차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

Lecture 8: LCD drive AHB Interface, SRAM/BRAM AHB Interface

Xuan-Truong Nguyen

Road map

Review

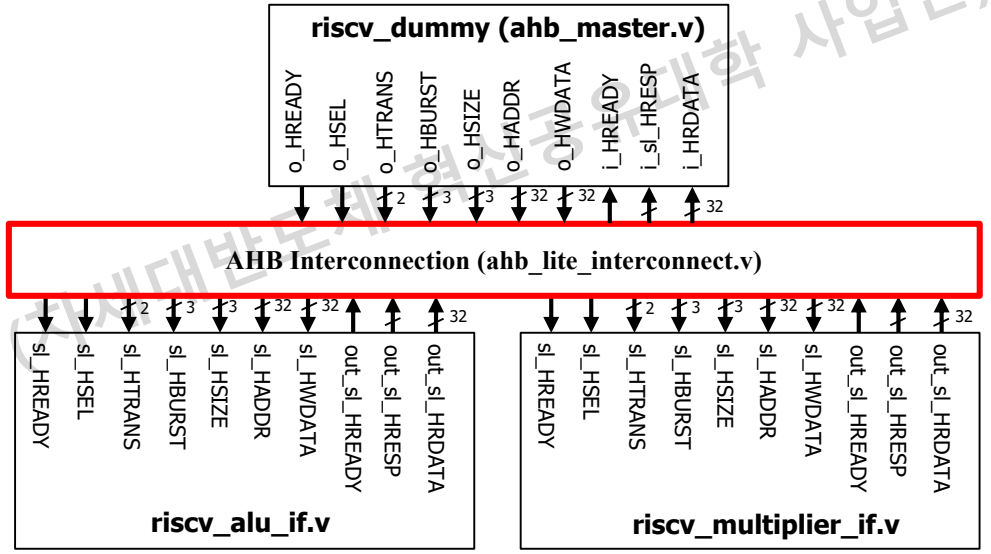
LCD Drive AHB interface

SRAM AHB Interface

Optimized frame buffer

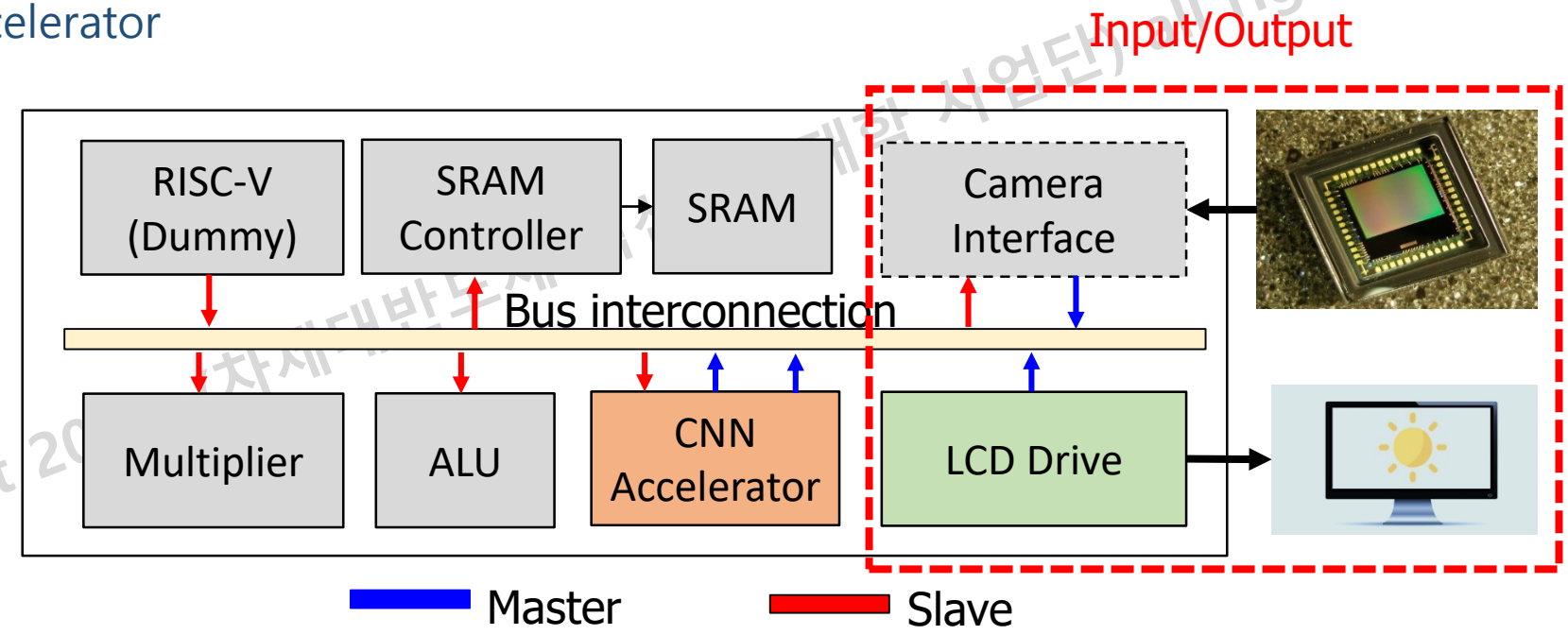
Bus Interconnection (Lecture 6)

- Build a target system that consists of an AHB master and two AHB slaves
  - Master: RISC-V dummy
  - Slaves: ALU and multiplier
- Masters and slaves are connected by AHB interconnect.



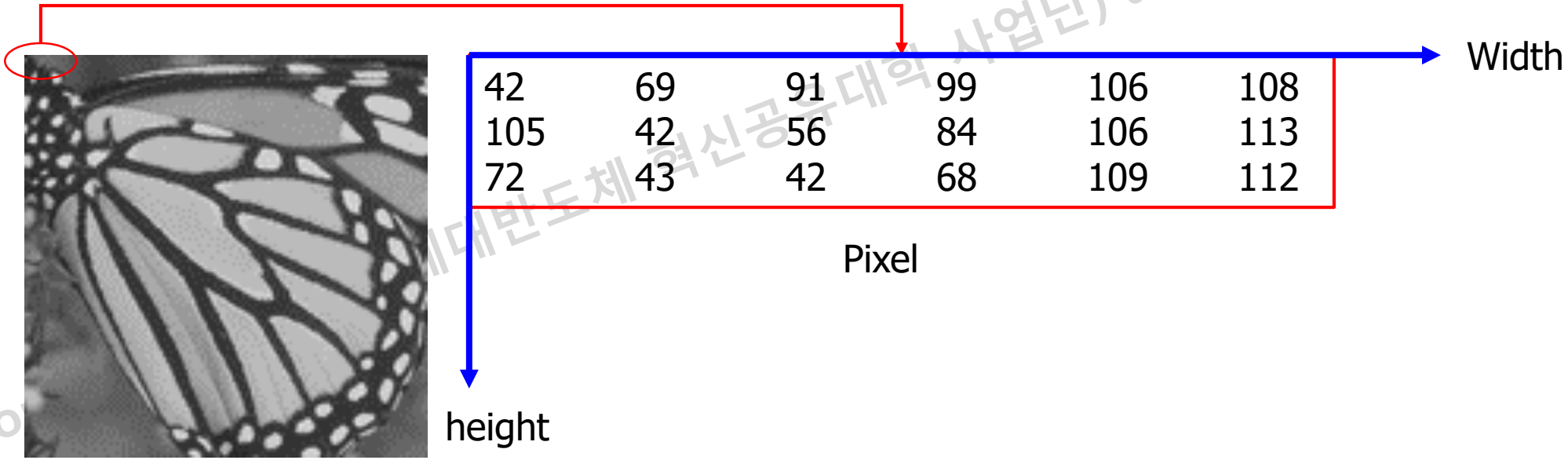
System block diagram (Lecture 7)

- System with a bus architecture:
  - Master (RISC-V), Slave Memory: SRAM
  - Custom IPs: Multiplier, ALU (single transfer)
  - Input: Camera Interface → Assume that an image is stored in memory
  - Output: LCD Drive
  - CNN accelerator



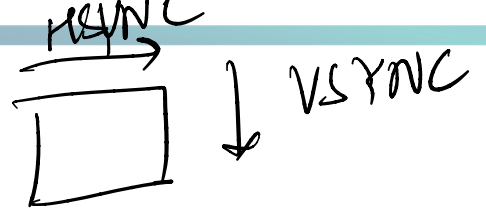
Input/output: Image

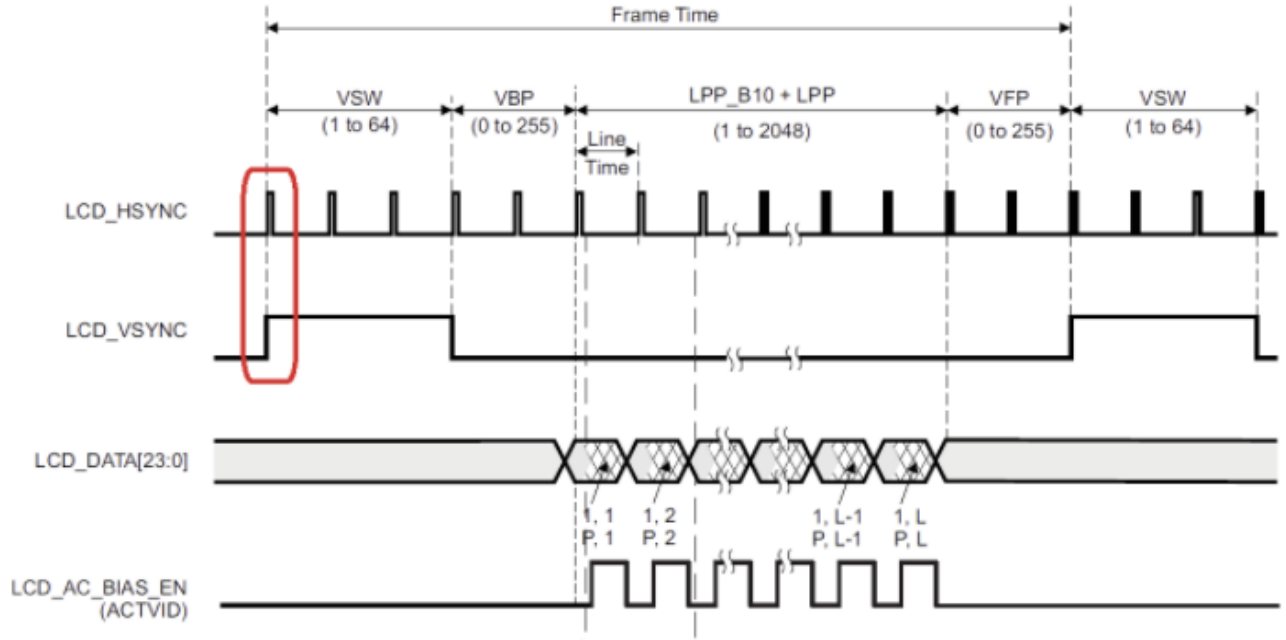
- What is an image?
  - "An image is an artifact that **depicts visual perception**, such as a photograph or a 2D picture that resembles a subject—usually a physical object—and thus provides a depiction of it.", **Wikipedia**.
- In the context of signal processing, an image is a distributed amplitude of color(s)



Output: LCD drive interface signals

- LCD_CLK: Clock
- LCD_VSYNC: the *frame* synchronization signal, manages *vertical* scanning, and acts as the frame update strobe.
- LCD_HSYNC: the *line* synchronization signal, manages *horizontal* line scanning, and acts as the line display strobe.
- LCD_DE: the DE signal indicates to the LCD-TFT that the data in the RGB bus is valid and must be latched to be drawn.
- Pixel RGB: The LTDC interface can be configured to output more than one color depth.

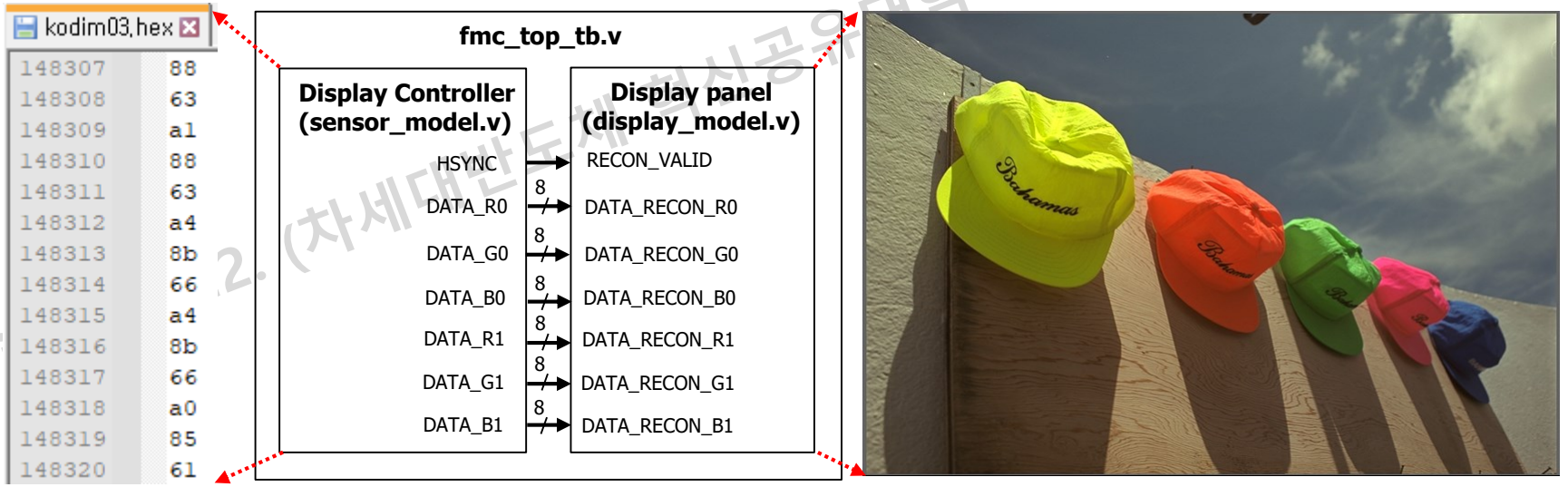




Note: In the context of this lab, HSYNC is used as a valid signal (DE)*

LCD drive

- Sensor model
  - Load a hexadecimal file, i.e. /img/kodim03.hex
  - Generate signals: VSYNC, HSYNC, RGB data (two pixels)
- Display panel
  - Capture signals from controller and display them.

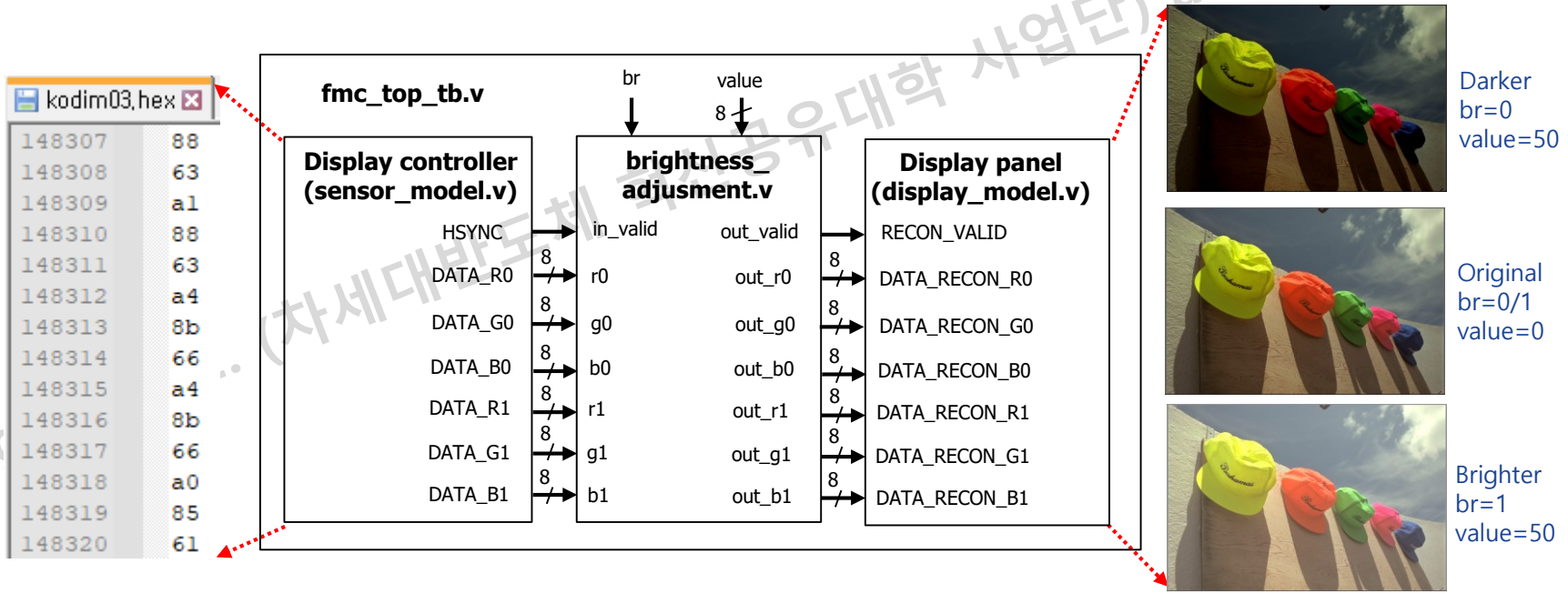


LCD drive: Brightness adjustment

- Baseline: Sensor model+ display panel

⇒ Control the brightness of the display panel.

- Two modes:
  - Brighter: Increase the pixel values (br=1, value)
  - Darker: Decrease the pixel values (br=0, value)



Reversible color transform (RCT)

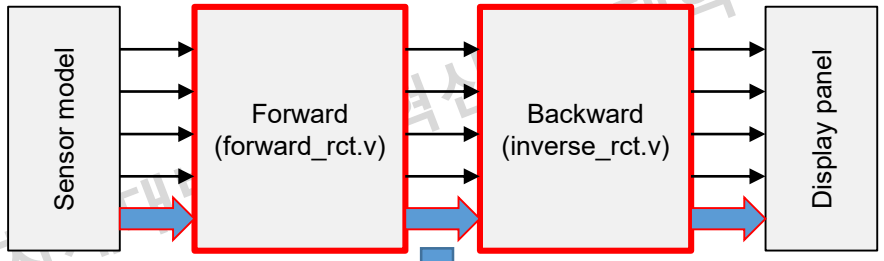
- Implement a reversible color transform (RCT)
  - Color channels are usually similar
  - In image compression, RCT is used to convert between two color domains
    - Forward RCT: RGB to YCbCr
    - Backward RCT: YCbCr to RGB

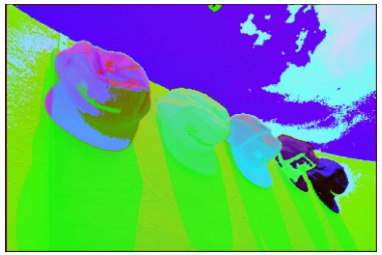
4:2:2 or 4:2:0 not this one!

⇒ Store pixels in the (Y0, Y1, Cb0, Cr0) instead of (R0, G0, B0, R1, G1, B1)

32 bit vs 48 bit

compressed image (less data)





Lecture plan

- Today we will
  - Connect an LCD drive to the system.
  - Utilize the forward and back color transform modules.
- Labs
  - Lab 1: Make an AHB interface of the LCD drive.
  - Lab 2: Make a memory module with an AHB interface
  - Lab 3: Frame memory compression

Road map

Review

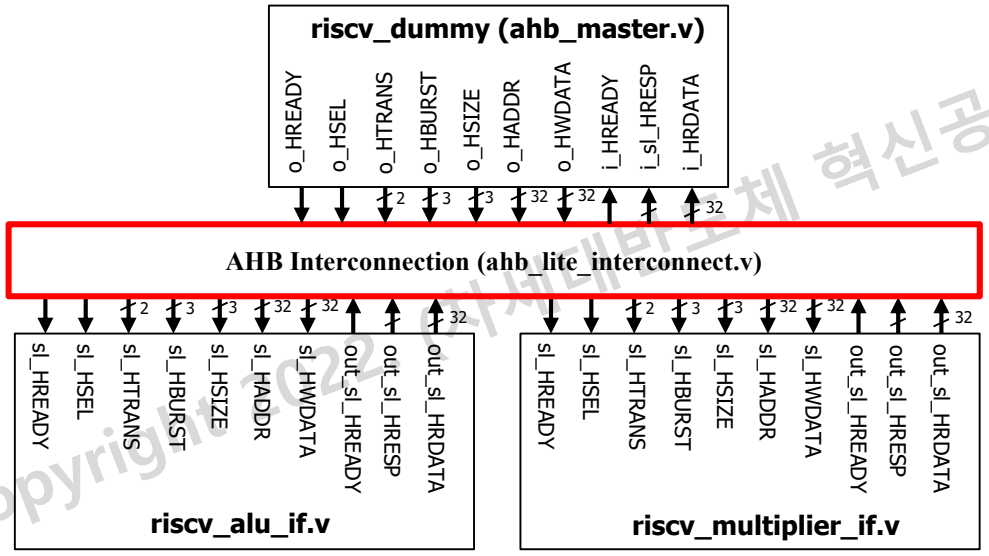
LCD Drive AHB interface

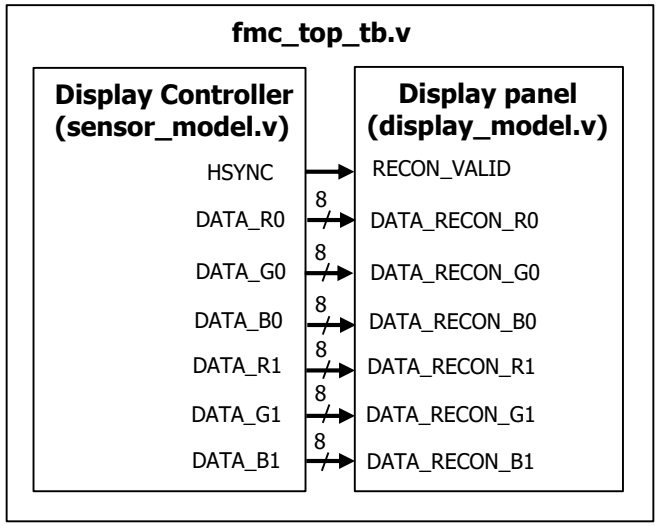
SRAM AHB Interface

Optimized frame buffer

Motivation

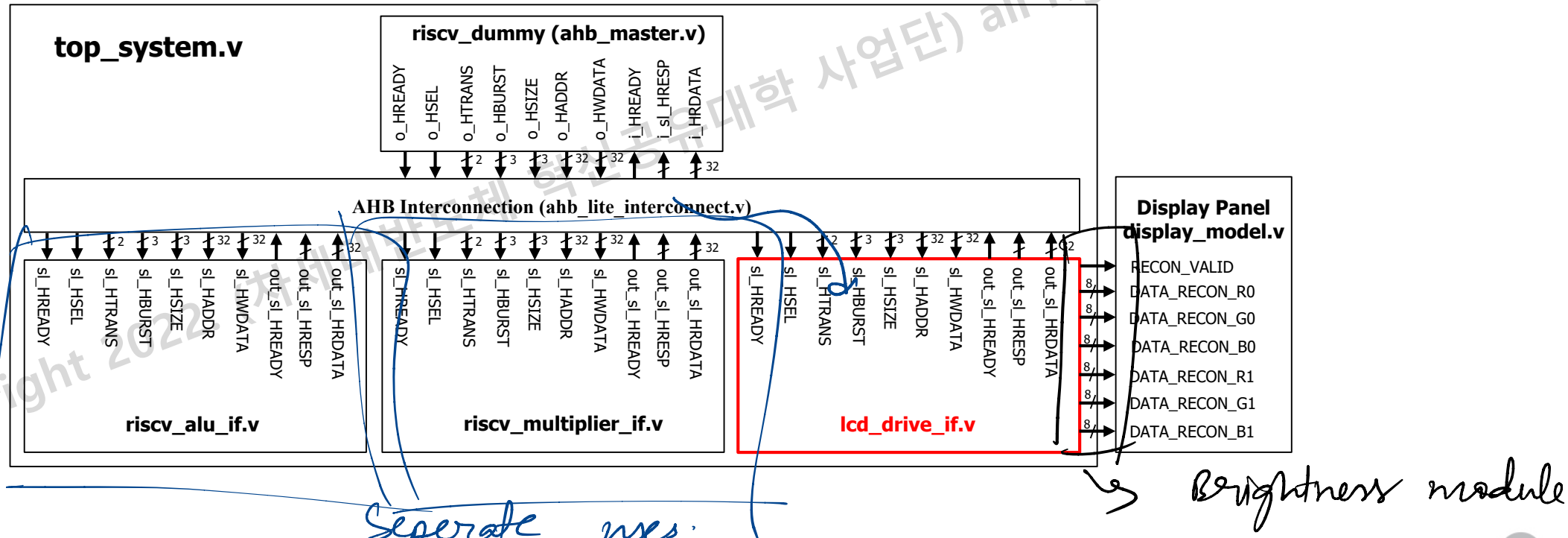
- Two separate codes
  - A system with one master and two slaves connected by a bus interconnect.
  - A display model and a display panel
- How to connect both of them?





Lab 1: LCD drive's AHB interface

- Build the target system that consists of an AHB master and three AHB slaves
  - Master: RISC-V dummy
  - Slaves: ALU, multiplier, and LCD drive
- Masters and slaves are connected by an AHB Interconnect.

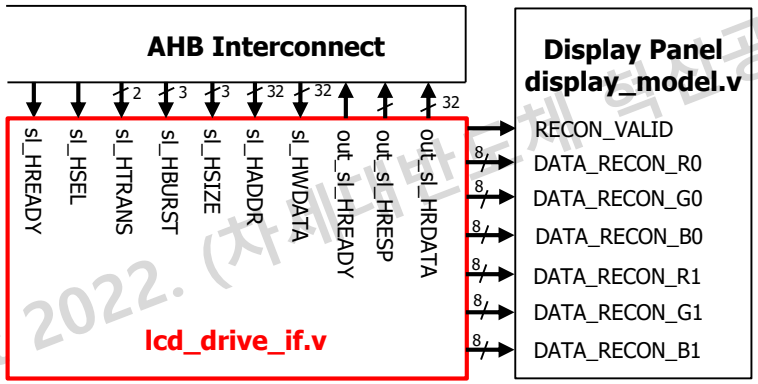


How to build and test the target system?

- Procedure
  1. Create a custom IP (lcd_drive_if.v) and add it into the top system
     - Create the AHB slave interface of LCD drive (lcd_drive_if.v).
     - Add its base address and the register map (map.v).
     - Add the LCD drive IP to the Bus interconnect (top_system.v).
  2. Generate data signals (lcd_drive_if.v)
     - Generate vsync, hsync, r0, b0, g0, r1, g1, b1.
  3. Test the system
     - Baseline mode
     - Brightness adjustment

Custom IP (lcd_drive_if.v)

- How to build an AHB interface of LCD drive?
- Input/output signals
  - AHB slave signals.
  - LCD drive signals




```
module lcd_drive_if #(
    parameter W_ADDR = 32,
    parameter W_DATA = 32,
    parameter WB_DATA = 4,
    parameter W_WB_DATA = 2,
    parameter W_CNT = 16,
    parameter DEF_HPROT = {`PROT_NOTCACHE,
    `PROT_UNBUF, `PROT_USER, `PROT_DATA},
    parameter IMG_PIX_W = 8)
(
    //CLOCK
    HCLK,
    HRESETn,
    //input signals of control port(slave)
    sl_HREADY,
    sl_HSEL,
    sl_HTRANS,
    sl_HBURST,
    sl_HSIZE,
    sl_HADDR,
    sl_HWRITE,
    sl_HWDATA,
    //output signals of control port(slave)
    out_sl_HREADY,
    out_sl_HRESP,
    out_sl_HRDATA,
    //LCD Drive
    out_valid,
    out_r0, out_g0, out_b0, out_r1, out_g1, out_b1
);
```

Custom IP (lcd_drive_if.v)

- Construct registers and register map based on parameters
  - Width, height, delays, frame size, brightness mode and value, start.

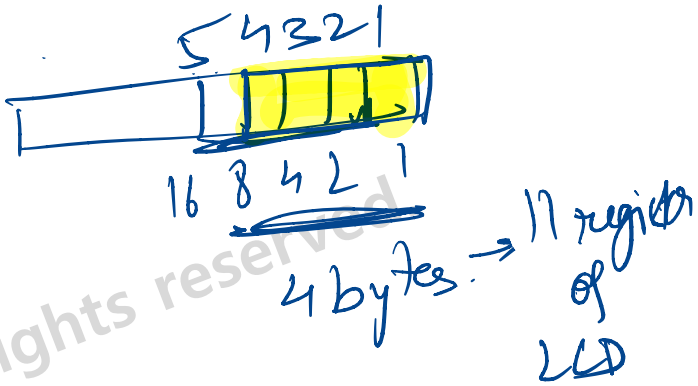
```
parameter WIDTH      = 768,
          HEIGHT     = 512,
          START_UP_DELAY = 100,
          VSYNC_CYCLE = 3,
          VSYNC_DELAY = 3,
          HSYNC_DELAY = 160,
          FRAME_TRANS_DELAY = 200,
          DATA_COUNT = WIDTH * HEIGHT/2;
localparam W_SIZE   = 12;
localparam W_FRAME_SI    = 2 * W_SIZE + 1;
localparam W_DELAY =
```

```
localparam N_REGS = 11;
localparam W_REGS = 4;

localparam LCD_DRIVE_WIDTH              = 0;
localparam LCD_DRIVE_HEIGHT             = 1;
localparam LCD_DRIVE_START_UP_DELAY     = 2;
localparam LCD_DRIVE_VSYNC_CYCLE        = 3;
localparam LCD_DRIVE_VSYNC_DELAY        = 4;
localparam LCD_DRIVE_HSYNC_DELAY        = 5;
localparam LCD_DRIVE_FRAME_TRANS_DELAY  = 6;
localparam LCD_DRIVE_DATA_COUNT         = 7;
localparam LCD_DRIVE_START              = 8;
localparam LCD_DRIVE_BR_MODE            = 9;
localparam LCD_DRIVE_BR_VALUE           = 10;
```

```
reg [W_REGS-1:0] q_sel_sl_reg;
reg q_ld_sl_reg;

reg [W_SIZE-1 :0] q_width;
reg [W_SIZE-1 :0] q_height;
reg [W_DELAY-1:0] q_start_up_delay;
reg [W_DELAY-1:0] q_vsync_cycle;
reg [W_DELAY-1:0] q_vsync_delay;
reg [W_DELAY-1:0] q_hsync_delay;
reg [W_DELAY-1:0] q_frame_trans_delay;
reg [W_FRAME_SIZE-1:0] q_data_count;
reg q_br_mode;
reg [IMG_PIX_W-1:0] q_br_value;
reg q_start;
```

Address decoder (lcd_drive_if.v)

- At each AHB slave IP, use an OFFSET to access a register
  - Offset is 4 bytes, so it ignores W_WB_DATA LSB bits
  - Access all registers, so it needs W_REGS bits.
    - Example: W_REGS = 4 to access 11 registers at LCD Drive




```
//------------------------------------------------------------
// Decode Stage: Address Phase
//------------------------------------------------------------
always @(posedge HCLK or negedge HRESETn)
begin
    if(~HRESETn)
    begin
        //control
        q_sel_sl_reg <= 0;
        q_ld_sl_reg <= 1'b0;
    end
    else begin
        if(sl_HSEL && sl_HREADY && ((sl_HTRANS == `TRANS_NONSEQ) |
        begin
            q_sel_sl_reg <= sl_HADDR[W_REGS+W_WB_DATA-1:W_WB_DATA];
            q_ld_sl_reg <= sl_HWRITE;
        end
        else begin
            q_ld_sl_reg <= 1'b0;
        end
    end
end
```

4+1-1:1

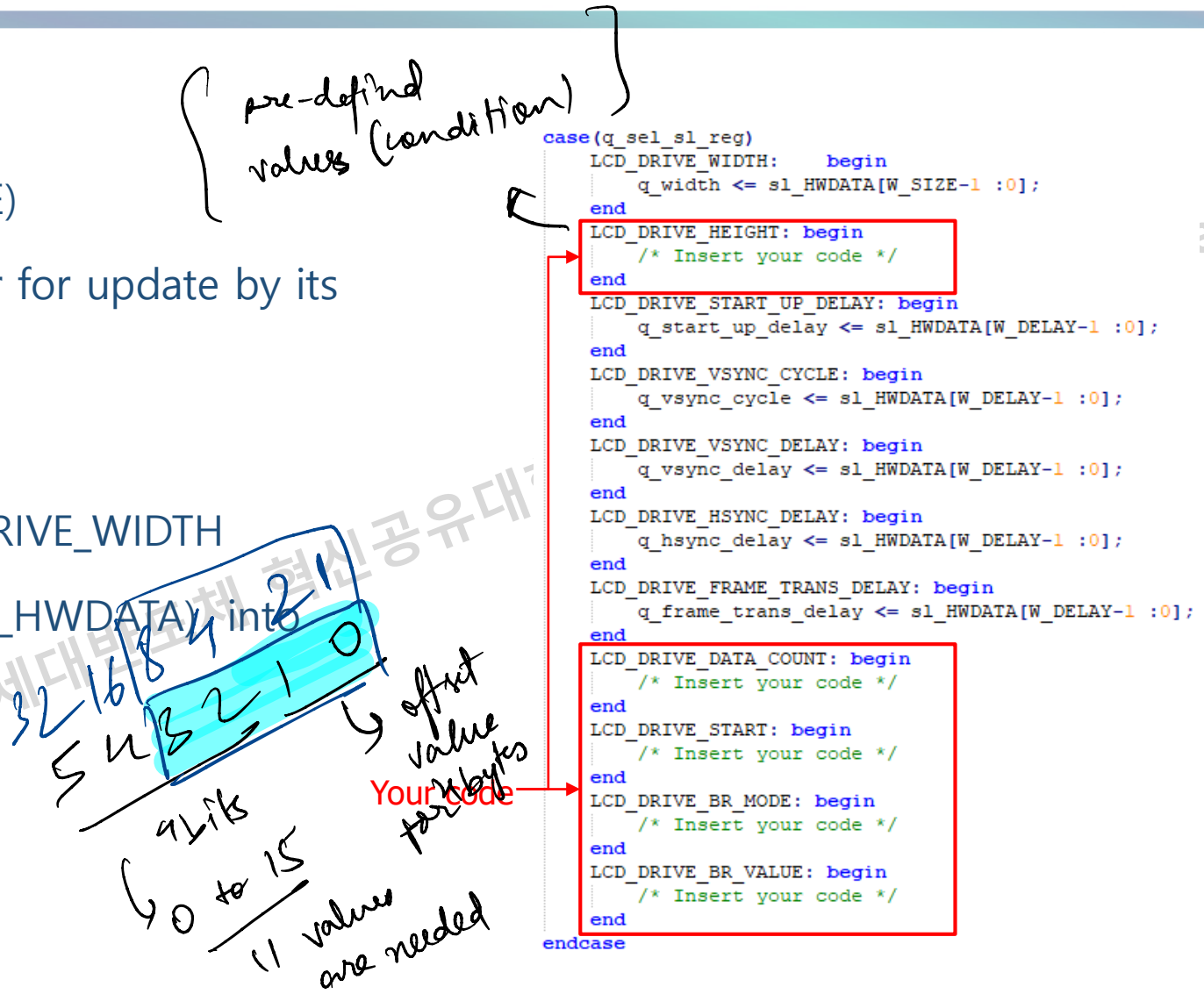
4:1

```
localparam N_REGS = 11;
localparam W_REGS = 4;

localparam LCD_DRIVE_WIDTH              = 0;
localparam LCD_DRIVE_HEIGHT             = 1;
localparam LCD_DRIVE_START_UP_DELAY     = 2;
localparam LCD_DRIVE_VSYNC_CYCLE        = 3;
localparam LCD_DRIVE_VSYNC_DELAY        = 4;
localparam LCD_DRIVE_HSYNC_DELAY        = 5;
localparam LCD_DRIVE_FRAME_TRANS_DELAY  = 6;
localparam LCD_DRIVE_DATA_COUNT         = 7;
localparam LCD_DRIVE_START              = 8;
localparam LCD_DRIVE_BR_MODE            = 9;
localparam LCD_DRIVE_BR_VALUE           = 10;
```

Data phase: Write operation

- Write operation
  - q_ld_sl_reg = 1 (WRITE)
  - Select a target register for update by its address (q_sel_sl_reg)
- For example

  If q_sel_sl_reg == LCD_DRIVE_WIDTH
  - Copy an input data (sl_HWDATA) into the register q_width



Data phase: Read operation

- Read operation
  - Select a target register for update by its address (q_sel_sl_reg)
- For example
  - If q_sel_sl_reg == LCD_DRIVE_WIDTH
    - Copy q_width to out_sl_HRDATA

```
always @*
begin:rdata
    out_sl_HRDATA = 32'h0;
    case(q_sel_sl_reg)
        LCD_DRIVE_WIDTH:   begin
            out_sl_HRDATA = q_width;
        end
        LCD_DRIVE_HEIGHT: begin
            /* Insert your code */
        end
        LCD_DRIVE_START_UP_DELAY: begin
            out_sl_HRDATA = q_start_up_delay;
        end
        LCD_DRIVE_VSYNC_CYCLE: begin
            out_sl_HRDATA = q_vsync_cycle;
        end
        LCD_DRIVE_VSYNC_DELAY: begin
            out_sl_HRDATA = q_vsync_delay;
        end
        LCD_DRIVE_HSYNC_DELAY: begin
            out_sl_HRDATA = q_hsync_delay;
        end
        LCD_DRIVE_FRAME_TRANS_DELAY: begin
            out_sl_HRDATA = q_frame_trans_delay;
        end
        LCD_DRIVE_DATA_COUNT: begin
            /* Insert your code */
        end
        LCD_DRIVE_START: begin
            /* Insert your code */
        end
        LCD_DRIVE_BR_MODE: begin
            /* Insert your code */
        end
        LCD_DRIVE_BR_VALUE: begin
            /* Insert your code */
        end
    endcase
end
```

Your code

Map (map.v)

- IP addressing
  - ALU
  - Multiplier
  - LCD drive
- Register maps of Slaves are visible to Masters

map.v

```
`define RISCV_ALU_BASE_ADDR          32'hE000_0000
`define RISCV_MULTIPLIER_BASE_ADDR   32'hE000_1000
`define RISCV_LCD_DRIVE_BASE_ADDR    32'hE100_0000

`define RISCV_MASK_ALU_BASE_ADDR         32'hFFFF_F000
`define RISCV_MASK_MULTIPLIER_BASE_ADDR  32'hFFFF_F000
`define RISCV_MASK_LCD_DRIVE_BASE_ADDR   32'hFF00_0000
//------------------------------------------------------------
`define RISCV_REG_ALU_OP_I   (`RISCV_ALU_BASE_ADDR + 32'h00)
`define RISCV_REG_ALU_A_I    (`RISCV_ALU_BASE_ADDR + 32'h04)
`define RISCV_REG_ALU_B_I    (`RISCV_ALU_BASE_ADDR + 32'h08)
`define RISCV_REG_ALU_P_O    (`RISCV_ALU_BASE_ADDR + 32'h0C)

//------------------------------------------------------------
`define RISCV_REG_MUL_OP_I       (`RISCV_MULTIPLIER_BASE_ADDR + 32'h00)
`define RISCV_REG_MUL_A_I        (`RISCV_MULTIPLIER_BASE_ADDR + 32'h04)
`define RISCV_REG_MUL_B_I        (`RISCV_MULTIPLIER_BASE_ADDR + 32'h08)
`define RISCV_REG_MUL_A_SIGNED   (`RISCV_MULTIPLIER_BASE_ADDR + 32'h0C)
`define RISCV_REG_MUL_B_SIGNED   (`RISCV_MULTIPLIER_BASE_ADDR + 32'h10)
`define RISCV_REG_MUL_P_O_LOW    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h14)
`define RISCV_REG_MUL_P_O_HIGH   (`RISCV_MULTIPLIER_BASE_ADDR + 32'h18)
`define RISCV_REG_MUL_STALL_W    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h1C)

//------------------------------------------------------------
`define LCD_DRIVE_WIDTH              (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h00)
`define LCD_DRIVE_HEIGHT             (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h04)
`define LCD_DRIVE_START_UP_DELAY     (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h08)
`define LCD_DRIVE_VSYNC_CYCLE        (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h0C)
`define LCD_DRIVE_VSYNC_DELAY        (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h10)
`define LCD_DRIVE_HSYNC_DELAY        (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h14)
`define LCD_DRIVE_FRAME_TRANS_DELAY  (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h18)
`define LCD_DRIVE_DATA_COUNT         (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h1C)
`define LCD_DRIVE_START              (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h20)
`define LCD_DRIVE_BR_MODE            (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h24)
`define LCD_DRIVE_BR_VALUE           (`RISCV_LCD_DRIVE_BASE_ADDR + 32'h28)
```

lcd_drive_if.v

```
reg [W_REGS-1:0] q_sel_sl_reg;
reg q_ld_sl_reg;

reg [W_SIZE-1 :0] q_width;
reg [W_SIZE-1 :0] q_height;
reg [W_DELAY-1:0] q_start_up_delay;
reg [W_DELAY-1:0] q_vsync_cycle;
reg [W_DELAY-1:0] q_vsync_delay;
reg [W_DELAY-1:0] q_hsync_delay;
reg [W_DELAY-1:0] q_frame_trans_delay;
reg [W_FRAME_SIZE-1:0] q_data_count;
reg q_br_mode;
reg [IMG_PIX_W-1:0] q_br_value;
reg q_start;
```

Top system (top_system.v)

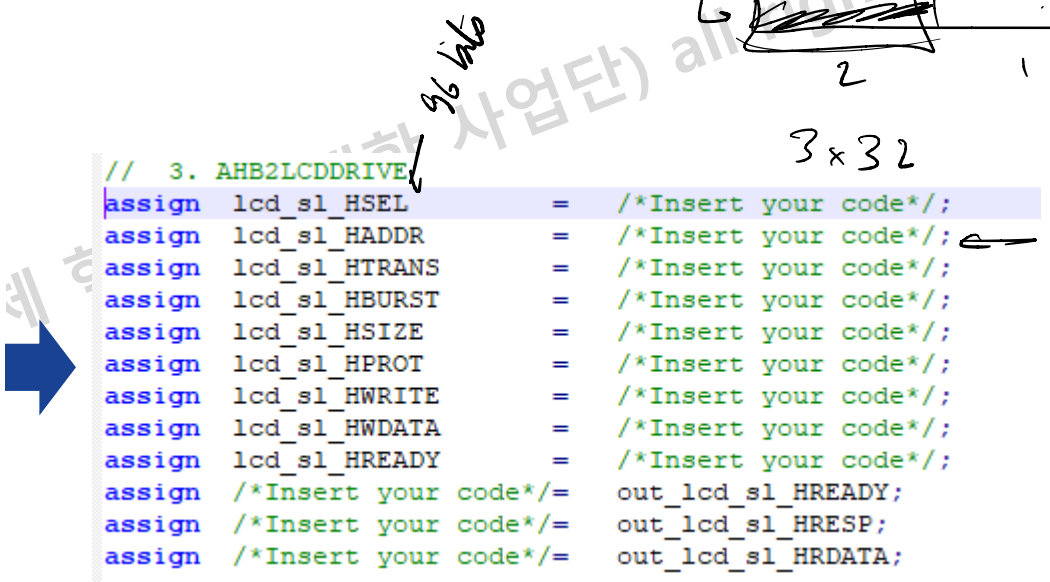
- Create the AHB slave interface of LCD drive (lcd_drive_if.v).
- Add its base address and the register map (map.v).
- Add the LCD drive IP to the bus interconnect (top_system.v).
  - Three slaves

```
parameter N_MASTER  =   1;
parameter W_MASTER  =   $clog2(N_MASTER); //GetBitWidth(N_MASTER);
parameter N_SLAVE   =   3;
parameter W_SLAVE   =   $clog2(N_SLAVE); //GetBitWidth(N_SLAVE);

parameter ADDR_START_MAP = {
        `RISCV_LCD_DRIVE_BASE_ADDR,         // 2
        `RISCV_MULTIPLIER_BASE_ADDR,        // 1
        `RISCV_ALU_BASE_ADDR                // 0
    };

parameter ADDR_END_MAP   = {
        `RISCV_LCD_DRIVE_BASE_ADDR,         // 2
        `RISCV_MULTIPLIER_BASE_ADDR,        // 1
        `RISCV_ALU_BASE_ADDR                // 0
    };


parameter ADDR_MASK = {
        `RISCV_MASK_LCD_DRIVE_BASE_ADDR,    // 2
        `RISCV_MASK_MULTIPLIER_BASE_ADDR,   // 1
        `RISCV_MASK_ALU_BASE_ADDR           // 0
    };
```

Top system (top_system.v)

- Connect a slave' ports to AHB Interconnect
  - Ports from a slave are connected to specified ports of AHB bus.
- Completing the connections between LCD drive and AHB bus

```
// LCD Drive
lcd_drive_if u_lcd_drive_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(lcd_sl_HREADY),
    .sl_HSEL(lcd_sl_HSEL),
    .sl_HTRANS(lcd_sl_HTRANS),
    .sl_HBURST(lcd_sl_HBURST),
    .sl_HSIZE(lcd_sl_HSIZE),
    .sl_HADDR(lcd_sl_HADDR),
    .sl_HWRITE(lcd_sl_HWRITE),
    .sl_HWDATA(lcd_sl_HWDATA),
    .out_sl_HREADY(out_lcd_sl_HREADY),
    .out_sl_HRESP( out_lcd_sl_HRESP),
    .out_sl_HRDATA(out_lcd_sl_HRDATA),
    .out_valid(out_valid),
    .out_r0(out_r0),
    .out_g0(out_g0),
    .out_b0(out_b0),
    .out_r1(out_r1),
    .out_g1(out_g1),
    .out_b1(out_b1)
    );
```



```
//  3. AHB2LCDDRIVE
assign  lcd_sl_HSEL         =   /*Insert your code*/;
assign  lcd_sl_HADDR        =   /*Insert your code*/;
assign  lcd_sl_HTRANS       =   /*Insert your code*/;
assign  lcd_sl_HBURST       =   /*Insert your code*/;
assign  lcd_sl_HSIZE        =   /*Insert your code*/;
assign  lcd_sl_HPROT        =   /*Insert your code*/;
assign  lcd_sl_HWRITE       =   /*Insert your code*/;
assign  lcd_sl_HWDATA       =   /*Insert your code*/;
assign  lcd_sl_HREADY       =   /*Insert your code*/;
assign  /*Insert your code*/=   out_lcd_sl_HREADY;
assign  /*Insert your code*/=   out_lcd_sl_HRESP;
assign  /*Insert your code*/=   out_lcd_sl_HRDATA;
```

Test bench (top_system_tb.v)

- In the test bench, the top system is connected to the display panel.

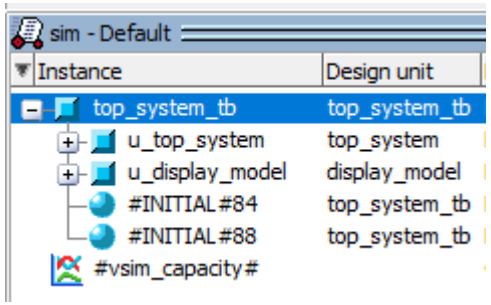
```
`timescale 1ns / 100ps
module top_system_tb;
        reg HCLK, HRESETn;                                  //**** module input
        reg [3:0]           alu_op_i;                       //**** control signals
        reg [31:0]          alu_a_i, alu_b_i;
        reg [31:0]          alu_p_o;
        wire out_valid;
        wire [IMG_PIX_W-1:0] out_r0, out_g0, out_b0, out_r1, out_g1, out_b1;
```

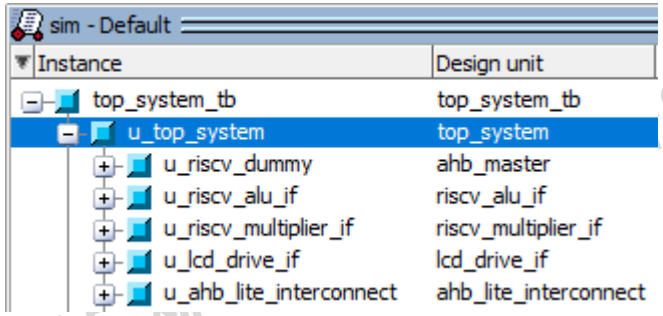
```
top_system
u_top_system (
     .HRESETn(HRESETn)
    ,.HCLK    (HCLK)
    ,/*output*/ .out_valid(out_valid)
    ,/*output [IMG_PIX_W-1:0]*/ .out_r0(out_r0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_g0(out_g0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_b0(out_b0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_r1(out_r1)
    ,/*output [IMG_PIX_W-1:0]*/ .out_g1(out_g1)
    ,/*output [IMG_PIX_W-1:0]*/ .out_b1(out_b1)
);
```

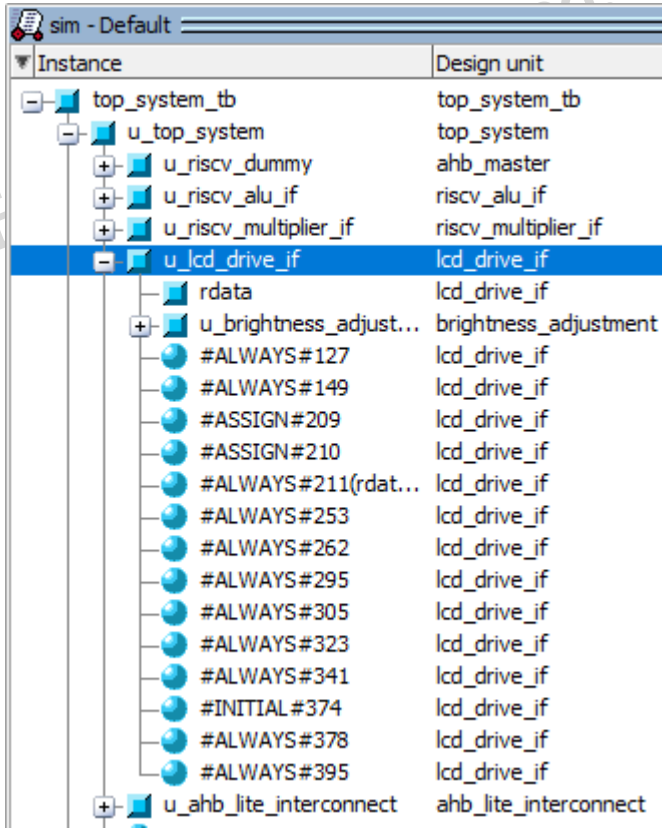
```
display_model #(.INFILE(`OUTPUTFILENAME))
u_display_model(
    ./*input */HCLK(HCLK),
    ./*input */HRESETn(HRESETn),
    ./*input */RECON_VALID(out_valid),
    ./*input [7:0]  */DATA_RECON_R0(out_r0),
    ./*input [7:0]  */DATA_RECON_G0(out_g0),
    ./*input [7:0]  */DATA_RECON_B0(out_b0),
    ./*input [7:0]  */DATA_RECON_R1(out_r1),
    ./*input [7:0]  */DATA_RECON_G1(out_g1),
    ./*input [7:0]  */DATA_RECON_B1(out_b1),
    ./*output  */DEC_DONE()
);
```

Test bench (top_system_tb.v)

- Test bench: the top system is connected to a display panel
- The top system includes: a master, ALU, multiplier, LCD drive, and bus interconnect
- LCD drive interface
  - Brightness adjustment







Test cases

- Master accesses ALU
  - Smaller-Less-Than (SLT) and addition (ADD) requests.

Select ALU &
AHB read and write

```
#(8*p)
        //sl_HSEL_alu = 1'b1;
        //sl_HSEL_multiplier = 1'b0;
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);     // Read the result

#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O, alu_p_o);     // Read the result
```

Test cases

- The Master accesses Multiplier
  - A multiplication (MUL) request.

```
    #(8*p)
            //sl_HSEL_alu = 1'b0;                                   Select multiplier &
            //sl_HSEL_multiplier = 1'b1;                            AHB read and write
    #(8*p)
            alu_a_i = 32'h7;
            alu_b_i = 32'h9;
            alu_op_i = `ALU_MULL;
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i     ); // Write the first operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i     ); // Write the second operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i); // Write the first operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i); // Write the second operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );    // Write the operation
    #(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o ); // Read the result

  end
```

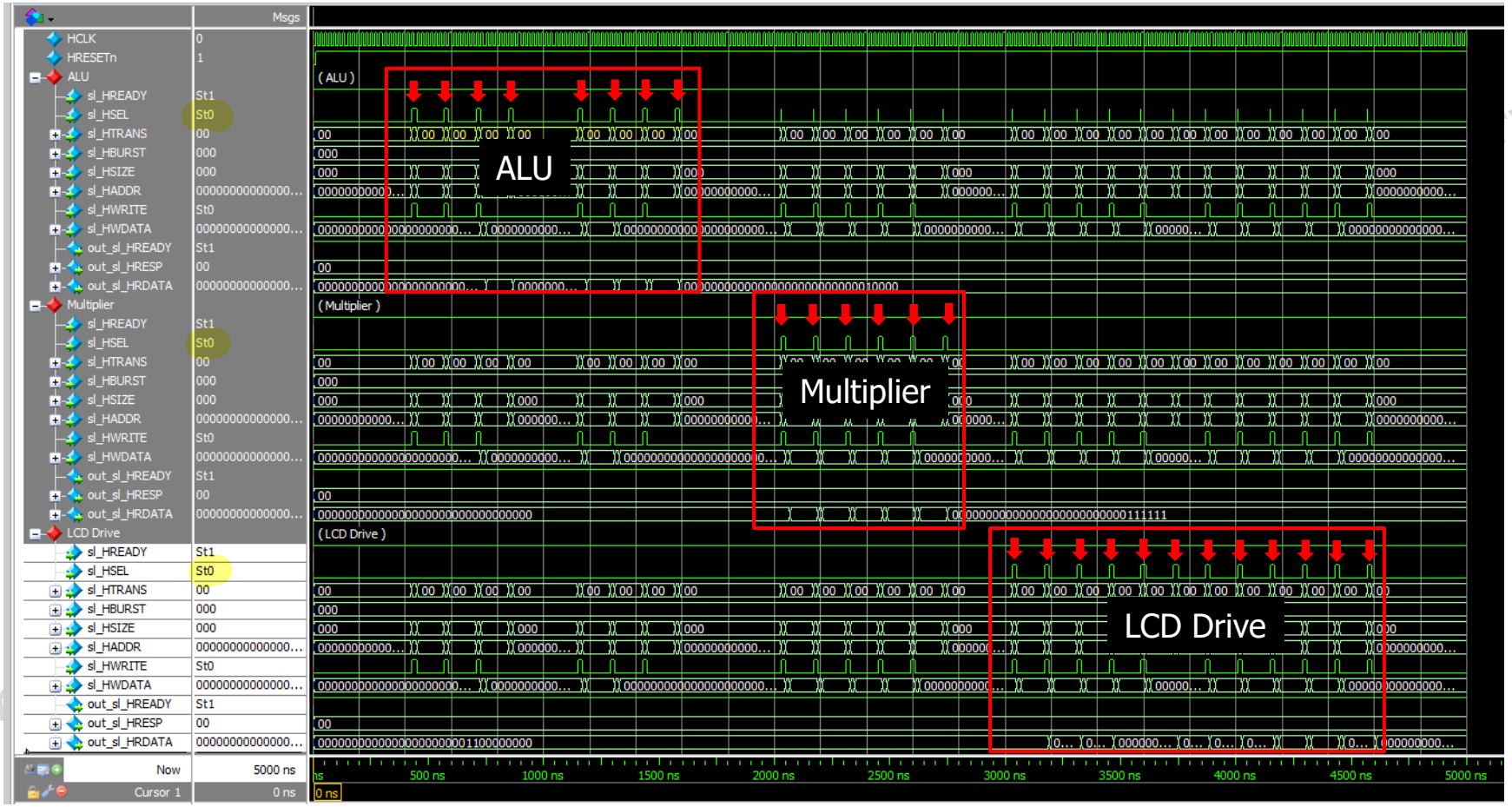
Test cases

- Configure registers of LCD drive
  - Width, height, frame size, data count
  - Vertical and horizontal synchronization delays.
  - Brightness mode and adjustment value
- After setting configuration registers of LCD drive, RISC-V starts a frame.

```
#(8*p)
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_WIDTH , q_width );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_HEIGHT , q_height );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START_UP_DELAY, q_start_up_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_VSYNC_CYCLE, q_vsync_cycle );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_VSYNC_DELAY, q_vsync_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite( `LCD_DRIVE_HSYNC_DELAY, q_hsync_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_FRAME_TRANS_DELAY, q_frame_trans_delay);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_DATA_COUNT, q_data_count );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_BR_MODE, q_br_mode);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_BR_VALUE, q_br_value );
// Start a frame
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START, 1'b1);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START, 1'b0);
```

Waveform

- Do a simulation with time = 5,000 ns



How to build and test the target system?

- Procedure
  1. Create a custom IP (lcd_drive_if.v) and add it into the top system
     - Create the AHB slave interface of LCD drive (lcd_drive_if.v).
     - Add its base address and the register map (map.v).
     - Add the LCD drive IP to the Bus interconnect (top_system.v).
  2. Generate data signals (lcd_drive_if.v)
     - Generate vsync, hsync, r0, b0, g0, r1, g1, b1.
  3. Test the system
     - Baseline mode
     - Brightness adjustment

TODO: LCD drive signals (lcd_drive_if.v)

- Generate data signals vsync, hsync, r0, b0, g0, r1, g1, b1.
  - Complete codes related to FSM by using configuration registers.

```
always @(*) begin
    case(cstate)
    ST_IDLE: begin
                //if(/*Insert your code*/)
                //    nstate = ST_VSYNC;
                //else
                //    nstate = ST_IDLE;
        end
        ST_VSYNC: begin
                if(ctrl_vsync_cnt == q_start_up_delay)
                    nstate = ST_HSYNC;
                else
                    nstate = ST_VSYNC;
        end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /*Insert your code*/)
                //    nstate = ST_DATA;
                //else
                //    nstate = ST_HSYNC;
        end
        ST_DATA: begin
                if(end_frame)     //end of frame
                    nstate = ST_IDLE;
                else begin
                    //if(col == /*Insert your code*/)//end of line
                    //    nstate = ST_HSYNC;
                    //else
                    //    nstate = ST_DATA;
                end
        end
        default: nstate = ST_IDLE;
    endcase
end
```

```
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        row <= 0;
        col <= 0;
    end
    else begin
        if(ctrl_data_run) begin
            if(col == q_width - 2) begin
                row <= row + 1;
            end
            //if(col == /*Insert your code*/)
            //  col <= 0;
            //else
            //  col <= col + 2;
        end
    end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        data_count <= 0;
    end
    else begin
        if(ctrl_data_run)
            data_count <= data_count + 1;
    end
end
//assign end_frame = (data_count == /*Insert your code*/)? 1'b1: 1'b0;
```

Finite State Machine

- Update the current state (cstate) by the next state (nstate)
  - Sequential logic
- Decide the next state based on the current state and other conditions.
  - Combinational logic

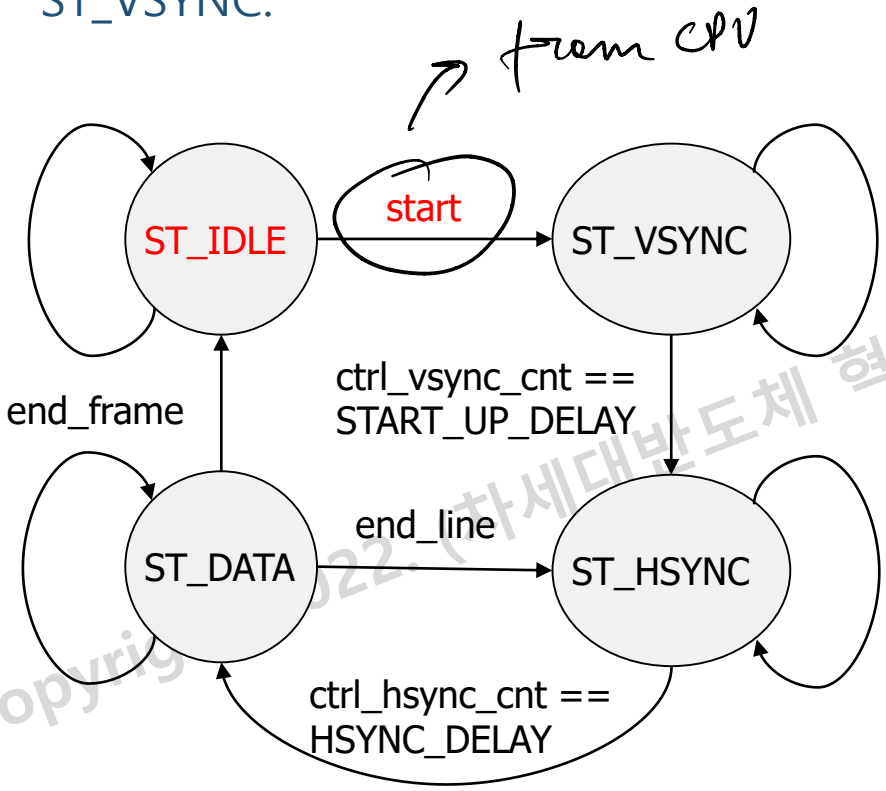
```
always @ (posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        cstate <= ST_IDLE;
    end
    else begin
        cstate <= nstate;
    end
end

always @ (*) begin
    case(cstate)
        ST_IDLE: begin
                if(start)
                    nstate = ST_VSYNC;
                else
                    nstate = ST_IDLE;
        end
        ...
        default: nstate = ST_IDLE;
    endcase
end
```

Update the current state (cstate) by the next state (nstate)

Decide the next state based on the current state and other conditions.

Finite State Machine

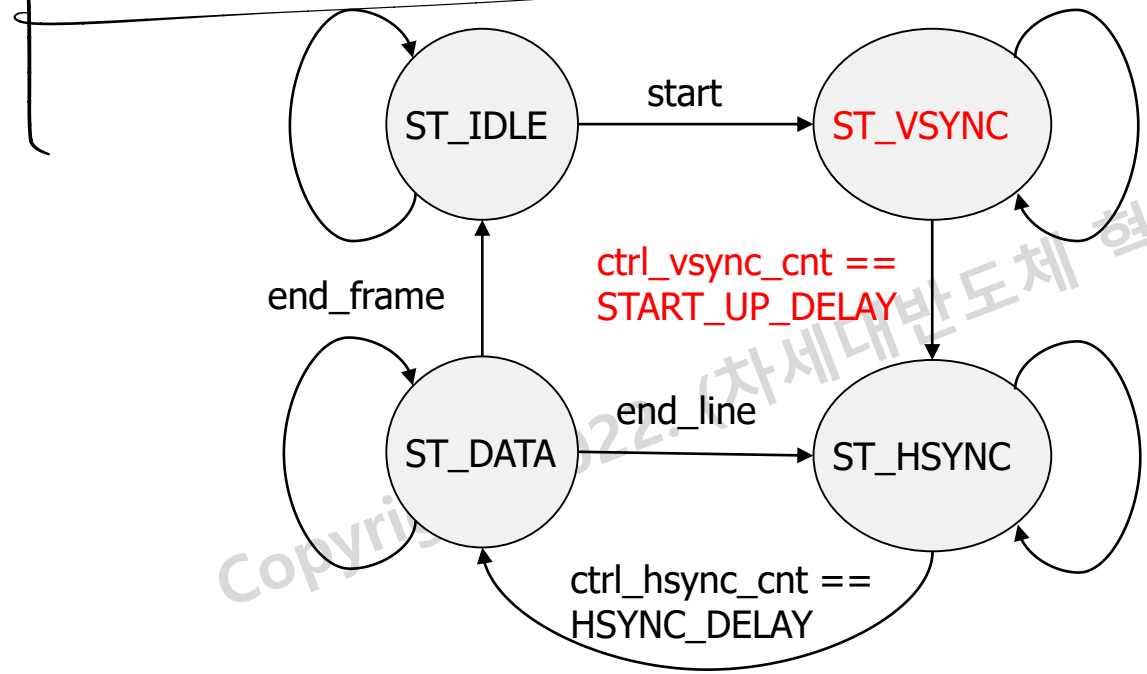
- ST_IDLE:
  - FSM is initialized at ST_IDLE
  - When "start" goes HIGH, FSM moves to ST_VSYNC.




```
always @(*) begin
    case(cstate)
        ST_IDLE: begin
                if(start)
                    nstate = ST_VSYNC;
                else
                    nstate = ST_IDLE;
            end
        ST_VSYNC: begin
                if(ctrl_vsync_cnt == q_start_up_delay)
                    nstate = ST_HSYNC;
                else
                    nstate = ST_VSYNC;
            end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /* Insert your code here */)
                //    nstate = /* Insert your code here */;
                //else
                //    nstate = ST_HSYNC;
            end
        ST_DATA: begin
                if(ctrl_done) begin   //end of frame
                    //nstate = /* Insert your code here */;
                end
                else begin
                    if(col == /* Insert your code here */)    //end of line
                        nstate = ST_HSYNC;
                    else
                        nstate = ST_DATA;
                end
            end
        default: nstate = ST_IDLE;
    endcase
end
```

Finite State Machine

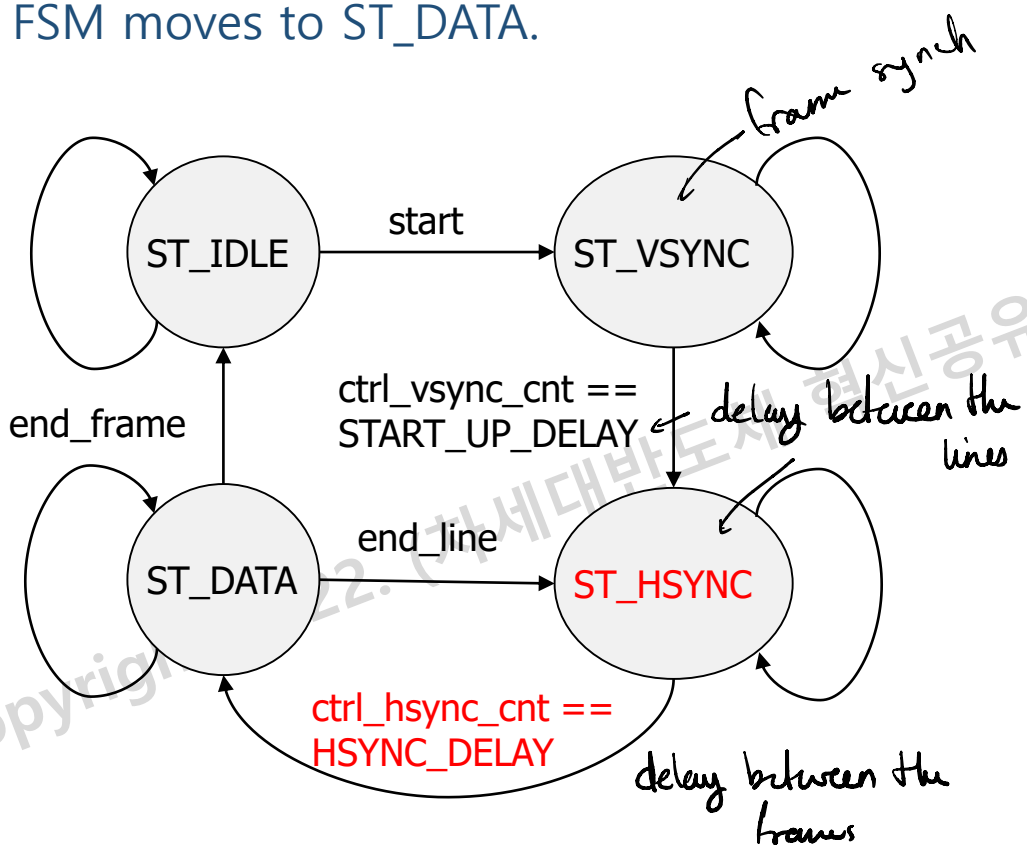
- ST_VSYNC: Frame synchronization
  - Start up for a frame
  - There is an VSYNC counter.
  - When the counter reaches to START_UP_DELAY, FSM moves to ST_HSYNC.




```
always @(*) begin
   case(cstate)
      ST_IDLE: begin
            if(start)
               nstate = ST_VSYNC;
            else
               nstate = ST_IDLE;
      end
        ST_VSYNC: begin
               if(ctrl_vsync_cnt == q_start_up_delay)
                  nstate = ST_HSYNC;
               else
                  nstate = ST_VSYNC;
        end
      ST_HSYNC: begin
            //if(ctrl_hsync_cnt == /* Insert your code here */)
            //    nstate = /* Insert your code here */;
            //else
            //    nstate = ST_HSYNC;
      end
      ST_DATA: begin
            if(ctrl_done) begin   //end of frame
                //nstate = /* Insert your code here */;
            end
            else begin
               if(col == /* Insert your code here */)    //end of line
                  nstate = ST_HSYNC;
               else
                  nstate = ST_DATA;
            end
      end
      default: nstate = ST_IDLE;
   endcase
end
```

Finite State Machine

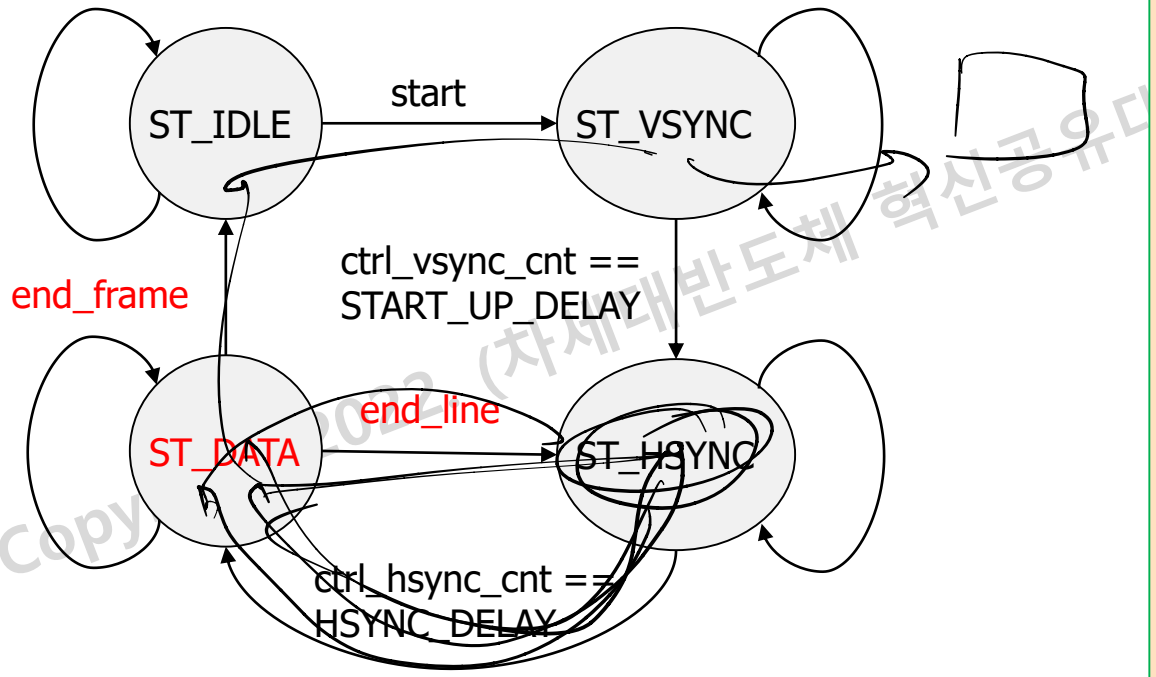
- ST_HSYNC: Line synchronization
  - There is an HSYNC counter.
  - When the counter reaches to HSYNC_DELAY, FSM moves to ST_DATA.



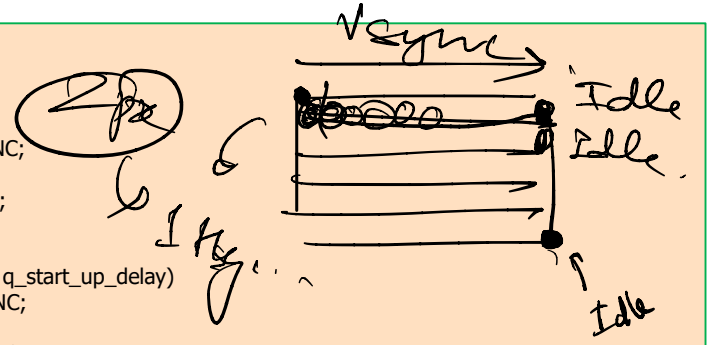

```
always @(*) begin
   case(cstate)
      ST_IDLE: begin
            if(start)
               nstate = ST_VSYNC;
            else
               nstate = ST_IDLE;
      end
      ST_VSYNC: begin
            if(ctrl_vsync_cnt == q_start_up_delay)
               nstate = ST_HSYNC;
            else
               nstate = ST_VSYNC;
      end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /* Insert your code here */)
                //    nstate = /* Insert your code here */;
                //else
                //    nstate = ST_HSYNC;
        end
      ST_DATA: begin
            if(ctrl_done) begin   //end of frame
                //nstate = /* Insert your code here */;
            end
            else begin
               if(col == /* Insert your code here */)    //end of line
                  nstate = ST_HSYNC;
               else
                  nstate = ST_DATA;
            end
      end
      default: nstate = ST_IDLE;
   endcase
end
```

Finite State Machine

- ST_DATA: Sending pixel data
  - There are two counters
  - Line data counter: if it reaches to end of line, FSM moves to ST_HSYNC.
  - Frame data counter: if it reaches to end of frame, FSM moves to ST_IDLE.




```
always @(*) begin
    case(cstate)
        ST_IDLE: begin
            if(start)
                nstate = ST_VSYNC;
            else
                nstate = ST_IDLE;
        end
        ST_VSYNC: begin
            if(ctrl_vsync_cnt == q_start_up_delay)
                nstate = ST_HSYNC;
            else
                nstate = ST_VSYNC;
        end
        ST_HSYNC: begin
            //if(ctrl_hsync_cnt == /* Insert your code here */)
            //    nstate = /* Insert your code here */;
            //else
            //    nstate = ST_HSYNC;
        end
        ST_DATA: begin
            if(ctrl_done) begin   //end of frame
                //nstate = /* Insert your code here */;
            end
            else begin
                //if(col == /* Insert your code here */;)    //end of line
                    nstate = ST_HSYNC;
                else
                    nstate = ST_DATA;
            end
        end
        default: nstate = ST_IDLE;
    endcase
end
```



Frame buffer (lcd_drive_if.v)

- Frame buffer
  1. Read a frame from a hex file
  2. Save pixel data to frame buffers (org_R, org_G, org_B)
  3. Generate LCD drive signals
     - VSYNC, HSYNC
     - DATA_R0, DATA_G0, DATA_B0
     - DATA_R1, DATA_G1, DATA_B1

```
// Read the input file to memory
initial begin
    $readmemh(INFILE, total_memory,0,sizeOfLengthReal_frame-1);
end
// Parse the input pixels
always@(q_start) begin
    if(q_start == 1'b1) begin
        for(i=0; i<WIDTH*HEIGHT*3 ; i=i+1) begin
            temp_BMP[i] = total_memory[i]; //read bmp format image
        end

        for(i=0; i<HEIGHT; i=i+1) begin
            for(j=0; j<WIDTH; j=j+1) begin
                org_R[WIDTH*i+j] = temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+0];
                org_G[WIDTH*i+j] = temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+1];
                org_B[WIDTH*i+j] = temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+2];
            end
        end
    end
end

// Output
always @(*) begin
    VSYNC   = 1'b0;
    HSYNC   = 1'b0;
    DATA_R0 = 0;
    DATA_G0 = 0;
    DATA_B0 = 0;
    DATA_R1 = 0;
    DATA_G1 = 0;
    DATA_B1 = 0;
    if(ctrl_data_run) begin
        VSYNC   = 1'b0;
        HSYNC   = 1'b1;
        DATA_R0 = org_R[WIDTH * row + col   ];
        DATA_G0 = org_G[WIDTH * row + col   ];
        DATA_B0 = org_B[WIDTH * row + col   ];
        DATA_R1 = org_R[WIDTH * row + col +1];
        DATA_G1 = org_G[WIDTH * row + col +1];
        DATA_B1 = org_B[WIDTH * row + col +1];
    end
end
```

Brightness adjustment (lcd_drive_if.v)

- Inputs
  - From the frame buffer: HSYNC, DATA_R0, DATA_G0, DATA_B0, DATA_R1, DATA_G1, DATA_B1
  - From RISC-V: brightness mode (q_br_mode) and value (q_br_value).
- Outputs: To the display panel
  - out_valid
  - out_r0, out_g0, out_b0, out_r0, out_g0, out_b0

```
//------------------------------------------------
// Brightness Adjustment
//------------------------------------------------
brightness_adjustment  #(.IMG_PIX_W(IMG_PIX_W),.WAVE_PIX_W(WAVE_PIX_W))
u_brightness_adjustment(
    /*input*/ .clk(HCLK),
    /*input*/ .rst_n(HRESETn),
    /*input*/ .in_valid(HSYNC),
    /*input*/ .mode(q_br_mode),
    /*input [IMG_PIX_W-1:0]*/ .value(q_br_value),
    /*input [IMG_PIX_W-1:0]*/ .r0(DATA_R0),
    /*input [IMG_PIX_W-1:0]*/ .g0(DATA_G0),
    /*input [IMG_PIX_W-1:0]*/ .b0(DATA_B0),
    /*input [IMG_PIX_W-1:0]*/ .r1(DATA_R1),
    /*input [IMG_PIX_W-1:0]*/ .g1(DATA_G1),
    /*input [IMG_PIX_W-1:0]*/ .b1(DATA_B1),
    /*output*/ .out_valid(out_valid),
    /*output [IMG_PIX_W-1:0]*/ .out_r0(out_r0),
    /*output [IMG_PIX_W-1:0]*/ .out_g0(out_g0),
    /*output [IMG_PIX_W-1:0]*/ .out_b0(out_b0),
    /*output [IMG_PIX_W-1:0]*/ .out_r1(out_r1),
    /*output [IMG_PIX_W-1:0]*/ .out_g1(out_g1),
    /*output [IMG_PIX_W-1:0]*/ .out_b1(out_b1)
);
```

Test cases

- Configure registers of LCD drive
  - Width, height, frame size, data count
  - Vertical and horizontal synchronization delays.
  - Brightness mode (q_br_mode: 0/1) and adjustment value (q_br_value: 50/100)
- After setting configuration registers of LCD drive, RISCV starts a frame.

```
#(8*p)
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_WIDTH , q_width );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_HEIGHT , q_height );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START_UP_DELAY, q_start_up_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_VSYNC_CYCLE, q_vsync_cycle );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_VSYNC_DELAY, q_vsync_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite( `LCD_DRIVE_HSYNC_DELAY, q_hsync_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_FRAME_TRANS_DELAY, q_frame_trans_delay);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_DATA_COUNT, q_data_count );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_BR_MODE, q_br_mode);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_BR_VALUE, q_br_value );
// Start a frame
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START, 1'b1);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START, 1'b0);
```

Results

- Integrate the model with the controller model and the display model
- Run 6ms and show the results at different options

darker, 50

darker, 100

brighter, 50

brighter, 100

Road map

Review

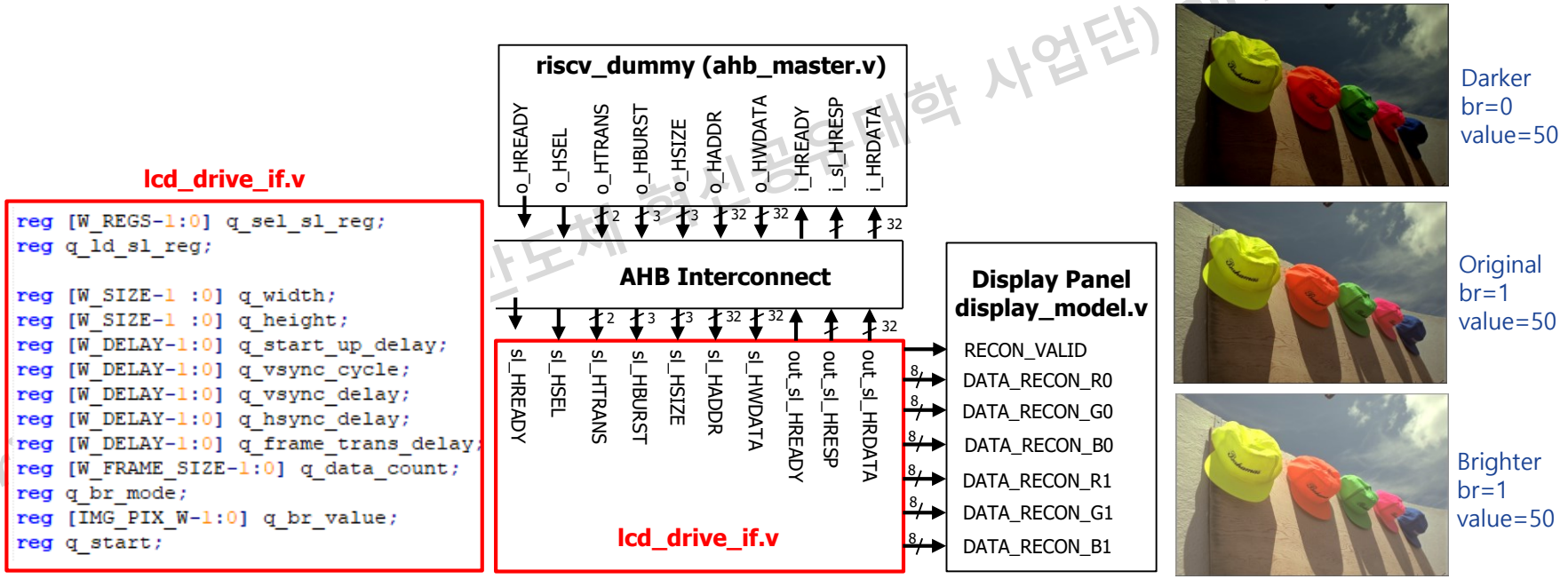
LCD Drive AHB interface

SRAM AHB Interface

Optimized frame buffer

Motivation

- Build an LCD drive IP and integrate it into the system
- RISC-V Master can configure the registers of LCD drive IP
  - Change the brightness mode and the adjustment value.
  - Display the image.
- Can we do better?



Motivation: Frame buffer (lcd_drive_if.v)

- Frame buffer

1. Read a frame from a hex file
2. Save pixel data to frame buffers (org_R, org_G, org_B)
3. Generate LCD drive signals

- VSYNC, HSYNC
- DATA_R0, DATA_G0, DATA_B0
- DATA_R1, DATA_G1, DATA_B1

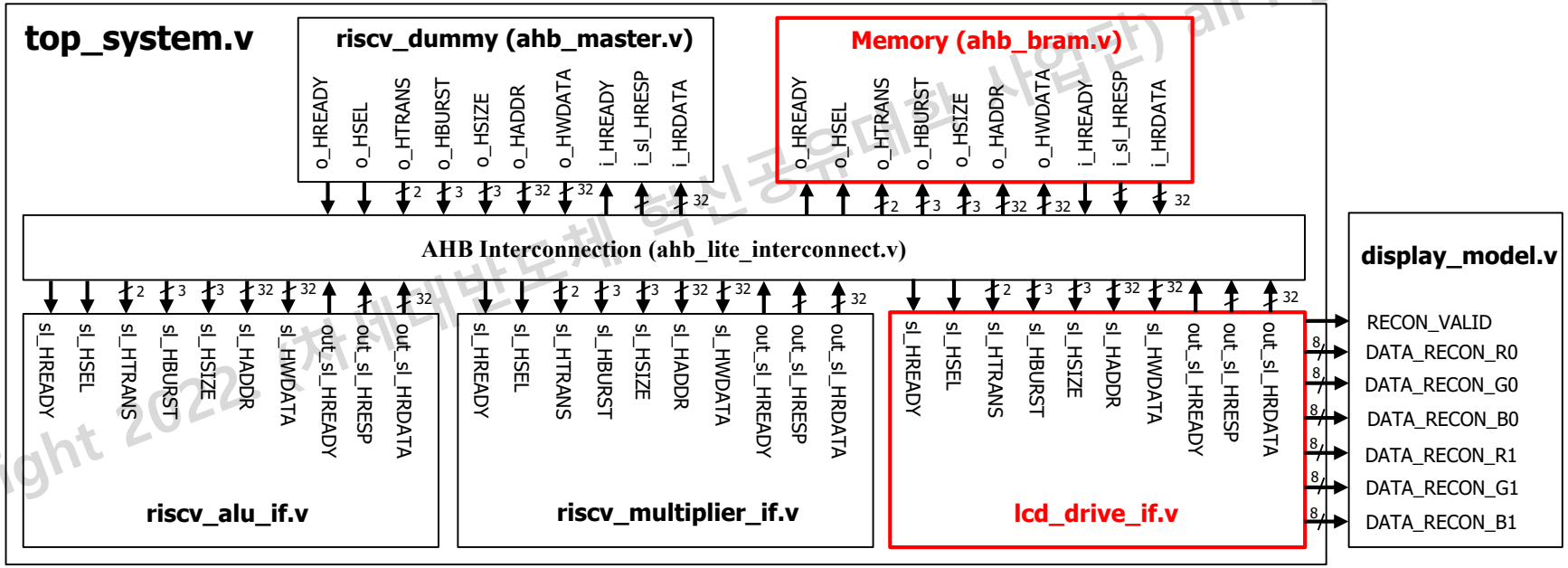
- Limitation
  - An input image is fixed

⇒ We need to re-design the frame buffer of the LCD drive IP which can display an *on-demand* image.

```
// Read the input file to memory
initial begin
    $readmemh(INFILE, total_memory,0,sizeOfLengthReal_frame-1);
end
// Parse the input pixels
always@(q_start) begin
    if(q_start == 1'b1) begin
        for(i=0; i<WIDTH*HEIGHT*3 ; i=i+1) begin
            temp_BMP[i] = total_memory[i]; //read bmp format image
        end

        for(i=0; i<HEIGHT; i=i+1) begin
            for(j=0; j<WIDTH; j=j+1) begin
                org_R[WIDTH*i+j] = temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+0];
                org_G[WIDTH*i+j] = temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+1];
                org_B[WIDTH*i+j] = temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+2];
            end
        end
    end
end

// Output
always @(*) begin
    VSYNC   = 1'b0;
    HSYNC   = 1'b0;
    DATA_R0 = 0;
    DATA_G0 = 0;
    DATA_B0 = 0;
    DATA_R1 = 0;
    DATA_G1 = 0;
    DATA_B1 = 0;
    if(ctrl_data_run) begin
        VSYNC   = 1'b0;
        HSYNC   = 1'b1;
        DATA_R0 = org_R[WIDTH * row + col   ];
        DATA_G0 = org_G[WIDTH * row + col   ];
        DATA_B0 = org_B[WIDTH * row + col   ];
        DATA_R1 = org_R[WIDTH * row + col +1];
        DATA_G1 = org_G[WIDTH * row + col +1];
        DATA_B1 = org_B[WIDTH * row + col +1];
    end
end
```

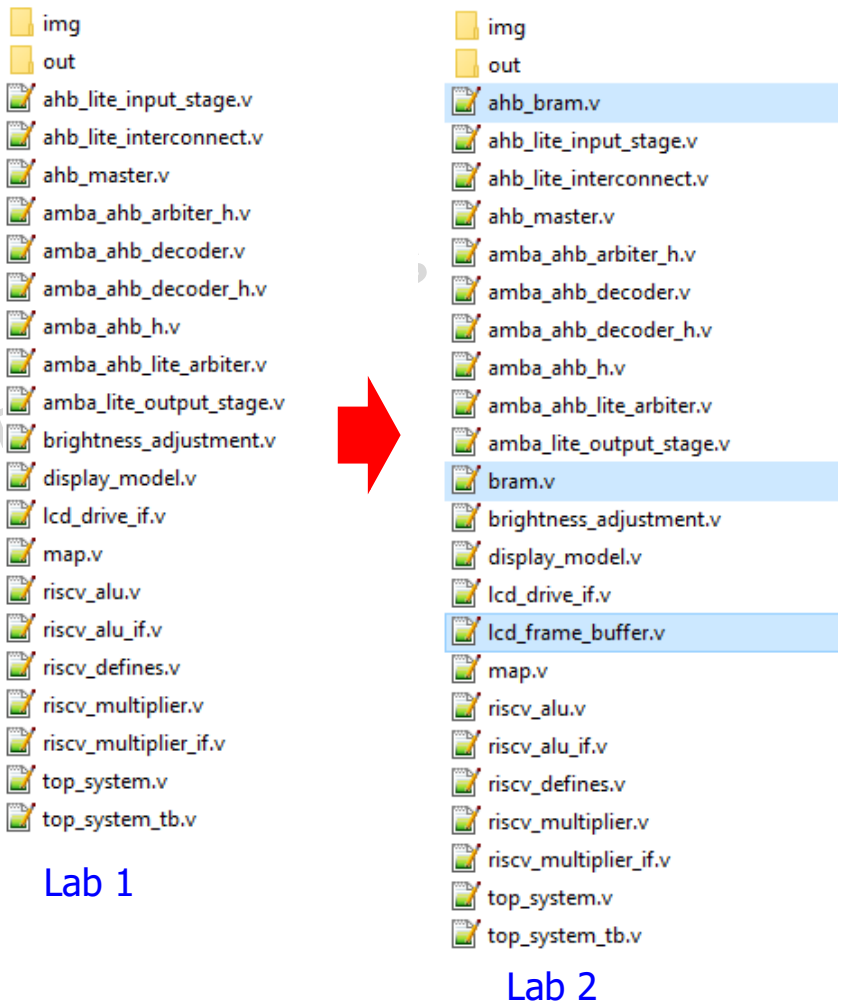
Top system (top_system.v)

- Build a target system that consists of an AHB master and four AHB slaves
  - Master: RISC-V dummy
  - Slaves: ALU, multiplier, LCD Drive, AHB block ram (BRAM).
- Masters and slaves are connected by AHB Interconnect.



Lab 2 vs Lab 1

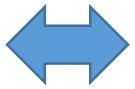
- Add three new files
  - bram.v: Block RAM (BRAM) model, memory
  - ahb_bram.v: AHB interface to BRAM
  - lcd_frame_buffer.v: a buffer to store an frame image.



Memory (ahb_bram.v)

- Memory: AHB Block RAM (BRAM)
  - AHB slave interface
  - Block ram (bram.v)
    - Read/Write Enable (en)
    - Address (addr)
    - Write data (din)
    - Write enable (we)
    - Read data (dout).

```
module bram(
    clk,     // Clock input
    en,      // RAM enable (select)
    addr,    // Address input(word addressing)
    din,     // Data input
    we,      // Write enable
    dout     // Data output
);
```



```
module ahb_bram(
    //CLOCK
    HCLK,
    HRESETn,
    //input signals of control port(slave)
    HREADY,
    HSEL,
    HTRANS,
    HBURST,
    HSIZE,
    HADDR,
    HWRITE,
    HWDATA,
    //output signals of control port(slave)
    out_HREADY,
    out_HRESP,
    out_HRDATA,
    // Memory interface
    sram_we,
    sram_en,
    sram_addr,
    sram_wdata,
    sram_rdata
);
```

Single port ram (bram.v)

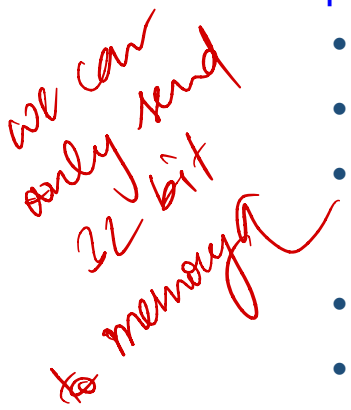
- Assume that an input image is stored in memory or a sing port block ram (bram.v)
  - Read/Write Enable (en)
  - Address (addr)
  - Write data (din)
  - Write enable (we)
  - Read data (dout).

- Memory cell (q_mem) can be initialized from the file INIT_FILE

```
parameter W_DATA = 32;
//RAM cell size(# of words)
parameter N_WORD = 16;
parameter W_WORD = 4;
parameter EN_LOAD_INIT_FILE = 1'b0;
parameter INIT_FILE = "img/kodim03_32bit.hex";

input                  clk;   // Clock input
input                  en;    // RAM enable (select)
input  [W_WORD-1:0]    addr;  // Address input(word addressing) prior to read
input  [W_DATA-1:0]    din;   // Data input
input                  we;    // Write enable
output [W_DATA-1:0]    dout;  // Data output

reg [W_DATA-1:0] q_mem[N_WORD-1:0] /* synthesis syn_ramstyle="block_ram" */;
reg [W_DATA-1:0] dout = 32'h0000_0000;

always @(posedge clk)
begin
    if(en & we)
        q_mem[addr] <= din;
    else if(en)
        dout <=  q_mem[addr];
end

// synopsys translate_off
initial
begin
    if(EN_LOAD_INIT_FILE)
    begin
        $readmemh(INIT_FILE, q_mem);
    end
end
// synopsys translate_on
```

Frame buffer (lcd_frame_buffer.v)

- Frame buffer works like a memory for buffering an image before display
  - Inputs
    - en: Enable signal
    - we: Write enable (Read: 0, Write:1)
    - din[23:0]: R, G, B pixels
      - R: din[23:16], B: din[15:8], G=din[7:0]
    - addr: Request address
    - addr_dual_pixel: address to access two pixels per cycle
  - Outputs
    - dout[23:0]
    - dout_dual_pixel[47:0]
  - Memory/Buffer
    - q_mem[WIDTH*HEIGHT-1:0][23:0]

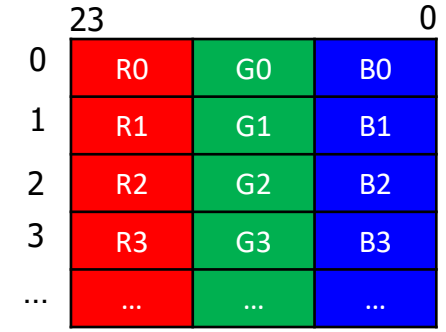



```
module lcd_frame_buffer(
    clk,
    en,
    addr,
    din,
    we,
    dout,
    addr_dual_pixel,
    dout_dual_pixel
);

    parameter W_DATA = 24;  //R: 0~7, G:8~15, B:16~23
    parameter WIDTH = 768;
    parameter HEIGHT = 512;
    parameter N_WORD = WIDTH * HEIGHT;
    parameter W_WORD = $clog2(N_WORD);

    input                 clk;              // Clock input
    input                 en;               // RAM enable (select)
    input  [W_WORD-1:0]   addr;             // Address input(word addressing) p
    input  [W_DATA-1:0]   din;              // Data input
    input                 we;               // Write enable
    output [W_DATA-1:0]   dout;             // Data output
    input  [W_WORD-1:0]   addr_dual_pixel;  // Address input(word addressing) p
    output [2*W_DATA-1:0]   dout_dual_pixel;  // Data output

    reg [W_DATA-1:0] q_mem[N_WORD-1:0]      /* synthesis syn_ramstyle="block_ra
    reg [W_DATA-1:0] dout = 32'h0000_0000;

    always @(posedge clk)
    begin
        if(en & we)
            q_mem[addr] <= din;
        else if(en)
            dout <=  q_mem[addr];
    end
    assign dout_dual_pixel = {q_mem[addr_dual_pixel+1],q_mem[addr_dual_pixel]};
endmodule
```

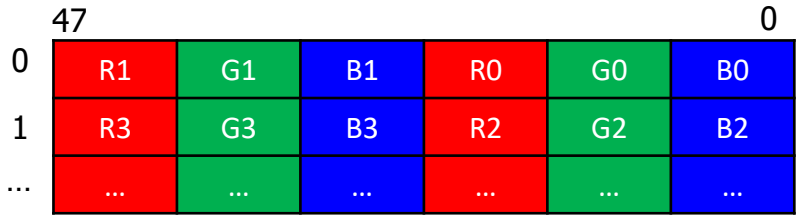
Frame buffer (lcd_drive_if.v)

- Frame buffer (lcd_frame_buffer.v):
  - Similar to a block ram
  - Read/write request (en)
  - Address (addr)
  - Write data (din)
  - Write enable (we)
  - Read data (dout).

```
assign bram_WE = (q_state == ST_WRITE);
assign bram_EN = bram_WE || ((q_state == ST_READ_WAIT) || (en_ahb && !sl_HWRITE));
assign bram_ADDR = bram_WE ? q_bram_addr : bram_addr;
assign addr_dual_pixel = {data_count[W_WORD-2:0],1'b0};
lcd_frame_buffer
u_frame_buffer(
    .clk(HCLK),
    .en(bram_EN),
    .addr(bram_ADDR),
    .din(sl_HWDATA[23:0]),
    .we(bram_WE),
    .dout(out_rgb_pixel),
    .addr_dual_pixel(addr_dual_pixel),
    .dout_dual_pixel(out_rgb_pixel_dual)
);
```



- Buffer
  - Number of words: WIDTH * HEIGHT
  - Each word has three bytes of R, G and B.
  - Special read ports to access two pixels in one cycle



TODO: Top system (top_system.v)

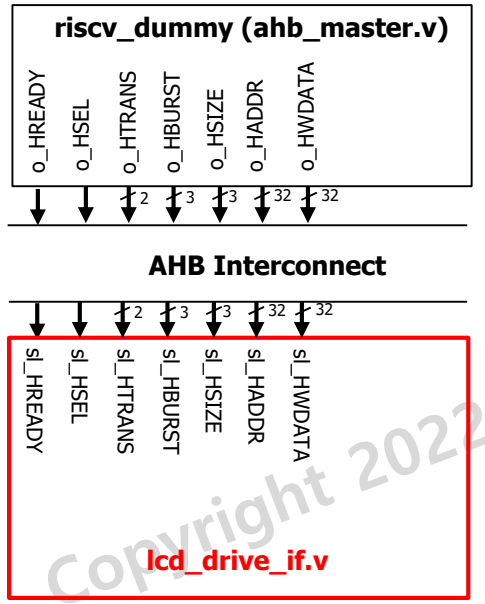
- Connect a slave' ports to AHB Interconnect
  - Ports from a slave are connected to specified ports of AHB Bus.
- Completing the connections between AHB BRAM and AHB Bus

```
//  4. AHB2MEM
assign  mem_sl_HSEL         =   /*Insert your code*/;
assign  mem_sl_HADDR        =   /*Insert your code*/;
assign  mem_sl_HTRANS       =   /*Insert your code*/;
assign  mem_sl_HBURST       =   /*Insert your code*/;
assign  mem_sl_HSIZE        =   /*Insert your code*/;
assign  mem_sl_HPROT        =   /*Insert your code*/;
assign  mem_sl_HWRITE       =   /*Insert your code*/;
assign  mem_sl_HWDATA       =   /*Insert your code*/;
assign  mem_sl_HREADY       =   /*Insert your code*/;
assign  /*Insert your code*/=   out_mem_sl_HREADY;
assign  /*Insert your code*/=   out_mem_sl_HRESP;
assign  /*Insert your code*/=   out_mem_sl_HRDATA;
```

```
// SRAM Controller
ahb_bram #(
.W_DATA(W_DATA),
.N_WORD(N_WORD),
.W_WORD(W_WORD),
.EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
.INIT_FILE(INIT_FILE))
u_ahb_bram
(
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .HREADY(mem_sl_HREADY),
    .HSEL(mem_sl_HSEL),
    .HTRANS(mem_sl_HTRANS),
    .HBURST(mem_sl_HBURST),
    .HSIZE(mem_sl_HSIZE),
    .HADDR(mem_sl_HADDR),
    .HWRITE(mem_sl_HWRITE),
    .HWDATA(mem_sl_HWDATA),
    .out_HREADY(out_mem_sl_HREADY),
    .out_HRESP(out_mem_sl_HRESP),
    .out_HRDATA(out_mem_sl_HRDATA),
    .sram_en(sram_en),
    .sram_we(sram_we),
    .sram_addr(sram_addr),
    .sram_rdata(sram_rdata),
    .sram_wdata(sram_wdata)
);
```

TODO: Top system (top_system.v)

- Connect a slave' ports to AHB Interconnect
  - Ports from a slave are connected to specified ports of AHB Bus.
- Completing the connections between LCD Drive and AHB Bus

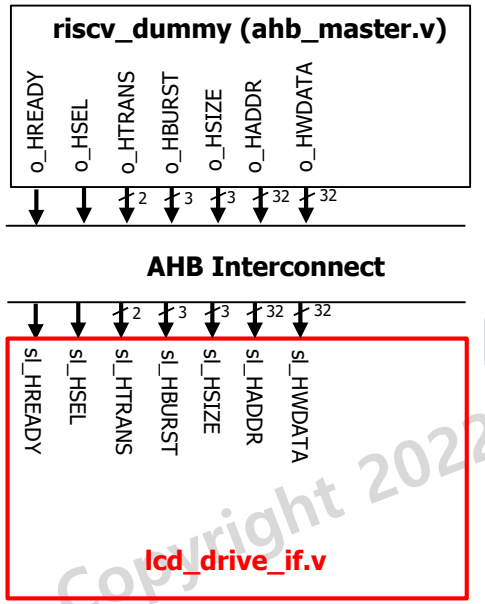



```
// LCD Drive
lcd_drive_if u_lcd_drive_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(lcd_sl_HREADY),
    .sl_HSEL(lcd_sl_HSEL),
    .sl_HTRANS(lcd_sl_HTRANS),
    .sl_HBURST(lcd_sl_HBURST),
    .sl_HSIZE(lcd_sl_HSIZE),
    .sl_HADDR(lcd_sl_HADDR),
    .sl_HWRITE(lcd_sl_HWRITE),
    .sl_HWDATA(lcd_sl_HWDATA),
    .out_sl_HREADY(out_lcd_sl_HREADY),
    .out_sl_HRESP( out_lcd_sl_HRESP),
    .out_sl_HRDATA(out_lcd_sl_HRDATA),
    .out_valid(out_valid),
    .out_r0(out_r0),
    .out_g0(out_g0),
    .out_b0(out_b0),
    .out_r1(out_r1),
    .out_g1(out_g1),
    .out_b1(out_b1)
    );
```

```
//  3. AHB2LCDDRIVE
assign  lcd_sl_HSEL         =   /*Insert your code*/;
assign  lcd_sl_HADDR        =   /*Insert your code*/;
assign  lcd_sl_HTRANS       =   /*Insert your code*/;
assign  lcd_sl_HBURST       =   /*Insert your code*/;
assign  lcd_sl_HSIZE        =   /*Insert your code*/;
assign  lcd_sl_HPROT        =   /*Insert your code*/;
assign  lcd_sl_HWRITE       =   /*Insert your code*/;
assign  lcd_sl_HWDATA       =   /*Insert your code*/;
assign  lcd_sl_HREADY       =   /*Insert your code*/;
assign  /*Insert your code*/=   out_lcd_sl_HREADY;
assign  /*Insert your code*/=   out_lcd_sl_HRESP;
assign  /*Insert your code*/=   out_lcd_sl_HRDATA;
```

TODO: LCD Drive signals (lcd_drive_if.v)

- Complete codes related to the Finte State machine (FSM) of the LCD drive




```
always @(*) begin
    case(cstate)
    ST_IDLE: begin
                if(q_start)
                    nstate = ST_VSYNC;
                else
                    nstate = ST_IDLE;
            end
    ST_VSYNC: begin
                if(ctrl_vsync_cnt == q_start_up_delay)
                    nstate = ST_HSYNC;
                else
                    nstate = ST_VSYNC;
            end
    ST_HSYNC: begin
                if(ctrl_hsync_cnt == /*Insert your code*/)
                    nstate = ST_DATA;
                else
                    nstate = ST_HSYNC;
            end
    ST_DATA: begin
                if(end_frame)     //end of frame
                    nstate = ST_IDLE;
                else begin
                    if(col == /*Insert your code*/)//end of
                        nstate = ST_HSYNC;
                    else
                        nstate = ST_DATA;
                end
            end
    default: nstate = ST_IDLE;
    endcase
end
```

```
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        row <= 0;
        col <= 0;
    end
    else begin
        if(ctrl_data_run) begin
            if(col == q_width - 2) begin
                row <= row + 1;
            end
            if(col == /*Insert your code*/)
                col <= 0;
            else
                col <= col + 2;
        end
    end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        data_count <= 0;
    end
    else begin
        if(ctrl_data_run)
            data_count <= data_count + 1;
    end
end
assign end_frame = (data_count == /*Insert your code*/)? 1'b1: 1'b0;
```

Frame buffer (lcd_drive_if.v)

- Generate LCD drive signals
  - VSYNC, HSYNC
  - DATA_R0, DATA_G0, DATA_B0
  - DATA_R1, DATA_G1, DATA_B1

```
lcd_frame_buffer
u_frame_buffer(
    .clk(HCLK),
    .en(bram_EN),
    .addr(bram_ADDR),
    .din(sl_HWDATA[23:0]),
    .we(bram_WE),
    .dout(out_rgb_pixel),
    .addr_dual_pixel(addr_dual_pixel),
    .dout_dual_pixel(out_rgb_pixel_dual)
);
//// Output
always @(*) begin
    VSYNC   = 1'b0;
    HSYNC   = 1'b0;
    DATA_R0 = 0;
    DATA_G0 = 0;
    DATA_B0 = 0;
    DATA_R1 = 0;
    DATA_G1 = 0;
    DATA_B1 = 0;
    if(ctrl_data_run) begin
        VSYNC   = 1'b0;
        HSYNC   = 1'b1;
        DATA_R0 = out_rgb_pixel_dual[16+:IMG_PIX_W];
        DATA_G0 = out_rgb_pixel_dual[ 8+:IMG_PIX_W];
        DATA_B0 = out_rgb_pixel_dual[ 0+:IMG_PIX_W];
        //DATA_R1 = out_rgb_pixel_dual[/*Insert your code*/];
        //DATA_G1 = out_rgb_pixel_dual[/*Insert your code*/];
        //DATA_B1 = out_rgb_pixel_dual[/*Insert your code*/];
    end
end
```

Brightness adjustment (lcd_drive_if.v)

- Inputs:
  - **From the frame buffer:**
    - HSYNC
    - DATA_R0, DATA_G0, DATA_B0
    - DATA_R1, DATA_G1, DATA_B1
  - **From RISC-V:**
    - brightness mode (q_br_mode)
    - value (q_br_value).
- Outputs: **To the display panel**
  - out_valid
  - out_r0, out_g0, out_b0
  - out_r1, out_g1, out_b1

```
//-------------------------------------------------
// Brightness Adjustment
//-------------------------------------------------
brightness_adjustment  #(.IMG_PIX_W(IMG_PIX_W),.WAVE_PIX_W(WAVE_PIX_W))
u_brightness_adjustment(
    /*input*/ .clk(HCLK),
    /*input*/ .rst_n(HRESETn),
    /*input*/ .in_valid(HSYNC),
    /*input*/ .mode(q_br_mode),
    /*input [IMG_PIX_W-1:0]*/ .value(q_br_value),
    /*input [IMG_PIX_W-1:0]*/ .r0(DATA_R0),
    /*input [IMG_PIX_W-1:0]*/ .g0(DATA_G0),
    /*input [IMG_PIX_W-1:0]*/ .b0(DATA_B0),
    /*input [IMG_PIX_W-1:0]*/ .r1(DATA_R1),
    /*input [IMG_PIX_W-1:0]*/ .g1(DATA_G1),
    /*input [IMG_PIX_W-1:0]*/ .b1(DATA_B1),
    /*output*/ .out_valid(out_valid),
    /*output [IMG_PIX_W-1:0]*/ .out_r0(out_r0),
    /*output [IMG_PIX_W-1:0]*/ .out_g0(out_g0),
    /*output [IMG_PIX_W-1:0]*/ .out_b0(out_b0),
    /*output [IMG_PIX_W-1:0]*/ .out_r1(out_r1),
    /*output [IMG_PIX_W-1:0]*/ .out_g1(out_g1),
    /*output [IMG_PIX_W-1:0]*/ .out_b1(out_b1)
);
```

Test bench

- In the test bench, the top system is connected to the display panel.

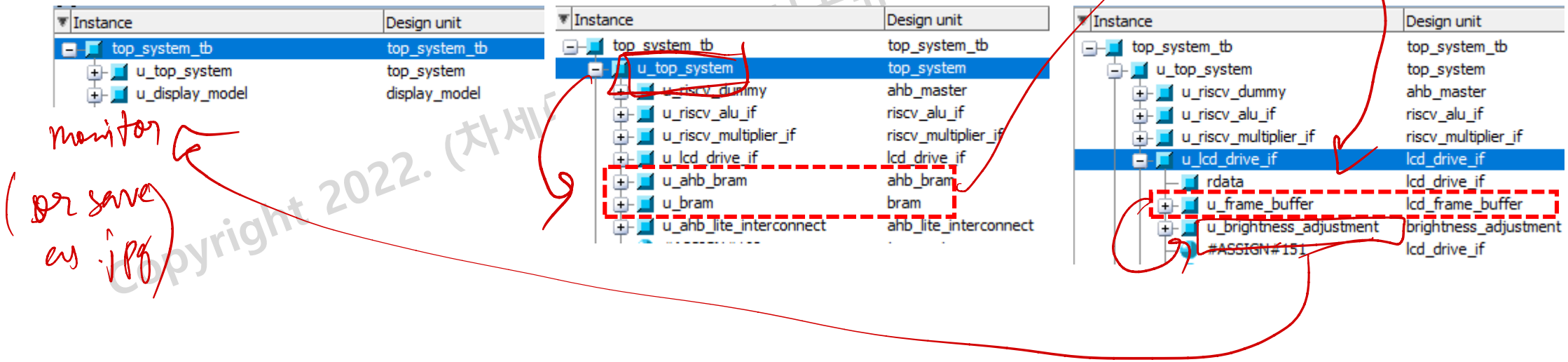
```
`timescale 1ns / 100ps
module top_system_tb;
        reg HCLK, HRESETn;                                  //**** module input
        reg [3:0]           alu_op_i;                       //**** control signals
        reg [31:0]          alu_a_i, alu_b_i;
        reg [31:0]          alu_p_o;
        wire out_valid;
        wire [IMG_PIX_W-1:0] out_r0, out_g0, out_b0, out_r1, out_g1, out_b1;
```

```
top_system
u_top_system (
     .HRESETn(HRESETn)
    ,.HCLK    (HCLK)
    ,/*output*/ .out_valid(out_valid)
    ,/*output [IMG_PIX_W-1:0]*/ .out_r0(out_r0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_g0(out_g0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_b0(out_b0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_r1(out_r1)
    ,/*output [IMG_PIX_W-1:0]*/ .out_g1(out_g1)
    ,/*output [IMG_PIX_W-1:0]*/ .out_b1(out_b1)
);
```

```
display_model #(.INFILE(`OUTPUTFILENAME))
u_display_model(
    ./*input */HCLK(HCLK),
    ./*input */HRESETn(HRESETn),
    ./*input */RECON_VALID(out_valid),
    ./*input [7:0]  */DATA_RECON_R0(out_r0),
    ./*input [7:0]  */DATA_RECON_G0(out_g0),
    ./*input [7:0]  */DATA_RECON_B0(out_b0),
    ./*input [7:0]  */DATA_RECON_R1(out_r1),
    ./*input [7:0]  */DATA_RECON_G1(out_g1),
    ./*input [7:0]  */DATA_RECON_B1(out_b1),
    ./*output  */DEC_DONE()
);
```

Test bench (top_system_tb.v)

- Test bench: the top system is connected to a display panel
- The top system includes: a master, ALU, multiplier, LCD Drive, Bus interconnect, and two new modules: BRAM and AHB BRAM
- LCD drive interface
  - Brightness adjustment
  - Frame buffer



Test cases

- Master accesses ALU
  - Comparison and addition requests.

Select ALU &
AHB read and write

```
#(8*p)
        //sl_HSEL_alu = 1'b1;
        //sl_HSEL_multiplier = 1'b0;
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i); // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);    // Read the result

#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i); // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O, alu_p_o);    // Read the result
```

Test cases

- Master accesses Multiplier
  - A multiplication request.

```
#(8*p)                                                                    Select multiplier &
                //sl_HSEL_alu = 1'b0;                                     AHB read and write
                //sl_HSEL_multiplier = 1'b1;
#(8*p)
                alu_a_i = 32'h7;
                alu_b_i = 32'h9;
                alu_op_i = `ALU_MULL;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i     ); // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i     ); // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i); // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i); // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );    // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o ); // Read the result
```

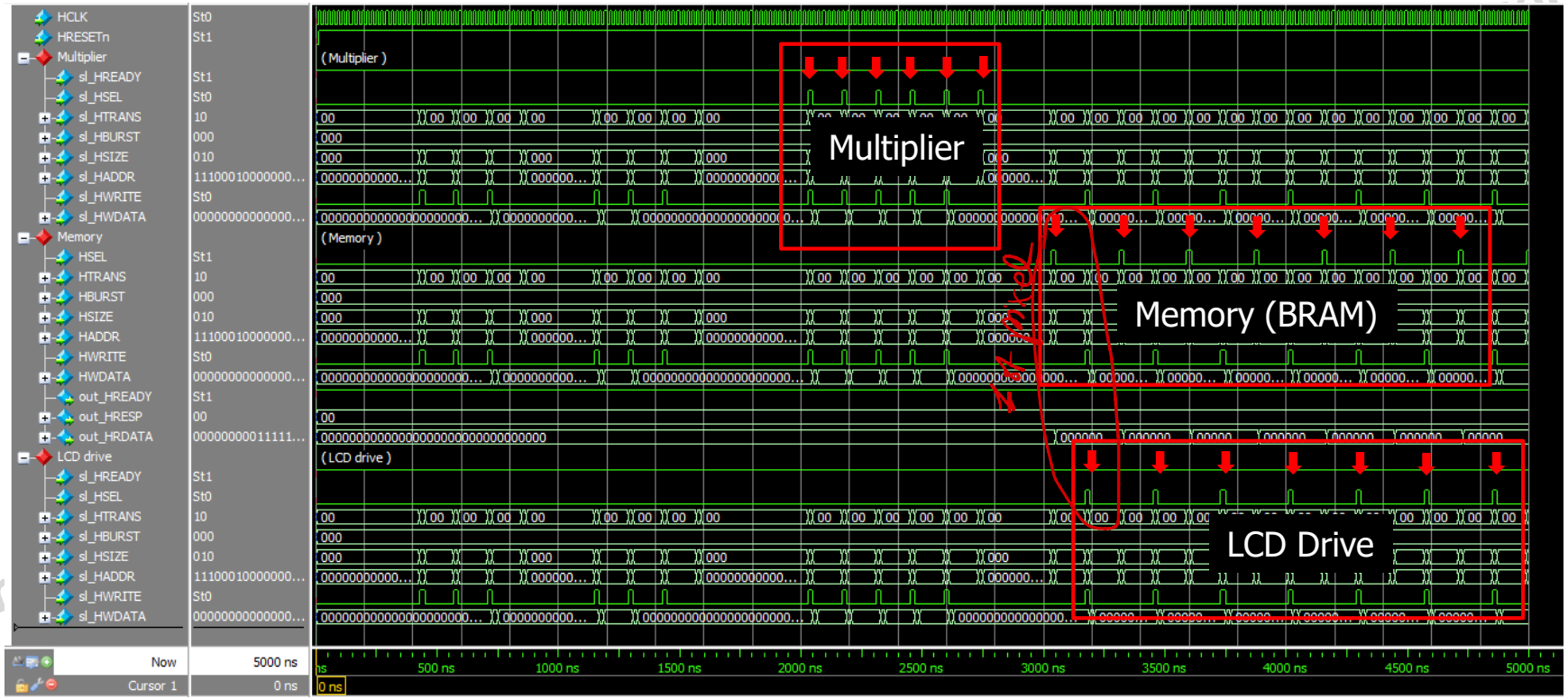
Test cases

- RISC-V
  - Reads a pixel of an image from memory and store it into rdata.
  - Write rdata to the frame buffer of
- An image has WIDTH * HEIGHT pixels
  - WIDTH * HEIGHT read and write operations

```
#(8*p)
// Load data to the frame buffer of LCD drive
for(i = 0; i < WIDTH * HEIGHT; i=i+1) begin
    #(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_MEMORY_BASE_ADDR + 4*i, rdata );
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_IMG_DATA + 4*i, rdata );
end
```

Waveform

- Complete all connections of LCD Drive and AHB BRAM.
- Do a simulation with time = 5,000 ns



Test cases

- Configure registers of LCD drive
  - Width, height, frame size, data count
  - Vertical and horizontal synchronization delays.
  - Brightness mode (q_br_mode: 0/1) and adjustment value (q_br_value: 50/100)
- After setting configuration registers of LCD drive, RISCV starts a frame.

```
#(8*p)
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_WIDTH , q_width );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_HEIGHT , q_height );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START_UP_DELAY, q_start_up_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_VSYNC_CYCLE, q_vsync_cycle );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_VSYNC_DELAY, q_vsync_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite( `LCD_DRIVE_HSYNC_DELAY, q_hsync_delay );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_FRAME_TRANS_DELAY, q_frame_trans_delay);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_DATA_COUNT, q_data_count );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_BR_MODE, q_br_mode);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_BR_VALUE, q_br_value );
// Start a frame
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START, 1'b1);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`LCD_DRIVE_START, 1'b0);
```

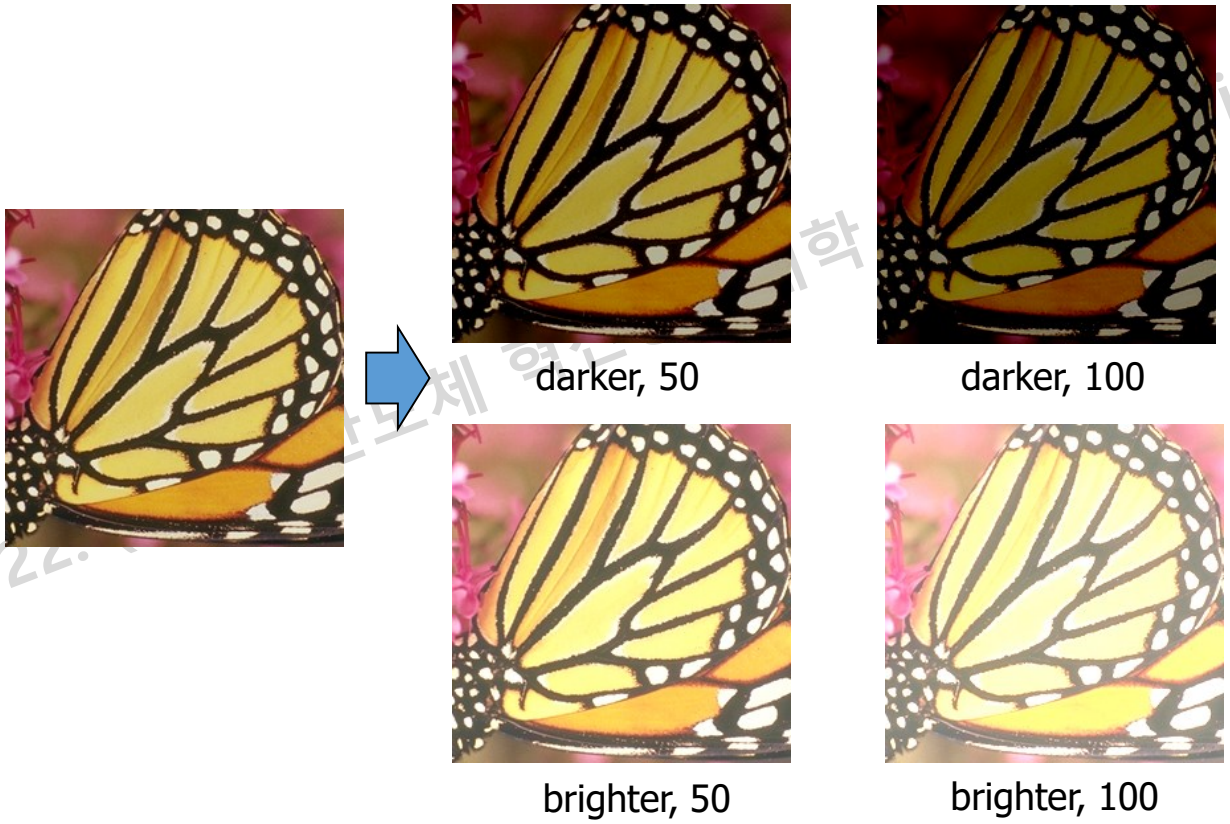
Experimental result (baseline)

- Complete all connections of LCD Drive and AHB BRAM.
- Do a simulation with time = 20 ms

```
always @(*) begin
    VSYNC   = 1'b0;
    HSYNC   = 1'b0;
    DATA_R0 = 0;
    DATA_G0 = 0;
    DATA_B0 = 0;
    DATA_R1 = 0;
    DATA_G1 = 0;
    DATA_B1 = 0;
    if(ctrl_data_run) begin
        VSYNC   = 1'b0;
        HSYNC   = 1'b1;
        DATA_R0 = out_rgb_pixel_dual[16+:IMG_PIX_W];
        DATA_G0 = out_rgb_pixel_dual[ 8+:IMG_PIX_W];
        DATA_B0 = out_rgb_pixel_dual[ 0+:IMG_PIX_W];
        //DATA_R1 = out_rgb_pixel_dual[40+:IMG_PIX_W];
        //DATA_G1 = out_rgb_pixel_dual[32+:IMG_PIX_W];
        //DATA_B1 = out_rgb_pixel_dual[24+:IMG_PIX_W];
    end
end
```

To do …

- Complete the missing codes
- Test the top system by running simulation in 20ms
- Show the simulation results at different options



Road map

Review

LCD Drive AHB interface

SRAM AHB Interface

Optimized frame buffer

Recall: Reversible color transform (RCT)

- The pixel data in a small region (aka block) usually are similar.
- In image compression, reversible color transform is used to convert between two color domains
  - Forward: RGB to YCbCr
  - Backward: YCbCr to RGB

Red

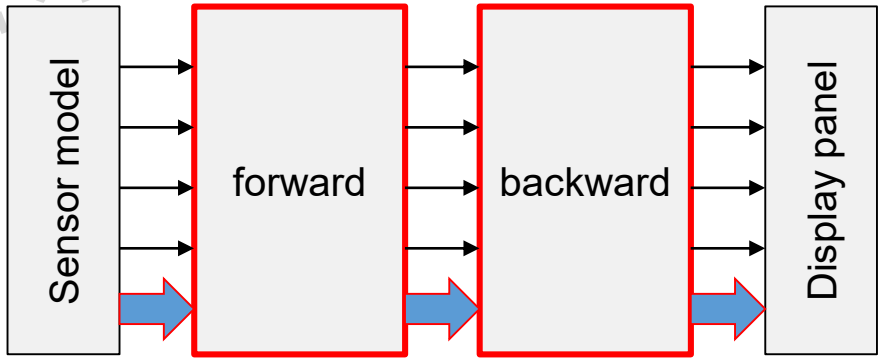
225	224	222	219	222	222	222	223
224	223	224	222	220	225	223	220
219	219	223	225	226	225	223	221
223	224	221	224	224	220	223	219
224	224	221	224	220	223	224	222
224	224	224	224	221	224	224	224
221	224	224	223	224	224	224	224
224	221	221	224	224	224	221	220

Green

204	203	201	195	199	199	202	202
205	204	206	201	200	204	202	200
200	200	203	204	205	204	202	197
204	204	202	204	206	202	204	199
206	206	202	204	202	204	205	201
205	206	206	205	203	205	208	206
202	206	205	204	205	206	206	206
206	203	203	205	205	206	202	201

Blue

99	98	95	90	94	94	93	97
99	99	102	98	94	99	95	94
96	98	102	103	102	101	97	95
99	103	103	105	104	98	98	93
102	104	103	103	98	99	101	98
101	104	104	101	97	101	106	104
98	102	101	99	99	102	104	104
102	97	95	99	99	102	98	99



Forward RCT (rct_forward.v)

- Convert pixels from the RGB domain to the YCbCr domain
- Inputs
  - Clock, reset
  - Input pixels r0, g0, b0, r1, g1, b1 and a valid signal (in_valid)
- Outputs: y0, y1, cb0, cb1 and an output valid signal (out_valid)

```
module forward_rct
#(
    parameter IMG_PIX_W  = 8,
              WAVE_PIX_W = 10
)
(
    input clk,
    input rst_n,
    input in_valid,
    input [IMG_PIX_W-1:0] r0, g0, b0,
    input [IMG_PIX_W-1:0] r1, g1, b1,
    output reg out_valid,
    output reg [WAVE_PIX_W-1:0] y0, y1, cb0, cr0
);
```

Forward RCT (rct_forward.v)

- Convert pixels from the RGB domain to the YCbCr domain
- $\begin{bmatrix} y \\ cb \\ cr \end{bmatrix} = \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 0 & -1 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$

- Down sampling:
  - Ignore cb1, cr1

```
/*Signals*/
wire [WAVE_PIX_W-1:0] tmp_y0 = {2'b00, r0} + {1'b0, g0,1'b0} + {2'b00, b0};//r0 + (g0<<<1) + b0
wire [WAVE_PIX_W-1:0] tmp_y1 = {2'b00, r1} + {1'b0, g1,1'b0} + {2'b00, b1};//r1 + (g1<<<1) + b1
wire [WAVE_PIX_W-1:0] tmp_cb = {2'b00, b0} - {2'b00, g0};//b0-g0
wire [WAVE_PIX_W-1:0] tmp_cr = {2'b00, r0} - {2'b00, g0};//r0-g0
/*Sequential logic*/
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        y0  <= 0;
        y1  <= 0;
        cb0 <= 0;
        cr0 <= 0;
        out_valid <= 0;
    end else begin
        if(in_valid) begin
            y0  <= {2'b0,tmp_y0[9:2]};
            //y1  <= /*Insert your code here*/;
            cb0 <= tmp_cb;
            //cr0 <= /*Insert your code here*/;
            out_valid <= 1;
        end else begin
            y0  <= 0;
            y1  <= 0;
            cb0 <= 0;
            cr0 <= 0;
            out_valid <= 0;
        end
    end
end
```

Backward/inverse RCT (rct_inverse.v)

- Convert pixels from the YCbCr domain to the RGB domain
- Inputs
  - Clock, reset
  - Input pixels y0, cb0, cr0, y1, cb1, cr1 and a valid signal (in_valid)
- Outputs: r0, g0, b0, r1, g1, b1 and an output valid signal (out_valid)

```
module inverse_rct
#(
    parameter IMG_PIX_W  = 8,
              WAVE_PIX_W = 10
)
(
    input clk,
    input rst_n,
    input in_valid,
    input signed [WAVE_PIX_W-1:0] y0, cb0, cr0,
    input signed [WAVE_PIX_W-1:0] y1, cb1, cr1,
    //input [WAVE_PIX_W-1:0] y0, cb0, cr0,
    //input [WAVE_PIX_W-1:0] y1, cb1, cr1,
    output reg out_valid,
    output [IMG_PIX_W-1:0] r0, g0, b0,
    output [IMG_PIX_W-1:0] r1, g1, b1
);
```

RCT backward/inverse (rct_inverse.v)

- Convert pixels from the RGB domain to the YCbCr domain

$$tmp = Y - (Cb + Cr) \gg 2$$
$$R = Cr + tmp$$
$$G = tmp$$
$$B = Cb + tmp$$

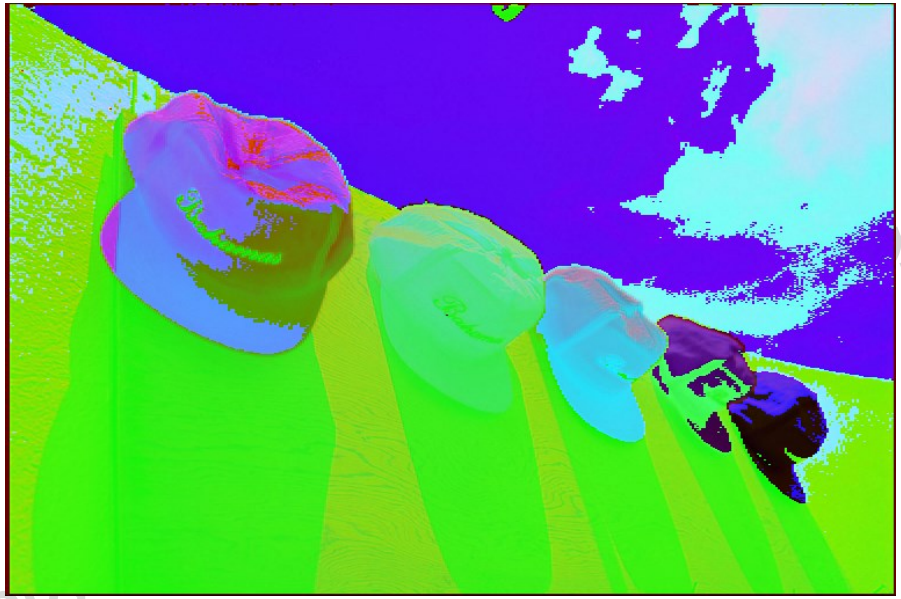
- Clipping
  - If the pixel is larger than 255, assign it to 255
  - If the pixel is smaller than 0, assign it to zero

```
assign r0 = (r0_tmp[WAVE_PIX_W-1])? 0 : (r0_tmp[WAVE_PIX_W-2])? 255 : r0_tmp[IMG_PIX_W-1:0];
assign g0 = (g0_tmp[WAVE_PIX_W-1])? 0 : (g0_tmp[WAVE_PIX_W-2])? 255 : g0_tmp[IMG_PIX_W-1:0];
assign b0 = (b0_tmp[WAVE_PIX_W-1])? 0 : (b0_tmp[WAVE_PIX_W-2])? 255 : b0_tmp[IMG_PIX_W-1:0];
assign r1 = (r1_tmp[WAVE_PIX_W-1])? 0 : (r1_tmp[WAVE_PIX_W-2])? 255 : r1_tmp[IMG_PIX_W-1:0];
assign g1 = (g1_tmp[WAVE_PIX_W-1])? 0 : (g1_tmp[WAVE_PIX_W-2])? 255 : g1_tmp[IMG_PIX_W-1:0];
assign b1 = (b1_tmp[WAVE_PIX_W-1])? 0 : (b1_tmp[WAVE_PIX_W-2])? 255 : b1_tmp[IMG_PIX_W-1:0];
```

```
/*Combinational logic*/
assign tmp0 = y0 - ((cb0 + cr0) >>> 2);
assign tmp1 = y1 - ((cb1 + cr1) >>> 2);

/*Sequential logic*/
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        r0_tmp <= 0;
        g0_tmp <= 0;
        b0_tmp <= 0;
        r1_tmp <= 0;
        g1_tmp <= 0;
        b1_tmp <= 0;
        out_valid <= 0;
    end else begin
        if(in_valid) begin
            r0_tmp <= cr0 + tmp0;
            g0_tmp <= tmp0;
            b0_tmp <= cb0 + tmp0;
            //r1_tmp <= /*Insert your code here*/;
            //g1_tmp <= /*Insert your code here*/;
            //b1_tmp <= /*Insert your code here*/;
            out_valid <= 1;
        end else begin
            r0_tmp <= 0;
            g0_tmp <= 0;
            b0_tmp <= 0;
            r1_tmp <= 0;
            g1_tmp <= 0;
            b1_tmp <= 0;
            out_valid <= 0;
        end
    end
end
```

Results

- Implement RCT forward and backward modules.
- Run 6 ms (fmc_rct_top_tb.v)
- Check the output images

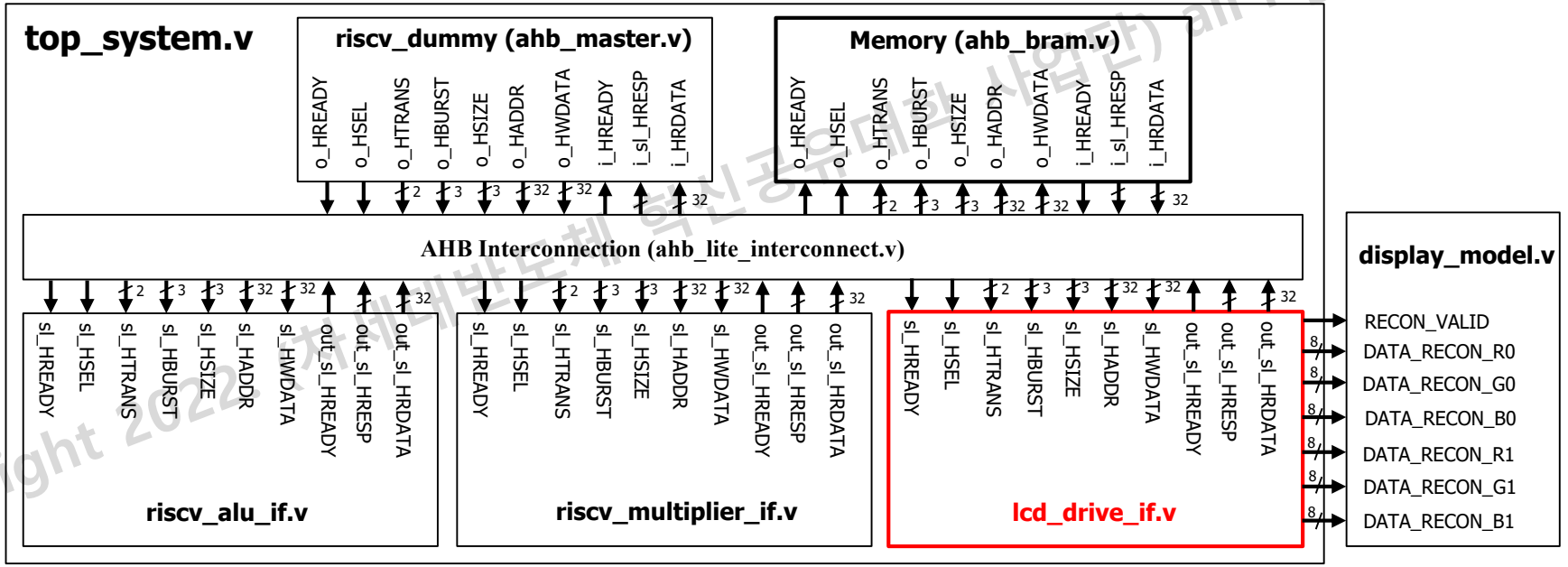


./out/kodim03_ycbcr.bmp

./out/kodim03.bmp

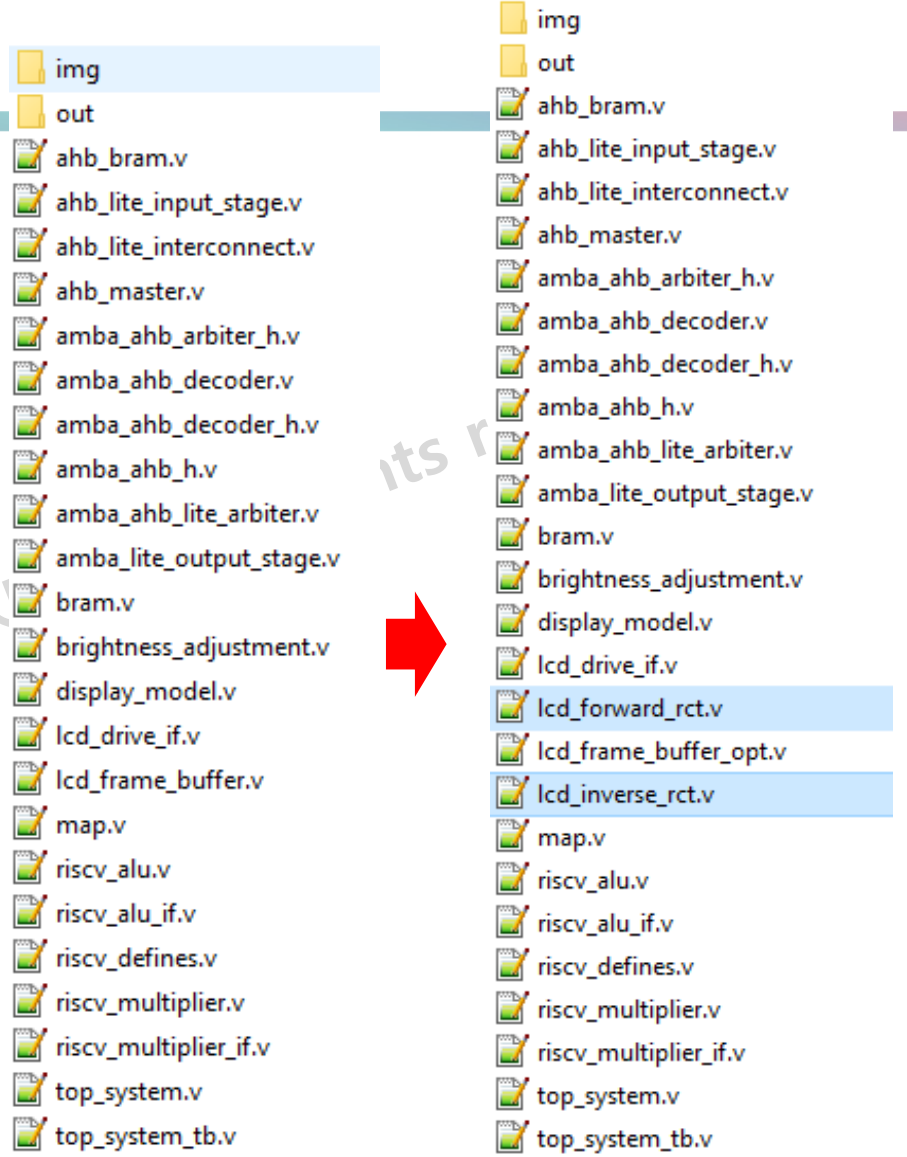
Lab 3: Compressed frame buffer

- Build a target system that consists of an AHB master and four AHB slaves
  - Master: RISC-V dummy
  - Slaves: ALU, multiplier, Optimized LCD Drive, AHB block ram (BRAM).
- Masters and slaves are connected by AHB Interconnect.



Lab 3 vs Lab 2

- Reuse files from Lab 2
  - bram.v: Block RAM (BRAM) model, memory
  - ahb_bram.v: AHB interface to BRAM
  - top_system.v
- Add two new files
  - lcd_forward_rct.v: forward RCT
  - lcd_inverse_rct.v: backward/inverse RCT

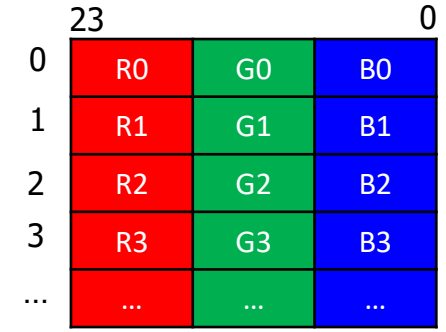


Lab 2 Lab 3

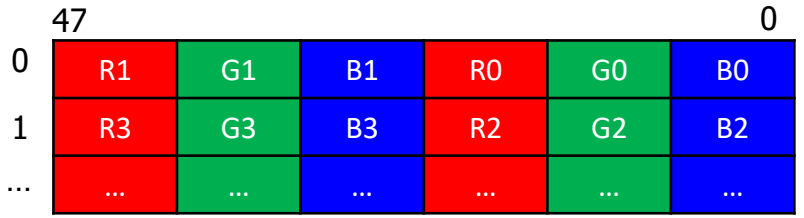
Previous Frame buffer (lcd_drive_if.v)

- Frame buffer (lcd_frame_buffer.v):
  - Similar to a block ram
  - Read/write request (en)
  - Address (addr)
  - Write data (din)
  - Write enable (we)
  - Read data (dout).

```
assign bram_WE = (q_state == ST_WRITE);
assign bram_EN = bram_WE || ((q_state == ST_READ_WAIT) || (en_ahb && !sl_HWRITE));
assign bram_ADDR = bram_WE ? q_bram_addr : bram_addr;
assign addr_dual_pixel = {data_count[W_WORD-2:0],1'b0};
lcd_frame_buffer
u_frame_buffer(
    .clk(HCLK),
    .en(bram_EN),
    .addr(bram_ADDR),
    .din(sl_HWDATA[23:0]),
    .we(bram_WE),
    .dout(out_rgb_pixel),
    .addr_dual_pixel(addr_dual_pixel),
    .dout_dual_pixel(out_rgb_pixel_dual)
);
```



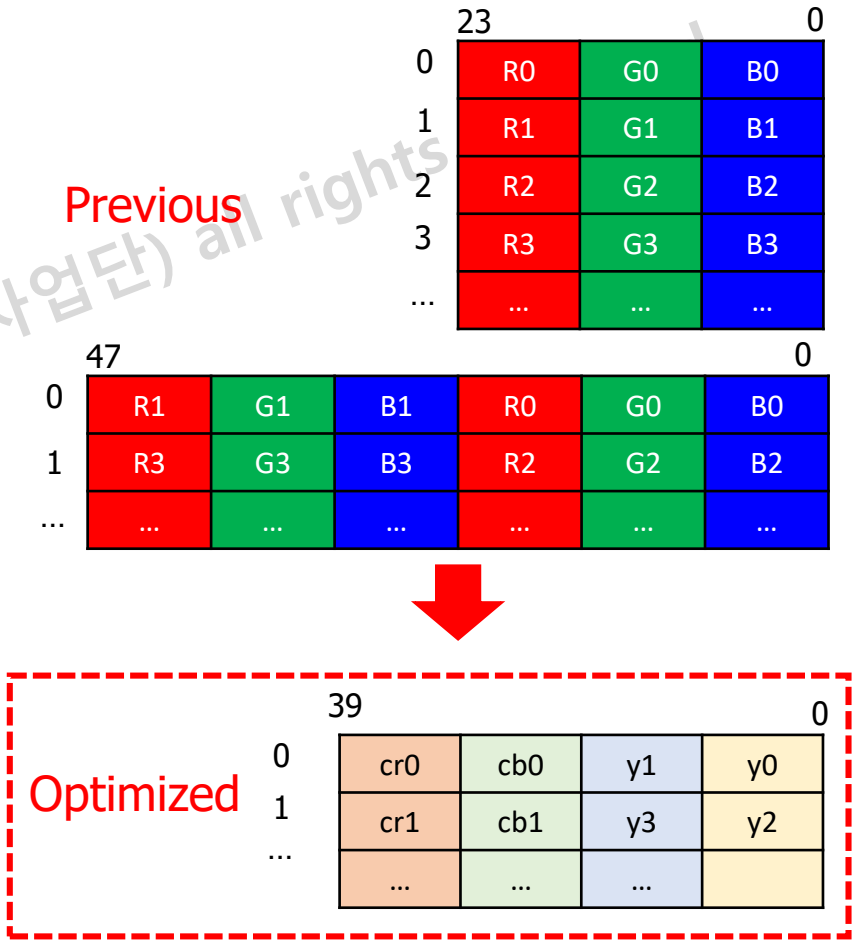
- Buffer
  - Number of words: WIDTH * HEIGHT
  - Each word has three bytes of R, G and B.
  - Special read ports to access two pixels in one cycle



Optimized Frame Buffer (lcd_drive_if.v)

- Encode
  - Use **forward RCT** to convert (R0, G0, B0, R1, G1, B1) to (Y0, Y1, Cb, Cr)
  - Store compressed pixels to the frame buffer

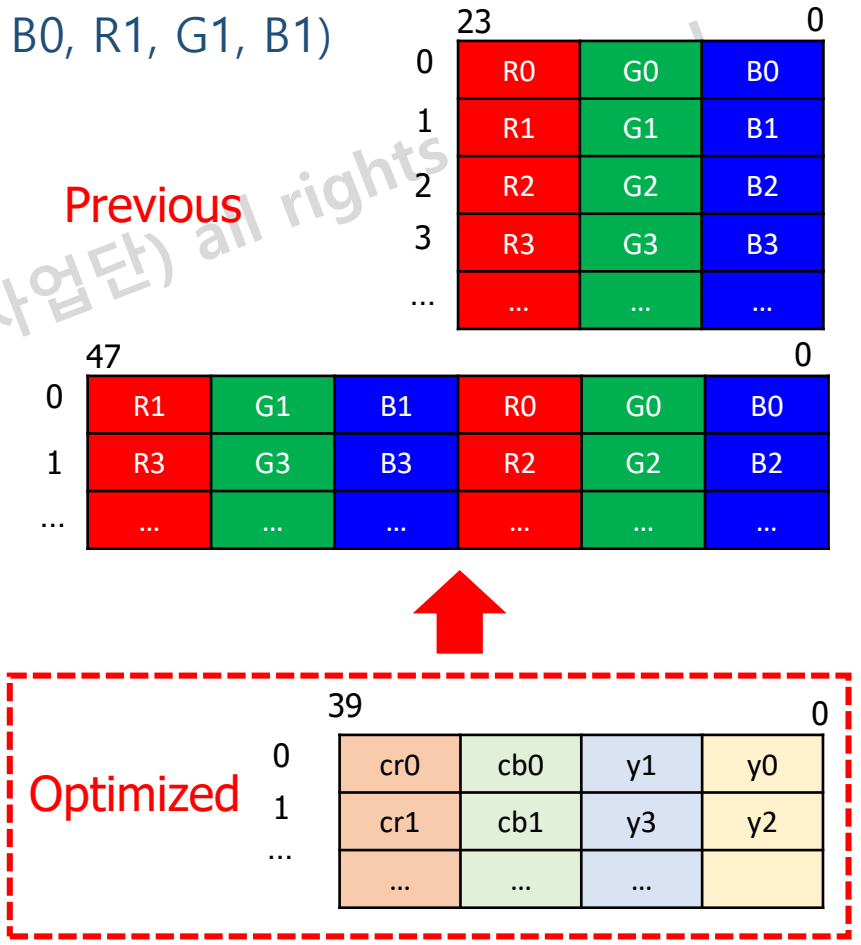
```
lcd_forward_rct u_forward_rct ( //{{{
    /* input*/.clk            (HCLK       ),
    /* input*/.rst_n          (HRESETn    ),
    /* input*/.in_valid       (in_valid_rct      ),
    /* input*/.r0             (in_pixel_rct[16+:IMG_PIX_W]),
    /* input*/.g0             (in_pixel_rct[ 8+:IMG_PIX_W]),
    /* input*/.b0             (in_pixel_rct[ 0+:IMG_PIX_W]),
    /* input*/.r1             (/*Insert your code*/),
    /* input*/.g1             (/*Insert your code*/),
    /* input*/.b1             (/*Insert your code*/),
    /*output*/.out_valid      (rct_o_valid),
    /*output*/.y0             (rct_dout_y0),
    /*output*/.y1             (rct_dout_y1),
    /*output*/.cb0            (rct_dout_cb),
    /*output*/.cr0            (rct_dout_cr)
); //}}}

assign rct_o_pixel = {rct_dout_cr,rct_dout_cb,rct_dout_y1,rct_dout_y0};
lcd_frame_buffer_opt
u_frame_buffer(
    .clk(HCLK),
    .en(rct_o_valid),
    .addr({1'b0,q_bram_addr[W_WORD-1:1]}),
    .din(rct_o_pixel),
    .we(rct_o_valid),
    .dout(),
    .addr_dual_pixel(addr_dual_pixel),
    .dout_dual_pixel(out_rgb_pixel_dual)
);
```



Optimized Frame Buffer (lcd_drive_if.v)

- Decode
  - Load compressed pixels
  - Use **inverse RCT** to convert (Y0, Y1, Cb, Cr) to (R0, G0, B0, R1, G1, B1)

```
u_frame_buffer(
    .clk(HCLK),
    .en(rct_o_valid),
    .addr({1'b0,q_bram_addr[W_WORD-1:1]}),
    .din(rct_o_pixel),
    .we(rct_o_valid),
    .dout(),
    .addr_dual_pixel(addr_dual_pixel),
    .dout_dual_pixel(out_rgb_pixel_dual)
);

lcd_inverse_rct u_inverse_rct
(
    ./*input */clk(HCLK),
    ./*input */rst_n(HRESETn),
    ./*input */in_valid(ctrl_data_run),
    ./*input signed [WAVE_PIX_W-1:0] */y0 (out_rgb_pixel_dual[ 0+:WAVE_PIX_W]),
    ./*input signed [WAVE_PIX_W-1:0] */cb0(out_rgb_pixel_dual[20+:WAVE_PIX_W]),
    ./*input signed [WAVE_PIX_W-1:0] */cr0(out_rgb_pixel_dual[30+:WAVE_PIX_W]),
    ./*input signed [WAVE_PIX_W-1:0] */y1 (/*Insert your code*/),
    ./*input signed [WAVE_PIX_W-1:0] */cb1(/*Insert your code*/),
    ./*input signed [WAVE_PIX_W-1:0] */cr1(/*Insert your code*/),
    ./*output reg */out_valid(recon_valid),
    ./*output [IMG_PIX_W-1:0] */r0(recon_data_R0),
    ./*output [IMG_PIX_W-1:0] */g0(recon_data_G0),
    ./*output [IMG_PIX_W-1:0] */b0(recon_data_B0),
    ./*output [IMG_PIX_W-1:0] */r1(recon_data_R1),
    ./*output [IMG_PIX_W-1:0] */g1(recon_data_G1),
    ./*output [IMG_PIX_W-1:0] */b1(recon_data_B1)
);
```



Optimized Frame Buffer (lcd_frame_buffer_opt.v)

- Frame buffer works like a memory for buffering an image before display
  - Inputs
    - en: Enable signal
    - we: Write enable (Read: 0, Write:1)
    - din[39:0]: Y0, Y1, Cb, Cr pixels
    - addr: Request address
    - addr_dual_pixel: address to access two pixels per cycle
  - Outputs
    - dout[39:0]
  - Memory/Buffer
    - q_mem[WIDTH*HEIGHT/2-1:0][39:0]

```
module lcd_frame_buffer_opt(
    clk,
    en,
    addr,
    din,
    we,
    dout,
    addr_dual_pixel,
    dout_dual_pixel
);
    parameter WAVE_PIX_W = 10;
    parameter W_DATA = 4*WAVE_PIX_W; // Cr:39~30, Cb:29~20, Y1:19~10, Y0:9~0
    parameter WIDTH = 768;
    parameter HEIGHT = 512;
    parameter N_WORD = WIDTH * HEIGHT/2;
    parameter W_WORD = $clog2(N_WORD);

    input                  clk;              // Clock input
    input                  en;               // RAM enable (select)
    input  [W_WORD-1:0]    addr;             // Address input(word addressing)
    input  [W_DATA-1:0]    din;              // Data input
    input                  we;               // Write enable
    output [W_DATA-1:0]    dout;             // Data output
    input  [W_WORD-1:0]    addr_dual_pixel;  // Address input(word addressing)
    output [W_DATA-1:0]    dout_dual_pixel;  // Data output

    reg [W_DATA-1:0] q_mem[N_WORD-1:0]      /* synthesis syn_ramstyle="block_r
    reg [W_DATA-1:0] dout = 0;

    always @(posedge clk)
    begin
        if(en & we)
            q_mem[addr] <= din;
        else if(en)
            dout <=  q_mem[addr];
    end
    assign dout_dual_pixel = q_mem[addr_dual_pixel];
endmodule
```

Brightness adjustment (lcd_drive_if.v)

- Inputs:
  - **From the frame buffer:**
    - HSYNC
    - DATA_R0, DATA_G0, DATA_B0
    - DATA_R1, DATA_G1, DATA_B1
  - **From RISC-V:**
    - brightness mode (q_br_mode)
    - value (q_br_value).
- Outputs: **To the display panel**
  - out_valid
  - out_r0, out_g0, out_b0
  - out_r1, out_g1, out_b1

```
//-------------------------------------------------
// Brightness Adjustment
//-------------------------------------------------
brightness_adjustment  #(.IMG_PIX_W(IMG_PIX_W),.WAVE_PIX_W(WAVE_PIX_W))
u_brightness_adjustment(
    /*input*/ .clk(HCLK),
    /*input*/ .rst_n(HRESETn),
    /*input*/ .in_valid(HSYNC),
    /*input*/ .mode(q_br_mode),
    /*input [IMG_PIX_W-1:0]*/ .value(q_br_value),
    /*input [IMG_PIX_W-1:0]*/ .r0(DATA_R0),
    /*input [IMG_PIX_W-1:0]*/ .g0(DATA_G0),
    /*input [IMG_PIX_W-1:0]*/ .b0(DATA_B0),
    /*input [IMG_PIX_W-1:0]*/ .r1(DATA_R1),
    /*input [IMG_PIX_W-1:0]*/ .g1(DATA_G1),
    /*input [IMG_PIX_W-1:0]*/ .b1(DATA_B1),
    /*output*/ .out_valid(out_valid),
    /*output [IMG_PIX_W-1:0]*/ .out_r0(out_r0),
    /*output [IMG_PIX_W-1:0]*/ .out_g0(out_g0),
    /*output [IMG_PIX_W-1:0]*/ .out_b0(out_b0),
    /*output [IMG_PIX_W-1:0]*/ .out_r1(out_r1),
    /*output [IMG_PIX_W-1:0]*/ .out_g1(out_g1),
    /*output [IMG_PIX_W-1:0]*/ .out_b1(out_b1)
);
```

Test bench

- In the test bench, the top system is connected to the display panel.

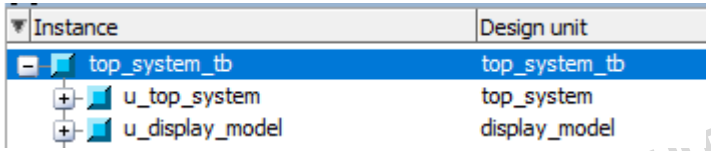
```
`timescale 1ns / 100ps
module top_system_tb;
        reg HCLK, HRESETn;                                  //**** module input
        reg [3:0]           alu_op_i;                       //**** control signals
        reg [31:0]          alu_a_i, alu_b_i;
        reg [31:0]          alu_p_o;
        wire out_valid;
        wire [IMG_PIX_W-1:0] out_r0, out_g0, out_b0, out_r1, out_g1, out_b1;
```

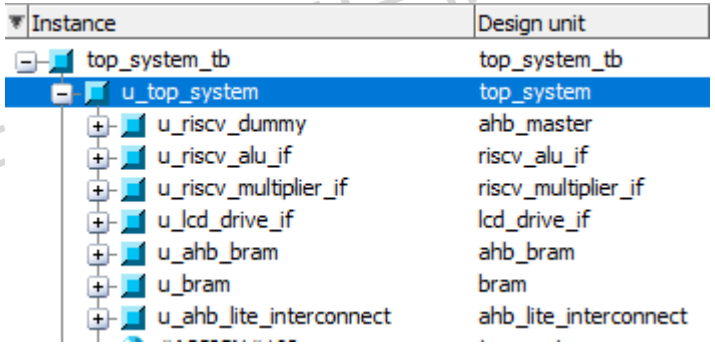
```
top_system
u_top_system (
     .HRESETn(HRESETn)
    ,.HCLK    (HCLK)
    ,/*output*/ .out_valid(out_valid)
    ,/*output [IMG_PIX_W-1:0]*/ .out_r0(out_r0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_g0(out_g0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_b0(out_b0)
    ,/*output [IMG_PIX_W-1:0]*/ .out_r1(out_r1)
    ,/*output [IMG_PIX_W-1:0]*/ .out_g1(out_g1)
    ,/*output [IMG_PIX_W-1:0]*/ .out_b1(out_b1)
);
```

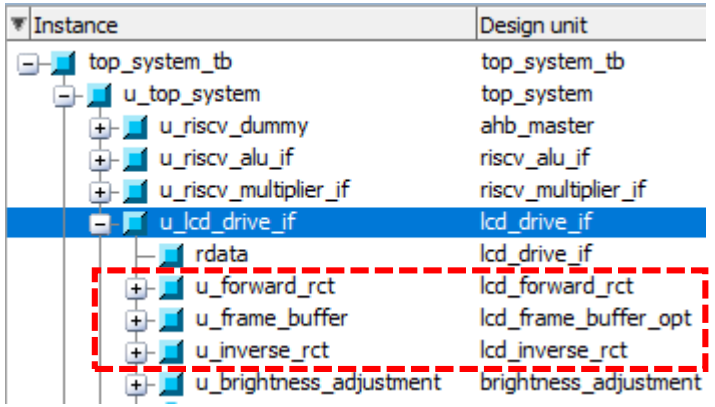
```
display_model #(.INFILE(`OUTPUTFILENAME))
u_display_model(
    ./*input */HCLK(HCLK),
    ./*input */HRESETn(HRESETn),
    ./*input */RECON_VALID(out_valid),
    ./*input [7:0]  */DATA_RECON_R0(out_r0),
    ./*input [7:0]  */DATA_RECON_G0(out_g0),
    ./*input [7:0]  */DATA_RECON_B0(out_b0),
    ./*input [7:0]  */DATA_RECON_R1(out_r1),
    ./*input [7:0]  */DATA_RECON_G1(out_g1),
    ./*input [7:0]  */DATA_RECON_B1(out_b1),
    ./*output  */DEC_DONE()
);
```

Test bench (top_system_tb.v)

- Test bench: the top system is connected to a display panel
- The top system includes: a master, ALU, multiplier, LCD Drive, Bus interconnect, BRAM and AHB BRAM
- LCD drive interface
  - Brightness adjustment
  - Frame buffer, forward RCT and backward RCT







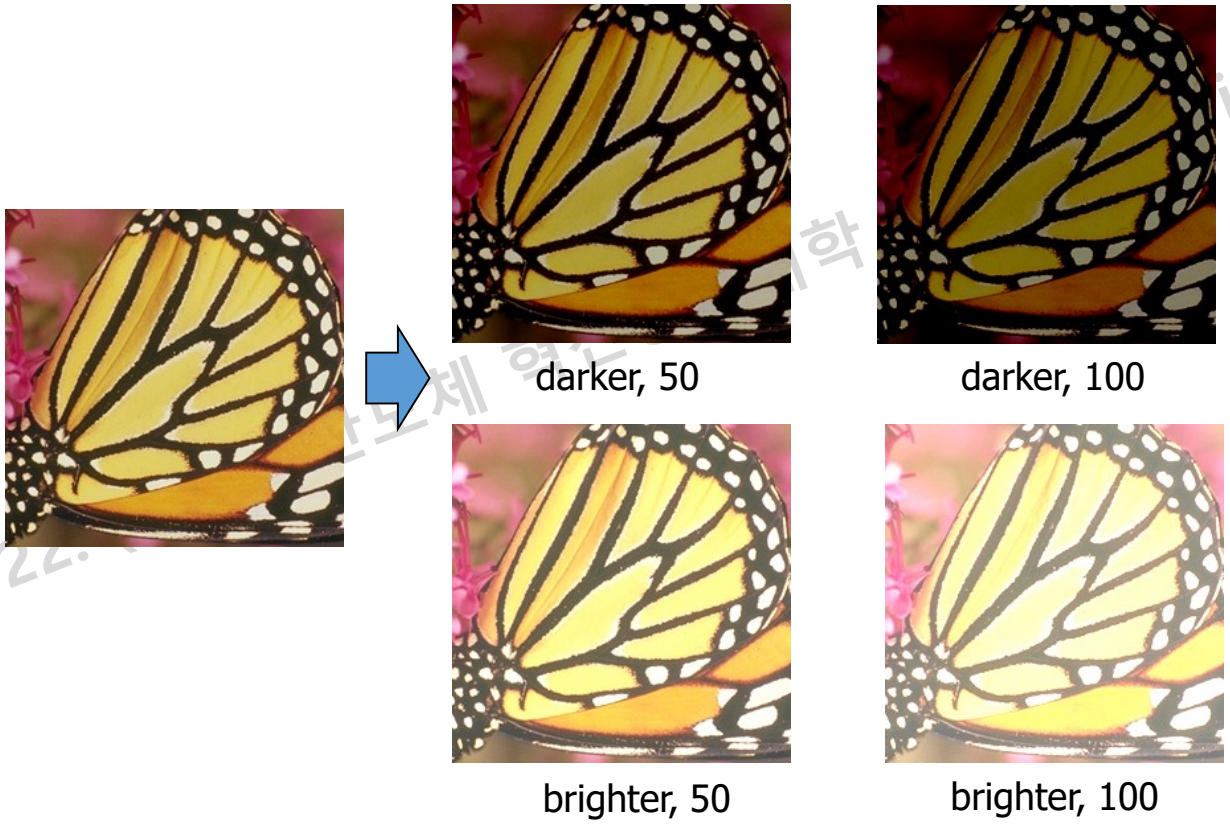
Experimental result (baseline)

- Complete all connections of LCD Drive and AHB BRAM.
- Do a simulation with time = 20 ms

```
u_frame_buffer(
    .clk(HCLK),
    .en(rct_o_valid),
    .addr({1'b0,q_bram_addr[W_WORD-1:1]}),
    .din(rct_o_pixel),
    .we(rct_o_valid),
    .dout(),
    .addr_dual_pixel(addr_dual_pixel),
    .dout_dual_pixel(out_rgb_pixel_dual)
);

lcd_inverse_rct u_inverse_rct
(
    ./*input */clk(HCLK),
    ./*input */rst_n(HRESETn),
    ./*input */in_valid(ctrl_data_run),
    ./*input signed [WAVE_PIX_W-1:0] */y0 (out_rgb_pixel_dual[ 0+:WAVE_PIX_W]),
    ./*input signed [WAVE_PIX_W-1:0] */cb0(out_rgb_pixel_dual[20+:WAVE_PIX_W]),
    ./*input signed [WAVE_PIX_W-1:0] */cr0(out_rgb_pixel_dual[30+:WAVE_PIX_W]),
    ./*input signed [WAVE_PIX_W-1:0] */y1 (/*Insert your code*/),
    ./*input signed [WAVE_PIX_W-1:0] */cb1(/*Insert your code*/),
    ./*input signed [WAVE_PIX_W-1:0] */cr1(/*Insert your code*/),
    ./*output reg */out_valid(recon_valid),
    ./*output [IMG_PIX_W-1:0] */r0(recon_data_R0),
    ./*output [IMG_PIX_W-1:0] */g0(recon_data_G0),
    ./*output [IMG_PIX_W-1:0] */b0(recon_data_B0),
    ./*output [IMG_PIX_W-1:0] */r1(recon_data_R1),
    ./*output [IMG_PIX_W-1:0] */g1(recon_data_G1),
    ./*output [IMG_PIX_W-1:0] */b1(recon_data_B1)
);
```

To do …

- Complete the missing codes
- Test the top system by running simulation in 20ms
- Show the simulation results at different options



Frame buffer

- Baseline: RGB W×H×3×8 = 24 WH
- Optimized: YCbCr + sampling (W×H/2)×4×10 = 20 WH
  - Reduce the frame buffer by 16.67% (=4/24).
- What is the cost? No free-lunch
  - Calculate the difference between two images (*demo_psnr_lcd.m*).
  - Root mean square error (rmse) = 0.6525
  - Peak Signal to Noise Ratio (PSNR) = 20×lg(255/rmse) = 51.84 (dB)

Baseline

Optimized

Difference