//顶层文件

module zstar(

inout [14:0]DDR\_addr,

inout [2:0]DDR\_ba,

inout DDR\_cas\_n,

inout DDR\_ck\_n,

inout DDR\_ck\_p,

inout DDR\_cke,

inout DDR\_cs\_n,

inout [3:0]DDR\_dm,

inout [31:0]DDR\_dq,

inout [3:0]DDR\_dqs\_n,

inout [3:0]DDR\_dqs\_p,

inout DDR\_odt,

inout DDR\_ras\_n,

inout DDR\_reset\_n,

inout DDR\_we\_n,

//ImageSensor图像接口

output image\_sensor\_xclk, //输出时钟

input image\_sensor\_pclk, //视频时钟

input image\_sensor\_vsync, //视频场同步信号，高电平有效（有效视频传输时该信号拉低）

input image\_sensor\_href, //视频行同步信号

input[7:0] image\_sensor\_data, //视频数据总线

output image\_sensor\_scl, //串行配置IIC时钟信号

inout image\_sensor\_sda, //串行配置IIC数据信号

output image\_sensor\_reset\_n, //复位接口，低电平有效

output image\_sensor\_pwdn, //低功耗使能信号，高电平有效

//ImageSensor2图像接口

output image2\_sensor\_xclk, //输出时钟

input image2\_sensor\_pclk, //视频时钟

input image2\_sensor\_vsync, //视频场同步信号，高电平有效（有效视频传输时该信号拉低）

input image2\_sensor\_href, //视频行同步信号

input[7:0] image2\_sensor\_data, //视频数据总线

output image2\_sensor\_scl, //串行配置IIC时钟信号

inout image2\_sensor\_sda, //串行配置IIC数据信号

output image2\_sensor\_reset\_n, //复位接口，低电平有效

output image2\_sensor\_pwdn, //低功耗使能信号，高电平有效

//VGA驱动接口

output[4:0] vga\_r,

output[5:0] vga\_g,

output[4:0] vga\_b,

output[2:0] vga\_rgb,

output vga\_hsy,vga\_vsy,

output vga\_clk,

output adv7123\_blank\_n,

output adv7123\_sync\_n,

//LED指示灯接口

output[0:0] led //用于测试的LED指示灯

);

wire clk\_25m;

wire clk\_50m;

wire clk\_100m;

wire clk\_75m;

wire sys\_rst\_n;

wire AXI\_HP0\_ACLK;

wire [31:0]AXI\_HP0\_araddr;

wire [1:0]AXI\_HP0\_arburst;

wire [3:0]AXI\_HP0\_arcache;

wire [5:0]AXI\_HP0\_arid;

wire [3:0]AXI\_HP0\_arlen;

wire [1:0]AXI\_HP0\_arlock;

wire [2:0]AXI\_HP0\_arprot;

wire [3:0]AXI\_HP0\_arqos;

wire AXI\_HP0\_arready;

wire [2:0]AXI\_HP0\_arsize;

wire AXI\_HP0\_arvalid;

wire [31:0]AXI\_HP0\_awaddr;

wire [1:0]AXI\_HP0\_awburst;

wire [3:0]AXI\_HP0\_awcache;

wire [5:0]AXI\_HP0\_awid;

wire [3:0]AXI\_HP0\_awlen;

wire [1:0]AXI\_HP0\_awlock;

wire [2:0]AXI\_HP0\_awprot;

wire [3:0]AXI\_HP0\_awqos;

wire AXI\_HP0\_awready;

wire [2:0]AXI\_HP0\_awsize;

wire AXI\_HP0\_awvalid;

wire [5:0]AXI\_HP0\_bid;

wire AXI\_HP0\_bready;

wire [1:0]AXI\_HP0\_bresp;

wire AXI\_HP0\_bvalid;

wire [63:0]AXI\_HP0\_rdata;

wire [5:0]AXI\_HP0\_rid;

wire AXI\_HP0\_rlast;

wire AXI\_HP0\_rready;

wire [1:0]AXI\_HP0\_rresp;

wire AXI\_HP0\_rvalid;

wire [63:0]AXI\_HP0\_wdata;

wire [5:0]AXI\_HP0\_wid;

wire AXI\_HP0\_wlast;

wire AXI\_HP0\_wready;

wire [7:0]AXI\_HP0\_wstrb;

wire AXI\_HP0\_wvalid;

wire AXI\_HP1\_ACLK;

wire [31:0]AXI\_HP1\_araddr;

wire [1:0]AXI\_HP1\_arburst;

wire [3:0]AXI\_HP1\_arcache;

wire [5:0]AXI\_HP1\_arid;

wire [3:0]AXI\_HP1\_arlen;

wire [1:0]AXI\_HP1\_arlock;

wire [2:0]AXI\_HP1\_arprot;

wire [3:0]AXI\_HP1\_arqos;

wire AXI\_HP1\_arready;

wire [2:0]AXI\_HP1\_arsize;

wire AXI\_HP1\_arvalid;

wire [31:0]AXI\_HP1\_awaddr;

wire [1:0]AXI\_HP1\_awburst;

wire [3:0]AXI\_HP1\_awcache;

wire [5:0]AXI\_HP1\_awid;

wire [3:0]AXI\_HP1\_awlen;

wire [1:0]AXI\_HP1\_awlock;

wire [2:0]AXI\_HP1\_awprot;

wire [3:0]AXI\_HP1\_awqos;

wire AXI\_HP1\_awready;

wire [2:0]AXI\_HP1\_awsize;

wire AXI\_HP1\_awvalid;

wire [5:0]AXI\_HP1\_bid;

wire AXI\_HP1\_bready;

wire [1:0]AXI\_HP1\_bresp;

wire AXI\_HP1\_bvalid;

wire [63:0]AXI\_HP1\_rdata;

wire [5:0]AXI\_HP1\_rid;

wire AXI\_HP1\_rlast;

wire AXI\_HP1\_rready;

wire [1:0]AXI\_HP1\_rresp;

wire AXI\_HP1\_rvalid;

wire [63:0]AXI\_HP1\_wdata;

wire [5:0]AXI\_HP1\_wid;

wire AXI\_HP1\_wlast;

wire AXI\_HP1\_wready;

wire [7:0]AXI\_HP1\_wstrb;

wire AXI\_HP1\_wvalid;

assign AXI\_HP0\_awid = 6'b00\_0001;

assign AXI\_HP0\_wid = 6'b00\_0001;

assign AXI\_HP0\_arid = 6'b00\_0001;

assign AXI\_HP1\_awid = 6'b00\_0010;

assign AXI\_HP1\_wid = 6'b00\_0010;

assign AXI\_HP1\_arid = 6'b00\_0010;

assign AXI\_HP0\_ACLK = clk\_100m;

assign AXI\_HP1\_ACLK = clk\_100m;

assign image\_sensor\_xclk = clk\_25m;

assign image2\_sensor\_xclk = clk\_25m;

//////////////////////////////////////////////////////////////////////////////////

//PS系统

zstar\_zynq\_ps u1\_zstar\_zynq\_ps\_i(

.AXI\_HP0\_ACLK(AXI\_HP0\_ACLK),

.AXI\_HP0\_araddr(AXI\_HP0\_araddr),

.AXI\_HP0\_arburst(AXI\_HP0\_arburst),

.AXI\_HP0\_arcache(AXI\_HP0\_arcache),

.AXI\_HP0\_arid(AXI\_HP0\_arid),

.AXI\_HP0\_arlen(AXI\_HP0\_arlen),

.AXI\_HP0\_arlock(AXI\_HP0\_arlock),

.AXI\_HP0\_arprot(AXI\_HP0\_arprot),

.AXI\_HP0\_arqos(AXI\_HP0\_arqos),

.AXI\_HP0\_arready(AXI\_HP0\_arready),

.AXI\_HP0\_arsize(AXI\_HP0\_arsize),

.AXI\_HP0\_arvalid(AXI\_HP0\_arvalid),

.AXI\_HP0\_awaddr(AXI\_HP0\_awaddr),

.AXI\_HP0\_awburst(AXI\_HP0\_awburst),

.AXI\_HP0\_awcache(AXI\_HP0\_awcache),

.AXI\_HP0\_awid(AXI\_HP0\_awid),

.AXI\_HP0\_awlen(AXI\_HP0\_awlen),

.AXI\_HP0\_awlock(AXI\_HP0\_awlock),

.AXI\_HP0\_awprot(AXI\_HP0\_awprot),

.AXI\_HP0\_awqos(AXI\_HP0\_awqos),

.AXI\_HP0\_awready(AXI\_HP0\_awready),

.AXI\_HP0\_awsize(AXI\_HP0\_awsize),

.AXI\_HP0\_awvalid(AXI\_HP0\_awvalid),

.AXI\_HP0\_bid(AXI\_HP0\_bid),

.AXI\_HP0\_bready(AXI\_HP0\_bready),

.AXI\_HP0\_bresp(AXI\_HP0\_bresp),

.AXI\_HP0\_bvalid(AXI\_HP0\_bvalid),

.AXI\_HP0\_rdata(AXI\_HP0\_rdata),

.AXI\_HP0\_rid(AXI\_HP0\_rid),

.AXI\_HP0\_rlast(AXI\_HP0\_rlast),

.AXI\_HP0\_rready(AXI\_HP0\_rready),

.AXI\_HP0\_rresp(AXI\_HP0\_rresp),

.AXI\_HP0\_rvalid(AXI\_HP0\_rvalid),

.AXI\_HP0\_wdata(AXI\_HP0\_wdata),

.AXI\_HP0\_wid(AXI\_HP0\_wid),

.AXI\_HP0\_wlast(AXI\_HP0\_wlast),

.AXI\_HP0\_wready(AXI\_HP0\_wready),

.AXI\_HP0\_wstrb(AXI\_HP0\_wstrb),

.AXI\_HP0\_wvalid(AXI\_HP0\_wvalid),

.AXI\_HP1\_ACLK(AXI\_HP1\_ACLK),

.AXI\_HP1\_araddr(AXI\_HP1\_araddr),

.AXI\_HP1\_arburst(AXI\_HP1\_arburst),

.AXI\_HP1\_arcache(AXI\_HP1\_arcache),

.AXI\_HP1\_arid(AXI\_HP1\_arid),

.AXI\_HP1\_arlen(AXI\_HP1\_arlen),

.AXI\_HP1\_arlock(AXI\_HP1\_arlock),

.AXI\_HP1\_arprot(AXI\_HP1\_arprot),

.AXI\_HP1\_arqos(AXI\_HP1\_arqos),

.AXI\_HP1\_arready(AXI\_HP1\_arready),

.AXI\_HP1\_arsize(AXI\_HP1\_arsize),

.AXI\_HP1\_arvalid(AXI\_HP1\_arvalid),

.AXI\_HP1\_awaddr(AXI\_HP1\_awaddr),

.AXI\_HP1\_awburst(AXI\_HP1\_awburst),

.AXI\_HP1\_awcache(AXI\_HP1\_awcache),

.AXI\_HP1\_awid(AXI\_HP1\_awid),

.AXI\_HP1\_awlen(AXI\_HP1\_awlen),

.AXI\_HP1\_awlock(AXI\_HP1\_awlock),

.AXI\_HP1\_awprot(AXI\_HP1\_awprot),

.AXI\_HP1\_awqos(AXI\_HP1\_awqos),

.AXI\_HP1\_awready(AXI\_HP1\_awready),

.AXI\_HP1\_awsize(AXI\_HP1\_awsize),

.AXI\_HP1\_awvalid(AXI\_HP1\_awvalid),

.AXI\_HP1\_bid(AXI\_HP1\_bid),

.AXI\_HP1\_bready(AXI\_HP1\_bready),

.AXI\_HP1\_bresp(AXI\_HP1\_bresp),

.AXI\_HP1\_bvalid(AXI\_HP1\_bvalid),

.AXI\_HP1\_rdata(AXI\_HP1\_rdata),

.AXI\_HP1\_rid(AXI\_HP1\_rid),

.AXI\_HP1\_rlast(AXI\_HP1\_rlast),

.AXI\_HP1\_rready(AXI\_HP1\_rready),

.AXI\_HP1\_rresp(AXI\_HP1\_rresp),

.AXI\_HP1\_rvalid(AXI\_HP1\_rvalid),

.AXI\_HP1\_wdata(AXI\_HP1\_wdata),

.AXI\_HP1\_wid(AXI\_HP1\_wid),

.AXI\_HP1\_wlast(AXI\_HP1\_wlast),

.AXI\_HP1\_wready(AXI\_HP1\_wready),

.AXI\_HP1\_wstrb(AXI\_HP1\_wstrb),

.AXI\_HP1\_wvalid(AXI\_HP1\_wvalid),

.DDR\_addr(DDR\_addr),

.DDR\_ba(DDR\_ba),

.DDR\_cas\_n(DDR\_cas\_n),

.DDR\_ck\_n(DDR\_ck\_n),

.DDR\_ck\_p(DDR\_ck\_p),

.DDR\_cke(DDR\_cke),

.DDR\_cs\_n(DDR\_cs\_n),

.DDR\_dm(DDR\_dm),

.DDR\_dq(DDR\_dq),

.DDR\_dqs\_n(DDR\_dqs\_n),

.DDR\_dqs\_p(DDR\_dqs\_p),

.DDR\_odt(DDR\_odt),

.DDR\_ras\_n(DDR\_ras\_n),

.DDR\_reset\_n(DDR\_reset\_n),

.DDR\_we\_n(DDR\_we\_n),

.FCLK\_CLK\_25M(clk\_25m),

.FCLK\_CLK\_50M(clk\_50m),

.FCLK\_CLK\_100M(clk\_100m),

.FCLK\_CLK\_75M(clk\_75m),

.FCLK\_RESET\_N(sys\_rst\_n)

);

//////////////////////////////////////////////////////////////////////////////////

//ImageSensor图像采集控制模块

//ImageSensor数据写入DDR3接口

wire image\_ddr3\_wren;

wire image\_ddr3\_line\_end;

wire image\_ddr3\_clr;

wire[15:0] image\_ddr3\_wrdb;

wire image\_ddr3\_wready;

wire image\_ddr3\_frame\_start;

wire image\_ddr3\_frame\_end;

image\_controller u2\_image\_controller(

.clk(clk\_50m), //时钟信号

.rst\_n(sys\_rst\_n), //复位信号

//ImageSensor图像采集接口

.image\_sensor\_pclk(image\_sensor\_pclk),

.image\_sensor\_vsync(~image\_sensor\_vsync),

.image\_sensor\_href(image\_sensor\_href),

.image\_sensor\_data(image\_sensor\_data),

//ImageSensor串行配置接口

.image\_sensor\_scl(image\_sensor\_scl),

.image\_sensor\_sda(image\_sensor\_sda),

//ImageSensor复位与低功耗接口

.image\_sensor\_reset\_n(image\_sensor\_reset\_n),

.image\_sensor\_pwdn(image\_sensor\_pwdn),

//ImageSensor数据写入DDR3接口

.image\_ddr3\_wready(image\_ddr3\_wready),

.image\_ddr3\_wren(image\_ddr3\_wren),

.image\_ddr3\_wrdb(image\_ddr3\_wrdb),

.image\_ddr3\_line\_end(image\_ddr3\_line\_end),

.image\_ddr3\_frame\_start(image\_ddr3\_frame\_start),

.image\_ddr3\_frame\_end(image\_ddr3\_frame\_end),

.image\_ddr3\_clr(image\_ddr3\_clr)

);

//////////////////////////////////////////////////////////////////////////////////

//Bayer2RGB处理模块

wire w\_rgb\_image\_rst;

wire w\_rgb\_image\_vld;

wire[23:0] w\_rgb\_image\_data;

bayer2rgb u3\_bayer2rgb(

.clk(clk\_50m),

.rst\_n(sys\_rst\_n),

//input Image Data Flow

.i\_bayer\_image\_vld(image\_ddr3\_wren),

.o\_bayer\_image\_tready(image\_ddr3\_wready),

.i\_bayer\_image\_data(image\_ddr3\_wrdb[7:0]),

.i\_bayer\_image\_sof(image\_ddr3\_frame\_start),

.i\_bayer\_image\_eof(image\_ddr3\_frame\_end),

.i\_bayer\_image\_eol(image\_ddr3\_line\_end),

//output Image Data Flow

.o\_rgb\_image\_rst(w\_rgb\_image\_rst),

.o\_rgb\_image\_vld(w\_rgb\_image\_vld),

.o\_rgb\_image\_data(w\_rgb\_image\_data)

);

//////////////////////////////////////////////////////////////////////////////////

//AXI HP0主机--写

axi\_hp0\_wr #(

.STAR\_ADDR(32'h0100\_0000))

u4\_axi\_hp0\_wr(

// Outputs

.AXI\_awaddr (AXI\_HP0\_awaddr[31:0]),

.AXI\_awlen (AXI\_HP0\_awlen[3:0]),

.AXI\_awsize (AXI\_HP0\_awsize[2:0]),

.AXI\_awburst (AXI\_HP0\_awburst[1:0]),

.AXI\_awlock (AXI\_HP0\_awlock[1:0]),

.AXI\_awcache (AXI\_HP0\_awcache[3:0]),

.AXI\_awprot (AXI\_HP0\_awprot[2:0]),

.AXI\_awqos (AXI\_HP0\_awqos[3:0]),

.AXI\_awvalid (AXI\_HP0\_awvalid),

.AXI\_wdata (AXI\_HP0\_wdata[63:0]),

.AXI\_wstrb (AXI\_HP0\_wstrb[7:0]),

.AXI\_wlast (AXI\_HP0\_wlast),

.AXI\_wvalid (AXI\_HP0\_wvalid),

.AXI\_bready (AXI\_HP0\_bready),

// Inputs

.rst\_n (sys\_rst\_n),

.i\_clk (clk\_50m),

.i\_data\_rst\_n (~image\_ddr3\_clr),

.i\_data\_en (w\_rgb\_image\_vld),

.i\_data ({w\_rgb\_image\_data[23:19],w\_rgb\_image\_data[7:2],w\_rgb\_image\_data[15:11]}),

.AXI\_clk (AXI\_HP0\_ACLK),

.AXI\_awready (AXI\_HP0\_awready),

.AXI\_wready (AXI\_HP0\_wready),

.AXI\_bid (AXI\_HP0\_bid[5:0]),

.AXI\_bresp (AXI\_HP0\_bresp[1:0]),

.AXI\_bvalid (AXI\_HP0\_bvalid)

);

//////////////////////////////////////////////////////////////////////////////////

//ImageSensor图像增强处理

//ImageSensor数据写入DDR3接口

wire w\_gamma\_image\_vld;

wire[23:0] w\_gamma\_image\_data;

gamma\_correction u5\_gamma\_correction(

// Outputs

.o\_gamma\_image\_vld (w\_gamma\_image\_vld),

.o\_gamma\_image\_data (w\_gamma\_image\_data[23:0]),

// Inputs

.clk (clk\_50m),

.rst\_n (sys\_rst\_n),

//.i\_rgb\_image\_rst (w\_rgb\_image\_rst),

.i\_rgb\_image\_vld (w\_rgb\_image\_vld),

.i\_rgb\_image\_data (w\_rgb\_image\_data[23:0])

);

//Gamma图像laplace transform锐化处理

//Gamma数据写入DDR3接口

wire image2\_ddr3\_wren;

wire[15:0] image2\_ddr3\_wrdb;

laplace\_transform uut\_laplace\_transform(

// Outputs

.o\_image\_ddr3\_wren (image2\_ddr3\_wren),

.o\_image\_ddr3\_wrdb\_modify (image2\_ddr3\_wrdb),

// Inputs

.clk (clk\_50m),

.rst\_n (sys\_rst\_n),

.i\_image\_ddr3\_wren (w\_gamma\_image\_vld),

.i\_image\_ddr3\_line\_end (image\_ddr3\_line\_end),

.i\_image\_ddr3\_clr (image\_ddr3\_clr),

.i\_image\_ddr3\_wrdb ({w\_gamma\_image\_data[23:19],w\_gamma\_image\_data[7:2],w\_gamma\_image\_data[15:11]}));

//////////////////////////////////////////////////////////////////////////////////

//AXI HP1主机--写

axi\_hp0\_wr #(

.STAR\_ADDR(32'h0200\_0000))

u6\_axi\_hp1\_wr(

// Outputs

.AXI\_awaddr (AXI\_HP1\_awaddr[31:0]),

.AXI\_awlen (AXI\_HP1\_awlen[3:0]),

.AXI\_awsize (AXI\_HP1\_awsize[2:0]),

.AXI\_awburst (AXI\_HP1\_awburst[1:0]),

.AXI\_awlock (AXI\_HP1\_awlock[1:0]),

.AXI\_awcache (AXI\_HP1\_awcache[3:0]),

.AXI\_awprot (AXI\_HP1\_awprot[2:0]),

.AXI\_awqos (AXI\_HP1\_awqos[3:0]),

.AXI\_awvalid (AXI\_HP1\_awvalid),

.AXI\_wdata (AXI\_HP1\_wdata[63:0]),

.AXI\_wstrb (AXI\_HP1\_wstrb[7:0]),

.AXI\_wlast (AXI\_HP1\_wlast),

.AXI\_wvalid (AXI\_HP1\_wvalid),

.AXI\_bready (AXI\_HP1\_bready),

// Inputs

.rst\_n (sys\_rst\_n),

.i\_clk (clk\_50m),

.i\_data\_rst\_n (~image\_ddr3\_clr),

.i\_data\_en (image2\_ddr3\_wren),

.i\_data (image2\_ddr3\_wrdb[15:0]),

.AXI\_clk (AXI\_HP1\_ACLK),

.AXI\_awready (AXI\_HP1\_awready),

.AXI\_wready (AXI\_HP1\_wready),

.AXI\_bid (AXI\_HP1\_bid[5:0]),

.AXI\_bresp (AXI\_HP1\_bresp[1:0]),

.AXI\_bvalid (AXI\_HP1\_bvalid)

);

//////////////////////////////////////////////////////////////////////////////////

//液晶显示驱动模块

wire lcd\_synclk; //LCD驱动模块同步时钟

wire[15:0] lcd\_rfdb1; //输出到LCD模块待显示的DDR3读出数据

wire[15:0] lcd\_rfdb2; //输出到LCD模块待显示的DDR3读出数据

wire lcd\_rfreq1; //LCD模块发出的读FIFO请求信号，高电平有效

wire lcd\_rfreq2; //LCD模块发出的读FIFO请求信号，高电平有效

wire lcd\_rfclr; //LCD模块发出的读FIFO复位，低电平有效

lcd\_driver u7\_lcd\_driver(

.clk\_25m(clk\_25m),

.clk\_50m(clk\_50m),

.clk\_65m(),

.clk\_75m(clk\_75m),

.clk\_108m(),

.clk\_130m(),

.rst\_n(sys\_rst\_n),

.vga\_r(vga\_r),

.vga\_g(vga\_g),

.vga\_b(vga\_b),

.vga\_rgb(vga\_rgb),

.vga\_hsy(vga\_hsy),

.vga\_vsy(vga\_vsy),

.vga\_clk(vga\_clk),

.adv7123\_blank\_n(adv7123\_blank\_n),

.adv7123\_sync\_n(adv7123\_sync\_n),

.lcd\_synclk(lcd\_synclk),

.lcd\_rfdb1(lcd\_rfdb1),

.lcd\_rfdb2(lcd\_rfdb2),

.lcd\_rfreq1(lcd\_rfreq1),

.lcd\_rfreq2(lcd\_rfreq2),

.lcd\_rfclr(lcd\_rfclr)

);

//////////////////////////////////////////////////////////////////////////////////HP0 READ

//AXI HP0主机--读

axi\_hp0\_rd #(

.STAR\_ADDR(32'h0100\_0000))

u8\_axi\_hp0\_rd(

// Outputs

.AXI\_araddr (AXI\_HP0\_araddr),

.AXI\_arburst (AXI\_HP0\_arburst),

.AXI\_arcache (AXI\_HP0\_arcache),

.AXI\_arlen (AXI\_HP0\_arlen),

.AXI\_arlock (AXI\_HP0\_arlock),

.AXI\_arprot (AXI\_HP0\_arprot),

.AXI\_arqos (AXI\_HP0\_arqos),

.AXI\_arsize (AXI\_HP0\_arsize),

.AXI\_arvalid (AXI\_HP0\_arvalid),

.AXI\_rready (AXI\_HP0\_rready),

// Inputs

.rst\_n (sys\_rst\_n),

.i\_clk (lcd\_synclk),

.i\_data\_rst\_n (~lcd\_rfclr),

.i\_data\_rden (lcd\_rfreq1),

.o\_data (lcd\_rfdb1),

.AXI\_clk (AXI\_HP0\_ACLK),

.AXI\_arready (AXI\_HP0\_arready),

.AXI\_rdata (AXI\_HP0\_rdata),

.AXI\_rid (AXI\_HP0\_rid),

.AXI\_rlast (AXI\_HP0\_rlast),

.AXI\_rresp (AXI\_HP0\_rresp),

.AXI\_rvalid (AXI\_HP0\_rvalid)

);

//////////////////////////////////////////////////////////////////////////////////HP1 READ

//AXI HP1主机--读

axi\_hp0\_rd #(

.STAR\_ADDR(32'h0200\_0000))

u9\_axi\_hp1\_rd(

// Outputs

.AXI\_araddr (AXI\_HP1\_araddr),

.AXI\_arburst (AXI\_HP1\_arburst),

.AXI\_arcache (AXI\_HP1\_arcache),

.AXI\_arlen (AXI\_HP1\_arlen),

.AXI\_arlock (AXI\_HP1\_arlock),

.AXI\_arprot (AXI\_HP1\_arprot),

.AXI\_arqos (AXI\_HP1\_arqos),

.AXI\_arsize (AXI\_HP1\_arsize),

.AXI\_arvalid (AXI\_HP1\_arvalid),

.AXI\_rready (AXI\_HP1\_rready),

// Inputs

.rst\_n (sys\_rst\_n),

.i\_clk (lcd\_synclk),

.i\_data\_rst\_n (~lcd\_rfclr),

.i\_data\_rden (lcd\_rfreq2),

.o\_data (lcd\_rfdb2),

.AXI\_clk (AXI\_HP1\_ACLK),

.AXI\_arready (AXI\_HP1\_arready),

.AXI\_rdata (AXI\_HP1\_rdata),

.AXI\_rid (AXI\_HP1\_rid),

.AXI\_rlast (AXI\_HP1\_rlast),

.AXI\_rresp (AXI\_HP1\_rresp),

.AXI\_rvalid (AXI\_HP1\_rvalid)

);

//////////////////////////////////////////////////////////////////////////////////

//LED闪烁逻辑产生模块例化

led\_controller u10\_led\_controller(

.clk(clk\_25m),

.rst\_n(sys\_rst\_n),

.led(led[0])

);

Endmodule

//摄像头配置文件

module image\_controller(

input clk, //50MHz时钟

input rst\_n, //复位信号，低电平有效

//ImageSensor图像采集接口

(\*mark\_debug = "true"\*) input image\_sensor\_pclk, //视频时钟

(\*mark\_debug = "true"\*) input image\_sensor\_vsync, //视频场同步信号，高电平有效（有效视频传输时该信号拉低）

(\*mark\_debug = "true"\*) input image\_sensor\_href, //视频行同步信号

(\*mark\_debug = "true"\*) input[7:0] image\_sensor\_data, //视频数据总线

//ImageSensor串行配置接口

(\*mark\_debug = "true"\*) output image\_sensor\_scl, //串行配置IIC时钟信号

(\*mark\_debug = "true"\*) inout image\_sensor\_sda, //串行配置IIC数据信号

//ImageSensor复位与低功耗接口

output image\_sensor\_reset\_n, //复位接口，低电平有效

output image\_sensor\_pwdn, //低功耗使能信号，高电平有效

//ImageSensor数据写入DDR3接口

input image\_ddr3\_wready,

output image\_ddr3\_wren,

output image\_ddr3\_line\_end,

output image\_ddr3\_frame\_start,

output image\_ddr3\_frame\_end,

output image\_ddr3\_clr,

output[15:0] image\_ddr3\_wrdb

);

//低功耗使能信号，高电平有效

assign image\_sensor\_pwdn = ~rst\_n;

//复位接口，低电平有效

assign image\_sensor\_reset\_n = rst\_n;

////////////////////////////////////////////////////

//IIC寄存器初始化配置

wire tiic\_init\_done; //IIC配置完成标志位，高电平有效

I2C\_OV5640\_Init\_RAW uut\_I2C\_OV5640\_Init\_RAW(

.clk(clk), //50MHz

.rst\_n(rst\_n), //Global Reset

.i2c\_sclk(image\_sensor\_scl), //I2C CLOCK

.i2c\_sdat(image\_sensor\_sda), //I2C DATA

.config\_done(tiic\_init\_done) //Config Done

);

////////////////////////////////////////////////////

//视频输入缓存控制

image\_capture uut\_image\_capture(

.clk(clk), //时钟信号

.rst\_n(rst\_n & tiic\_init\_done), //复位信号

//ImageSensor采集接口

.image\_sensor\_pclk(image\_sensor\_pclk), //视频时钟

.image\_sensor\_vsync(image\_sensor\_vsync), //视频场同步信号，高电平有效（有效视频传输时该信号拉低）

.image\_sensor\_href(image\_sensor\_href), //视频行同步信号

.image\_sensor\_data(image\_sensor\_data), //视频数据总线

//ImageSensor数据写入DDR3接口

.image\_ddr3\_wready(image\_ddr3\_wready),

.image\_ddr3\_wren(image\_ddr3\_wren),

.image\_ddr3\_line\_end(image\_ddr3\_line\_end),

.image\_ddr3\_frame\_start(image\_ddr3\_frame\_start),

.image\_ddr3\_frame\_end(image\_ddr3\_frame\_end),

.image\_ddr3\_clr(image\_ddr3\_clr),

.image\_ddr3\_wrdb(image\_ddr3\_wrdb)

);

endmodule

//灰度值转RGB

module bayer2rgb(

input clk,

input rst\_n,

//input Image Data Flow

input i\_bayer\_image\_vld,

output o\_bayer\_image\_tready,

input[7:0] i\_bayer\_image\_data,

input i\_bayer\_image\_sof,

input i\_bayer\_image\_eof,

input i\_bayer\_image\_eol,

//output Image Data Flow

output reg o\_rgb\_image\_rst,

output o\_rgb\_image\_vld,

output[23:0] o\_rgb\_image\_data

);

reg[15:0] cnt;

reg[5:0] i\_axi\_ctrl\_awaddr;

reg i\_axi\_ctrl\_awvalid;

wire o\_axi\_ctrl\_awready;

reg[31:0] i\_axi\_ctrl\_awdata;

reg i\_axi\_ctrl\_wvalid;

wire o\_axi\_ctrl\_wready;

parameter IMAGE\_WIDTH = 32'd640;////32'd2944;//32'd2880;//

parameter IMAGE\_HIGHT = 32'd480;////32'd2000;//32'd1620;//

////////////////////////////////////////////////////

//demoasic IP例化

v\_demosaic\_0 uut\_v\_demosaic\_0 (

.s\_axi\_CTRL\_AWADDR (i\_axi\_ctrl\_awaddr), // input wire [5 : 0] s\_axi\_CTRL\_AWADDR

.s\_axi\_CTRL\_AWVALID (i\_axi\_ctrl\_awvalid), // input wire s\_axi\_CTRL\_AWVALID

.s\_axi\_CTRL\_AWREADY (o\_axi\_ctrl\_awready), // output wire s\_axi\_CTRL\_AWREADY

.s\_axi\_CTRL\_WDATA (i\_axi\_ctrl\_awdata), // input wire [31 : 0] s\_axi\_CTRL\_WDATA

.s\_axi\_CTRL\_WSTRB (4'hf/\*s\_axi\_CTRL\_WSTRB\*/), // input wire [3 : 0] s\_axi\_CTRL\_WSTRB

.s\_axi\_CTRL\_WVALID (i\_axi\_ctrl\_wvalid), // input wire s\_axi\_CTRL\_WVALID

.s\_axi\_CTRL\_WREADY (o\_axi\_ctrl\_wready), // output wire s\_axi\_CTRL\_WREADY

.s\_axi\_CTRL\_BRESP (/\*s\_axi\_CTRL\_BRESP\*/), // output wire [1 : 0] s\_axi\_CTRL\_BRESP

.s\_axi\_CTRL\_BVALID (/\*s\_axi\_CTRL\_BVALID\*/), // output wire s\_axi\_CTRL\_BVALID

.s\_axi\_CTRL\_BREADY (1'b1/\*s\_axi\_CTRL\_BREADY\*/), // input wire s\_axi\_CTRL\_BREADY

.s\_axi\_CTRL\_ARADDR (6'd0), // input wire [5 : 0] s\_axi\_CTRL\_ARADDR

.s\_axi\_CTRL\_ARVALID (1'b0), // input wire s\_axi\_CTRL\_ARVALID

.s\_axi\_CTRL\_ARREADY (), // output wire s\_axi\_CTRL\_ARREADY

.s\_axi\_CTRL\_RDATA (), // output wire [31 : 0] s\_axi\_CTRL\_RDATA

.s\_axi\_CTRL\_RRESP (/\*s\_axi\_CTRL\_RRESP\*/), // output wire [1 : 0] s\_axi\_CTRL\_RRESP

.s\_axi\_CTRL\_RVALID (), // output wire s\_axi\_CTRL\_RVALID

.s\_axi\_CTRL\_RREADY (1'b1/\*s\_axi\_CTRL\_RREADY\*/), // input wire s\_axi\_CTRL\_RREADY

.ap\_clk (clk), // input wire ap\_clk

.ap\_rst\_n (rst\_n), // input wire ap\_rst\_n

.interrupt (/\*interrupt\*/), // output wire interrupt

.s\_axis\_video\_TVALID (i\_bayer\_image\_vld), // input wire s\_axis\_video\_TVALID

.s\_axis\_video\_TREADY (o\_bayer\_image\_tready), // output wire s\_axis\_video\_TREADY

.s\_axis\_video\_TDATA (i\_bayer\_image\_data), // input wire [7 : 0] s\_axis\_video\_TDATA

.s\_axis\_video\_TKEEP (1'b1/\*s\_axis\_video\_TKEEP\*/), // input wire [0 : 0] s\_axis\_video\_TKEEP

.s\_axis\_video\_TSTRB (1'b1/\*s\_axis\_video\_TSTRB\*/), // input wire [0 : 0] s\_axis\_video\_TSTRB

.s\_axis\_video\_TUSER (i\_bayer\_image\_sof), // input wire [0 : 0] s\_axis\_video\_TUSER

.s\_axis\_video\_TLAST (i\_bayer\_image\_eol), // input wire [0 : 0] s\_axis\_video\_TLAST

.s\_axis\_video\_TID (1'b1/\*s\_axis\_video\_TID\*/), // input wire [0 : 0] s\_axis\_video\_TID

.s\_axis\_video\_TDEST (1'b1/\*s\_axis\_video\_TDEST\*/), // input wire [0 : 0] s\_axis\_video\_TDEST

.m\_axis\_video\_TVALID (o\_rgb\_image\_vld), // output wire m\_axis\_video\_TVALID

.m\_axis\_video\_TREADY (1'b1/\*m\_axis\_video\_TREADY\*/), // input wire m\_axis\_video\_TREADY

.m\_axis\_video\_TDATA (o\_rgb\_image\_data), // output wire [23 : 0] m\_axis\_video\_TDATA

.m\_axis\_video\_TKEEP (/\*m\_axis\_video\_TKEEP\*/), // output wire [2 : 0] m\_axis\_video\_TKEEP

.m\_axis\_video\_TSTRB (/\*m\_axis\_video\_TSTRB\*/), // output wire [2 : 0] m\_axis\_video\_TSTRB

.m\_axis\_video\_TUSER (/\*m\_axis\_video\_TUSER\*/), // output wire [0 : 0] m\_axis\_video\_TUSER

.m\_axis\_video\_TLAST (/\*m\_axis\_video\_TLAST\*/), // output wire [0 : 0] m\_axis\_video\_TLAST

.m\_axis\_video\_TID (/\*m\_axis\_video\_TID\*/), // output wire [0 : 0] m\_axis\_video\_TID

.m\_axis\_video\_TDEST (/\*m\_axis\_video\_TDEST\*/) // output wire [0 : 0] m\_axis\_video\_TDEST

);

////////////////////////////////////////////////////

//demoasic IP初始化

always @(posedge clk)

if(!rst\_n) cnt <= 16'd0;

else if(cnt < 16'hffff) cnt <= cnt+1'b1;

wire timer\_1 = (cnt == 16'h8000);

wire timer\_2 = (cnt == 16'h8004);

wire timer\_3 = (cnt == 16'h9000);

wire timer\_4 = (cnt == 16'h9004);

wire timer\_5 = (cnt == 16'ha000);

wire timer\_6 = (cnt == 16'ha004);

wire timer\_7 = (cnt == 16'hb000);

wire timer\_8 = (cnt == 16'hb004);

always @(posedge clk)

if(!rst\_n) begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b0;

i\_axi\_ctrl\_awdata <= 32'd0;

i\_axi\_ctrl\_wvalid <= 1'b0;

end

//register 0x0010 (active width) = IMAGE\_WIDTH

else if(timer\_1) begin

i\_axi\_ctrl\_awaddr <= 6'h10;

i\_axi\_ctrl\_awvalid <= 1'b1;

i\_axi\_ctrl\_awdata <= 32'd0;

i\_axi\_ctrl\_wvalid <= 1'b0;

end

else if(timer\_2) begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b0;

i\_axi\_ctrl\_awdata <= IMAGE\_WIDTH;

i\_axi\_ctrl\_wvalid <= 1'b1;

end

//register 0x0018 (active height) = IMAGE\_HIGHT

else if(timer\_3) begin

i\_axi\_ctrl\_awaddr <= 6'h18;

i\_axi\_ctrl\_awvalid <= 1'b1;

i\_axi\_ctrl\_awdata <= 32'd0;

i\_axi\_ctrl\_wvalid <= 1'b0;

end

else if(timer\_4) begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b0;

i\_axi\_ctrl\_awdata <= IMAGE\_HIGHT;

i\_axi\_ctrl\_wvalid <= 1'b1;

end

//register 0x0028 (bayer phase) = 0- RG/GB, 1 - GR/BG, 2 - GB/RG, 3- BG/GR

else if(timer\_5) begin

i\_axi\_ctrl\_awaddr <= 6'h28;

i\_axi\_ctrl\_awvalid <= 1'b1;

i\_axi\_ctrl\_awdata <= 32'd0;

i\_axi\_ctrl\_wvalid <= 1'b0;

end

else if(timer\_6) begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b0;

i\_axi\_ctrl\_awdata <= 32'd1; //GR/BG

//i\_axi\_ctrl\_awdata <= 32'd2; //GB/RG

//i\_axi\_ctrl\_awdata <= 32'd3; // BG/GR

i\_axi\_ctrl\_wvalid <= 1'b1;

end

//register 0 (ctrl): bit7=1 (auto\_restart)

else if(timer\_7) begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b1;

i\_axi\_ctrl\_awdata <= 32'd0;

i\_axi\_ctrl\_wvalid <= 1'b0;

end

else if(timer\_8) begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b0;

i\_axi\_ctrl\_awdata <= 32'h0000\_0081;

i\_axi\_ctrl\_wvalid <= 1'b1;

end

else begin

i\_axi\_ctrl\_awaddr <= 6'd0;

i\_axi\_ctrl\_awvalid <= 1'b0;

i\_axi\_ctrl\_awdata <= 32'd0;

i\_axi\_ctrl\_wvalid <= 1'b0;

end

////////////////////////////////////////////////////

//延时计数器，产生复位信号

reg[11:0] dly;

always @(posedge clk)

if(!rst\_n) dly <= 12'd0;

else if(i\_bayer\_image\_eof) dly <= 12'd1;

else if(dly != 12'd0) dly <= dly+1'b1;

else dly <= 12'd0;

always @(posedge clk)

if(!rst\_n) o\_rgb\_image\_rst <= 1'b0;

else if((dly >= 12'd3200) && (dly <= 12'd3300)) o\_rgb\_image\_rst <= 1'b1;

else o\_rgb\_image\_rst <= 1'b0;

endmodule

//AXI总线配置

module axi\_hp0\_wr#(

parameter STAR\_ADDR = 32'h0100\_0000)

(

//系统信号

input rst\_n,

//写入DDR3的数据

input i\_clk,

input i\_data\_rst\_n,

input i\_data\_en,

input[15:0] i\_data,

//AXI总线时钟

input AXI\_clk,

//AXI写地址通道

output reg[31:0] AXI\_awaddr,

output[3:0] AXI\_awlen,

output[2:0] AXI\_awsize,

output[1:0] AXI\_awburst,

output[1:0] AXI\_awlock,

output[3:0] AXI\_awcache,

output[2:0] AXI\_awprot,

output[3:0] AXI\_awqos,

output reg AXI\_awvalid,

input AXI\_awready,

//AXI写数据通道

output[63:0] AXI\_wdata,

output[7:0] AXI\_wstrb,

output reg AXI\_wlast,

output reg AXI\_wvalid,

input AXI\_wready,

//AXI写响应通道

input[5:0] AXI\_bid,

input[1:0] AXI\_bresp,

input AXI\_bvalid,

output AXI\_bready

);

//------------------------------------------------------------------------------------

//内部信号申明

wire[7:0] wrfifo\_data\_count; //FIFO可读数据数量

reg wrfifo\_rden; //FIFO读数据使能信号

reg[7:0] wrdata\_num; //写入DDR3数据计数器

reg[3:0] cstate,nstate; //状态寄存器

parameter AXI\_BURST\_LEN = 16;

parameter STATE\_RST = 4'h0;

parameter STATE\_IDLE = 4'h1;

parameter STATE\_WADD = 4'h2;

parameter STATE\_WDAT = 4'h3;

parameter WRITE\_DONE = 4'h4;

//-------------------------------------------------------------------------------

//缓存FIFO，将数据从i\_clk转到AXI\_clk时钟域，位宽从16bit转到64bit

fifo\_generator\_0 uut\_fifo\_generator\_0 (

.rst(~i\_data\_rst\_n), // input wire rst

.wr\_clk(i\_clk), // input wire wr\_clk

.rd\_clk(AXI\_clk), // input wire rd\_clk

.din(i\_data), // input wire [15 : 0] din

.wr\_en(i\_data\_en), // input wire wr\_en

.rd\_en(wrfifo\_rden), // input wire rd\_en

.dout(AXI\_wdata), // output wire [63 : 0] dout

.full(), // output wire full

.empty(), // output wire empty

.rd\_data\_count(wrfifo\_data\_count) // output wire [7 : 0] rd\_data\_count

);

//------------------------------------------------------------------------------------

//将i\_data\_rst\_n在AXI\_clk时钟域打一拍

reg w\_data\_rst\_n;

always @(posedge AXI\_clk)

w\_data\_rst\_n <= i\_data\_rst\_n;

//-------------------------------------------------------------------------------

//AXI写状态机

always @(posedge AXI\_clk or negedge rst\_n) begin

if(~rst\_n)begin

cstate <= STATE\_RST;

end

else begin

cstate <= nstate;

end

end

always @( \* ) begin

case(cstate)

STATE\_RST: begin

if(w\_data\_rst\_n) nstate = STATE\_IDLE;

else nstate = STATE\_RST;

end

STATE\_IDLE: begin

if(!w\_data\_rst\_n) nstate = STATE\_RST;

else if(wrfifo\_data\_count >= 8'd16) nstate = STATE\_WADD;

else nstate = STATE\_IDLE;

end

STATE\_WADD: begin

if(AXI\_awvalid && AXI\_awready) nstate = STATE\_WDAT;

else nstate = STATE\_WADD;

end

STATE\_WDAT: begin

if(wrdata\_num >= (AXI\_BURST\_LEN+1)) nstate = WRITE\_DONE;

else nstate = STATE\_WDAT;

end

WRITE\_DONE: begin

nstate = STATE\_IDLE;

end

default: begin

nstate = STATE\_RST;

end

endcase

end

//1个burst写入数据的个数计数

always @(posedge AXI\_clk) begin

if (!rst\_n) wrdata\_num <= 'b0;

else if(cstate == STATE\_WDAT) begin

if(wrdata\_num == 8'd0) wrdata\_num <= wrdata\_num + 1'b1;

else if((wrdata\_num <= AXI\_BURST\_LEN) && AXI\_wready && AXI\_wvalid) wrdata\_num <= wrdata\_num + 1'b1;

else wrdata\_num <= wrdata\_num;

end

else wrdata\_num <= 'b0;

end

//-------------------------------------------------------------------------------

//FIFO读取使能信号产生

always @(\*) begin

if (cstate == STATE\_WDAT) begin

if(wrdata\_num == 8'd0) wrfifo\_rden <= 1'b1;

else if((wrdata\_num >= 8'd1) && (wrdata\_num < AXI\_BURST\_LEN) && AXI\_wready && AXI\_wvalid) wrfifo\_rden <= 1'b1;

else wrfifo\_rden <= 1'b0;

end

else wrfifo\_rden <= 1'b0;

end

//-------------------------------------------------------------------------------

//AXI总线写数据时序产生

//写地址产生

always @(posedge AXI\_clk)begin

if(cstate == STATE\_RST) AXI\_awaddr <= STAR\_ADDR;

else if(AXI\_awvalid && AXI\_awready) AXI\_awaddr <= AXI\_awaddr + AXI\_BURST\_LEN \* 8;

end

//写地址有效信号产生

always @(posedge AXI\_clk) begin

if (!rst\_n) AXI\_awvalid <= 1'b0;

else if(cstate == STATE\_WADD) begin

if(AXI\_awvalid && AXI\_awready) AXI\_awvalid <= 1'b0;

else AXI\_awvalid <= 1'b1;

end

else AXI\_awvalid <= 1'b0;

end

//写数据有效信号产生

always @(posedge AXI\_clk) begin

if (!rst\_n) AXI\_wvalid <= 1'b0;

else if((wrdata\_num >= 8'd1) && (wrdata\_num < AXI\_BURST\_LEN)) AXI\_wvalid <= 1'b1;

else if((wrdata\_num == AXI\_BURST\_LEN) && !AXI\_wready) AXI\_wvalid <= 1'b1;

else AXI\_wvalid <= 1'b0;

end

//写最后一个数据有效信号产生

always @(posedge AXI\_clk) begin

if (!rst\_n) AXI\_wlast <= 1'b0;

else if((wrdata\_num == (AXI\_BURST\_LEN - 1)) && AXI\_wready && AXI\_wvalid) AXI\_wlast <= 1'b1;

else if((wrdata\_num == AXI\_BURST\_LEN) && !AXI\_wready) AXI\_wlast <= 1'b1;

else AXI\_wlast <= 1'b0;

end

//-------------------------------------------------------------------------------

//AXI HP总线固定赋值设置

assign AXI\_awsize = 3'b011; //8 Bytes per burst

assign AXI\_awburst = 2'b01;

assign AXI\_awlock = 2'b00;

assign AXI\_awcache = 4'b0010;

assign AXI\_awprot = 3'h0;

assign AXI\_awqos = 4'h0;

assign AXI\_wstrb = 8'hff;

assign AXI\_bready = 1'b1;

assign AXI\_awlen = AXI\_BURST\_LEN - 1;

endmodule

//Gamma矫正模块

module gamma\_correction(

input clk,

input rst\_n,

//input i\_rgb\_image\_rst,

input i\_rgb\_image\_vld,

input[23:0] i\_rgb\_image\_data,

output reg o\_gamma\_image\_vld,

output[23:0] o\_gamma\_image\_data

);

////////////////////////////////////////////////////

//输出使能信号产生

reg r\_rgb\_image\_vld;

always @(posedge clk)

if(!rst\_n) r\_rgb\_image\_vld <= 1'b0;

else r\_rgb\_image\_vld <= i\_rgb\_image\_vld;

always @(posedge clk)

if(!rst\_n) o\_gamma\_image\_vld <= 1'b0;

else o\_gamma\_image\_vld <= r\_rgb\_image\_vld;

////////////////////////////////////////////////////

//gamma LUT ROM for R

blk\_mem\_gen\_0 uut\_blk\_mem\_gen\_r (

.clka(clk), // input wire clka

.ena(i\_rgb\_image\_vld), // input wire ena

.addra(i\_rgb\_image\_data[23:16]), // input wire [7 : 0] addra

.douta(o\_gamma\_image\_data[23:16]) // output wire [7 : 0] douta

);

////////////////////////////////////////////////////

//gamma LUT ROM for G

blk\_mem\_gen\_0 uut\_blk\_mem\_gen\_g (

.clka(clk), // input wire clka

.ena(i\_rgb\_image\_vld), // input wire ena

.addra(i\_rgb\_image\_data[7:0]), // input wire [7 : 0] addra

.douta(o\_gamma\_image\_data[7:0]) // output wire [7 : 0] douta

);

////////////////////////////////////////////////////

//gamma LUT ROM for B

blk\_mem\_gen\_0 uut\_blk\_mem\_gen\_b (

.clka(clk), // input wire clka

.ena(i\_rgb\_image\_vld), // input wire ena

.addra(i\_rgb\_image\_data[15:8]), // input wire [7 : 0] addra

.douta(o\_gamma\_image\_data[15:8]) // output wire [7 : 0] douta

);

endmodule

//Laplace边缘提取模块

module laplace\_transform(

input clk, //50MHz时钟

input rst\_n, //复位信号，低电平有效

//Image Data flow from Image Sensor

input i\_image\_ddr3\_wren,

input i\_image\_ddr3\_line\_end,

input i\_image\_ddr3\_clr,

input[15:0] i\_image\_ddr3\_wrdb,

//Image Data flow after laplace transform

output reg o\_image\_ddr3\_wren,

//output[15:0] o\_image\_ddr3\_wrdb

output[15:0] o\_image\_ddr3\_wrdb\_modify

);

parameter IMAGE\_WIDTH = 10'd640;

parameter IMAGE\_HIGHT = 10'd480;

//modify

wire [15:0]o\_image\_ddr3\_wrdb;

wire [7:0] Gray\_Temp1, Gray\_Temp;

assign Gray\_Temp1 = {o\_image\_ddr3\_wrdb[15:11],0} +

o\_image\_ddr3\_wrdb[10:5] +

{o\_image\_ddr3\_wrdb[4:0],0};

assign Gray\_Temp = Gray\_Temp1[7:2];

assign o\_image\_ddr3\_wrdb\_modify = {Gray\_Temp[5:1],Gray\_Temp,Gray\_Temp[5:1]};

////////////////////////////////////////////////////

//数据行计数器

reg[9:0] r\_line\_cnt;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_line\_cnt <= 10'd0;

else if(i\_image\_ddr3\_clr) r\_line\_cnt <= 10'd0;

else if(i\_image\_ddr3\_line\_end) r\_line\_cnt <= r\_line\_cnt+1'b1;

else ;

////////////////////////////////////////////////////

//FIFO for cache 1 line image data

reg r\_fifo1\_rd\_en;

wire[15:0] w\_fifo1\_dout;

wire[9:0] w\_fifo1\_data\_count;

fifo\_generator\_3 uut1\_fifo\_generator\_3 (

.clk(clk), // input wire clk

.srst(!rst\_n || i\_image\_ddr3\_clr), // input wire srst

.din(i\_image\_ddr3\_wrdb), // input wire [15 : 0] din

.wr\_en(i\_image\_ddr3\_wren), // input wire wr\_en

.rd\_en(r\_fifo1\_rd\_en), // input wire rd\_en

.dout(w\_fifo1\_dout), // output wire [15 : 0] dout

.full(), // output wire full

.empty(), // output wire empty

.data\_count(w\_fifo1\_data\_count) // output wire [9 : 0] data\_count

);

reg r\_fifo2\_wr\_en;

reg r\_fifo2\_rd\_en;

wire[15:0] w\_fifo2\_dout;

wire[9:0] w\_fifo2\_data\_count;

fifo\_generator\_3 uut2\_fifo\_generator\_3 (

.clk(clk), // input wire clk

.srst(!rst\_n || i\_image\_ddr3\_clr), // input wire srst

.din(w\_fifo1\_dout), // input wire [15 : 0] din

.wr\_en(r\_fifo2\_wr\_en), // input wire wr\_en

.rd\_en(r\_fifo2\_rd\_en), // input wire rd\_en

.dout(w\_fifo2\_dout), // output wire [15 : 0] dout

.full(), // output wire full

.empty(), // output wire empty

.data\_count(w\_fifo2\_data\_count) // output wire [9 : 0] data\_count

);

////////////////////////////////////////////////////

//连续读出640个数据计数控制状态机

parameter RFIFO\_RESET = 3'd0;

parameter RFIFO\_IDLE = 3'd1;

parameter RFIFO\_RDDB1 = 3'd2;

parameter RFIFO\_RDDB2 = 3'd3;

parameter RFIFO\_WAIT = 3'd4;

parameter RFIFO\_RDDB3 = 3'd5;

reg[2:0] rfifo\_state;

reg[9:0] dcnt; //读FIFO数据个数计数器

reg[9:0] laplace\_num;

reg[3:0] dly\_cnt;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) rfifo\_state <= RFIFO\_RESET;

else if(i\_image\_ddr3\_clr) rfifo\_state <= RFIFO\_RESET;

else begin

case(rfifo\_state)

RFIFO\_RESET: if(w\_fifo1\_data\_count >= IMAGE\_WIDTH) rfifo\_state <= RFIFO\_RDDB1;

else rfifo\_state <= RFIFO\_RESET;

RFIFO\_RDDB1: if(dcnt >= (IMAGE\_WIDTH-1)) rfifo\_state <= RFIFO\_IDLE;

else rfifo\_state <= RFIFO\_RDDB1;

RFIFO\_IDLE: if(r\_line\_cnt >= IMAGE\_HIGHT) rfifo\_state <= RFIFO\_WAIT;

else if(w\_fifo2\_data\_count >= IMAGE\_WIDTH) rfifo\_state <= RFIFO\_RDDB2;

else rfifo\_state <= RFIFO\_IDLE;

RFIFO\_RDDB2: if(r\_line\_cnt >= IMAGE\_HIGHT) rfifo\_state <= RFIFO\_WAIT;

else if(dcnt >= (IMAGE\_WIDTH-1)) rfifo\_state <= RFIFO\_IDLE;

else rfifo\_state <= RFIFO\_RDDB2;

RFIFO\_WAIT: if(dly\_cnt == 4'hf) rfifo\_state <= RFIFO\_RDDB3;

else rfifo\_state <= RFIFO\_WAIT;

RFIFO\_RDDB3: if(dcnt >= (IMAGE\_WIDTH-1)) rfifo\_state <= RFIFO\_RESET;

else rfifo\_state <= RFIFO\_RDDB3;

default: rfifo\_state <= RFIFO\_IDLE;

endcase

end

always @(posedge clk or negedge rst\_n)

if(!rst\_n) dly\_cnt <= 4'd0;

else if(rfifo\_state == RFIFO\_WAIT) dly\_cnt <= dly\_cnt+1'b1;

else dly\_cnt <= 4'd0;

//读FIFO数据个数计数器

always @(posedge clk or negedge rst\_n)

if(!rst\_n) dcnt <= 10'd0;

else if((rfifo\_state == RFIFO\_IDLE) || (rfifo\_state == RFIFO\_RESET)) dcnt <= 10'd0;

else if(((rfifo\_state == RFIFO\_RDDB1) || (rfifo\_state == RFIFO\_RDDB2)) && i\_image\_ddr3\_wren) dcnt <= dcnt+1'b1;

else if(rfifo\_state == RFIFO\_RDDB3) dcnt <= dcnt+1'b1;

else dcnt <= 10'd0;

//laplace transform数据计数器

always @(posedge clk or negedge rst\_n)

if(!rst\_n) laplace\_num <= 10'd0;

else if(dcnt == 10'd4) laplace\_num <= 10'd1;

else if(laplace\_num != 10'd0) begin

if(laplace\_num < IMAGE\_WIDTH) laplace\_num <= laplace\_num+1'b1;

else laplace\_num <= 10'd0;

end

else laplace\_num <= 10'd0;

//读FIFO1使能信号产生逻辑

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_fifo1\_rd\_en <= 1'b0;

else if(((rfifo\_state == RFIFO\_RDDB1) || (rfifo\_state == RFIFO\_RDDB2)) && i\_image\_ddr3\_wren) r\_fifo1\_rd\_en <= 1'b1;

else if((rfifo\_state == RFIFO\_RDDB3) && (dcnt < IMAGE\_WIDTH)) r\_fifo1\_rd\_en <= 1'b1;

else r\_fifo1\_rd\_en <= 1'b0;

//写FIFO2是能信号产生逻辑

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_fifo2\_wr\_en <= 1'b0;

else r\_fifo2\_wr\_en <= r\_fifo1\_rd\_en;

//读FIFO2使能信号产生逻辑

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_fifo2\_rd\_en <= 1'b0;

else if(((rfifo\_state == RFIFO\_RDDB2) || (rfifo\_state == RFIFO\_RDDB3)) && i\_image\_ddr3\_wren) r\_fifo2\_rd\_en <= 1'b1;

else r\_fifo2\_rd\_en <= 1'b0;

////////////////////////////////////////////////////

//图像缓存3拍

reg[15:0] data\_temp\_line\_1[2:0];

reg[15:0] data\_temp\_line\_2[2:0];

reg[15:0] data\_temp\_line\_3[4:0];

always @(posedge clk) begin

data\_temp\_line\_1[0] <= w\_fifo2\_dout;

data\_temp\_line\_1[1] <= data\_temp\_line\_1[0];

data\_temp\_line\_1[2] <= data\_temp\_line\_1[1];

end

always @(posedge clk) begin

data\_temp\_line\_2[0] <= w\_fifo1\_dout;

data\_temp\_line\_2[1] <= data\_temp\_line\_2[0];

data\_temp\_line\_2[2] <= data\_temp\_line\_2[1];

end

always @(posedge clk) begin

data\_temp\_line\_3[0] <= i\_image\_ddr3\_wrdb;

data\_temp\_line\_3[1] <= data\_temp\_line\_3[0];

data\_temp\_line\_3[2] <= data\_temp\_line\_3[1];

data\_temp\_line\_3[3] <= data\_temp\_line\_3[2];

data\_temp\_line\_3[4] <= data\_temp\_line\_3[3];

end

////////////////////////////////////////////////////

//图像输出laplace transform运算

reg[9:0] sum\_a\_r,sum\_b\_r1,sum\_b\_r2;

reg[9:0] sum\_a\_g,sum\_b\_g1,sum\_b\_g2;

reg[9:0] sum\_a\_b,sum\_b\_b1,sum\_b\_b2;

reg[15:0] laplace\_result;

always @(posedge clk) begin

sum\_a\_r <= {2'b00,data\_temp\_line\_2[1][15:11],3'b000};

sum\_a\_g <= {1'b0,data\_temp\_line\_2[1][10:5],3'b000};

sum\_a\_b <= {2'b00,data\_temp\_line\_2[1][4:0],3'b000};

end

always @(posedge clk) begin

sum\_b\_r1 <= {5'd0,data\_temp\_line\_2[0][15:11]} + {5'd0,data\_temp\_line\_2[2][15:11]} + {5'd0,data\_temp\_line\_1[1][15:11]} + {5'd0,data\_temp\_line\_3[3][15:11]};

sum\_b\_g1 <= {4'd0,data\_temp\_line\_2[0][10:5]} + {4'd0,data\_temp\_line\_2[2][10:5]} + {4'd0,data\_temp\_line\_1[1][10:5]} + {4'd0,data\_temp\_line\_3[3][10:5]};

sum\_b\_b1 <= {5'd0,data\_temp\_line\_2[0][4:0]} + {5'd0,data\_temp\_line\_2[2][4:0]} + {5'd0,data\_temp\_line\_1[1][4:0]} + {5'd0,data\_temp\_line\_3[3][4:0]};

sum\_b\_r2 <= {5'd0,data\_temp\_line\_1[0][15:11]} + {5'd0,data\_temp\_line\_1[2][15:11]} + {5'd0,data\_temp\_line\_3[2][15:11]} + {5'd0,data\_temp\_line\_3[4][15:11]};

sum\_b\_g2 <= {4'd0,data\_temp\_line\_1[0][10:5]} + {4'd0,data\_temp\_line\_1[2][10:5]} + {4'd0,data\_temp\_line\_3[2][10:5]} + {4'd0,data\_temp\_line\_3[4][10:5]};

sum\_b\_b2 <= {5'd0,data\_temp\_line\_1[0][4:0]} + {5'd0,data\_temp\_line\_1[2][4:0]} + {5'd0,data\_temp\_line\_3[2][4:0]} + {5'd0,data\_temp\_line\_3[4][4:0]};

end

wire[9:0] temp\_r = sum\_a\_r-(sum\_b\_r1+sum\_b\_r2);

wire[9:0] temp\_g = sum\_a\_g-(sum\_b\_g1+sum\_b\_g2);

wire[9:0] temp\_b = sum\_a\_b-(sum\_b\_b1+sum\_b\_b2);

always @(posedge clk) begin

if((laplace\_num == 10'd1) || (laplace\_num == IMAGE\_WIDTH)) laplace\_result <= data\_temp\_line\_2[2]; //第1列和最后1列使用原值

else begin

if(sum\_a\_r < (sum\_b\_r1+sum\_b\_r2)) laplace\_result[15:11] <= 5'd0;

else if(temp\_r[9:5] != 5'd0) laplace\_result[15:11] <= 5'b1\_1111;

else laplace\_result[15:11] = temp\_r[4:0];

if(sum\_a\_g < (sum\_b\_g1+sum\_b\_g2)) laplace\_result[10:5] <= 6'd0;

else if(temp\_g[9:6] != 4'd0) laplace\_result[10:5] <= 6'b11\_1111;

else laplace\_result[10:5] = temp\_g[5:0];

if(sum\_a\_b < (sum\_b\_b1+sum\_b\_b2)) laplace\_result[4:0] <= 5'd0;

else if(temp\_b[9:5] != 5'd0) laplace\_result[4:0] <= 5'b1\_1111;

else laplace\_result[4:0] = temp\_b[4:0];

end

end

////////////////////////////////////////////////////

//图像输出有效信号产生

reg r\_image\_ddr3\_wren;

reg[3:0] r\_laplace\_line\_wren;

reg r\_last\_line\_wren;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_last\_line\_wren <= 1'b0;

else if((rfifo\_state == RFIFO\_RDDB3) && (dcnt < IMAGE\_WIDTH)) r\_last\_line\_wren <= 1'b1;

else r\_last\_line\_wren <= 1'b0;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_laplace\_line\_wren <= 4'd0;

else begin

r\_laplace\_line\_wren[3:1] <= r\_laplace\_line\_wren[2:0];

if((rfifo\_state == RFIFO\_RDDB2) && (r\_line\_cnt > 10'd1) && i\_image\_ddr3\_wren) r\_laplace\_line\_wren[0] <= 1'b1;

else r\_laplace\_line\_wren[0] <= 1'b0;

end

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_image\_ddr3\_wren <= 1'b0;

else if(r\_laplace\_line\_wren[3]) r\_image\_ddr3\_wren <= 1'b1;

else if(r\_last\_line\_wren) r\_image\_ddr3\_wren <= 1'b1;

else r\_image\_ddr3\_wren <= 1'b0;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) o\_image\_ddr3\_wren <= 1'b0;

else if((r\_line\_cnt == 10'd0) && i\_image\_ddr3\_wren) o\_image\_ddr3\_wren <= 1'b1;

else if(r\_image\_ddr3\_wren) o\_image\_ddr3\_wren <= 1'b1;

else o\_image\_ddr3\_wren <= 1'b0;

////////////////////////////////////////////////////

//图像数据输出

reg[15:0] r\_image\_ddr3\_wrdb;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) r\_image\_ddr3\_wrdb <= 16'd0;

else if(r\_line\_cnt == 10'd0) r\_image\_ddr3\_wrdb <= i\_image\_ddr3\_wrdb;

else if(r\_image\_ddr3\_wren && (r\_line\_cnt >= IMAGE\_HIGHT)) r\_image\_ddr3\_wrdb <= w\_fifo1\_dout;

else ;

reg output\_link;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) output\_link <= 1'b0;

else if(r\_line\_cnt == 10'd0) output\_link <= 1'b1;

else if(r\_image\_ddr3\_wren && (r\_line\_cnt >= IMAGE\_HIGHT)) output\_link <= 1'b1;

else output\_link <= 1'b0;

assign o\_image\_ddr3\_wrdb = output\_link ? r\_image\_ddr3\_wrdb:laplace\_result;

Endmodule

//VGA输出控制模块

module lcd\_driver(

//系统时钟与复位信号

input clk\_25m,

input clk\_50m,

input clk\_65m,

input clk\_75m,

input clk\_108m,

input clk\_130m,

input rst\_n,

//VGA驱动接口

output[4:0] vga\_r,

output[5:0] vga\_g,

output[4:0] vga\_b,

output[2:0] vga\_rgb,

output reg vga\_hsy,vga\_vsy,

output vga\_clk,

output adv7123\_blank\_n,

output adv7123\_sync\_n,

//LCD与FIFO的接口

output lcd\_synclk,

input[15:0] lcd\_rfdb1, //FIFO读出数据总线

input[15:0] lcd\_rfdb2, //FIFO读出数据总线

output reg lcd\_rfreq1, //FIFO读请求信号

output reg lcd\_rfreq2, //FIFO读请求信号

output reg lcd\_rfclr //FIFO复位信号，高电平有效

);

//-----------------------------------------------------------

wire clk;

assign vga\_clk = ~clk;

assign lcd\_synclk = clk;

assign vga\_rgb = 3'b000;

//-----------------------------------------------------------

//`define VGA\_640\_480

//`define VGA\_800\_600

//`define VGA\_1024\_768

`define VGA\_1280\_720

//`define VGA\_1280\_960

//`define VGA\_1280\_1024

//`define VGA\_1920\_1080

//-----------------------------------------------------------

`ifdef VGA\_640\_480

//VGA Timing 640\*480 & 25MHz & 60Hz

assign clk = clk\_25m;

parameter VGA\_HTT = 12'd800-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd96; //Hor Sync Time

parameter VGA\_HBP = 12'd48;//+12'd16; //Hor Back Porch

parameter VGA\_HVT = 12'd640; //Hor Valid Time

parameter VGA\_HFP = 12'd16; //Hor Front Porch

parameter VGA\_VTT = 12'd525-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd2; //Ver Sync Time

parameter VGA\_VBP = 12'd33;//-12'd4; //Ver Back Porch

parameter VGA\_VVT = 12'd480; //Ver Valid Time

parameter VGA\_VFP = 12'd10; //Ver Front Porch

`endif

`ifdef VGA\_800\_600

//VGA Timing 800\*600 & 50MHz & 72Hz

assign clk = clk\_50m;

parameter VGA\_HTT = 12'd1040-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd120; //Hor Sync Time

parameter VGA\_HBP = 12'd64; //Hor Back Porch

parameter VGA\_HVT = 12'd800; //Hor Valid Time

parameter VGA\_HFP = 12'd56; //Hor Front Porch

parameter VGA\_VTT = 12'd666-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd6; //Ver Sync Time

parameter VGA\_VBP = 12'd23; //Ver Back Porch

parameter VGA\_VVT = 12'd600; //Ver Valid Time

parameter VGA\_VFP = 12'd37; //Ver Front Porch

`endif

`ifdef VGA\_1024\_768

//VGA Timing 1024\*768 & 65MHz & 60Hz

assign clk = clk\_65m;

parameter VGA\_HTT = 12'd1344-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd136; //Hor Sync Time

parameter VGA\_HBP = 12'd160; //Hor Back Porch

parameter VGA\_HVT = 12'd1024; //Hor Valid Time

parameter VGA\_HFP = 12'd24; //Hor Front Porch

parameter VGA\_VTT = 12'd806-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd6; //Ver Sync Time

parameter VGA\_VBP = 12'd29; //Ver Back Porch

parameter VGA\_VVT = 12'd768; //Ver Valid Time

parameter VGA\_VFP = 12'd3; //Ver Front Porch

`endif

`ifdef VGA\_1280\_720

//VGA Timing 1280\*720 & 75MHz & 60Hz

assign clk = clk\_75m;

parameter VGA\_HTT = 12'd1648-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd80; //Hor Sync Time

parameter VGA\_HBP = 12'd216; //Hor Back Porch

parameter VGA\_HVT = 12'd1280; //Hor Valid Time

parameter VGA\_HFP = 12'd72; //Hor Front Porch

parameter VGA\_VTT = 12'd750-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd5; //Ver Sync Time

parameter VGA\_VBP = 12'd22; //Ver Back Porch

parameter VGA\_VVT = 12'd720; //Ver Valid Time

parameter VGA\_VFP = 12'd3; //Ver Front Porch

`endif

`ifdef VGA\_1280\_1024

//VGA Timing 1280\*960 & 108MHz & 60Hz

assign clk = clk\_108m;

parameter VGA\_HTT = 12'd1688-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd112; //Hor Sync Time

parameter VGA\_HBP = 12'd248; //Hor Back Porch

parameter VGA\_HVT = 12'd1280; //Hor Valid Time

parameter VGA\_HFP = 12'd48; //Hor Front Porch

parameter VGA\_VTT = 12'd1066-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd3; //Ver Sync Time

parameter VGA\_VBP = 12'd38; //Ver Back Porch

parameter VGA\_VVT = 12'd1024; //Ver Valid Time

parameter VGA\_VFP = 12'd1; //Ver Front Porch

`endif

`ifdef VGA\_1280\_960

//VGA Timing 1280\*1024 & 108MHz & 60Hz

assign clk = clk\_108m;

parameter VGA\_HTT = 12'd1800-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd112; //Hor Sync Time

parameter VGA\_HBP = 12'd312; //Hor Back Porch

parameter VGA\_HVT = 12'd1280; //Hor Valid Time

parameter VGA\_HFP = 12'd96; //Hor Front Porch

parameter VGA\_VTT = 12'd1000-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd3; //Ver Sync Time

parameter VGA\_VBP = 12'd36; //Ver Back Porch

parameter VGA\_VVT = 12'd960; //Ver Valid Time

parameter VGA\_VFP = 12'd1; //Ver Front Porch

`endif

`ifdef VGA\_1920\_1080

//VGA Timing 1920\*1080 & 130MHz & 60Hz

assign clk = clk\_130m;

parameter VGA\_HTT = 12'd2000-12'd1; //Hor Total Time

parameter VGA\_HST = 12'd12; //Hor Sync Time

parameter VGA\_HBP = 12'd40; //Hor Back Porch

parameter VGA\_HVT = 12'd1920; //Hor Valid Time

parameter VGA\_HFP = 12'd28; //Hor Front Porch

parameter VGA\_VTT = 12'd1105-12'd1; //Ver Total Time

parameter VGA\_VST = 12'd4; //Ver Sync Time

parameter VGA\_VBP = 12'd18; //Ver Back Porch

parameter VGA\_VVT = 12'd1080; //Ver Valid Time

parameter VGA\_VFP = 12'd3; //Ver Front Porch

`endif

//-----------------------------------------------------------

//x and y counter

reg[11:0] xcnt;

reg[11:0] ycnt;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) xcnt <= 12'd0;

else if(xcnt >= VGA\_HTT) xcnt <= 12'd0;

else xcnt <= xcnt+1'b1;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) ycnt <= 12'd0;

else if(xcnt == VGA\_HTT) begin

if(ycnt >= VGA\_VTT) ycnt <= 12'd0;

else ycnt <= ycnt+1'b1;

end

else ;

//-----------------------------------------------------------

//hsy and vsy generate

always @(posedge clk or negedge rst\_n)

if(!rst\_n) vga\_hsy <= 1'b0;

else if(xcnt < VGA\_HST) vga\_hsy <= 1'b1;

else vga\_hsy <= 1'b0;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) vga\_vsy <= 1'b0;

else if(ycnt < VGA\_VST) vga\_vsy <= 1'b1;

else vga\_vsy <= 1'b0;

//-----------------------------------------------------------

//vga valid signal generate

reg vga\_valid;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) vga\_valid <= 1'b0;

else if((xcnt >= (VGA\_HST+VGA\_HBP)) && (xcnt < (VGA\_HST+VGA\_HBP+VGA\_HVT))

&& (ycnt >= (VGA\_VST+VGA\_VBP)) && (ycnt < (VGA\_VST+VGA\_VBP+VGA\_VVT)))

vga\_valid <= 1'b1;

else vga\_valid <= 1'b0;

assign adv7123\_blank\_n = vga\_valid;

assign adv7123\_sync\_n = 1'b0;

//--------------------------------------------------

//FIFO读请求信号和复位信号产生

always @(posedge clk or negedge rst\_n)

if(!rst\_n) lcd\_rfreq1 <= 1'b0;

else if((xcnt >= (VGA\_HST+VGA\_HBP-2)) && (xcnt < (VGA\_HST+VGA\_HBP+640-2))

&& (ycnt >= (VGA\_VST+VGA\_VBP+120)) && (ycnt < (VGA\_VST+VGA\_VBP+VGA\_VVT-120)))

lcd\_rfreq1 <= 1'b1;

else lcd\_rfreq1 <= 1'b0;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) lcd\_rfreq2 <= 1'b0;

else if((xcnt >= (VGA\_HST+VGA\_HBP+640-2)) && (xcnt < (VGA\_HST+VGA\_HBP+VGA\_HVT-2))

&& (ycnt >= (VGA\_VST+VGA\_VBP+120)) && (ycnt < (VGA\_VST+VGA\_VBP+VGA\_VVT-120)))

lcd\_rfreq2 <= 1'b1;

else lcd\_rfreq2 <= 1'b0;

always @(posedge clk or negedge rst\_n)

if(!rst\_n) lcd\_rfclr <= 1'b1;

else if(ycnt == 12'd0) lcd\_rfclr <= 1'b1;

else lcd\_rfclr <= 1'b0;

//-----------------------------------------------------------

//display corlor generate

reg[4:0] vga\_rdb;

reg[5:0] vga\_gdb;

reg[4:0] vga\_bdb;

reg[15:0] lcd\_db\_rgb; // LCD色彩显示寄存器

always @ (posedge clk or negedge rst\_n)

if(!rst\_n) {vga\_rdb,vga\_gdb,vga\_bdb} <= 16'd0;

else if((ycnt >= (VGA\_VST+VGA\_VBP)) && (ycnt < (VGA\_VST+VGA\_VBP+VGA\_VVT)) && (xcnt >= (VGA\_HST+VGA\_HBP-1)) && (xcnt < (VGA\_HST+VGA\_HBP+VGA\_HVT-1))) begin

if((ycnt >= (VGA\_VST+VGA\_VBP+120)) && (ycnt < (VGA\_VST+VGA\_VBP+VGA\_VVT-120))) begin

if((xcnt >= (VGA\_HST+VGA\_HBP-1)) && (xcnt < (VGA\_HST+VGA\_HBP+640-1))) {vga\_rdb,vga\_gdb,vga\_bdb} <= lcd\_rfdb1;

else if((xcnt >= (VGA\_HST+VGA\_HBP+640-1)) && (xcnt < (VGA\_HST+VGA\_HBP+VGA\_HVT-1))) {vga\_rdb,vga\_gdb,vga\_bdb} <= lcd\_rfdb2;

else {vga\_rdb,vga\_gdb,vga\_bdb} <= 16'h0000;

end

else {vga\_rdb,vga\_gdb,vga\_bdb} <= {5'h10,6'h20,5'h10};

end

else {vga\_rdb,vga\_gdb,vga\_bdb} <= 16'd0;

/\*else if((ycnt >= (VGA\_VST+VGA\_VBP+120)) && (ycnt < (VGA\_VST+VGA\_VBP+VGA\_VVT-120))) begin

if((xcnt >= (VGA\_HST+VGA\_HBP-1)) && (xcnt < (VGA\_HST+VGA\_HBP+640-1))) {vga\_rdb,vga\_gdb,vga\_bdb} <= lcd\_rfdb1;

else if((xcnt >= (VGA\_HST+VGA\_HBP+640-1)) && (xcnt < (VGA\_HST+VGA\_HBP+VGA\_HVT-1))) {vga\_rdb,vga\_gdb,vga\_bdb} <= lcd\_rfdb2;

else {vga\_rdb,vga\_gdb,vga\_bdb} <= 16'd0;

end

else {vga\_rdb,vga\_gdb,vga\_bdb} <= 16'd0;\*/

//-----------------------------------------------------------

//corlor data and clock generate

assign vga\_r = vga\_valid ? vga\_rdb:5'd0;

assign vga\_g = vga\_valid ? vga\_gdb:6'd0;

assign vga\_b = vga\_valid ? vga\_bdb:5'd0;

endmodule