //输入 8 位数据
  input [6:0] reg_max_fixed_8,  //输入 7 位的最大值
  input [7:0] fixed_q,  //输入定点方案
  output reg [7:0] data_out //输出 8 位数据
);

reg [7:0] sub_data_result;  //存放定点数据减法结果
reg [7:0] sub_fixed_result; //存放定点方案减法结果
reg fixed_signed; //存放定点减法符号，为 0 则输入定点方案大与等于 131，为 1 则输入定点方案小于
131
reg [15:0] fixed_shift_buffer;  //存放定点转换中间值，高八位随后相或作为溢出位
reg [8:0] fixed_shift_result;//存放定点转换结果，最高位为溢出位，后 8 位为数据

reg [8:0] data_in_buffer; //fixed_shift_result 缓存
reg [7:0] data_rom_out; //查表输出
reg [7:0] data_rom_buf1,data_rom_buf2,data_rom_buf3;  //为了和 operation_fixed_all 模块的
流水线同步,缓冲

//最大值减去输入数据,得到 sub_data_result
always @(posedge clk)
begin
```

```
    if(!data_in[7]) //最高位为符号位,根据符号选择使用减法还是加法
            sub_data_result<={1'b0,reg_max_fixed_8}-{1'b0,data_in[6:0]};
    else
            sub_data_result<={1'b0,reg_max_fixed_8}+{1'b0,data_in[6:0]};
end

//定点方案减去 131 的绝对值为 sub_fixed_result,fixed_signed 存放符号
always @(posedge clk)
begin
    if(fixed_q>=8'd131)
        begin
            sub_fixed_result<=fixed_q-8'd131;
            fixed_signed<=1'b0;
        end
     else if(fixed_q<8'd131)
        begin
            sub_fixed_result<=8'd131-fixed_q;
            fixed_signed<=1'b1;
        end
     else
        begin
            sub_fixed_result<=sub_fixed_result;
            fixed_signed<=fixed_signed;
        end
end

//根据定点方案减法结果进行移位,化为 3+5 定点方案
always @(posedge clk)
begin
    if(sub_fixed_result>8'd8 && (!fixed_signed) )
        fixed_shift_buffer<=16'hff_00;   //左移数据 8 位以上,溢出则将高 8 位置 1
     else if(sub_fixed_result>8'd8 &&   fixed_signed    )
        fixed_shift_buffer<=16'h00_00;   //右移数据 8 位以上,直接舍去为 0
     else if(sub_fixed_result<=8'd8 && (!fixed_signed))
        fixed_shift_buffer<=sub_data_result<<sub_fixed_result;   //左移  sub_fixed_result
     else if(sub_fixed_result<=8'd8 &&   fixed_signed   )
        fixed_shift_buffer<=sub_data_result>>sub_fixed_result;   //右移  sub_fixed_result
     else
        fixed_shift_buffer<=fixed_shift_buffer;
end

//定点转换结果输出
always @(posedge clk)
begin
```

```
            begin
                fixed_shift_result[8]<= |fixed_shift_buffer[15:8]; //高 8 位相或作为溢出位
                fixed_shift_result[7:0]<=fixed_shift_buffer[7:0];
            end
end

//缓冲
always @(posedge clk)
begin
    data_in_buffer<=fixed_shift_result;
end

//rom 表
always @(posedge clk)
begin
if(data_in_buffer[8])
    data_rom_out<=0;
else
    begin
    case(data_in_buffer[7:0])
    0:data_rom_out<=255;
    1:data_rom_out<=248;
    2:data_rom_out<=240;
    3:data_rom_out<=233;
    4:data_rom_out<=226;
    5:data_rom_out<=219;
    6:data_rom_out<=212;
    7:data_rom_out<=206;
    8:data_rom_out<=199;
    9:data_rom_out<=193;
    10:data_rom_out<=187;
    11:data_rom_out<=181;
    12:data_rom_out<=176;
    13:data_rom_out<=170;
    14:data_rom_out<=165;
    15:data_rom_out<=160;
    16:data_rom_out<=155;
    17:data_rom_out<=150;
    18:data_rom_out<=146;
    19:data_rom_out<=141;
    20:data_rom_out<=137;
    21:data_rom_out<=132;
    22:data_rom_out<=128;
    23:data_rom_out<=124;
```

```
24:data_rom_out<=121;
25:data_rom_out<=117;
26:data_rom_out<=113;
27:data_rom_out<=110;
28:data_rom_out<=106;
29:data_rom_out<=103;
30:data_rom_out<=100;
31:data_rom_out<=97;
32:data_rom_out<=94;
33:data_rom_out<=91;
34:data_rom_out<=88;
35:data_rom_out<=85;
36:data_rom_out<=83;
37:data_rom_out<=80;
38:data_rom_out<=78;
39:data_rom_out<=75;
40:data_rom_out<=73;
41:data_rom_out<=71;
42:data_rom_out<=69;
43:data_rom_out<=66;
44:data_rom_out<=64;
45:data_rom_out<=62;
46:data_rom_out<=60;
47:data_rom_out<=59;
48:data_rom_out<=57;
49:data_rom_out<=55;
50:data_rom_out<=53;
51:data_rom_out<=52;
52:data_rom_out<=50;
53:data_rom_out<=49;
54:data_rom_out<=47;
55:data_rom_out<=46;
56:data_rom_out<=44;
57:data_rom_out<=43;
58:data_rom_out<=41;
59:data_rom_out<=40;
60:data_rom_out<=39;
61:data_rom_out<=38;
62:data_rom_out<=37;
63:data_rom_out<=35;
64:data_rom_out<=34;
65:data_rom_out<=33;
66:data_rom_out<=32;
67:data_rom_out<=31;
```

```
68:data_rom_out<=30;
69:data_rom_out<=29;
70:data_rom_out<=28;
71:data_rom_out<=28;
72:data_rom_out<=27;
73:data_rom_out<=26;
74:data_rom_out<=25;
75:data_rom_out<=24;
76:data_rom_out<=24;
77:data_rom_out<=23;
78:data_rom_out<=22;
79:data_rom_out<=21;
80:data_rom_out<=21;
81:data_rom_out<=20;
82:data_rom_out<=20;
83:data_rom_out<=19;
84:data_rom_out<=18;
85:data_rom_out<=18;
86:data_rom_out<=17;
87:data_rom_out<=17;
88:data_rom_out<=16;
89:data_rom_out<=16;
90:data_rom_out<=15;
91:data_rom_out<=15;
92:data_rom_out<=14;
93:data_rom_out<=14;
94:data_rom_out<=13;
95:data_rom_out<=13;
96:data_rom_out<=13;
97:data_rom_out<=12;
98:data_rom_out<=12;
99:data_rom_out<=11;
100:data_rom_out<=11;
101:data_rom_out<=11;
102:data_rom_out<=10;
103:data_rom_out<=10;
104:data_rom_out<=10;
105:data_rom_out<=9;
106:data_rom_out<=9;
107:data_rom_out<=9;
108:data_rom_out<=9;
109:data_rom_out<=8;
110:data_rom_out<=8;
111:data_rom_out<=8;
```

```
112:data_rom_out<=8;
113:data_rom_out<=7;
114:data_rom_out<=7;
115:data_rom_out<=7;
116:data_rom_out<=7;
117:data_rom_out<=7;
118:data_rom_out<=6;
119:data_rom_out<=6;
120:data_rom_out<=6;
121:data_rom_out<=6;
122:data_rom_out<=6;
123:data_rom_out<=5;
124:data_rom_out<=5;
125:data_rom_out<=5;
126:data_rom_out<=5;
127:data_rom_out<=5;
128:data_rom_out<=5;
129:data_rom_out<=4;
130:data_rom_out<=4;
131:data_rom_out<=4;
132:data_rom_out<=4;
133:data_rom_out<=4;
134:data_rom_out<=4;
135:data_rom_out<=4;
136:data_rom_out<=4;
137:data_rom_out<=3;
138:data_rom_out<=3;
139:data_rom_out<=3;
140:data_rom_out<=3;
141:data_rom_out<=3;
142:data_rom_out<=3;
143:data_rom_out<=3;
144:data_rom_out<=3;
145:data_rom_out<=3;
146:data_rom_out<=3;
147:data_rom_out<=3;
148:data_rom_out<=2;
149:data_rom_out<=2;
150:data_rom_out<=2;
151:data_rom_out<=2;
152:data_rom_out<=2;
153:data_rom_out<=2;
154:data_rom_out<=2;
155:data_rom_out<=2;
```

```
156:data_rom_out<=2;
157:data_rom_out<=2;
158:data_rom_out<=2;
159:data_rom_out<=2;
160:data_rom_out<=2;
161:data_rom_out<=2;
162:data_rom_out<=2;
163:data_rom_out<=2;
164:data_rom_out<=1;
165:data_rom_out<=1;
166:data_rom_out<=1;
167:data_rom_out<=1;
168:data_rom_out<=1;
169:data_rom_out<=1;
170:data_rom_out<=1;
171:data_rom_out<=1;
172:data_rom_out<=1;
173:data_rom_out<=1;
174:data_rom_out<=1;
175:data_rom_out<=1;
176:data_rom_out<=1;
177:data_rom_out<=1;
178:data_rom_out<=1;
179:data_rom_out<=1;
180:data_rom_out<=1;
181:data_rom_out<=1;
182:data_rom_out<=1;
183:data_rom_out<=1;
184:data_rom_out<=1;
185:data_rom_out<=1;
186:data_rom_out<=1;
187:data_rom_out<=1;
188:data_rom_out<=1;
189:data_rom_out<=1;
190:data_rom_out<=1;
191:data_rom_out<=1;
192:data_rom_out<=1;
193:data_rom_out<=1;
194:data_rom_out<=1;
195:data_rom_out<=1;
196:data_rom_out<=1;
197:data_rom_out<=1;
198:data_rom_out<=1;
default:data_rom_out<=0;
```

```
    endcase
    end
end

//多打 4 拍
always @ (posedge clk)
begin
  data_rom_buf1<=data_rom_out;
  data_rom_buf2<=data_rom_buf1;
  data_rom_buf3=data_rom_buf2;
  data_out<=data_rom_buf3;
end

endmodule
```

5.divider_fixed_all.v

```
//实现结果为 16 定点除法器功能，十六级流水，每一级流水求得结果中的一位
module divider_fixed_all
(
  input clk;  //输入时钟和复位信号
  input [15:0] dividend;  //输入被除数
  input [33:0] accumulator; //输入累加寄存器存储的值,作为除数
  output reg [15:0] division_result;  //存放除法最终结果
);

wire [16:0] dividend_lls_1; //存放左移 1 位的被除数,用于计算 reg_result_0

//寄存商的中间结果
reg reg_result_0;
reg [1:0]reg_result_1;
reg [2:0]reg_result_2;
reg [3:0]reg_result_3;
reg [4:0]reg_result_4;
reg [5:0]reg_result_5;
reg [6:0]reg_result_6;
reg [7:0]reg_result_7;
reg [8:0]reg_result_8;
reg [9:0]reg_result_9;
reg [10:0]reg_result_10;
reg [11:0]reg_result_11;
reg [12:0]reg_result_12;
reg [13:0]reg_result_13;
```

```
reg [14:0]reg_result_14;

//寄存被除数的中间结果
reg [15:0] reg_dividend_0;
reg [15:0] reg_dividend_1;
reg [16:0] reg_dividend_2;
reg [17:0] reg_dividend_3;
reg [18:0] reg_dividend_4;
reg [19:0] reg_dividend_5;
reg [20:0] reg_dividend_6;
reg [21:0] reg_dividend_7;
reg [22:0] reg_dividend_8;
reg [23:0] reg_dividend_9;
reg [24:0] reg_dividend_10;
reg [25:0] reg_dividend_11;
reg [26:0] reg_dividend_12;
reg [27:0] reg_dividend_13;
reg [28:0] reg_dividend_14;

//左移一位,用以计算除法结果第一位
assign dividend_lls_1=dividend<<1;

//第 1 级流水
always @(posedge clk)
begin
    if(dividend_lls_1>=accumulator)   //比较被除数和除数的大小
        begin
        reg_result_0<=1;   //第 1 位结果为 1
        reg_dividend_0<=(dividend_lls_1-accumulator)<<1;   //被除数减去除数并左移
        end
  else
        begin
        reg_result_0<=0;   //第 1 位结果为 0
        reg_dividend_0<=dividend_lls_1<<1;   //被除数直接左移
        end
end

//第 2 级流水
always @(posedge clk)
begin
    if((reg_dividend_0)>=accumulator)   //比较被除数和除数的大小
        begin
        reg_result_1[0]<=1; //第 2 位结果为 1
        reg_dividend_1<=(reg_dividend_0-accumulator)<<1;   //被除数减去除数并左移
```

```
        reg_result_1[1]<=reg_result_0;   //将上一级流水结果左移
        end
   else
        begin
        reg_result_1[0]<=0; //第 2 位结果为 0
        reg_dividend_1<=reg_dividend_0<<1;   //被除数直接左移
        reg_result_1[1]<=reg_result_0;   //将上一级流水结果左移
        end
end

//第 3 级流水
always @(posedge clk)
begin
     if((reg_dividend_1)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_2[0]<=1; //第 3 位结果为 1
        reg_dividend_2<=(reg_dividend_1-accumulator)<<1;   //被除数减去除数并左移
        reg_result_2[2:1]<=reg_result_1[1:0]; //将上一级流水结果左移
        end
     else
        begin
        reg_result_2[0]<=0; //第 3 位结果为 0
        reg_dividend_2<=reg_dividend_1<<1;   //被除数直接左移
        reg_result_2[2:1]<=reg_result_1[1:0]; //将上一级流水结果左移
        end
end

//第 4 级流水
always @(posedge clk)
begin
     if((reg_dividend_2)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_3[0]<=1; //第 4 位结果为 1
        reg_dividend_3<=(reg_dividend_2-accumulator)<<1;   //被除数减去除数并左移
        reg_result_3[3:1]<=reg_result_2[2:0]; //将上一级流水结果左移
        end
     else
        begin
        reg_result_3[0]<=0; //第 4 位结果为 0
        reg_dividend_3<=reg_dividend_2<<1;   //被除数直接左移
        reg_result_3[3:1]<=reg_result_2[2:0]; //将上一级流水结果左移
        end
end
```

```
//第 5 级流水
always @(posedge clk)
begin
    if((reg_dividend_3)>=accumulator)   //比较被除数和除数的大小
        begin
        reg_result_4[0]<=1; //第 5 位结果为 1
        reg_dividend_4<=(reg_dividend_3-accumulator)<<1;   //被除数减去除数并左移
        reg_result_4[4:1]<=reg_result_3[3:0]; //将上一级流水结果左移
        end
  else
        begin
        reg_result_4[0]<=0; //第 5 位结果为 0
        reg_dividend_4<=reg_dividend_3<<1;   //被除数直接左移
        reg_result_4[4:1]<=reg_result_3[3:0]; //将上一级流水结果左移
        end
end

//第 6 级流水
always @(posedge clk)
begin
     if((reg_dividend_4)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_5[0]<=1; //第 6 位结果为 1
        reg_dividend_5<=(reg_dividend_4-accumulator)<<1;   //被除数减去除数并左移
        reg_result_5[5:1]<=reg_result_4[4:0];   //将上一级流水结果左移
        end
  else
        begin
        reg_result_5[0]<=0; //第 6 位结果为 0
        reg_dividend_5<=reg_dividend_4<<1;   //被除数直接左移
        reg_result_5[5:1]<=reg_result_4[4:0];   //将上一级流水结果左移
        end
end

//第 7 级流水
always @(posedge clk)
begin
     if((reg_dividend_5)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_6[0]<=1;//第 7 位结果为 1
        reg_dividend_6<=(reg_dividend_5-accumulator)<<1; //除数减去被除数并左移
        reg_result_6[6:1]<=reg_result_5[5:0];//将上一级流水结果左移
        end
  else

`
```

```
        begin
        reg_result_6[0]<=0; //第 7 位结果为 0
        reg_dividend_6<=reg_dividend_5<<1;   //被除数直接左移
        reg_result_6[6:1]<=reg_result_5[5:0];   //将上一级流水结果左移
        end
end

//第 8 级流水
always @(posedge clk)
begin
    if((reg_dividend_6)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_7[0]<=1; //第 8 位结果为 1
        reg_dividend_7<=(reg_dividend_6-accumulator)<<1;//除数减去被除数并左移
        reg_result_7[7:1]<=reg_result_6[6:0];   //将上一级流水结果左移
        end
    else
        begin
        reg_result_7[0]<=0; //第 8 位结果为 0
        reg_dividend_7<=reg_dividend_6<<1;//被除数直接左移
        reg_result_7[7:1]<=reg_result_6[6:0];   //将上一级流水结果左移
        end
end

//第 9 级流水
always @(posedge clk)
begin
    if((reg_dividend_7)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_8[0]<=1; //第 9 位结果为 1
        reg_dividend_8<=(reg_dividend_7-accumulator)<<1;//除数减去被除数并左移
        reg_result_8[8:1]<=reg_result_7[7:0];//将上一级流水结果左移
        end
    else
        begin
        reg_result_8[0]<=0; //第 9 位结果为 0
        reg_dividend_8<=reg_dividend_7<<1;   //被除数直接左移
        reg_result_8[8:1]<=reg_result_7[7:0]; //将上一级流水结果左移
        end
end

//第 10 级流水
always @(posedge clk)
begin

`
```

```
    if((reg_dividend_8)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_9[0]<=1; //第 10 位结果为 1
      reg_dividend_9<=(reg_dividend_8-accumulator)<<1;//除数减去被除数并左移
      end
    else
      begin
      reg_result_9[0]<=0; //第 10 位结果为 0
      reg_dividend_9<=reg_dividend_8<<1;
      end

   reg_result_9[9:1]<=reg_result_8[8:0];   //将上一级流水结果左移

end

//第 11 级流水
always @(posedge clk)
begin

    if((reg_dividend_9)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_10[0]<=1; //第 11 位结果为 1
      reg_dividend_10<=(reg_dividend_9-accumulator)<<1;//除数减去被除数并左移
      end
    else
      begin
      reg_result_10[0]<=0; //第 11 位结果为 0
      reg_dividend_10<=reg_dividend_9<<1;
      end

    reg_result_10[10:1]<=reg_result_9[9:0];   //将上一级流水结果左移

end

//第 12 级流水
always @(posedge clk)
begin
    if((reg_dividend_10)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_11[0]<=1; //第 12 位结果为 1
      reg_dividend_11<=(reg_dividend_10-accumulator)<<1;//除数减去被除数并左移
      end
    else
      begin
```

```
        reg_result_11[0]<=0; //第 12 位结果为 0
        reg_dividend_11<=reg_dividend_10<<1;
        end

    reg_result_11[11:1]<=reg_result_10[10:0];   //将上一级流水结果左移

end

//第 13 级流水
always @(posedge clk)
begin
    if((reg_dividend_11)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_12[0]<=1; //第 13 位结果为 1
        reg_dividend_12<=(reg_dividend_11-accumulator)<<1;//除数减去被除数并左移
        end
    else
        begin
        reg_result_12[0]<=0; //第 13 位结果为 0
        reg_dividend_12<=reg_dividend_11<<1;
        end

    reg_result_12[12:1]<=reg_result_11[11:0];   //将上一级流水结果左移

end

//第 14 级流水
always @(posedge clk)
begin
    if((reg_dividend_12)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_13[0]<=1; //第 14 位结果为 1
        reg_dividend_13<=(reg_dividend_12-accumulator)<<1;//除数减去被除数并左移
        end
    else
        begin
        reg_result_13[0]<=0; //第 14 位结果为 0
        reg_dividend_13<=reg_dividend_12<<1;
        end

    reg_result_13[13:1]<=reg_result_12[12:0];   //将上一级流水结果左移

end
```

```
//第 15 级流水
always @(posedge clk)
begin
    if((reg_dividend_13)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_14[0]<=1; //第 15 位结果为 1
      reg_dividend_14<=(reg_dividend_13-accumulator)<<1;//除数减去被除数并左移
      end
    else
      begin
      reg_result_14[0]<=0; //第 15 位结果为 0
      reg_dividend_14<=reg_dividend_13<<1;
      end

    reg_result_14[14:1]<=reg_result_13[13:0];   //将上一级流水结果左移

end

//第 16 级流水
always @(posedge clk)
begin

    if(reg_dividend_14>=accumulator) //比较被除数和除数的大小
      division_result[0]<=1; //第 16 位结果为 1
    else
      division_result[0]<=0; //第 16 位结果为 0

    division_result[15:1]<=reg_result_14[14:0];   //将上一级流水结果左移

    end


endmodule
```

6.divider_fixed_8.v

```
//实现结果为 16 定点除法器功能,数据处理采用 8 级流水,每一级流水产生结果的一位,结果表示范围为
[0,1)
//同时为了与 divider_fixed_all 的十六级流水一致,缓存 8 个 clk
module divider_fixed_8
(
  input clk;   //输入时钟和复位信号
  input [7:0] dividend; //输入被除数
  input [28:0] accumulator; //输入累加寄存器存储的值,作为除数
```

`

```
   output reg [7:0] division_result; //存放除法最终结果
);

wire [8:0] dividend_lls_1;   //存放左移 1 位的被除数,用于计算 reg_result_0

//寄存商的中间结果
reg reg_result_0;
reg [1:0]reg_result_1;
reg [2:0]reg_result_2;
reg [3:0]reg_result_3;
reg [4:0]reg_result_4;
reg [5:0]reg_result_5;
reg [6:0]reg_result_6;
reg [7:0]reg_result_7;

//寄存被除数的中间结果
reg [9:0] reg_dividend_0;
reg [10:0] reg_dividend_1;
reg [11:0] reg_dividend_2;
reg [12:0] reg_dividend_3;
reg [13:0] reg_dividend_4;
reg [14:0] reg_dividend_5;
reg [15:0] reg_dividend_6;

//用于缓存 8 个节拍
reg [7:0] division_result_buffer0;
reg [7:0] division_result_buffer1;
reg [7:0] division_result_buffer2;
reg [7:0] division_result_buffer3;
reg [7:0] division_result_buffer4;
reg [7:0] division_result_buffer5;
reg [7:0] division_result_buffer6;

//左移一位,用以计算除法结果第一位
assign dividend_lls_1=dividend<<1;

//第 1 级流水
always @(posedge clk)
begin
    if(dividend_lls_1>=accumulator)   //比较被除数和除数的大小
        begin
        reg_result_0<=1;   //第 1 位结果为 1
        reg_dividend_0<=(dividend_lls_1-accumulator)<<1;   //被除数减去除数并左移
        end

`
```

```
  else
      begin
      reg_result_0<=0;  //第 1 位结果为 0
      reg_dividend_0<=dividend_lls_1<<1;  //被除数直接左移
      end
end

//第 2 级流水
always @(posedge clk)
begin
   if((reg_dividend_0)>=accumulator)  //比较被除数和除数的大小
      begin
      reg_result_1[0]<=1; //第 2 位结果为 1
      reg_dividend_1<=(reg_dividend_0-accumulator)<<1;  //被除数减去除数并左移
      reg_result_1[1]<=reg_result_0;  //将上一级流水结果左移
      end
  else
      begin
      reg_result_1[0]<=0; //第 2 位结果为 0
      reg_dividend_1<=reg_dividend_0<<1;  //被除数直接左移
      reg_result_1[1]<=reg_result_0;  //将上一级流水结果左移
      end
end

//第 3 级流水
always @(posedge clk)
begin
    if((reg_dividend_1)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_2[0]<=1; //第 3 位结果为 1
      reg_dividend_2<=(reg_dividend_1-accumulator)<<1;  //被除数减去除数并左移
      reg_result_2[2:1]<=reg_result_1[1:0];  //将上一级流水结果左移
      end
    else
      begin
      reg_result_2[0]<=0; //第 3 位结果为 0
      reg_dividend_2<=reg_dividend_1<<1;  //被除数直接左移
      reg_result_2[2:1]<=reg_result_1[1:0];  //将上一级流水结果左移
      end
end

//第 4 级流水
always @(posedge clk)
begin

`
```

```
    if((reg_dividend_2)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_3[0]<=1; //第 4 位结果为 1
      reg_dividend_3<=(reg_dividend_2-accumulator)<<1;   //被除数减去除数并左移
      reg_result_3[3:1]<=reg_result_2[2:0];   //将上一级流水结果左移
      end
    else
      begin
      reg_result_3[0]<=0; //第 4 位结果为 0
      reg_dividend_3<=reg_dividend_2<<1;   //被除数直接左移
      reg_result_3[3:1]<=reg_result_2[2:0];   //将上一级流水结果左移
      end
end

//第 5 级流水
always @(posedge clk)
begin
   if((reg_dividend_3)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_4[0]<=1; //第 5 位结果为 1
      reg_dividend_4<=(reg_dividend_3-accumulator)<<1;   //被除数减去除数并左移
      reg_result_4[4:1]<=reg_result_3[3:0];   //将上一级流水结果左移
      end
  else
      begin
      reg_result_4[0]<=0; //第 5 位结果为 0
      reg_dividend_4<=reg_dividend_3<<1;   //被除数直接左移
      reg_result_4[4:1]<=reg_result_3[3:0];   //将上一级流水结果左移
      end
end

//第 6 级流水
always @(posedge clk)
begin
    if((reg_dividend_4)>=accumulator) //比较被除数和除数的大小
      begin
      reg_result_5[0]<=1; //第 6 位结果为 1
      reg_dividend_5<=(reg_dividend_4-accumulator)<<1;   //被除数减去除数并左移
      reg_result_5[5:1]<=reg_result_4[4:0];   //将上一级流水结果左移
      end
  else
      begin
      reg_result_5[0]<=0; //第 6 位结果为 0
      reg_dividend_5<=reg_dividend_4<<1;   //被除数直接左移

`
```

```
        reg_result_5[5:1]<=reg_result_4[4:0];   //将上一级流水结果左移
        end
end

//第 7 级流水
always @(posedge clk)
begin
     if((reg_dividend_5)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_6[0]<=1; //第 7 位结果为 1
        reg_dividend_6<=(reg_dividend_5-accumulator)<<1;   //除数减去被除数并左移
        reg_result_6[6:1]<=reg_result_5[5:0];   //将上一级流水结果左移
        end
  else
        begin
        reg_result_6[0]<=0; //第 7 位结果为 0
        reg_dividend_6<=reg_dividend_5<<1;   //被除数直接左移
        reg_result_6[6:1]<=reg_result_5[5:0];   //将上一级流水结果左移
        end
end

//第 8 级流水
always @(posedge clk)
begin
     if((reg_dividend_6)>=accumulator) //比较被除数和除数的大小
        begin
        reg_result_7[0]<=1; //第 8 位结果为 1
        reg_result_7[7:1]<=reg_result_6[6:0];   //将上一级流水结果左移
        end
     else
        begin
        reg_result_7[0]<=0; //第 8 位结果为 0
        reg_result_7[7:1]<=reg_result_6[6:0];   //将上一级流水结果左移
        end
end

//数据缓存 8 个 clk
always @(posedge clk)
begin
     division_result_buffer0<=reg_result_7;
     division_result_buffer1<=division_result_buffer0;
     division_result_buffer2<=division_result_buffer1;
     division_result_buffer3<=division_result_buffer2;
     division_result_buffer4<=division_result_buffer3;
```

```
        division_result_buffer5<=division_result_buffer4;
        division_result_buffer6<=division_result_buffer5;
        division_result<=division_result_buffer6;
    end

    endmodule
```

7.find_max.v

```
module find_max
(
  input clk,  //时钟信号
  input rst,  //复位信号
  input init_txn_pulse_state, //顶层状态机启动信号,用于中间比较寄存器的复位
  input [63:0] data_in_64,  //输入总线 64 位数据
  input find_max_enable,  //模块使能信号
  input fixed_shift,  //8/16 定点方式切换
  input init_start_find_max,  //寻找最大值状态机启动信号
  output reg max_result_done, //寻找最大值状态机完成信号
  output reg [15:0] max_result  //输出最大值结果
);

//中间比较寄存器,暂时缓存比较后的多通道最大值
reg [14:0] compare_u0,compare_u1,compare_u2,compare_u3;
reg [6:0] compare_u4,compare_u5,compare_u6,compare_u7;

reg state_compare;  //状态机的状态
reg [2:0] count;  //计数器
reg [14:0] compare_buffer;  //缓存选通的数据

//将输入总线的 64 位数据根据定点方式切分成对应 4 或者 8 个有效的数据
wire [15:0] data_in_u0,data_in_u1,data_in_u2,data_in_u3;
wire [7:0] data_in_u4,data_in_u5,data_in_u6,data_in_u7;

assign data_in_u0[14:0]=fixed_shift ? data_in_64[6:0]   : data_in_64[14:0]  ;
assign data_in_u1[14:0]=fixed_shift ? data_in_64[14:8]  : data_in_64[30:16] ;
assign data_in_u2[14:0]=fixed_shift ? data_in_64[22:16] : data_in_64[46:32] ;
assign data_in_u3[14:0]=fixed_shift ? data_in_64[30:24] : data_in_64[62:48] ;
assign data_in_u4[6:0]=data_in_64[38:32];
assign data_in_u5[6:0]=data_in_64[46:40];
assign data_in_u6[6:0]=data_in_64[54:48];
assign data_in_u7[6:0]=data_in_64[62:56];
```

```
assign data_in_u0[15]=fixed_shift ? data_in_64[7]      : data_in_64[15] ;
assign data_in_u1[15]=fixed_shift ? data_in_64[15]     : data_in_64[31] ;
assign data_in_u2[15]=fixed_shift ? data_in_64[23]     : data_in_64[47] ;
assign data_in_u3[15]=fixed_shift ? data_in_64[31]     : data_in_64[63] ;
assign data_in_u4[7]= data_in_64[39] ;
assign data_in_u5[7]= data_in_64[47] ;
assign data_in_u6[7]= data_in_64[55] ;
assign data_in_u7[7]= data_in_64[63] ;

//通过计数器,实现需要比较的数据的选通
always @(*)
begin
  case(count)
  0:compare_buffer<=compare_u0;
  1:compare_buffer<=compare_u1;
  2:compare_buffer<=compare_u2;
  3:compare_buffer<=compare_u3;
  4:compare_buffer<=compare_u4;
  5:compare_buffer<=compare_u5;
  6:compare_buffer<=compare_u6;
  7:compare_buffer<=compare_u7;
  default:compare_buffer<=0;
  endcase
end

//寻找最大值状态机
always @(posedge clk)
begin
  if( rst==0 || init_txn_pulse_state==1)   //通过顶层状态机或者复位信号同步复位
    begin
    state_compare<=0;
    max_result_done<=0;
    max_result<=0;
    count<=0;
    end
  else
  begin
    case(state_compare)
    0:
    begin
      if(init_start_find_max) //检测到启动信号,进入下一个状态,同时计数器清零,最大值赋值为第一个
中间比较寄存器的结果
        begin
          max_result<=compare_u0;
```

```
            state_compare<=1;
            count<=0;
          end
      end

      1:
      begin
        count<=count+1; //计数器加 1 来选通所需数据

        if(max_result < compare_buffer)
          max_result<=compare_buffer; //比较得较大值

        if(fixed_shift && count==7) //当定点方式是 8 位,比较 8 个数
          begin
          state_compare<=0; //返回空闲态
          max_result_done<=1; //发出完成信号
          end
        else if( (!fixed_shift) && count==3)  //当定点方式是 16 位,比较 4 个数
          begin
          state_compare<=0; //返回空闲态
          max_result_done<=1; //发出完成信号
          end
        else
          begin
          state_compare<=state_compare;
          max_result_done<=max_result_done;
          end

      end
      default:state_compare<=0;
      endcase
    end
end

//以下 8 个 always 语句块:用于单通道比较
//利用 顶层状态机信号 init_txn_pulse_state 进行复位
//单通道比较时,判断是否为负数,如果是负数的话,较大值不变,如果是正数,进行比较,保存较大值
always @(posedge clk)
begin
  if( rst==0 || init_txn_pulse_state==1 )
    compare_u0<=0;
  else if(find_max_enable && data_in_u0[15])
    compare_u0<=compare_u0;
  else if(find_max_enable && compare_u0<data_in_u0)
```

```
      compare_u0<=data_in_u0;
    else
      compare_u0<=compare_u0;
end

always @(posedge clk)
begin
    if( rst==0 || init_txn_pulse_state==1 )
      compare_u1<=0;
    else if(find_max_enable && data_in_u1[15])
      compare_u1<=compare_u1;
    else if(find_max_enable && compare_u1<data_in_u1)
      compare_u1<=data_in_u1;
    else
      compare_u1<=compare_u1;
end

always @(posedge clk)
begin
    if( rst==0 || init_txn_pulse_state==1 )
      compare_u2<=0;
    else if(find_max_enable && data_in_u2[15])
      compare_u2<=compare_u2;
    else if(find_max_enable && compare_u2<data_in_u2)
      compare_u2<=data_in_u2;
    else
      compare_u2<=compare_u2;
end

always @(posedge clk)
begin
    if( rst==0 || init_txn_pulse_state==1 )
      compare_u3<=0;
    else if(find_max_enable && data_in_u3[15])
      compare_u3<=compare_u3;
    else if(find_max_enable && compare_u3<data_in_u3)
      compare_u3<=data_in_u3;
    else
      compare_u3<=compare_u3;
end

always @(posedge clk)
begin
    if( rst==0 || init_txn_pulse_state==1 )
```

```
      compare_u4<=0;
   else if(find_max_enable && data_in_u4[7])
      compare_u4<=compare_u4;
   else if(find_max_enable && compare_u4<data_in_u4)
      compare_u4<=data_in_u4;
   else
      compare_u4<=compare_u4;
end

always @(posedge clk)
begin
   if( rst==0 || init_txn_pulse_state==1 )
      compare_u5<=0;
   else if(find_max_enable && data_in_u5[7])
      compare_u5<=compare_u5;
   else if(find_max_enable && compare_u5<data_in_u5)
      compare_u5<=data_in_u5;
   else
      compare_u5<=compare_u5;
end

always @(posedge clk)
begin
   if( rst==0 || init_txn_pulse_state==1 )
      compare_u6<=0;
   else if(find_max_enable && data_in_u6[7])
      compare_u6<=compare_u6;
   else if(find_max_enable && compare_u6<data_in_u6)
      compare_u6<=data_in_u6;
   else
      compare_u6<=compare_u6;
end

always @(posedge clk)
begin
   if( rst==0 || init_txn_pulse_state==1 )
      compare_u7<=0;
   else if(find_max_enable && data_in_u7[7])
      compare_u7<=compare_u7;
   else if(find_max_enable && compare_u7<data_in_u7)
      compare_u7<=data_in_u7;
   else
      compare_u7<=compare_u7;
end
```

```
endmodule
```

8.accumulator_final_result.v

```
module accumulator_final_result
(
  input clk,   //时钟信号
  input rst,   //复位信号
  input init_txn_pulse_state, //顶层状态机启动信号,同时对次顶层状态机和相应的寄存器复位
  input fixed_shift_flag, //定点方式切换标志位,用于 8 定点和 16 定点的切换
  input init_start_accumulator, //启动累加状态机
  input [31:0] accumulator_0,
  input [31:0] accumulator_1,
  input [31:0] accumulator_2,
  input [31:0] accumulator_3,
  input [25:0] accumulator_4,
  input [25:0] accumulator_5,
  input [25:0] accumulator_6,
  input [25:0] accumulator_7, //以上 8 个寄存器用于存储单通道累加值
  output reg [33:0] accumulator_result, //该寄存器为最终累加结果寄存器,用于存储累加状态机完成
时,最终得到的累加值
  output reg accumulator_done //累加状态机完成信号,传递给顶层状态机使其进入下一个状态
);

reg state;//状态
reg [31:0] accumulator_buffer;   //用于选通操作数的缓存值
reg [2:0] cnt; //选择累加的操作数

always @(*)
begin
    case(cnt)
    0:accumulator_buffer<=accumulator_0;
    1:accumulator_buffer<=accumulator_1;
    2:accumulator_buffer<=accumulator_2;
    3:accumulator_buffer<=accumulator_3;
    4:accumulator_buffer<=accumulator_4;
    5:accumulator_buffer<=accumulator_5;
    6:accumulator_buffer<=accumulator_6;
    7:accumulator_buffer<=accumulator_7;
    default:accumulator_buffer<=0;

`
```

```
    endcase
end

always @ (posedge clk)
begin
  if( rst==0 || init_txn_pulse_state==1)
    begin
    state<=0;
    accumulator_result<=0;
    accumulator_done<=0;
    cnt<=0;
    end
  else
  begin
    case(state)
    0:
    begin
      if(init_start_accumulator)  //累加状态机启动,进入工作状态,并且将相应寄存器进行复位
        begin
        state<=1;
        accumulator_done<=0;
        cnt<=0;
        accumulator_result<=0;
        end
    end

    1:
    begin
      if(fixed_shift_flag)  //如果标志位为 1,判断为 8 定点方式,进行 8 次累加,否则确定为 16 定点方式,
进行 4 次累加
        begin
          if(cnt==7)
            begin
            state<=0;
            accumulator_done<=1;  //状态机完成
            end
          else
            begin
            state<=state;
            accumulator_done<=accumulator_done;
            end
        end
        else
        begin
```

```
            if(cnt==3)
              begin
              state<=0;
              accumulator_done<=1;   //状态机完成
              end
            else
              begin
              state<=state;
              accumulator_done<=accumulator_done;
              end
          end

          accumulator_result<=accumulator_result+accumulator_buffer; //结果寄存器和选通的数
据进行累加

          cnt<=cnt+1; //选择下一个数

        end
        default:
        begin
        state<=0;
        accumulator_done<=0;
        cnt<=0;
        accumulator_result<=0;
        end
        endcase
    end
end

endmodule
```

9.fifo.v

```
module fifo
(
    input rst,   //同步复位信号
    input clk,   //时钟信号
    input wr_en,   //写使能
    input rd_en,   //读使能
    input [63:0] data_in, //输入数据
    output wire [63:0] data_out,   //输出数据
    output wire full, //满信号
```

`

```
  output wire empty,   //空信号
  output wire almost_full,   //将慢信号
  output wire almost_empty   //将空信号
);

reg [63:0] memory[0:31];   //存储 ram
reg [5:0] fifo_cnt; //计数器,用于对 ram 存储值的个数计数
reg [4:0] rd_p; //读指针
reg [4:0] wr_p; //写指针

assign full   = (fifo_cnt == 32)?1'b1:1'b0;
assign empty = (fifo_cnt == 0)?1'b1:1'b0;
assign almost_full   = (fifo_cnt == 31)?1'b1:1'b0;
assign almost_empty = (fifo_cnt == 1)?1'b1:1'b0;

always @(posedge clk)
begin
    if(!rst)
        fifo_cnt <= 0;
    else if((!full&&wr_en)&&(!empty&&rd_en))   //同时读写,计数器值不变
        fifo_cnt <= fifo_cnt;
    else if(!full && wr_en) //写一次,计数器值加上 1
        fifo_cnt <= fifo_cnt + 1;
    else if(!empty&& rd_en) //读一次,计数器值减去 1
        fifo_cnt <= fifo_cnt-1;
    else
        fifo_cnt <= fifo_cnt; //无读写操作,计数器值不变
end

assign data_out = memory[rd_p]; //输出数据就是读指针指向的那个 ram 存储单元

always @(posedge clk)
begin
    if(wr_en && !full)
        memory[wr_p] <= data_in;   //写一次,将写指针对应 ram 存储单元赋为输入数据值
end

always @(posedge clk)
begin
    if(!rst)
        wr_p <= 0;
    else
    begin
        if(!full && wr_en)   //进行写操作时,写指针加 1

`
```

```
            wr_p <= wr_p + 1;
        else
            wr_p <= wr_p;
    end
end

always @(posedge clk)
begin
    if(!rst)
        rd_p <= 0;
    else
    begin
        if(!empty && rd_en) //进行读操作时,读指针加 1
            rd_p <= rd_p + 1;
        else
            rd_p <= rd_p;
    end
end

endmodule
```

二、顶层测试搭建

1.control_test.v

```
`timescale 1 ns/ 1 ps

module control_test
(
  input clk,
  input rst,
  input init_all_test,
  input single_test_done,
  output reg init_test_softmax_module
);



integer i;
reg [15:0] tmp_data;
reg signed [15:0] tmp_signed_data;
integer fp_r;
integer fp_w;
reg [12:0] write_address;
```

```
integer cnt_differ_0;   //对误差绝对值为 0 计数
integer cnt_differ_1;   //对误差绝对值为 1 计数
integer cnt_differ_2;   //对误差绝对值为 2 计数
integer cnt_differ_more;   //对误差绝对值为 3 计数

integer fp_log;

initial
begin

//创建用于记录误差分布,统计数目的文本
  fp_log=$fopen("/home/hubbery/log_differ_number.txt","w");

//启动信号初始化为 0
  init_test_softmax_module=0;

//顶层启动
  @(posedge init_all_test)
   begin
   /*-------------------------------------------------------------------------
                                    开始测试 rand0.1   16bit
   ------------------------------------------------------------------------- */

    //计数清零
    cnt_differ_0=0;
    cnt_differ_1=0;
    cnt_differ_2=0;
    cnt_differ_more=0;

    //启动 rand0.1 测试
    init_test_softmax_module=1;

    //写入 rand0.1 测试数据集
    fp_r=$fopen("/home/hubbery/fixed0.1.txt","r");
    fp_w=$fopen("/home/hubbery/datain.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);
```

```
    //写入 rand0.1 验证数据集
    fp_r=$fopen("/home/hubbery/rand_verify_0.1.txt","r");
    fp_w=$fopen("/home/hubbery/verify.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand0.1 控制寄存器
    fp_w=$fopen("/home/hubbery/registerin.txt","w");
    $fwrite(fp_w, "%h ",64'h0000_0100_007D_0000);
    $fwrite(fp_w, "\n");
    $fclose(fp_w);
  end

//关闭启动信号
@(posedge clk) init_test_softmax_module=0;

//记录比较结果
@(posedge single_test_done)
  begin
    fp_r=$fopen("/home/hubbery/output.txt","r");
    fp_w=$fopen("/home/hubbery/log_0.1_compare_16bit.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
      $fscanf(fp_r, "%d",tmp_signed_data);

      //统计误差,计数
      if(tmp_signed_data==0)
        cnt_differ_0=cnt_differ_0+1;
      else if( tmp_signed_data==1 || tmp_signed_data==-1 )
        cnt_differ_1=cnt_differ_1+1;
      else if( tmp_signed_data==2 || tmp_signed_data==-2 )
        cnt_differ_2=cnt_differ_2+1;
      else
        cnt_differ_more=cnt_differ_more+1;

      $fwrite(fp_w, "%d ",tmp_signed_data);
      $fwrite(fp_w, "\n");
    end
```

```
        $fclose(fp_r);
        $fclose(fp_w);
    end

    $fwrite(fp_log, "The result of test_rand0.1 (16bit):\n");
    $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "The sum is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
    $fwrite(fp_log, "\n\n\n");

/*------------------------------------------------------------------------
                                    开始测试 rand1   16bit
------------------------------------------------------------------------ */

    //计数清零
    cnt_differ_0=0;
    cnt_differ_1=0;
    cnt_differ_2=0;
    cnt_differ_more=0;

    //启动信号
    init_test_softmax_module=1;

    //写入 rand1 测试数据集
    fp_r=$fopen("/home/hubbery/fixed1.txt","r");
    fp_w=$fopen("/home/hubbery/datain.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand1 验证数据集
    fp_r=$fopen("/home/hubbery/rand_verify_1.txt","r");
    fp_w=$fopen("/home/hubbery/verify.txt","w");
```

```
for(i=0;i<4096;i=i+1)
begin
    $fscanf(fp_r, "%d",tmp_data);
    $fwrite(fp_w, "%d ",tmp_data);
    $fwrite(fp_w, "\n");
end
$fclose(fp_r);
$fclose(fp_w);

//写入 rand1 控制寄存器
fp_w=$fopen("/home/hubbery/registerin.txt","w");
$fwrite(fp_w, "%h ",64'h0000_0100_0081_0000);
$fwrite(fp_w, "\n");
$fclose(fp_w);



//关闭启动信号
@(posedge clk) init_test_softmax_module=0;

//记录比较结果
@(posedge single_test_done)
 begin
  fp_r=$fopen("/home/hubbery/output.txt","r");
  fp_w=$fopen("/home/hubbery/log_1_compare_16bit.txt","w");
  for(i=0;i<4096;i=i+1)
  begin
    $fscanf(fp_r, "%d",tmp_signed_data);

    //统计误差,计数
    if(tmp_signed_data==0)
      cnt_differ_0=cnt_differ_0+1;
    else if( tmp_signed_data==1 || tmp_signed_data==-1 )
      cnt_differ_1=cnt_differ_1+1;
    else if( tmp_signed_data==2 || tmp_signed_data==-2 )
      cnt_differ_2=cnt_differ_2+1;
    else
      cnt_differ_more=cnt_differ_more+1;

    $fwrite(fp_w, "%d ",tmp_signed_data);
    $fwrite(fp_w, "\n");
  end
  $fclose(fp_r);
  $fclose(fp_w);
 end
```

```
        $fwrite(fp_log, "The result of test_rand1 (16bit):\n");
        $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log,                                    "The                                    sum
is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
        $fwrite(fp_log, "\n\n\n");

/*--------------------------------------------------------------------------
                                    开始测试 rand5   16bit
 -------------------------------------------------------------------------- */

//计数清零
cnt_differ_0=0;
cnt_differ_1=0;
cnt_differ_2=0;
cnt_differ_more=0;

//启动信号
init_test_softmax_module=1;

//写入 rand5 测试数据集
fp_r=$fopen("/home/hubbery/fixed5.txt","r");
fp_w=$fopen("/home/hubbery/datain.txt","w");
for(i=0;i<4096;i=i+1)
begin
    $fscanf(fp_r, "%d",tmp_data);
    $fwrite(fp_w, "%d ",tmp_data);
    $fwrite(fp_w, "\n");
end
$fclose(fp_r);
$fclose(fp_w);

//写入 rand5 验证数据集
fp_r=$fopen("/home/hubbery/rand_verify_5.txt","r");
fp_w=$fopen("/home/hubbery/verify.txt","w");
for(i=0;i<4096;i=i+1)
begin
```

```
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand5 控制寄存器
    fp_w=$fopen("/home/hubbery/registerin.txt","w");
    $fwrite(fp_w, "%h ",64'h0000_0100_0083_0000);
    $fwrite(fp_w, "\n");
    $fclose(fp_w);


    //关闭启动信号
    @(posedge clk) init_test_softmax_module=0;

    //记录比较结果
    @(posedge single_test_done)
     begin
      fp_r=$fopen("/home/hubbery/output.txt","r");
      fp_w=$fopen("/home/hubbery/log_5_compare_16bit.txt","w");
      for(i=0;i<4096;i=i+1)
      begin
        $fscanf(fp_r, "%d",tmp_signed_data);

        //统计误差,计数
        if(tmp_signed_data==0)
          cnt_differ_0=cnt_differ_0+1;
        else if( tmp_signed_data==1 || tmp_signed_data==-1 )
          cnt_differ_1=cnt_differ_1+1;
        else if( tmp_signed_data==2 || tmp_signed_data==-2 )
          cnt_differ_2=cnt_differ_2+1;
        else
          cnt_differ_more=cnt_differ_more+1;

        $fwrite(fp_w, "%d ",tmp_signed_data);
        $fwrite(fp_w, "\n");
      end
      $fclose(fp_r);
      $fclose(fp_w);

     end
```

```
 $fwrite(fp_log, "The result of test_rand5 (16bit):\n");
 $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "The sum is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
 $fwrite(fp_log, "\n\n\n");

/*-------------------------------------------------------------------------
                                    开始测试 rand10   16bit
------------------------------------------------------------------------- */

    //计数清零
    cnt_differ_0=0;
    cnt_differ_1=0;
    cnt_differ_2=0;
    cnt_differ_more=0;

    //启动信号
    init_test_softmax_module=1;

    //写入 rand10 测试数据集
    fp_r=$fopen("/home/hubbery/fixed10.txt","r");
    fp_w=$fopen("/home/hubbery/datain.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand10 验证数据集
    fp_r=$fopen("/home/hubbery/rand_verify_10.txt","r");
    fp_w=$fopen("/home/hubbery/verify.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
```

```
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand10 控制寄存器
    fp_w=$fopen("/home/hubbery/registerin.txt","w");
    $fwrite(fp_w, "%h ",64'h0000_0100_0084_0000);
    $fwrite(fp_w, "\n");
    $fclose(fp_w);


    //关闭启动信号
    @(posedge clk) init_test_softmax_module=0;

    //记录比较结果
    @(posedge single_test_done)
     begin
      fp_r=$fopen("/home/hubbery/output.txt","r");
      fp_w=$fopen("/home/hubbery/log_10_compare_16bit.txt","w");
      for(i=0;i<4096;i=i+1)
      begin
        $fscanf(fp_r, "%d",tmp_signed_data);

        //统计误差,计数
        if(tmp_signed_data==0)
          cnt_differ_0=cnt_differ_0+1;
        else if( tmp_signed_data==1 || tmp_signed_data==-1 )
          cnt_differ_1=cnt_differ_1+1;
        else if( tmp_signed_data==2 || tmp_signed_data==-2 )
          cnt_differ_2=cnt_differ_2+1;
        else
          cnt_differ_more=cnt_differ_more+1;

        $fwrite(fp_w, "%d ",tmp_signed_data);
        $fwrite(fp_w, "\n");
      end
      $fclose(fp_r);
      $fclose(fp_w);
    end

     $fwrite(fp_log, "The result of test_rand10 (16bit):\n");
     $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
     $fwrite(fp_log, "\n");
```

```
        $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log,
"The sum is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
        $fwrite(fp_log, "\n\n\n");



    /*-------------------------------------------------------------------------
                                        开始测试 rand0.1    8bit
    ------------------------------------------------------------------------- */

     //计数清零
     cnt_differ_0=0;
     cnt_differ_1=0;
     cnt_differ_2=0;
     cnt_differ_more=0;

     //启动 rand0.1 测试
     init_test_softmax_module=1;

     //写入 rand0.1 测试数据集
     fp_r=$fopen("/home/hubbery/fixed0.1.txt","r");
     fp_w=$fopen("/home/hubbery/datain.txt","w");
     for(i=0;i<4096;i=i+1)
     begin
          $fscanf(fp_r, "%d",tmp_data);
          $fwrite(fp_w, "%d ",tmp_data);
          $fwrite(fp_w, "\n");
     end
     $fclose(fp_r);
     $fclose(fp_w);

     //写入 rand0.1 验证数据集
     fp_r=$fopen("/home/hubbery/rand_verify_0.1.txt","r");
     fp_w=$fopen("/home/hubbery/verify.txt","w");
     for(i=0;i<4096;i=i+1)
     begin
          $fscanf(fp_r, "%d",tmp_data);
          $fwrite(fp_w, "%d ",tmp_data);
```

```
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand0.1 控制寄存器
    fp_w=$fopen("/home/hubbery/registerin.txt","w");
    $fwrite(fp_w, "%h ",64'h0000_0101_007C_0000);
    $fwrite(fp_w, "\n");
    $fclose(fp_w);

  //关闭启动信号
  @(posedge clk) init_test_softmax_module=0;

  //记录比较结果
  @(posedge single_test_done)
   begin
    fp_r=$fopen("/home/hubbery/output.txt","r");
    fp_w=$fopen("/home/hubbery/log_0.1_compare_8bit.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
      $fscanf(fp_r, "%d",tmp_signed_data);

      //统计误差,计数
      if(tmp_signed_data==0)
        cnt_differ_0=cnt_differ_0+1;
      else if( tmp_signed_data==1 || tmp_signed_data==-1 )
        cnt_differ_1=cnt_differ_1+1;
      else if( tmp_signed_data==2 || tmp_signed_data==-2 )
        cnt_differ_2=cnt_differ_2+1;
      else
        cnt_differ_more=cnt_differ_more+1;

      $fwrite(fp_w, "%d ",tmp_signed_data);
      $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);
  end

  $fwrite(fp_log, "The result of test_rand0.1 (8bit):\n");
  $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
  $fwrite(fp_log, "\n");
  $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
```

```
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
    $fwrite(fp_log, "\n");
    $fwrite(fp_log, "The sum is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
    $fwrite(fp_log, "\n\n\n");

/*--------------------------------------------------------------------------
                                    开始测试 rand1   8bit
-------------------------------------------------------------------------- */

    //计数清零
    cnt_differ_0=0;
    cnt_differ_1=0;
    cnt_differ_2=0;
    cnt_differ_more=0;

    //启动信号
    init_test_softmax_module=1;

    //写入 rand1 测试数据集
    fp_r=$fopen("/home/hubbery/fixed1.txt","r");
    fp_w=$fopen("/home/hubbery/datain.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand1 验证数据集
    fp_r=$fopen("/home/hubbery/rand_verify_1.txt","r");
    fp_w=$fopen("/home/hubbery/verify.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);
```

```
//写入 rand1 控制寄存器
fp_w=$fopen("/home/hubbery/registerin.txt","w");
$fwrite(fp_w, "%h ",64'h0000_0101_0080_0000);
$fwrite(fp_w, "\n");
$fclose(fp_w);


//关闭启动信号
@(posedge clk) init_test_softmax_module=0;

//记录比较结果
@(posedge single_test_done)
 begin
  fp_r=$fopen("/home/hubbery/output.txt","r");
  fp_w=$fopen("/home/hubbery/log_1_compare_8bit.txt","w");
  for(i=0;i<4096;i=i+1)
  begin
    $fscanf(fp_r, "%d",tmp_signed_data);

    //统计误差,计数
    if(tmp_signed_data==0)
      cnt_differ_0=cnt_differ_0+1;
    else if( tmp_signed_data==1 || tmp_signed_data==-1 )
      cnt_differ_1=cnt_differ_1+1;
    else if( tmp_signed_data==2 || tmp_signed_data==-2 )
      cnt_differ_2=cnt_differ_2+1;
    else
      cnt_differ_more=cnt_differ_more+1;

    $fwrite(fp_w, "%d ",tmp_signed_data);
    $fwrite(fp_w, "\n");
  end
  $fclose(fp_r);
  $fclose(fp_w);
 end

 $fwrite(fp_log, "The result of test_rand1 (8bit):\n");
 $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
 $fwrite(fp_log, "\n");
```

```
        $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log,                                    "The                                    sum
is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
        $fwrite(fp_log, "\n\n\n");

/*--------------------------------------------------------------------------
                                    开始测试 rand5   8bit
 -------------------------------------------------------------------------- */

//计数清零
cnt_differ_0=0;
cnt_differ_1=0;
cnt_differ_2=0;
cnt_differ_more=0;

//启动信号
init_test_softmax_module=1;

//写入 rand5 测试数据集
fp_r=$fopen("/home/hubbery/fixed5.txt","r");
fp_w=$fopen("/home/hubbery/datain.txt","w");
for(i=0;i<4096;i=i+1)
begin
    $fscanf(fp_r, "%d",tmp_data);
    $fwrite(fp_w, "%d ",tmp_data);
    $fwrite(fp_w, "\n");
end
$fclose(fp_r);
$fclose(fp_w);

//写入 rand5 验证数据集
fp_r=$fopen("/home/hubbery/rand_verify_5.txt","r");
fp_w=$fopen("/home/hubbery/verify.txt","w");
for(i=0;i<4096;i=i+1)
begin
    $fscanf(fp_r, "%d",tmp_data);
    $fwrite(fp_w, "%d ",tmp_data);
    $fwrite(fp_w, "\n");
end
$fclose(fp_r);
$fclose(fp_w);

//写入 rand5 控制寄存器

`
```

```
fp_w=$fopen("/home/hubbery/registerin.txt","w");
$fwrite(fp_w, "%h ",64'h0000_0101_0082_0000);
$fwrite(fp_w, "\n");
$fclose(fp_w);


//关闭启动信号
@(posedge clk) init_test_softmax_module=0;

//记录比较结果
@(posedge single_test_done)
 begin
  fp_r=$fopen("/home/hubbery/output.txt","r");
  fp_w=$fopen("/home/hubbery/log_5_compare_8bit.txt","w");
  for(i=0;i<4096;i=i+1)
  begin
    $fscanf(fp_r, "%d",tmp_signed_data);

    //统计误差,计数
    if(tmp_signed_data==0)
      cnt_differ_0=cnt_differ_0+1;
    else if( tmp_signed_data==1 || tmp_signed_data==-1 )
      cnt_differ_1=cnt_differ_1+1;
    else if( tmp_signed_data==2 || tmp_signed_data==-2 )
      cnt_differ_2=cnt_differ_2+1;
    else
      cnt_differ_more=cnt_differ_more+1;

    $fwrite(fp_w, "%d ",tmp_signed_data);
    $fwrite(fp_w, "\n");
  end
  $fclose(fp_r);
  $fclose(fp_w);

 end

 $fwrite(fp_log, "The result of test_rand5 (8bit):\n");
 $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
```

```
 $fwrite(fp_log, "\n");
 $fwrite(fp_log, "The sum is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
 $fwrite(fp_log, "\n\n\n");

/*--------------------------------------------------------------------------
                                    开始测试 rand10   8bit
-------------------------------------------------------------------------- */

    //计数清零
    cnt_differ_0=0;
    cnt_differ_1=0;
    cnt_differ_2=0;
    cnt_differ_more=0;

    //启动信号
    init_test_softmax_module=1;

    //写入 rand10 测试数据集
    fp_r=$fopen("/home/hubbery/fixed10.txt","r");
    fp_w=$fopen("/home/hubbery/datain.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand10 验证数据集
    fp_r=$fopen("/home/hubbery/rand_verify_10.txt","r");
    fp_w=$fopen("/home/hubbery/verify.txt","w");
    for(i=0;i<4096;i=i+1)
    begin
        $fscanf(fp_r, "%d",tmp_data);
        $fwrite(fp_w, "%d ",tmp_data);
        $fwrite(fp_w, "\n");
    end
    $fclose(fp_r);
    $fclose(fp_w);

    //写入 rand10 控制寄存器
    fp_w=$fopen("/home/hubbery/registerin.txt","w");
    $fwrite(fp_w, "%h ",64'h0000_0101_0083_0000);
```

```
        $fwrite(fp_w, "\n");
        $fclose(fp_w);


        //关闭启动信号
        @(posedge clk) init_test_softmax_module=0;

        //记录比较结果
        @(posedge single_test_done)
         begin
          fp_r=$fopen("/home/hubbery/output.txt","r");
          fp_w=$fopen("/home/hubbery/log_10_compare_8bit.txt","w");
          for(i=0;i<4096;i=i+1)
          begin
            $fscanf(fp_r, "%d",tmp_signed_data);

            //统计误差,计数
            if(tmp_signed_data==0)
              cnt_differ_0=cnt_differ_0+1;
            else if( tmp_signed_data==1 || tmp_signed_data==-1 )
              cnt_differ_1=cnt_differ_1+1;
            else if( tmp_signed_data==2 || tmp_signed_data==-2 )
              cnt_differ_2=cnt_differ_2+1;
            else
              cnt_differ_more=cnt_differ_more+1;

            $fwrite(fp_w, "%d ",tmp_signed_data);
            $fwrite(fp_w, "\n");
          end
          $fclose(fp_r);
          $fclose(fp_w);
        end

        $fwrite(fp_log, "The result of test_rand10 (8bit):\n");
        $fwrite(fp_log, "cnt_differ_0 = %d ",cnt_differ_0);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_1 = %d ",cnt_differ_1);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_2 = %d ",cnt_differ_2);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log, "cnt_differ_more = %d ",cnt_differ_more);
        $fwrite(fp_log, "\n");
        $fwrite(fp_log,                                    "The                                    sum
is %d",cnt_differ_0+cnt_differ_1+cnt_differ_2+cnt_differ_more);
```

```
        $fwrite(fp_log, "\n\n\n");

/*------------------------------------------------------------------------------ */

      $fclose(fp_log);

/*------------------------------------------------------------------------------ */

   $stop;
end

endmodule
```

2.test_softmax

(1)test_softmax_v1_0.v

```
`timescale 1 ns / 1 ps

      module test_softmax_v1_0 #
      (
            parameter C_M00_AXI_register_address = 32'h00FFFFF8,
            parameter   C_M00_AXI_TARGET_SLAVE_BASE_ADDR     = 32'h01000000,
            parameter integer C_M00_AXI_BURST_LEN      = 16,
            parameter integer C_M00_AXI_ID_WIDTH  = 1,
            parameter integer C_M00_AXI_ADDR_WIDTH   = 32,
            parameter integer C_M00_AXI_DATA_WIDTH    = 64,
            parameter integer C_M00_AXI_AWUSER_WIDTH      = 0,
            parameter integer C_M00_AXI_ARUSER_WIDTH      = 0,
            parameter integer C_M00_AXI_WUSER_WIDTH = 0,
            parameter integer C_M00_AXI_RUSER_WIDTH  = 0,
            parameter integer C_M00_AXI_BUSER_WIDTH  = 0
      )
      (
            input wire   m00_axi_init_axi_txn,
            output wire m00_axi_fixed_shift_signal,
            output wire   m00_axi_txn_done,
            input wire   m00_axi_aclk,
            input wire   m00_axi_aresetn,
            output wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_awid,
            output wire [C_M00_AXI_ADDR_WIDTH-1 : 0] m00_axi_awaddr,
            output wire [7 : 0] m00_axi_awlen,
            output wire [2 : 0] m00_axi_awsize,

`
```

```
        output wire [1 : 0] m00_axi_awburst,
        output wire    m00_axi_awlock,
        output wire [3 : 0] m00_axi_awcache,
        output wire [2 : 0] m00_axi_awprot,
        output wire [3 : 0] m00_axi_awqos,
        output wire [C_M00_AXI_AWUSER_WIDTH-1 : 0] m00_axi_awuser,
        output wire    m00_axi_awvalid,
        input wire    m00_axi_awready,
        output wire [C_M00_AXI_DATA_WIDTH-1 : 0] m00_axi_wdata,
        output wire [C_M00_AXI_DATA_WIDTH/8-1 : 0] m00_axi_wstrb,
        output wire    m00_axi_wlast,
        output wire [C_M00_AXI_WUSER_WIDTH-1 : 0] m00_axi_wuser,
        output wire    m00_axi_wvalid,
        input wire    m00_axi_wready,
        input wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_bid,
        input wire [1 : 0] m00_axi_bresp,
        input wire [C_M00_AXI_BUSER_WIDTH-1 : 0] m00_axi_buser,
        input wire    m00_axi_bvalid,
        output wire    m00_axi_bready,
        output wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_arid,
        output wire [C_M00_AXI_ADDR_WIDTH-1 : 0] m00_axi_araddr,
        output wire [7 : 0] m00_axi_arlen,
        output wire [2 : 0] m00_axi_arsize,
        output wire [1 : 0] m00_axi_arburst,
        output wire    m00_axi_arlock,
        output wire [3 : 0] m00_axi_arcache,
        output wire [2 : 0] m00_axi_arprot,
        output wire [3 : 0] m00_axi_arqos,
        output wire [C_M00_AXI_ARUSER_WIDTH-1 : 0] m00_axi_aruser,
        output wire    m00_axi_arvalid,
        input wire    m00_axi_arready,
        input wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_rid,
        input wire [C_M00_AXI_DATA_WIDTH-1 : 0] m00_axi_rdata,
        input wire [1 : 0] m00_axi_rresp,
        input wire    m00_axi_rlast,
        input wire [C_M00_AXI_RUSER_WIDTH-1 : 0] m00_axi_ruser,
        input wire    m00_axi_rvalid,
        output wire    m00_axi_rready
    );

    test_softmax_v1_0_M00_AXI # (
        .register_address(C_M00_AXI_register_address),
        .C_M_TARGET_SLAVE_BASE_ADDR(C_M00_AXI_TARGET_SLAVE_BASE_ADDR),
        .C_M_AXI_BURST_LEN(C_M00_AXI_BURST_LEN),
```

```
        .C_M_AXI_ID_WIDTH(C_M00_AXI_ID_WIDTH),
        .C_M_AXI_ADDR_WIDTH(C_M00_AXI_ADDR_WIDTH),
        .C_M_AXI_DATA_WIDTH(C_M00_AXI_DATA_WIDTH),
        .C_M_AXI_AWUSER_WIDTH(C_M00_AXI_AWUSER_WIDTH),
        .C_M_AXI_ARUSER_WIDTH(C_M00_AXI_ARUSER_WIDTH),
        .C_M_AXI_WUSER_WIDTH(C_M00_AXI_WUSER_WIDTH),
        .C_M_AXI_RUSER_WIDTH(C_M00_AXI_RUSER_WIDTH),
        .C_M_AXI_BUSER_WIDTH(C_M00_AXI_BUSER_WIDTH)
    ) test_softmax_v1_0_M00_AXI_inst (
        .INIT_AXI_TXN(m00_axi_init_axi_txn),
        .fixed_shift(m00_axi_fixed_shift_signal),
        .TXN_DONE(m00_axi_txn_done),
        .M_AXI_ACLK(m00_axi_aclk),
        .M_AXI_ARESETN(m00_axi_aresetn),
        .M_AXI_AWID(m00_axi_awid),
        .M_AXI_AWADDR(m00_axi_awaddr),
        .M_AXI_AWLEN(m00_axi_awlen),
        .M_AXI_AWSIZE(m00_axi_awsize),
        .M_AXI_AWBURST(m00_axi_awburst),
        .M_AXI_AWLOCK(m00_axi_awlock),
        .M_AXI_AWCACHE(m00_axi_awcache),
        .M_AXI_AWPROT(m00_axi_awprot),
        .M_AXI_AWQOS(m00_axi_awqos),
        .M_AXI_AWUSER(m00_axi_awuser),
        .M_AXI_AWVALID(m00_axi_awvalid),
        .M_AXI_AWREADY(m00_axi_awready),
        .M_AXI_WDATA(m00_axi_wdata),
        .M_AXI_WSTRB(m00_axi_wstrb),
        .M_AXI_WLAST(m00_axi_wlast),
        .M_AXI_WUSER(m00_axi_wuser),
        .M_AXI_WVALID(m00_axi_wvalid),
        .M_AXI_WREADY(m00_axi_wready),
        .M_AXI_BID(m00_axi_bid),
        .M_AXI_BRESP(m00_axi_bresp),
        .M_AXI_BUSER(m00_axi_buser),
        .M_AXI_BVALID(m00_axi_bvalid),
        .M_AXI_BREADY(m00_axi_bready),
        .M_AXI_ARID(m00_axi_arid),
        .M_AXI_ARADDR(m00_axi_araddr),
        .M_AXI_ARLEN(m00_axi_arlen),
        .M_AXI_ARSIZE(m00_axi_arsize),
        .M_AXI_ARBURST(m00_axi_arburst),
        .M_AXI_ARLOCK(m00_axi_arlock),
        .M_AXI_ARCACHE(m00_axi_arcache),
```

```
        .M_AXI_ARPROT(m00_axi_arprot),
        .M_AXI_ARQOS(m00_axi_arqos),
        .M_AXI_ARUSER(m00_axi_aruser),
        .M_AXI_ARVALID(m00_axi_arvalid),
        .M_AXI_ARREADY(m00_axi_arready),
        .M_AXI_RID(m00_axi_rid),
        .M_AXI_RDATA(m00_axi_rdata),
        .M_AXI_RRESP(m00_axi_rresp),
        .M_AXI_RLAST(m00_axi_rlast),
        .M_AXI_RUSER(m00_axi_ruser),
        .M_AXI_RVALID(m00_axi_rvalid),
        .M_AXI_RREADY(m00_axi_rready)
    );
    endmodule
```

(2)test_softmax_v1_0_M00_AXI.v

```
`timescale 1 ns / 1 ps

    module test_softmax_v1_0_M00_AXI #
    (
        parameter   register_address = 32'h00FFFFF8,
        parameter   C_M_TARGET_SLAVE_BASE_ADDR = 32'h40000000,
        parameter integer C_M_AXI_BURST_LEN   = 16,
        parameter integer C_M_AXI_ID_WIDTH     = 1,
        parameter integer C_M_AXI_ADDR_WIDTH       = 32,
        parameter integer C_M_AXI_DATA_WIDTH = 32,
        parameter integer C_M_AXI_AWUSER_WIDTH   = 0,
        parameter integer C_M_AXI_ARUSER_WIDTH   = 0,
        parameter integer C_M_AXI_WUSER_WIDTH    = 0,
        parameter integer C_M_AXI_RUSER_WIDTH    = 0,
        parameter integer C_M_AXI_BUSER_WIDTH    = 0
    )
    (
       input wire   INIT_AXI_TXN, //ip 核数据处理启动信号
       output wire fixed_shift,
        output reg   TXN_DONE,      //ip 核数据处理完成信号
        input wire   M_AXI_ACLK, //时钟信号
        input wire   M_AXI_ARESETN, //复位信号

`
```

```
        output wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_AWID,
        output wire [C_M_AXI_ADDR_WIDTH-1 : 0] M_AXI_AWADDR,
        output wire [7 : 0] M_AXI_AWLEN,
        output wire [2 : 0] M_AXI_AWSIZE,
        output wire [1 : 0] M_AXI_AWBURST,
        output wire   M_AXI_AWLOCK,
        output wire [3 : 0] M_AXI_AWCACHE,
        output wire [2 : 0] M_AXI_AWPROT,
        output wire [3 : 0] M_AXI_AWQOS,
        output wire [C_M_AXI_AWUSER_WIDTH-1 : 0] M_AXI_AWUSER,
        output wire   M_AXI_AWVALID,
        input wire   M_AXI_AWREADY,
        output wire [C_M_AXI_DATA_WIDTH-1 : 0] M_AXI_WDATA,
        output wire [C_M_AXI_DATA_WIDTH/8-1 : 0] M_AXI_WSTRB,
        output wire   M_AXI_WLAST,
        output wire [C_M_AXI_WUSER_WIDTH-1 : 0] M_AXI_WUSER,
        output wire   M_AXI_WVALID,
        input wire   M_AXI_WREADY,
        input wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_BID,
        input wire [1 : 0] M_AXI_BRESP,
        input wire [C_M_AXI_BUSER_WIDTH-1 : 0] M_AXI_BUSER,
        input wire   M_AXI_BVALID,
        output wire   M_AXI_BREADY,
        output wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_ARID,
        output wire [C_M_AXI_ADDR_WIDTH-1 : 0] M_AXI_ARADDR,
        output wire [7 : 0] M_AXI_ARLEN,
        output wire [2 : 0] M_AXI_ARSIZE,
        output wire [1 : 0] M_AXI_ARBURST,
        output wire   M_AXI_ARLOCK,
        output wire [3 : 0] M_AXI_ARCACHE,
        output wire [2 : 0] M_AXI_ARPROT,
        output wire [3 : 0] M_AXI_ARQOS,
        output wire [C_M_AXI_ARUSER_WIDTH-1 : 0] M_AXI_ARUSER,
        output wire   M_AXI_ARVALID, //
        input wire   M_AXI_ARREADY,
        input wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_RID,
        input wire [C_M_AXI_DATA_WIDTH-1 : 0] M_AXI_RDATA,
        input wire [1 : 0] M_AXI_RRESP,
        input wire   M_AXI_RLAST,
        input wire [C_M_AXI_RUSER_WIDTH-1 : 0] M_AXI_RUSER,
        input wire   M_AXI_RVALID,
        output wire   M_AXI_RREADY
    );
```

```
    function integer clogb2 (input integer bit_depth);
    begin
      for(clogb2=0; bit_depth>0; clogb2=clogb2+1)
        bit_depth = bit_depth >> 1;
      end
    endfunction

    localparam integer C_TRANSACTIONS_NUM = clogb2(C_M_AXI_BURST_LEN-1);

    localparam integer C_MASTER_LENGTH   = 13;

    localparam integer C_NO_BURSTS_REQ
= C_MASTER_LENGTH-clogb2((C_M_AXI_BURST_LEN*C_M_AXI_DATA_WIDTH/8)-1);

    reg [1:0] mst_exec_state;
    reg [1:0] single_write_state;
    reg [2:0] control_state;

    reg [C_M_AXI_ADDR_WIDTH-1 : 0]    axi_awaddr;
    reg       axi_awvalid;
    reg       axi_wlast;
    reg       axi_wvalid;
    reg       axi_bready;
    reg [C_TRANSACTIONS_NUM : 0]      write_index;
    wire [C_TRANSACTIONS_NUM+3 : 0]       burst_size_bytes;
    reg [C_NO_BURSTS_REQ+8 : 0]       write_burst_counter;
    reg       start_single_burst_write;
    reg       burst_write_active;
    wire      wnext;
    reg       init_txn_ff;
    reg       init_txn_ff2;
    reg       init_txn_edge;

    reg       init_txn_pulse_write;
    reg       writes_done;

    reg       init_single_write;
    reg       single_write_done;

    wire init_txn_pulse_state;

   wire [63:0] data_64;

   reg control_single_write;
```

```
 reg single_write_awvalid;
 reg single_write_wlast;
 reg single_write_wvalid;
 reg single_write_bready;

reg [63:0] register_content;

reg [15:0] memory [0:4095];
 reg [11:0] memory_p;
 reg [11:0] address;
 integer cnt,i;
 integer fp;

assign fixed_shift=register_content[32];

 initial
 begin
    forever
    begin
     @(posedge init_txn_pulse_state)
      begin
       fp=$fopen("/home/hubbery/datain.txt","r");
       address=0;
       while(!$feof(fp))
       begin
     for(i=0; i<4096; i=i+1)
          begin
          cnt = $fscanf (fp, "%d",memory[address]);
          address=address+1;
       end
     end
     $fclose(fp);

     fp=$fopen("/home/hubbery/registerin.txt","r");
     cnt=$fscanf (fp, "%h",register_content);
     $fclose(fp);
     end
  end
 end


 always @(posedge M_AXI_ACLK)
 begin
```

```
            if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
                memory_p <= 0 ;
            else if (wnext)
                memory_p <= memory_p + 1;
            else
                memory_p <= memory_p;
            end

  assign data_64
=register_content[32]  ?{memory[8*memory_p+7][15],memory[8*memory_p+7][13:8],me
mory[8*memory_p+6][15],memory[8*memory_p+6][13:8],memory[8*memory_p+5][15],m
emory[8*memory_p+5][13:8],memory[8*memory_p+4][15],memory[8*memory_p+4][13:8]
,memory[8*memory_p+3][15],memory[8*memory_p+3][13:8],memory[8*memory_p+2][1
5],memory[8*memory_p+2][13:8],memory[8*memory_p+1][15],memory[8*memory_p+1][
13:8],memory[8*memory_p][15],memory[8*memory_p][13:8]}                    :
{memory[4*memory_p+3],memory[4*memory_p+2],memory[4*memory_p+1],memory[4*
memory_p]} ;

      assign M_AXI_WDATA   = control_single_write ? register_content : data_64;

      assign M_AXI_AWID     = 'b0;
      assign M_AXI_AWADDR=       control_single_write       ?       register_address       :
(C_M_TARGET_SLAVE_BASE_ADDR + axi_awaddr) ;
      assign M_AXI_AWLEN   = control_single_write ?   0 : (C_M_AXI_BURST_LEN - 1);
      assign M_AXI_AWSIZE  = 3'b011;
      assign M_AXI_AWBURST     = 2'b01;
      assign M_AXI_AWLOCK = 1'b0;
      assign M_AXI_AWCACHE     = 4'b0010;
      assign M_AXI_AWPROT = 3'h0;
      assign M_AXI_AWQOS  = 4'h0;
      assign M_AXI_AWUSER = 'b1;
      assign M_AXI_AWVALID= control_single_write ? single_write_awvalid :axi_awvalid;
      assign M_AXI_WSTRB   ={(C_M_AXI_DATA_WIDTH/8){1'b1}};
      assign M_AXI_WLAST   = control_single_write ? single_write_wlast : axi_wlast ;
      assign M_AXI_WUSER   = 'b0;
      assign M_AXI_WVALID  = control_single_write ? single_write_wvalid : axi_wvalid;
      assign M_AXI_BREADY  = control_single_write ? single_write_bready : axi_bready;
      assign M_AXI_ARID     = 'b0;
      assign M_AXI_ARADDR = 'b0;
      assign M_AXI_ARLEN    = 'b0;
      assign M_AXI_ARSIZE   = 3'b011;
      assign M_AXI_ARBURST     = 2'b01;
      assign M_AXI_ARLOCK  = 1'b0;
      assign M_AXI_ARCACHE     = 4'b0010;
```

```
assign M_AXI_ARPROT  = 3'h0;
assign M_AXI_ARQOS   = 4'h0;
assign M_AXI_ARUSER  = 'b1;
assign M_AXI_ARVALID ='b0;
assign M_AXI_RREADY  = 'b0;
assign burst_size_bytes      = C_M_AXI_BURST_LEN * C_M_AXI_DATA_WIDTH/8;

assign init_txn_pulse_state  = (!init_txn_ff2) && init_txn_ff;


      always @(posedge M_AXI_ACLK)
      begin
            if (M_AXI_ARESETN == 0 )
            begin
                  init_txn_ff <= 1'b0;
                  init_txn_ff2 <= 1'b0;
            end
            else
            begin
                  init_txn_ff <= INIT_AXI_TXN;
                  init_txn_ff2 <= init_txn_ff;
            end
      end

      always @(posedge M_AXI_ACLK)
      begin
            if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
                  axi_awvalid <= 1'b0;
            end
            else if (~axi_awvalid && start_single_burst_write)
            begin
                  axi_awvalid <= 1'b1;
            end
            else if (M_AXI_AWREADY && axi_awvalid)
            begin
                  axi_awvalid <= 1'b0;
            end
            else
            begin
                  axi_awvalid <= axi_awvalid;
            end
      end
```

```
        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
            begin
              axi_awaddr <= 'b0;
            end
          else if (M_AXI_AWREADY && axi_awvalid)
            begin
              axi_awaddr <= axi_awaddr + burst_size_bytes;
            end
          else
            axi_awaddr <= axi_awaddr;
          end

        assign wnext = M_AXI_WREADY & axi_wvalid;

           always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
              axi_wvalid <= 1'b0;
            end
          else if (~axi_wvalid && start_single_burst_write)
            begin
              axi_wvalid <= 1'b1;
            end
          else if (wnext && axi_wlast)
            axi_wvalid <= 1'b0;
          else
            axi_wvalid <= axi_wvalid;
        end

        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
              axi_wlast <= 1'b0;
            end
           else if (((write_index == C_M_AXI_BURST_LEN-2 && C_M_AXI_BURST_LEN >= 2) &&
wnext) || (C_M_AXI_BURST_LEN == 1 ))
            begin
              axi_wlast <= 1'b1;
            end
          else if (wnext)
```

```
            axi_wlast <= 1'b0;
          else if (axi_wlast && C_M_AXI_BURST_LEN == 1)
            axi_wlast <= 1'b0;
          else
            axi_wlast <= axi_wlast;
        end

        always @(posedge M_AXI_ACLK)
        begin
           if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 || start_single_burst_write
== 1'b1)
            begin
              write_index <= 0;
            end
          else if (wnext && (write_index != C_M_AXI_BURST_LEN-1))
            begin
              write_index <= write_index + 1;
            end
          else
            write_index <= write_index;
        end

        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
            begin
              axi_bready <= 1'b0;
            end
          else if (M_AXI_BVALID && ~axi_bready)
            begin
              axi_bready <= 1'b1;
            end
          else if (axi_bready)
            begin
              axi_bready <= 1'b0;
            end
                else
            axi_bready <= axi_bready;
        end



        always @(posedge M_AXI_ACLK)
        begin
          if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1 )
```

```
        begin
          write_burst_counter <= 'b0;
        end
      else if (M_AXI_AWREADY && axi_awvalid)
        begin
          if (write_burst_counter[C_NO_BURSTS_REQ-register_content[32]] == 1'b0)
            begin
              write_burst_counter <= write_burst_counter + 1'b1;
            end
        end
      else
        write_burst_counter <= write_burst_counter;
    end

     always @(posedge M_AXI_ACLK)
    begin
      if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
        burst_write_active <= 1'b0;
      else if (start_single_burst_write)
        burst_write_active <= 1'b1;
      else if (M_AXI_BVALID && axi_bready)
        burst_write_active <= 0;
    end

    always @(posedge M_AXI_ACLK)
    begin
      if (M_AXI_ARESETN == 0 || init_txn_pulse_write == 1'b1)
        writes_done <= 1'b0;
                                          else          if          (M_AXI_BVALID          &&
(write_burst_counter[C_NO_BURSTS_REQ-register_content[32]]) && axi_bready)
        writes_done <= 1'b1;
      else
        writes_done <= writes_done;
    end

    always @ ( posedge M_AXI_ACLK)
    begin
      if (M_AXI_ARESETN == 1'b0 )
        begin
          mst_exec_state      <= 2'b00;
          start_single_burst_write <= 1'b0;
        end
      else
        begin
```

```
        case (mst_exec_state)

          2'b00:
            if ( init_txn_pulse_write == 1'b1)
              begin
                mst_exec_state   <= 2'b01;
              end
            else
              begin
                mst_exec_state   <= 2'b00;
              end

          2'b01:
            if (writes_done )
              begin
                mst_exec_state <= 2'b10;
              end
            else
              begin
                mst_exec_state   <= 2'b01;

                if (~axi_awvalid && ~start_single_burst_write && ~burst_write_active)
                  begin
                    start_single_burst_write <= 1'b1;
                  end
                else
                  begin
                    start_single_burst_write <= 1'b0;
                  end
                end

          2'b10:
            begin
              mst_exec_state <= 2'b00;
            end
          default :
            begin
              mst_exec_state   <= 2'b00;
            end
        endcase
      end
  end

always @(posedge M_AXI_ACLK)
```

```
begin
    if(M_AXI_ARESETN == 0 || init_txn_pulse_state)
        begin
        single_write_state<=0;
        single_write_awvalid<=0;
        single_write_wlast<=0;
        single_write_wvalid<=0;
        single_write_bready<=0;
        control_single_write<=0;
        single_write_done<=0;
        end
    else
        case(single_write_state)
        0:
        begin
            if(init_single_write)
                begin
                single_write_state<=single_write_state+1;
                control_single_write<=1;
                single_write_done<=0;
                end
        end

        1:
        begin
            if(single_write_awvalid && M_AXI_AWREADY )
                begin
                single_write_state<=single_write_state+1;
                single_write_awvalid<=0;
                end
            else
                begin
                single_write_awvalid<=1;
                single_write_state<=single_write_state;
                end
        end

        2:
        begin
            if(single_write_wvalid && M_AXI_WREADY )
                begin
                single_write_wlast<=0;
                single_write_wvalid<=0;
                single_write_state<=single_write_state+1;
```

```
                    end
                else
                    begin
                    single_write_wlast<=1;
                    single_write_wvalid<=1;
                    single_write_state<=single_write_state;
                    end
            end

            3:
            begin
                if(single_write_bready && M_AXI_BVALID )
                    begin
                    single_write_state<=0;
                    single_write_bready<=0;
                    control_single_write<=0;
                    single_write_done<=1;
                    end
                else
                    begin
                    single_write_bready<=1;
                    single_write_state<=single_write_state;
                    control_single_write<=1;
                    single_write_done<=0;
                    end
            end

            default:
            begin
            single_write_state<=0;
            single_write_awvalid<=0;
            single_write_wlast<=0;
            single_write_wvalid<=0;
            single_write_bready<=0;
            control_single_write<=0;
            single_write_done<=0;
            end
            endcase
    end

always @(posedge M_AXI_ACLK)
begin
    if(M_AXI_ARESETN == 0)
        begin
```

```
        control_state<=0;
        init_txn_pulse_write<=0;
        init_single_write<=0;
        TXN_DONE<=0;
        end
    else
        case(control_state)
        0:
        begin
            if(init_txn_pulse_state)
                begin
                control_state<=control_state+1;
                TXN_DONE<=0;
                init_single_write<=1;
                end
            else
                begin
                control_state<=control_state;
                TXN_DONE<=TXN_DONE;
                init_single_write<=0;
                end
        end

       1:
       begin
           if(init_single_write)
           begin
               init_single_write<=0;
               control_state<=control_state+1;
           end
       end


        2:
        begin
            if(single_write_done)
                begin
                control_state<=control_state+1;
                init_txn_pulse_write<=1;
                end
            else
                begin
                control_state<=control_state;
                init_txn_pulse_write<=init_txn_pulse_write;
```

```
                end
        end

        3:
        begin
            if(init_txn_pulse_write)
                begin
                    init_txn_pulse_write<=0;
                    control_state<=control_state+1;
                end
        end

        4:
        begin

            if(writes_done)
                begin
                    control_state<=0;
                    TXN_DONE<=1;
                end
            else
                begin
                    control_state<=control_state;
                    TXN_DONE<=TXN_DONE;
                end

        end

        default:
        begin
        control_state<=0;
        init_txn_pulse_write<=0;
        init_single_write<=0;
        TXN_DONE<=0;
        end
        endcase
end


    endmodule
```

3.compare_test

`

(1)compare_test_v1_0.v

```
`timescale 1 ns / 1 ps

	module compare_test_v1_0 #
	(
		parameter	C_M00_AXI_TARGET_SLAVE_BASE_ADDR	= 32'h01000000,
		parameter integer C_M00_AXI_BURST_LEN	= 16,
		parameter integer C_M00_AXI_ID_WIDTH	= 1,
		parameter integer C_M00_AXI_ADDR_WIDTH	= 32,
		parameter integer C_M00_AXI_DATA_WIDTH	= 64,
		parameter integer C_M00_AXI_AWUSER_WIDTH	= 0,
		parameter integer C_M00_AXI_ARUSER_WIDTH	= 0,
		parameter integer C_M00_AXI_WUSER_WIDTH	= 0,
		parameter integer C_M00_AXI_RUSER_WIDTH	= 0,
		parameter integer C_M00_AXI_BUSER_WIDTH	= 0
	)
	(
		input wire	m00_axi_init_axi_txn,
		input wire m00_axi_fixed_shift_signal,
		output wire	m00_axi_txn_done,
		input wire	m00_axi_aclk,
		input wire	m00_axi_aresetn,
		output wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_awid,
		output wire [C_M00_AXI_ADDR_WIDTH-1 : 0] m00_axi_awaddr,
		output wire [7 : 0] m00_axi_awlen,
		output wire [2 : 0] m00_axi_awsize,
		output wire [1 : 0] m00_axi_awburst,
		output wire	m00_axi_awlock,
		output wire [3 : 0] m00_axi_awcache,
		output wire [2 : 0] m00_axi_awprot,
		output wire [3 : 0] m00_axi_awqos,
		output wire [C_M00_AXI_AWUSER_WIDTH-1 : 0] m00_axi_awuser,
		output wire	m00_axi_awvalid,
		input wire	m00_axi_awready,
		output wire [C_M00_AXI_DATA_WIDTH-1 : 0] m00_axi_wdata,
		output wire [C_M00_AXI_DATA_WIDTH/8-1 : 0] m00_axi_wstrb,
		output wire	m00_axi_wlast,
		output wire [C_M00_AXI_WUSER_WIDTH-1 : 0] m00_axi_wuser,
		output wire	m00_axi_wvalid,
		input wire	m00_axi_wready,
		input wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_bid,
```

```
        input wire [1 : 0] m00_axi_bresp,
        input wire [C_M00_AXI_BUSER_WIDTH-1 : 0] m00_axi_buser,
        input wire    m00_axi_bvalid,
        output wire    m00_axi_bready,
        output wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_arid,
        output wire [C_M00_AXI_ADDR_WIDTH-1 : 0] m00_axi_araddr,
        output wire [7 : 0] m00_axi_arlen,
        output wire [2 : 0] m00_axi_arsize,
        output wire [1 : 0] m00_axi_arburst,
        output wire    m00_axi_arlock,
        output wire [3 : 0] m00_axi_arcache,
        output wire [2 : 0] m00_axi_arprot,
        output wire [3 : 0] m00_axi_arqos,
        output wire [C_M00_AXI_ARUSER_WIDTH-1 : 0] m00_axi_aruser,
        output wire    m00_axi_arvalid,
        input wire    m00_axi_arready,
        input wire [C_M00_AXI_ID_WIDTH-1 : 0] m00_axi_rid,
        input wire [C_M00_AXI_DATA_WIDTH-1 : 0] m00_axi_rdata,
        input wire [1 : 0] m00_axi_rresp,
        input wire    m00_axi_rlast,
        input wire [C_M00_AXI_RUSER_WIDTH-1 : 0] m00_axi_ruser,
        input wire    m00_axi_rvalid,
        output wire    m00_axi_rready
    );
    compare_test_v1_0_M00_AXI # (
        .C_M_TARGET_SLAVE_BASE_ADDR(C_M00_AXI_TARGET_SLAVE_BASE_ADDR),
        .C_M_AXI_BURST_LEN(C_M00_AXI_BURST_LEN),
        .C_M_AXI_ID_WIDTH(C_M00_AXI_ID_WIDTH),
        .C_M_AXI_ADDR_WIDTH(C_M00_AXI_ADDR_WIDTH),
        .C_M_AXI_DATA_WIDTH(C_M00_AXI_DATA_WIDTH),
        .C_M_AXI_AWUSER_WIDTH(C_M00_AXI_AWUSER_WIDTH),
        .C_M_AXI_ARUSER_WIDTH(C_M00_AXI_ARUSER_WIDTH),
        .C_M_AXI_WUSER_WIDTH(C_M00_AXI_WUSER_WIDTH),
        .C_M_AXI_RUSER_WIDTH(C_M00_AXI_RUSER_WIDTH),
        .C_M_AXI_BUSER_WIDTH(C_M00_AXI_BUSER_WIDTH)
    ) compare_test_v1_0_M00_AXI_inst (
        .INIT_AXI_TXN(m00_axi_init_axi_txn),
        .fixed_shift(m00_axi_fixed_shift_signal),
        .TXN_DONE(m00_axi_txn_done),
        .M_AXI_ACLK(m00_axi_aclk),
        .M_AXI_ARESETN(m00_axi_aresetn),
        .M_AXI_AWID(m00_axi_awid),
        .M_AXI_AWADDR(m00_axi_awaddr),
        .M_AXI_AWLEN(m00_axi_awlen),
```

```
        .M_AXI_AWSIZE(m00_axi_awsize),
        .M_AXI_AWBURST(m00_axi_awburst),
        .M_AXI_AWLOCK(m00_axi_awlock),
        .M_AXI_AWCACHE(m00_axi_awcache),
        .M_AXI_AWPROT(m00_axi_awprot),
        .M_AXI_AWQOS(m00_axi_awqos),
        .M_AXI_AWUSER(m00_axi_awuser),
        .M_AXI_AWVALID(m00_axi_awvalid),
        .M_AXI_AWREADY(m00_axi_awready),
        .M_AXI_WDATA(m00_axi_wdata),
        .M_AXI_WSTRB(m00_axi_wstrb),
        .M_AXI_WLAST(m00_axi_wlast),
        .M_AXI_WUSER(m00_axi_wuser),
        .M_AXI_WVALID(m00_axi_wvalid),
        .M_AXI_WREADY(m00_axi_wready),
        .M_AXI_BID(m00_axi_bid),
        .M_AXI_BRESP(m00_axi_bresp),
        .M_AXI_BUSER(m00_axi_buser),
        .M_AXI_BVALID(m00_axi_bvalid),
        .M_AXI_BREADY(m00_axi_bready),
        .M_AXI_ARID(m00_axi_arid),
        .M_AXI_ARADDR(m00_axi_araddr),
        .M_AXI_ARLEN(m00_axi_arlen),
        .M_AXI_ARSIZE(m00_axi_arsize),
        .M_AXI_ARBURST(m00_axi_arburst),
        .M_AXI_ARLOCK(m00_axi_arlock),
        .M_AXI_ARCACHE(m00_axi_arcache),
        .M_AXI_ARPROT(m00_axi_arprot),
        .M_AXI_ARQOS(m00_axi_arqos),
        .M_AXI_ARUSER(m00_axi_aruser),
        .M_AXI_ARVALID(m00_axi_arvalid),
        .M_AXI_ARREADY(m00_axi_arready),
        .M_AXI_RID(m00_axi_rid),
        .M_AXI_RDATA(m00_axi_rdata),
        .M_AXI_RRESP(m00_axi_rresp),
        .M_AXI_RLAST(m00_axi_rlast),
        .M_AXI_RUSER(m00_axi_ruser),
        .M_AXI_RVALID(m00_axi_rvalid),
        .M_AXI_RREADY(m00_axi_rready)
    );
    endmodule
```

(2)compare_test_v1_0_M00_AXI.v

```
`timescale 1 ns / 1 ps

	module compare_test_v1_0_M00_AXI #
	(
		parameter	C_M_TARGET_SLAVE_BASE_ADDR = 32'h01000000,
		parameter integer C_M_AXI_BURST_LEN	= 16,
		parameter integer C_M_AXI_ID_WIDTH	= 1,
		parameter integer C_M_AXI_ADDR_WIDTH	= 32,
		parameter integer C_M_AXI_DATA_WIDTH = 64,
		parameter integer C_M_AXI_AWUSER_WIDTH	= 0,
		parameter integer C_M_AXI_ARUSER_WIDTH	= 0,
		parameter integer C_M_AXI_WUSER_WIDTH	= 0,
		parameter integer C_M_AXI_RUSER_WIDTH	= 0,
		parameter integer C_M_AXI_BUSER_WIDTH	= 0
	)
	(
		input wire	INIT_AXI_TXN,
		input wire fixed_shift,
		output reg	TXN_DONE,
		input wire	M_AXI_ACLK,
		input wire	M_AXI_ARESETN,
		output wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_AWID,
		output wire [C_M_AXI_ADDR_WIDTH-1 : 0] M_AXI_AWADDR,
		output wire [7 : 0] M_AXI_AWLEN,
		output wire [2 : 0] M_AXI_AWSIZE,
		output wire [1 : 0] M_AXI_AWBURST,
		output wire	M_AXI_AWLOCK,
		output wire [3 : 0] M_AXI_AWCACHE,
		output wire [2 : 0] M_AXI_AWPROT,
		output wire [3 : 0] M_AXI_AWQOS,
		output wire [C_M_AXI_AWUSER_WIDTH-1 : 0] M_AXI_AWUSER,
		output wire	M_AXI_AWVALID,
		input wire	M_AXI_AWREADY,
		output wire [C_M_AXI_DATA_WIDTH-1 : 0] M_AXI_WDATA,
		output wire [C_M_AXI_DATA_WIDTH/8-1 : 0] M_AXI_WSTRB,
		output wire	M_AXI_WLAST,
		output wire [C_M_AXI_WUSER_WIDTH-1 : 0] M_AXI_WUSER,
		output wire	M_AXI_WVALID,
		input wire	M_AXI_WREADY,
		input wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_BID,
		input wire [1 : 0] M_AXI_BRESP,
```

```
        input wire [C_M_AXI_BUSER_WIDTH-1 : 0] M_AXI_BUSER,
        input wire    M_AXI_BVALID,
        output wire    M_AXI_BREADY,
        output wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_ARID,
        output wire [C_M_AXI_ADDR_WIDTH-1 : 0] M_AXI_ARADDR,
        output wire [7 : 0] M_AXI_ARLEN,
        output wire [2 : 0] M_AXI_ARSIZE,
        output wire [1 : 0] M_AXI_ARBURST,
        output wire    M_AXI_ARLOCK,
        output wire [3 : 0] M_AXI_ARCACHE,
        output wire [2 : 0] M_AXI_ARPROT,
        output wire [3 : 0] M_AXI_ARQOS,
        output wire [C_M_AXI_ARUSER_WIDTH-1 : 0] M_AXI_ARUSER,
        output wire    M_AXI_ARVALID,
        input wire    M_AXI_ARREADY,
        input wire [C_M_AXI_ID_WIDTH-1 : 0] M_AXI_RID,
        input wire [C_M_AXI_DATA_WIDTH-1 : 0] M_AXI_RDATA,
        input wire [1 : 0] M_AXI_RRESP,
        input wire    M_AXI_RLAST,
        input wire [C_M_AXI_RUSER_WIDTH-1 : 0] M_AXI_RUSER,
        input wire    M_AXI_RVALID,
        output wire    M_AXI_RREADY
    );

      function integer clogb2 (input integer bit_depth);
      begin
        for(clogb2=0; bit_depth>0; clogb2=clogb2+1)
          bit_depth = bit_depth >> 1;
        end
      endfunction

     localparam integer C_TRANSACTIONS_NUM = clogb2(C_M_AXI_BURST_LEN-1);
     localparam integer C_MASTER_LENGTH   = 13;
     localparam integer C_NO_BURSTS_REQ
= C_MASTER_LENGTH-clogb2((C_M_AXI_BURST_LEN*C_M_AXI_DATA_WIDTH/8)-1);

     reg [1:0] mst_exec_state;

    reg [C_M_AXI_ADDR_WIDTH-1 : 0]     axi_araddr;
    reg         axi_arvalid;
    reg         axi_rready;
    reg [C_TRANSACTIONS_NUM : 0]       read_index;
    wire [C_TRANSACTIONS_NUM+3 : 0]  burst_size_bytes;
    reg [C_NO_BURSTS_REQ : 0]      read_burst_counter;
```

```
reg         start_single_burst_read;
reg         reads_done;
reg         burst_read_active;
wire        rnext;
reg         init_txn_ff;
reg         init_txn_ff2;
reg         init_txn_edge;
wire        init_txn_pulse;

reg rnext_buffer;

reg   signed [15:0] compare_difference_0;
reg   signed [15:0] compare_difference_1;
reg   signed [15:0] compare_difference_2;
reg   signed [15:0] compare_difference_3;
reg   signed [15:0] compare_difference_4;
reg   signed [15:0] compare_difference_5;
reg   signed [15:0] compare_difference_6;
reg   signed [15:0] compare_difference_7;

reg [15:0] memory[0:4095];
reg [15:0] store_data[0:4099];
reg [11:0] address;
reg [11:0] memory_p;
reg [11:0] address_w;
integer cnt,i;
integer fp;

integer fp_write;

wire [15:0]m0,m1,m2,m3,m4,m5,m6,m7;
wire [15:0]d0,d1,d2,d3,d4,d5,d6,d7;

assign m0=fixed_shift ? memory[8*memory_p][15:8] : memory[4*memory_p] ;
assign m1=fixed_shift ? memory[8*memory_p+1][15:8] : memory[4*memory_p+1];
assign m2=fixed_shift ? memory[8*memory_p+2][15:8] : memory[4*memory_p+2];
assign m3=fixed_shift ? memory[8*memory_p+3][15:8] : memory[4*memory_p+3];
assign m4=memory[8*memory_p+4][15:8];
assign m5=memory[8*memory_p+5][15:8];
assign m6=memory[8*memory_p+6][15:8];
assign m7=memory[8*memory_p+7][15:8];



assign d0=fixed_shift ? M_AXI_RDATA[7:0] : M_AXI_RDATA[15:0];
```

```
assign d1=fixed_shift ? M_AXI_RDATA[15:8] : M_AXI_RDATA[31:16];
assign d2=fixed_shift ? M_AXI_RDATA[23:16] : M_AXI_RDATA[47:32];
assign d3=fixed_shift ? M_AXI_RDATA[31:24] : M_AXI_RDATA[63:48];
assign d4= M_AXI_RDATA[39:32];
assign d5= M_AXI_RDATA[47:40];
assign d6= M_AXI_RDATA[55:48];
assign d7= M_AXI_RDATA[63:56];

initial
begin
 forever
  begin
  @(posedge init_txn_pulse)
   begin
    fp=$fopen("/home/hubbery/verify.txt","r");
    address=0;
    while(!$feof(fp))
    begin
    for(i=0; i<4096; i=i+1)
         begin
         cnt = $fscanf (fp, "%d",memory[address]);
         address=address+1;
         end
    end
    $fclose(fp);
   end
  end
end

initial
 begin
 forever
 begin
 @(posedge init_txn_pulse)   fp_write =   $fopen("/home/hubbery/output.txt","w");
 @(posedge TXN_DONE)            $fclose(fp_write);
 end
 end


always @(posedge M_AXI_ACLK)
  begin
    if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1)
      begin
        compare_difference_0=0;
```

```
        compare_difference_1=0;
        compare_difference_2=0;
        compare_difference_3=0;
        compare_difference_4=0;
        compare_difference_5=0;
        compare_difference_6=0;
        compare_difference_7=0;
        memory_p=0;
      end
    else if(rnext)
      begin
        if(fixed_shift)
        begin
        compare_difference_0=m0-d0;
        compare_difference_1=m1-d1;
        compare_difference_2=m2-d2;
        compare_difference_3=m3-d3;
        compare_difference_4=m4-d4;
        compare_difference_5=m5-d5;
        compare_difference_6=m6-d6;
        compare_difference_7=m7-d7;

        $fwrite(fp_write, "%d ", compare_difference_0);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_1);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_2);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_3);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_4);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_5);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_6);
        $fwrite(fp_write, "\n");
        $fwrite(fp_write, "%d ", compare_difference_7);
        $fwrite(fp_write, "\n");
        memory_p=memory_p+1;
        end
        else
        begin
        compare_difference_0=m0-d0;
        compare_difference_1=m1-d1;
```

```
            compare_difference_2=m2-d2;
            compare_difference_3=m3-d3;

            $fwrite(fp_write, "%d ", compare_difference_0);
            $fwrite(fp_write, "\n");
            $fwrite(fp_write, "%d ", compare_difference_1);
            $fwrite(fp_write, "\n");
            $fwrite(fp_write, "%d ", compare_difference_2);
            $fwrite(fp_write, "\n");
            $fwrite(fp_write, "%d ", compare_difference_3);
            $fwrite(fp_write, "\n");
            memory_p=memory_p+1;
            end
        end
    end


assign M_AXI_AWID      = 'b0;
assign M_AXI_AWADDR = 0;
assign M_AXI_AWLEN    = C_M_AXI_BURST_LEN - 1;
assign M_AXI_AWSIZE   = clogb2((C_M_AXI_DATA_WIDTH/8)-1);
assign M_AXI_AWBURST      = 2'b01;
assign M_AXI_AWLOCK = 1'b0;
assign M_AXI_AWCACHE      = 4'b0010;
assign M_AXI_AWPROT = 3'h0;
assign M_AXI_AWQOS   = 4'h0;
assign M_AXI_AWUSER = 'b1;
assign M_AXI_AWVALID= 0;
assign M_AXI_WDATA   = 0;
assign M_AXI_WSTRB   = {(C_M_AXI_DATA_WIDTH/8){1'b1}};
assign M_AXI_WLAST   = 0;
assign M_AXI_WUSER   = 'b0;
assign M_AXI_WVALID  = 0;
assign M_AXI_BREADY  = 0;
assign M_AXI_ARID      = 'b0;
assign M_AXI_ARADDR = C_M_TARGET_SLAVE_BASE_ADDR + axi_araddr;
assign M_AXI_ARLEN    = C_M_AXI_BURST_LEN - 1;
assign M_AXI_ARSIZE   = clogb2((C_M_AXI_DATA_WIDTH/8)-1);
assign M_AXI_ARBURST      = 2'b01;
assign M_AXI_ARLOCK  = 1'b0;
assign M_AXI_ARCACHE      = 4'b0010;
assign M_AXI_ARPROT  = 3'h0;
assign M_AXI_ARQOS   = 4'h0;
assign M_AXI_ARUSER  = 'b1;
```

```
assign M_AXI_ARVALID = axi_arvalid;
assign M_AXI_RREADY  = axi_rready;
assign burst_size_bytes       = C_M_AXI_BURST_LEN * C_M_AXI_DATA_WIDTH/8;
assign init_txn_pulse    = (!init_txn_ff2) && init_txn_ff;



always @(posedge M_AXI_ACLK)
  begin
    if (M_AXI_ARESETN == 0 )
      begin
        init_txn_ff <= 1'b0;
        init_txn_ff2 <= 1'b0;
      end
    else
      begin
        init_txn_ff <= INIT_AXI_TXN;
        init_txn_ff2 <= init_txn_ff;
      end
  end

  always @(posedge M_AXI_ACLK)
  begin

    if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1 )
      begin
        axi_arvalid <= 1'b0;
      end
    else if (M_AXI_ARREADY && axi_arvalid)
      begin
        axi_arvalid <= 1'b0;
      end
    else
      axi_arvalid <= axi_arvalid;
  end

  always @(posedge M_AXI_ACLK)
  begin

    if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1 )
      begin
        axi_arvalid <= 1'b0;
      end
    else if (~axi_arvalid && start_single_burst_read)
```

```
      begin
        axi_arvalid <= 1'b1;
      end
    else if (M_AXI_ARREADY && axi_arvalid)
      begin
        axi_arvalid <= 1'b0;
      end
    else
      axi_arvalid <= axi_arvalid;
end


always @(posedge M_AXI_ACLK)
begin
  if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1)
    begin
      axi_araddr <= 'b0;
    end
  else if (M_AXI_ARREADY && axi_arvalid)
    begin
      axi_araddr <= axi_araddr + burst_size_bytes;
    end
  else
    axi_araddr <= axi_araddr;
end

assign rnext = M_AXI_RVALID && axi_rready;

always @(posedge M_AXI_ACLK)
begin
  if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1 )
    begin
      axi_rready <= 1'b0;
    end
  else if (M_AXI_RVALID)
    begin
        if (M_AXI_RLAST && axi_rready)
         begin
           axi_rready <= 1'b0;
         end
        else
          begin
            axi_rready <= 1'b1;
          end
```

```
      end
  end



  always @(posedge M_AXI_ACLK)
  begin
    if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1)
      begin
        read_burst_counter <= 'b0;
      end
    else if (M_AXI_ARREADY && axi_arvalid)
      begin
        if (read_burst_counter[C_NO_BURSTS_REQ-fixed_shift] == 1'b0)
          begin
            read_burst_counter <= read_burst_counter + 1'b1;
          end
      end
    else
      read_burst_counter <= read_burst_counter;
  end

  always @(posedge M_AXI_ACLK)
  begin
    if (M_AXI_ARESETN == 0 ||   init_txn_pulse== 1'b1 || start_single_burst_read)
      begin
        read_index <= 0;
      end
    else if (rnext && (read_index != C_M_AXI_BURST_LEN-1))
      begin
        read_index <= read_index + 1;
      end
    else
      read_index <= read_index;
  end



  always @(posedge M_AXI_ACLK)
 begin
    if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1)
      burst_read_active <= 1'b0;
    else if (start_single_burst_read)
      burst_read_active <= 1'b1;
    else if (M_AXI_RVALID && axi_rready && M_AXI_RLAST)
```

```
        burst_read_active <= 0;
      end

    always @(posedge M_AXI_ACLK)
    begin
      if (M_AXI_ARESETN == 0 || init_txn_pulse == 1'b1)
        reads_done <= 1'b0;
      else if (M_AXI_RVALID && axi_rready && (read_index == C_M_AXI_BURST_LEN-1) &&
(read_burst_counter[C_NO_BURSTS_REQ-fixed_shift]))
        reads_done <= 1'b1;
      else
        reads_done <= reads_done;
      end

     always @ ( posedge M_AXI_ACLK)
     begin
       if (M_AXI_ARESETN == 1'b0 )
         begin
           mst_exec_state         <= 0;
           start_single_burst_read   <= 1'b0;
           TXN_DONE<=0;
         end
       else
         begin
           case (mst_exec_state)

             0:
               if ( init_txn_pulse == 1'b1)
                 begin
                   mst_exec_state   <= 1;
                   TXN_DONE<=0;
                 end
               else
                 begin
                   mst_exec_state   <= 0;
                 end


             1:
               if (reads_done)
                 begin
                   mst_exec_state <= 0;
                   TXN_DONE<=1;
                 end
```

```
                else
                  begin
                    mst_exec_state  <= 1;

                    if (~axi_arvalid && ~burst_read_active && ~start_single_burst_read)
                      begin
                        start_single_burst_read <= 1'b1;
                      end
                   else
                     begin
                       start_single_burst_read <= 1'b0;
                     end
                  end

              default :
                begin
                  mst_exec_state  <= 0;
                end
            endcase
          end
      end


    endmodule
```

三、开发板验证

SDK c 语言代码

1.写入 sd 卡

```
#include <string.h>
#include "stdlib.h"
#include <stdio.h>
#include <math.h>
#include"xil_printf.h"
#include "platform.h"
#include "xparameters.h"
#include "xil_io.h"
#include "ff.h"
#include "xdevcfg.h"
#include "xgpiops.h"
```

`

```
static FATFS fatfs;
#define DDR_BASEARDDR   0x01000000

#define FILE_0 "rand0.txt"
#define FILE_0_RESULT "rand0_w.txt"
#define FILE_1 "rand1.txt"
#define FILE_1_RESULT "rand1_w.txt"
#define FILE_5 "rand5.txt"
#define FILE_5_RESULT "rand5_w.txt"
#define FILE_10 "rand10.txt"
#define FILE_10_RESULT "rand10_w.txt"

void SD_Init()
{
    f_mount(&fatfs,"",0);
}

int SD_Transfer_read(char *FileName,u32 DestinationAddress,u32 ByteLength,int number)
{
    FIL fil;

    UINT br;

    f_open(&fil,FileName,FA_READ);

    f_lseek(&fil, (DWORD) number);

    f_read(&fil, (void*)DestinationAddress,ByteLength,&br);

    f_close(&fil);

    return XST_SUCCESS;
}


void SD_Transfer_write(char *FileName)
{
    FIL fil;

    int j;
    int fixed_result;
    char fixed_result_string[8]={};
    int number=0;
```

```
	UINT bw;
	char shift_line[2]={13,10};
	f_open(&fil,FileName,FA_CREATE_ALWAYS | FA_WRITE);

	for(j=0;j<4096;j++)
	{
	 fixed_result=Xil_In16(DDR_BASEARDDR+j*2);

	 sprintf(fixed_result_string,"%d",fixed_result);

	    f_lseek(&fil, (DWORD) number);
	    number=number+strlen(fixed_result_string);
	    f_write(&fil,fixed_result_string,strlen(fixed_result_string),&bw);

	    f_lseek(&fil, (DWORD) number);
	    number=number+2;
	    f_write(&fil,shift_line,2,&bw);
	}

	f_close(&fil);

}

void disablecache()
{
	Xil_DCacheDisable();
	Xil_ICacheDisable();
}


int main()
{

	static XGpioPs psGpioInstancePtr;
	XGpioPs_Config* GpioConfigPtr;
	int xStatus;
	GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
	if(GpioConfigPtr == NULL)
	    return XST_FAILURE;
	xStatus		=		XGpioPs_CfgInitialize(&psGpioInstancePtr,GpioConfigPtr,
GpioConfigPtr->BaseAddr);
	if(XST_SUCCESS != xStatus)
	    print(" PS GPIO INIT FAILED \n\r");
	XGpioPs_SetDirectionPin(&psGpioInstancePtr, 54,1);

`
```

```
XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, 54,1);
XGpioPs_SetDirectionPin(&psGpioInstancePtr, 55,0);
XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, 55,0);

disablecache();
SD_Init();



float ff;
int ff_fixed;
char dest_str[12];
int number=0,j,k;

printf("Start read %s.\n\n",FILE_0);

for(j=0,number=0;j<4096;j++)
{
	SD_Transfer_read(FILE_0,(u32)dest_str,12,number);
	for(k=0;k<12;k++)
	{
		if(((int)dest_str[k])==10)
			number+=k+1;
	}

	ff=atof(dest_str);

	ff_fixed=ff*pow(2,15-(124-128));

	if(ff_fixed<0)
		ff_fixed=-ff_fixed+pow(2,15);

	Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
}

Xil_Out16(DDR_BASEARDDR-8,(u16)0);
Xil_Out8(DDR_BASEARDDR-6,(u8)124);
Xil_Out8(DDR_BASEARDDR-5,(u8)0);
Xil_Out8(DDR_BASEARDDR-4,(u8)0);
Xil_Out8(DDR_BASEARDDR-3,(u8)1);
Xil_Out16(DDR_BASEARDDR-2,(u16)0);



XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);
```

```
while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

SD_Transfer_write(FILE_0_RESULT);

printf("Success! Complete rand0.1.\n\n");



 printf("Start read %s.\n\n",FILE_1);

 for(j=0,number=0;j<4096;j++)
 {
       SD_Transfer_read(FILE_1,(u32)dest_str,12,number);
       for(k=0;k<12;k++)
       {
             if(((int)dest_str[k])==10)
                   number+=k+1;
       }

       ff=atof(dest_str);

       ff_fixed=ff*pow(2,15-(129-128));

       if(ff_fixed<0)
             ff_fixed=-ff_fixed+pow(2,15);

       Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
 }

 Xil_Out16(DDR_BASEARDDR-8,(u16)0);
 Xil_Out8(DDR_BASEARDDR-6,(u8)129);
 Xil_Out8(DDR_BASEARDDR-5,(u8)0);
 Xil_Out8(DDR_BASEARDDR-4,(u8)0);
 Xil_Out8(DDR_BASEARDDR-3,(u8)1);
 Xil_Out16(DDR_BASEARDDR-2,(u16)0);


XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);

while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);
```

```
SD_Transfer_write(FILE_1_RESULT);

printf("Success! Complete rand1.\n\n");


 printf("Start read %s.\n\n",FILE_5);

 for(j=0,number=0;j<4096;j++)
 {
       SD_Transfer_read(FILE_5,(u32)dest_str,12,number);
       for(k=0;k<12;k++)
       {
             if(((int)dest_str[k])==10)
                   number+=k+1;
       }

       ff=atof(dest_str);

       ff_fixed=ff*pow(2,15-(131-128));

       if(ff_fixed<0)
             ff_fixed=-ff_fixed+pow(2,15);

       Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
 }

 Xil_Out16(DDR_BASEARDDR-8,(u16)0);
 Xil_Out8(DDR_BASEARDDR-6,(u8)131);
 Xil_Out8(DDR_BASEARDDR-5,(u8)0);
 Xil_Out8(DDR_BASEARDDR-4,(u8)0);
 Xil_Out8(DDR_BASEARDDR-3,(u8)1);
 Xil_Out16(DDR_BASEARDDR-2,(u16)0);


XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);

while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

SD_Transfer_write(FILE_5_RESULT);

printf("Success! Complete rand5.\n\n");
```

```
    printf("Start read %s.\n\n",FILE_10);

    for(j=0,number=0;j<4096;j++)
    {
        SD_Transfer_read(FILE_10,(u32)dest_str,12,number);
        for(k=0;k<12;k++)
        {
            if(((int)dest_str[k])==10)
                number+=k+1;
        }

        ff=atof(dest_str);

        ff_fixed=ff*pow(2,15-(132-128));

        if(ff_fixed<0)
            ff_fixed=-ff_fixed+pow(2,15);

        Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
    }

    Xil_Out16(DDR_BASEARDDR-8,(u16)0);
    Xil_Out8(DDR_BASEARDDR-6,(u8)132);
    Xil_Out8(DDR_BASEARDDR-5,(u8)0);
    Xil_Out8(DDR_BASEARDDR-4,(u8)0);
    Xil_Out8(DDR_BASEARDDR-3,(u8)1);
    Xil_Out16(DDR_BASEARDDR-2,(u16)0);


   XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);

   while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
   XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

   SD_Transfer_write(FILE_10_RESULT);

   printf("Success! Complete rand10.\n\n");

   return 0;
}
```

2.统计误差分布

`

```
#include <string.h>
#include "stdlib.h"
#include <stdio.h>
#include <math.h>
#include"xil_printf.h"
#include "platform.h"
#include "xparameters.h"
#include "xil_io.h"
#include "ff.h"
#include "xdevcfg.h"
#include "xgpiops.h"

static FATFS fatfs;
#define DDR_BASEARDDR   0x01000000

#define FILE_0 "rand0.txt"

#define FILE_1 "rand1.txt"

#define FILE_5 "rand5.txt"

#define FILE_10 "rand10.txt"


void SD_Init()
{
    f_mount(&fatfs,"",0);
}

int SD_Transfer_read(char *FileName,u32 DestinationAddress,u32 ByteLength,int number)
{
    FIL fil;

    UINT br;

    f_open(&fil,FileName,FA_READ);

    f_lseek(&fil, (DWORD) number);

    f_read(&fil, (void*)DestinationAddress,ByteLength,&br);

    f_close(&fil);


`
```

```
	return XST_SUCCESS;
}


void SD_write_result(char *FileName)
{
	FIL fil;

	int j;
	int fixed_result;
	char fixed_result_string[8]={};
	int number=0;
	UINT bw;
	char shift_line[2]={13,10};
	f_open(&fil,FileName,FA_CREATE_ALWAYS | FA_WRITE);

	for(j=0;j<4096;j++)
	{
	 fixed_result=Xil_In16(DDR_BASEARDDR+j*2);

	 sprintf(fixed_result_string,"%d",fixed_result);

		f_lseek(&fil, (DWORD) number);
		number=number+strlen(fixed_result_string);
		f_write(&fil,fixed_result_string,strlen(fixed_result_string),&bw);

		f_lseek(&fil, (DWORD) number);
		number=number+2;
		f_write(&fil,shift_line,2,&bw);
	}

	f_close(&fil);

}

void disablecache()
{
	 Xil_DCacheDisable();
	 Xil_ICacheDisable();
}



int main()
```

```
{

        static XGpioPs psGpioInstancePtr;
        XGpioPs_Config* GpioConfigPtr;
        int xStatus;
        GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
        xStatus         =         XGpioPs_CfgInitialize(&psGpioInstancePtr,GpioConfigPtr,
GpioConfigPtr->BaseAddr);
        if(XST_SUCCESS != xStatus)
              print(" PS GPIO INIT FAILED \n\r");
        XGpioPs_SetDirectionPin(&psGpioInstancePtr, 54,1);
        XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, 54,1);
        XGpioPs_SetDirectionPin(&psGpioInstancePtr, 55,0);
        XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, 55,0);

        disablecache();
       SD_Init();

        float ff;
        int ff_fixed;
        char dest_str[12];
        int number=0,j,k;
        float max_tmp=0.0;
        float store_exp=0;
        float sum=0;
        float fixed_result;
        float c_model_fixed;
        char *file_name;
        u8 fixed_q;
        int counter_0=0,counter_1=0,counter_2=0,counter_more=0;


        file_name=FILE_0;
        fixed_q=124;
    /*  Start read   FILE_0 */
        printf("Start read %s.\n\n",file_name);
        counter_0=0;
        counter_1=0;
        counter_2=0;
        counter_more=0;
        sum=0;
        for(j=0,number=0;j<4096;j++)
        {
              SD_Transfer_read(file_name,(u32)dest_str,12,number);
```

```
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            max_tmp=(ff>max_tmp) ? ff : max_tmp ;
            ff_fixed=ff*pow(2,15-(fixed_q-128));
            if(ff_fixed<0)
                ff_fixed=-ff_fixed+pow(2,15);
//          printf("start read sd file:%d\n",j);
            Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
        }
        printf("start exp,and max:%f\n",max_tmp);
        Xil_Out16(DDR_BASEARDDR-8,(u16)0);
        Xil_Out8(DDR_BASEARDDR-6,(u8)fixed_q);
        Xil_Out8(DDR_BASEARDDR-5,(u8)0);
        Xil_Out8(DDR_BASEARDDR-4,(u8)0);
        Xil_Out8(DDR_BASEARDDR-3,(u8)1);
        Xil_Out16(DDR_BASEARDDR-2,(u16)0);


       XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);
       while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
       XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

        for(j=0,number=0;j<4096;j++)
            {
                SD_Transfer_read(file_name,(u32)dest_str,12,number);
                for(k=0;k<12;k++)
                {
                    if(((int)dest_str[k])==10)
                        number+=k+1;
                }

                ff=atof(dest_str);

                store_exp=expf(ff-max_tmp);
                sum=sum+store_exp;
            }
```

```
        for(j=0,number=0;j<4096;j++)
        {

            SD_Transfer_read(file_name,(u32)dest_str,12,number);
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            fixed_result = Xil_In16(DDR_BASEARDDR+j*2);
            ff=(expf(ff-max_tmp))/sum;
            c_model_fixed=ff*pow(2,16);
            fixed_result=fabs(fixed_result-c_model_fixed);
    //      printf("show sub error(fixed):%f\n",fixed_result);
            if(fixed_result<1)
                counter_0++;
            else if(fixed_result<2)
                counter_1++;
            else if(fixed_result<3)
                counter_2++;
            else
                counter_more++;
        }

        printf("Success! Complete %s.\n",file_name);

        printf("show the error counter:\n");
                            printf("0 to 1:%d\n1 to 2:%d\n2 to
3:%d\nmore :%d\n",counter_0,counter_1,counter_2,counter_more);
          printf("%s error total:%d\n",file_name,counter_0 + counter_1 + counter_2 +
counter_more);


        file_name=FILE_1;
        fixed_q=129;
    /*  Start read   FILE_1 */
        printf("Start read %s.\n\n",file_name);
        counter_0=0;
        counter_1=0;
```

```
        counter_2=0;
        counter_more=0;
        sum=0;
        for(j=0,number=0;j<4096;j++)
        {
            SD_Transfer_read(file_name,(u32)dest_str,12,number);
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            max_tmp=(ff>max_tmp) ? ff : max_tmp ;
            ff_fixed=ff*pow(2,15-(fixed_q-128));
            if(ff_fixed<0)
                ff_fixed=-ff_fixed+pow(2,15);
//          printf("start read sd file:%d\n",j);
            Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
        }
        printf("start exp,and max:%f\n",max_tmp);
        Xil_Out16(DDR_BASEARDDR-8,(u16)0);
        Xil_Out8(DDR_BASEARDDR-6,(u8)fixed_q);
        Xil_Out8(DDR_BASEARDDR-5,(u8)0);
        Xil_Out8(DDR_BASEARDDR-4,(u8)0);
        Xil_Out8(DDR_BASEARDDR-3,(u8)1);
        Xil_Out16(DDR_BASEARDDR-2,(u16)0);


       XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);
       while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
       XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

        for(j=0,number=0;j<4096;j++)
            {
                SD_Transfer_read(file_name,(u32)dest_str,12,number);
                for(k=0;k<12;k++)
                {
                    if(((int)dest_str[k])==10)
                        number+=k+1;
                }

                ff=atof(dest_str);
```

```
                store_exp=expf(ff-max_tmp);
                sum=sum+store_exp;
            }




        for(j=0,number=0;j<4096;j++)
        {

            SD_Transfer_read(file_name,(u32)dest_str,12,number);
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            fixed_result = Xil_In16(DDR_BASEARDDR+j*2);
            ff=(expf(ff-max_tmp))/sum;
            c_model_fixed=ff*pow(2,16);
            fixed_result=fabs(fixed_result-c_model_fixed);
//          printf("show sub error(fixed):%f\n",fixed_result);
            if(fixed_result<1)
                counter_0++;
            else if(fixed_result<2)
                counter_1++;
            else if(fixed_result<3)
                counter_2++;
            else
                counter_more++;
        }

        printf("Success! Complete %s.\n",file_name);

        printf("show the error counter:\n");
                            printf("0 to 1:%d\n1 to 2:%d\n2 to
3:%d\nmore :%d\n",counter_0,counter_1,counter_2,counter_more);
           printf("%s error total:%d\n",file_name,counter_0 + counter_1 + counter_2 +
counter_more);
```

```
        file_name=FILE_5;
        fixed_q=131;
/*      Start read   FILE_5 */
        printf("Start read %s.\n\n",file_name);
        counter_0=0;
        counter_1=0;
        counter_2=0;
        counter_more=0;
        sum=0;
        for(j=0,number=0;j<4096;j++)
        {
            SD_Transfer_read(file_name,(u32)dest_str,12,number);
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            max_tmp=(ff>max_tmp) ? ff : max_tmp ;
            ff_fixed=ff*pow(2,15-(fixed_q-128));
            if(ff_fixed<0)
                ff_fixed=-ff_fixed+pow(2,15);
//          printf("start read sd file:%d\n",j);
            Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
        }
        printf("start exp,and max:%f\n",max_tmp);
        Xil_Out16(DDR_BASEARDDR-8,(u16)0);
        Xil_Out8(DDR_BASEARDDR-6,(u8)fixed_q);
        Xil_Out8(DDR_BASEARDDR-5,(u8)0);
        Xil_Out8(DDR_BASEARDDR-4,(u8)0);
        Xil_Out8(DDR_BASEARDDR-3,(u8)1);
        Xil_Out16(DDR_BASEARDDR-2,(u16)0);


       XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);
       while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
       XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

        for(j=0,number=0;j<4096;j++)
```

```
        {
            SD_Transfer_read(file_name,(u32)dest_str,12,number);
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            store_exp=expf(ff-max_tmp);
            sum=sum+store_exp;
        }




    for(j=0,number=0;j<4096;j++)
    {

        SD_Transfer_read(file_name,(u32)dest_str,12,number);
        for(k=0;k<12;k++)
        {
            if(((int)dest_str[k])==10)
                number+=k+1;
        }

        ff=atof(dest_str);

        fixed_result = Xil_In16(DDR_BASEARDDR+j*2);
        ff=(expf(ff-max_tmp))/sum;
        c_model_fixed=ff*pow(2,16);
        fixed_result=fabs(fixed_result-c_model_fixed);
//      printf("show sub error(fixed):%f\n",fixed_result);
        if(fixed_result<1)
            counter_0++;
        else if(fixed_result<2)
            counter_1++;
        else if(fixed_result<3)
            counter_2++;
        else
            counter_more++;
    }
```

```
        printf("Success! Complete %s.\n",file_name);

        printf("show the error counter:\n");
                                printf("0 to 1:%d\n1 to 2:%d\n2 to
3:%d\nmore :%d\n",counter_0,counter_1,counter_2,counter_more);
         printf("%s error total:%d\n\n\n",file_name,counter_0 + counter_1 + counter_2 +
counter_more);


        file_name=FILE_10;
        fixed_q=132;
    /*  Start read   FILE_10 */
        printf("Start read %s.\n\n",file_name);
        counter_0=0;
        counter_1=0;
        counter_2=0;
        counter_more=0;
        sum=0;
        for(j=0,number=0;j<4096;j++)
        {
            SD_Transfer_read(file_name,(u32)dest_str,12,number);
            for(k=0;k<12;k++)
            {
                if(((int)dest_str[k])==10)
                    number+=k+1;
            }

            ff=atof(dest_str);

            max_tmp=(ff>max_tmp) ? ff : max_tmp ;
            ff_fixed=ff*pow(2,15-(fixed_q-128));
            if(ff_fixed<0)
                ff_fixed=-ff_fixed+pow(2,15);
    //          printf("start read sd file:%d\n",j);
            Xil_Out16(DDR_BASEARDDR+j*2,(u16)ff_fixed);
        }
        printf("start exp,and max:%f\n",max_tmp);
        Xil_Out16(DDR_BASEARDDR-8,(u16)0);
        Xil_Out8(DDR_BASEARDDR-6,(u8)fixed_q);
        Xil_Out8(DDR_BASEARDDR-5,(u8)0);
        Xil_Out8(DDR_BASEARDDR-4,(u8)0);
        Xil_Out8(DDR_BASEARDDR-3,(u8)1);
        Xil_Out16(DDR_BASEARDDR-2,(u16)0);
```

```
XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);
while(XGpioPs_ReadPin(&psGpioInstancePtr,55)==0) {;}
XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);

 for(j=0,number=0;j<4096;j++)
      {
           SD_Transfer_read(file_name,(u32)dest_str,12,number);
           for(k=0;k<12;k++)
           {
                if(((int)dest_str[k])==10)
                     number+=k+1;
           }

           ff=atof(dest_str);

           store_exp=expf(ff-max_tmp);
           sum=sum+store_exp;
      }




 for(j=0,number=0;j<4096;j++)
 {

      SD_Transfer_read(file_name,(u32)dest_str,12,number);
      for(k=0;k<12;k++)
      {
           if(((int)dest_str[k])==10)
                number+=k+1;
      }

      ff=atof(dest_str);

      fixed_result = Xil_In16(DDR_BASEARDDR+j*2);
      ff=(expf(ff-max_tmp))/sum;
      c_model_fixed=ff*pow(2,16);
      fixed_result=fabs(fixed_result-c_model_fixed);
//    printf("show sub error(fixed):%f\n",fixed_result);
      if(fixed_result<1)
           counter_0++;
      else if(fixed_result<2)
           counter_1++;
```

```
            else if(fixed_result<3)
                counter_2++;
            else
                counter_more++;
        }

        printf("Success! Complete %s.\n",file_name);

        printf("show the error counter:\n");
        printf("0 to 1:%d\n1 to 2:%d\n2 to 3:%d\nmore :%d\n",counter_0,counter_1,counter_2,counter_more);
        printf("%s error total:%d\n",file_name,counter_0 + counter_1 + counter_2 + counter_more);

    return 0;
}
```

四、matlab

1.不等间隔划分拟合区域

```
x=0:0.0001:8;
y=exp(-x);
plot(x,y,' ');
hold on;

x1=0.25:0.25:2;
y1=exp(-x1);
stem(x1,y1,' ');

x2=2.5:0.5:4;
y2=exp(-x2);
stem(x2,y2,' ');

x3=5:1:6;
y3=exp(-x3);
stem(x3,y3,' ');

x4=8;
y4=exp(-x4);
stem(x4,y4,' ');

hold off;
```

`

2.浮点转为定点

```
datain=importdata("rand1.txt");

for i=1:1:4096
        datain(i)=round(datain(i)*power(2,14));
end

for i=1:1:4096
    if datain(i) <0
        datain(i)=-datain(i)+power(2,15);
    end
end
dlmwrite('fixed1.txt',datain,'\n');
```

3.拟合二阶多项式系数

```
x_1=0:1:1024
x_2=1024:1:2048
x_3=2048:1:3072
x_4=3072:1:4096
x_5=4096:1:5120
x_6=5120:1:6144
x_7=6144:1:7168
x_8=7168:1:8192
x_9=8192:1:10240
x_10=10240:1:12288
x_11=12288:1:14336
x_12=14336:1:16384
x_13=16384:1:20480
x_14=20480:1:24576
x_15=24576:1:32768

x1=x_1/4096
x2=x_2/4096
x3=x_3/4096
x4=x_4/4096
x5=x_5/4096
x6=x_6/4096
x7=x_7/4096
x8=x_8/4096
x9=x_9/4096
x10=x_10/4096
```

`

```
x11=x_11/4096
x12=x_12/4096
x13=x_13/4096
x14=x_14/4096
x15=x_15/4096

y1=4096*exp(-x1)
y2=4096*exp(-x2)
y3=4096*exp(-x3)
y4=4096*exp(-x4)
y5=4096*exp(-x5)
y6=4096*exp(-x6)
y7=4096*exp(-x7)
y8=4096*exp(-x8)
y9=4096*exp(-x9)
y10=4096*exp(-x10)
y11=4096*exp(-x11)
y12=4096*exp(-x12)
y13=4096*exp(-x13)
y14=4096*exp(-x14)
y15=4096*exp(-x15)

p1=polyfit(x1,y1,2)
p2=polyfit(x2,y2,2)
p3=polyfit(x3,y3,2)
p4=polyfit(x4,y4,2)
p5=polyfit(x5,y5,2)
p6=polyfit(x6,y6,2)
p7=polyfit(x7,y7,2)
p8=polyfit(x8,y8,2)
p9=polyfit(x9,y9,2)
p10=polyfit(x10,y10,2)
p11=polyfit(x11,y11,2)
p12=polyfit(x12,y12,2)
p13=polyfit(x13,y13,2)
p14=polyfit(x14,y14,2)
p15=polyfit(x15,y15,2)

f1=polyval(p1,x1)
f2=polyval(p2,x2)
f3=polyval(p3,x3)
f4=polyval(p4,x4)
f5=polyval(p5,x5)
f6=polyval(p6,x6)
```

```
f7=polyval(p7,x7)
f8=polyval(p8,x8)
f9=polyval(p9,x9)
f10=polyval(p10,x10)
f11=polyval(p11,x11)
f12=polyval(p12,x12)
f13=polyval(p13,x13)
f14=polyval(p14,x14)
f15=polyval(p15,x15)


fw1=f1-y1
fw2=f2-y2
fw3=f3-y3
fw4=f4-y4
fw5=f5-y5
fw6=f6-y6
fw7=f7-y7
fw8=f8-y8
fw9=f9-y9
fw10=f10-y10
fw11=f11-y11
fw12=f12-y12
fw13=f13-y13
fw14=f14-y14
fw15=f15-y15

fa1=abs(fw1)./y1
fa2=abs(fw2)./y2
fa3=abs(fw3)./y3
fa4=abs(fw4)./y3
fa5=abs(fw5)./y5
fa6=abs(fw6)./y6
fa7=abs(fw7)./y7
fa8=abs(fw8)./y8
fa9=abs(fw9)./y9
fa10=abs(fw10)./y10
fa11=abs(fw11)./y11
fa12=abs(fw12)./y12
fa13=abs(fw13)./y13
fa14=abs(fw14)./y14
fa15=abs(fw15)./y15

figure(1)

`
```

```
plot(x1,fa1);
hold on
plot(x2,fa2);
plot(x3,fa3);
plot(x4,fa4);
plot(x5,fa5);
plot(x6,fa6);
plot(x7,fa7);
plot(x8,fa8);
plot(x9,fa9);
plot(x10,fa10);
plot(x11,fa11);
plot(x12,fa12);
plot(x13,fa13);
plot(x14,fa14);
plot(x15,fa15);
hold off

figure(2)
plot(x1,fw1);
hold on
plot(x2,fw2);
plot(x3,fw3);
plot(x4,fw4);
plot(x5,fw5);
plot(x6,fw6);
plot(x7,fw7);
plot(x8,fw8);
plot(x9,fw9);
plot(x10,fw10);
plot(x11,fw11);
plot(x12,fw12);
plot(x13,fw13);
plot(x14,fw14);
plot(x15,fw15);
hold off

p1
p2
p3
p4
p5
p6
p7
```

p8
p9
p10
p11
p12
p13
p14
p15

4.提供 softmax 标准结果

```
x=1:1:4096;

data_in_rand_0_1= importdata("rand0.1.txt");
data_in_rand_1= importdata("rand1.txt");
data_in_rand_5= importdata("rand5.txt");
data_in_rand_10= importdata("rand10.txt");

tmp=zeros(1,4096);
f_out=zeros(1,4096);

sum=0;
max=0;
for i=1:1:4096
      if data_in_rand_0_1(i) >max
            max=data_in_rand_0_1(i);
      end
end
for i=1:1:4096
    tmp(i)=exp( data_in_rand_0_1(i)-max );
    sum=sum+tmp(i);
end
for i=1:1:4096
   f_out(i)=tmp(i)/sum*power(2,16);
end
dlmwrite('rand_verify_0.1_f.txt',f_out,'\n');
```

`

```
sum=0;
max=0;
for i=1:1:4096
      if data_in_rand_1(i) >max
            max=data_in_rand_1(i);
      end
end
for i=1:1:4096
    tmp(i)=exp( data_in_rand_1(i)-max );
    sum=sum+tmp(i);
end
for i=1:1:4096
   f_out(i)=tmp(i)/sum*power(2,16);
end
dlmwrite('rand_verify_1_f.txt',f_out,'\n');

sum=0;
max=0;
for i=1:1:4096
      if data_in_rand_5(i) >max
            max=data_in_rand_5(i);
      end
end
for i=1:1:4096
    tmp(i)=exp( data_in_rand_5(i)-max );
    sum=sum+tmp(i);
end
for i=1:1:4096
   f_out(i)=tmp(i)/sum*power(2,16);
end
dlmwrite('rand_verify_5_f.txt',f_out,'\n');

sum=0;
max=0;
for i=1:1:4096
      if data_in_rand_10(i) >max
            max=data_in_rand_10(i);
      end
end
for i=1:1:4096
    tmp(i)=exp( data_in_rand_10(i)-max );
    sum=sum+tmp(i);
end
```

```
for i=1:1:4096
   f_out(i)=tmp(i)/sum*power(2,16);
end
dlmwrite('rand_verify_10_f.txt',f_out,'\n');
```

5.将误差以离散点的方式表示

```
x=1:1:4096;
f1= importdata("RAND0_W.TXT");
f2= importdata("RAND1_W.TXT");
f3= importdata("RAND5_W.TXT");
f4= importdata("RAND10_W.TXT");
v1=importdata("rand_verify_0.1_f.txt");
v2=importdata("rand_verify_1_f.txt");
v3=importdata("rand_verify_5_f.txt");
v4=importdata("rand_verify_10_f.txt");
c1=abs(f1-v1);
c2=abs(f2-v2);
c3=abs(f3-v3);
c4=abs(f4-v4);
figure(1);
plot(x, c1 ,'*');
figure(2);
plot(x, c2 ,'*');
figure(3);
plot(x, c3 ,'*');
figure(4);
plot(x, c4 ,'*');
```