DISPLAY ADAPTOR

Venkateswara Raju Datla[#1], Suhani Vaishnav[#2]

[#]*Computer Engineering Department, San Jose State University*
San Jose California

[1]venkateswararaju.datla@sjsu.edu

[2]suhani.vaishnav@sjsu.edu

Abstract— **This document delivers how to construct a display adaptor. The display is one of the most crucial peripherals in a system because it establishes a link between the user and the system. The implementation is using Verilog and multiple modules are integrated and governed by the controller to perform the display operation.**

Keywords— **Clock, Pixel Counter, Address Counter, Line Counter, Buffers, Active Image, Display Controller, Vertical Blanking and Multiplexer.**

I. INTRODUCTION

The display is one of the most crucial peripherals in a system because it establishes a link between the user and the system. The format on LCD or LED consists of an active image surrounded by vertical and horizontal blanking regions.
These blanking areas are made out of black pixels. On the other hand, each pixel in the active image is composed of eight-bit wide Red (R), Green (G) and Blue (B) components. The basic operation consists of fetching a 24-bit pixel from the system memory and placing it in the active image area one at a time. The image data, on the other hand, is continuously fed to the display buffers, Buf0 and Buf1, by a 32-bit wide system bus, WData[31:0], as shown in **Fig1.**

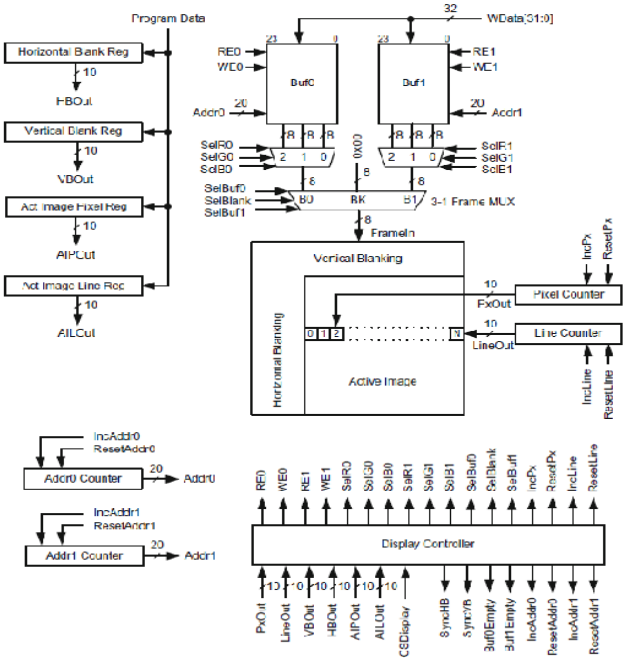

Fig1. The Display Adaptor unit data path

II. DESIGN

Several individual modules combine together to form a Display Adaptor. These individual modules are based on the data path shown and comprise of the line counter, address counter, buffers, multiplexers and controllers. These are then integrated into a top module. The unit is activated when a CSDisplay input is received form the system to engage the display. Following this, the display controller module first places the incoming pixels from the system bus to Buf0 (Fig1), and then transfers pixels from this buffer to the image frame. The controller also generates two timing attributes, SycnHB and SyncVB, to indicate start of the vertical blanking sections of the frame. This is done in order to synchronize the display adaptor with the monitor. Once active, the first component of the blanking pixel 0, 0x00, arrives at the frame in cycle 2. Within the same cycle, the SycnVB signal becomes logic 1, indicating the start of the vertical blanking for the frame. This continues until the end of the first vertical blanking line. The second blanking line follows the same pattern as the first one. Thus, the blanking section ends.

Following this the R-component(61 cycle) of the first image pixel comes into the image frame, followed by the G and the B components in cycles 64 and 67**,** respectively. Since the image data comes from the first display buffer, Buf0, the port assignment in the 3-1 MUX must be changed from port BK to port B0 by SelBuf0 = 1. Therefore, after the first image pixel is delivered to the frame, the address pointer for buffer0, Addr0, is incremented by one to be able to fetch the R-component of the next pixel from Buf0. After this the second line of the active image is delivered. Addr0 is incremented during this period to fetch pixels from Buf0 and form the second active image line. The rest of the image is delivered to the frame and a new frame is formed with SyncVB = 1. Afetr execution from Buf0, the data is written from file into Buf1 while it's blanking section is being executed and then the reading of active image from Buf1 initiates.

The controller is designed based on Moore type state machine in **Fig2.** It comprises of a string of states(39) , each responsible for delivering blank or active image pixels to the frame. The state machine keeps track of the pixel and line numbers in the frame and defines the boundaries between the blanking and the active image regions. Its functionality largely depends on the pixel and the line counter values. The state machine stays in the IDLE state until it is externally activated by CSDisplay = 1.

Once activated, the R-component of the first blank pixel enters the frame through the BK-port of the 3-1 MUX, and prompts SelBlank = 1 in cycle 2. This refers to the START0 state. In cycle 3, the G-component of the first blank pixel is delivered to the frame. This is shown as the VB0-G state in the state diagram. In this state, SelBlank = 1 in order to transmit the G-component of the first blank pixel to the frame. The B-component of the first blank pixel arrives in cycle 4, which corresponds to the VB0-B state. During this cycle, SelBlank = 1 and IncPx = 1 to increment the pixel counter by one. At this point, the controller checks if the end of the first vertical blanking line has been reached by forming PxOut = (HBOut + AIPOut – 2). Here, PxOut corresponds to the output of the pixel counter, HBOut corresponds to the number of horizontal blanking pixels in the Horizontal Blanking register, and AIPOut corresponds to the number of active image pixels in the Active Image Pixel register. Since the design doesnot have horizontal blanking, hence HBOut is not taken into consideration. If the end has not been reached, the machine keeps circling around the VB0-R, VB0-G and VB0-B states until PxOut becomes equal to (AIPOut – 2). During this period, every time the B-component of a blank pixel is delivered to the frame, the pixel counter increments by one. When the end point is detected, the state machine goes to the Reset VB0-R state where it delivers the R-component of the last blanking pixel that belongs to the first blanking line. However, the contents of the Vertical Blanking register, VBOut, needs to be checked prior to the start of the next vertical blanking line in case this register is programmed to have only one vertical blanking line. Therefore, while in the Reset VB0-G state, the line counter output, LineOut, is compared against (VBOut - 1). If the line counter output is less than (VBOut – 1), then the state machine first goes to the Reset VB0-B state. If LineOut is equal to (VBOut – 1), the state machine goes to the Switch0
VB-to-Active image state.

The state machine enters the R0 state to deliver the R-component of the first active image pixel from Buf0. In this state, port 0 of the 3-1 MUX (Fig1) at the output of Buf0 becomes active by SelR0 = 1, and port B0 of the 3-1 frame MUX becomes active by SelBuf0 = 1. The read enable input for Buf0 also stays at logic 1 by RE0 = 1. In next cycle the state machine goes to the G0 state where it delivers the G component of the first active image pixel to the frame. This cycle requires SelG0 = 1 to activate port 1 of the 3-1 MUX at the output of Buf0 while keeping SelBuf0 = 1 and RE0 = 1. Addr0 is also incremented in this state by IncAddr0 = 1. In cycle 67, the controller reaches the B0 state where it increments the pixel counter by IncPx = 1, selects port 2 of the 3-1 MUX at the output of Buf0 by SelB0 = 1, and maintains both SelBuf0 = 1 and RE0 = 1. In this state, the controller checks if the end of active image has been reached by comparing PxOut against (AIPOut – 2). If the controller finds PxOut < (AIPOut – 2), it goes back to the R0 state to retrieve more image pixels from Buf0. If the controller finds PxOut = (AIPOut – 2), it moves to the Reset R0 state in to deliver the last R-component of the image pixel, and generates SelR0 = 1, SelBuf0 = 1 and RE0 = 1. The state machine then moves to the Reset G0 state and the Reset B0 state. In the Reset B0 state, the controller resets the pixel counter by ResetPx = 1, increments the line counter by IncLine = 1, selects port 2 of the 3-1Buf0 MUX by SelB0 = 1, and keeps SelBuf0 = 1. While in this state, the controller checks to see if the active image is more than a single line or not, and compares the output of the line counter, LineOut, against (AILOut – 2). If the controller finds that LineOut < (AILOut – 2) it goes to R0 State. However, if the controller finds that LineOut = (AILOut – 2), it realizes that it will be processing the last line of the current frame it goes to the Last R1 state. Once the controller exhausts all the active image pixels in Buf0, it switches to Buf1 to construct the next image frame.

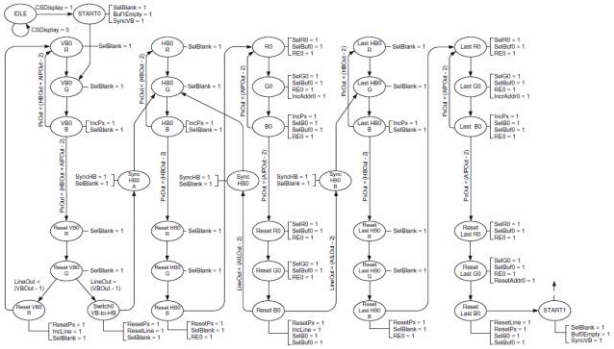

Fig2. State diagram for display controller

Conclusion

The functioning of the display adaptor and it's associated components is conveyed via this design using Verilog.
For test cases, a 10*10 image is used. The binary values are extracted using Hex Editor. After removing padding and the header, the values are stored into a file and the file is given as an input via SRAM, as WData to the unit. The output FrameIn of the unit is taken into a file and the header is added to the file. Upon adding the padding bits, the image can be viewed as a bmp file.

REFERENCES

[1] Ahmet Bindal, *Fundaments Of Computer Architecture and Design*, 5th ed

[2] https://en.wikipedia.org/wiki/BMP_file_format

[3] http://paulbourke.net/dataformats/bitmaps/

Appendix
Appendix-1

Verilog Code:

Display Adaptor Module(Top Module):

```
/*The top module for Display Adaptor*/
module topdisplay(clk, reset, CSDisplay,SyncVB,Buf0EMpty,Buf1Empty, Frameout);
output [7:0]Frameout;
input clk;
input reset;
input CSDisplay;
output SyncVB;
output Buf0EMpty;
output Buf1Empty;
/*Defining wires and registers*/
reg [7:0]Frameout;
wire [31:0] WDataw;
wire [19:0]Addrw0;
wire [19:0]Addrw1;
wire [7:0]mux0data0;
wire[7:0]mux0data1;
wire [7:0]mux0data2;
wire [7:0]mux1data0;
wire [7:0]mux1data1;
wire [7:0]mux1data2;
wire [7:0] muxbuf0out;
wire [7:0] muxbuf1out;
wire [9:0]Pxout;
wire[9:0]Lineout;
wire [7:0]frame;
wire IncAddr00;
wire IncAddr11;
wire [19:0]Addrr0;
wire [19:0]Addrr1;
/*Instantiating the multiple modules for data-path*/
counter c1(.clock(clk),.WE0(WE0),.reset(reset),.addrA(Addrw0));
buf1write bf1(.clock(clk),.WE1(WE1),.reset(reset),.addrB(Addrw1));
file_read fr(.WData(WDataw),.clock(clk));
/*Instantiating the Buffer 0*/
Buf0
buf0(.dataout00(mux0data0),.dataout01(mux0data1),.dataout02(mux0data2),.WData(WDataw),.addr0(Addrw0),.addr_read0(Addrr0),.WE0(WE0),.RE0(RE0),.clock(clk));
/*Instantiating the Buffer 1*/
Buf1
buf1(.dout00(mux1data0),.dout01(mux1data1),.dout02(mux1data2),.WData(WDataw),.addr1(Addrw1),.addr_read1(Addrr1),.WE1(WE1),.RE1(RE1),.clock(clk));
/*Instantiating the 3-1 Frame Multiplexer*/
Bigmux
bgmux(.FrameIn(frame),.SelBuf0(SelBuf0),.SelBlank(SelBlank),.SelBuf1(SelBuf1),.MuxBuf0(muxbuf0out),.MuxBuf1(muxbuf1out));
/*Instantiating the 3-1 Buffer 0 Multiplexer*/
Buf0mux buf0mx
(.MuxBuf0(muxbuf0out),.SelR0(SelR0),.SelG0(SelG0),.SelB0(SelB0),.IN0(mux0data2),.IN1(mux0data1),.IN2(mux0data0));
```

```
/*Instantiating the 3-1 Buffer 1 Multiplexer*/
Buf1mux
bf1mx(.MuxBuf1(muxbuf1out),.SelR1(SelR1),.SelG1(SelG1),.SelB1(SelB1),.IN0(mux1data2),.IN1(mux1data1),.IN2(mux1dat
a0));
/*Instantiating the Controller*/
statemachine fsm(.RE0(RE0),.WE0(WE0),.RE1(RE1),.WE1(WE1),.SelR0(SelR0),.SelG0(SelG0),.SelB0(SelB0),
.SelR1(SelR1),.SelG1(SelG1),.SelB1(SelB1),.SelBuf0(SelBuf0),
.SelBlank(SelBlank),.SelBuf1(SelBuf1),.IncPx(IncPx),.ResetPx(ResetPx),.IncLine(IncLine),.ResetLine(ResetLine),
.SyncVB(SyncVB),.Buf0EMpty(Buf0EMpty),.Buf1Empty(Buf1Empty),.IncAddr0(IncAddr00),.ResetAddr0(ResetAddr0),.Inc
Addr1(IncAddr11),.ResetAddr1(ResetAddr1),
.Pxout(Pxout),.Lineout(Lineout),.CSDisplay(CSDisplay),.clk(clk),.reset(reset));
/*Instantiating the Line Counter*/
linecounter l(.clock(clk),.ResetLine(ResetLine),.IncLine(IncLine),.LineOut(Lineout));
/*Instantiating the Pixel Counter */
pixelcounter p(.clock(clk), .ResetPx(ResetPx), .IncPx(IncPx),.PxOut(Pxout));
/*Instantiating the Address Counter 0*/
Addrcounter a0(.clock(clk),.ResetAddr(ResetAddr0),.IncAddr(IncAddr00),.Addr0(Addrr0));
/*Instantiating the Address Counter1*/
Addrcounter a1(.clock(clk),.ResetAddr(ResetAddr1),.IncAddr(IncAddr11),.Addr0(Addrr1));
/*Delivering output to the Frame buffer */
always @(posedge clk)
begin
Frameout <= frame;
end
endmodule
```

Test Bench of the Display Adaptor Module (Top Module) :

```
/*The test-bench for top module*/
module test_topdisplay;
wire [7:0]Frameout;
wire Buf0EMpty;
wire Buf1Empty;
wire SyncVB;
reg CSDisplay,clk,reset;
integer f,i;
wire [7:0]frame;
/*Instantiating the top module*/
topdisplay td1(clk,reset,CSDisplay,SyncVB,Buf0EMpty,Buf1Empty,Frameout);
/*Generating the clock signals*/
initial
begin
clk=0;
forever #5 clk=~clk;
end
/*Generating the CSDisplay*/
initial
begin
reset=1;
CSDisplay=0;
#4 reset=0;
#3
CSDisplay=1;
end
/*Writing the output to a text file*/
initial
begin
```

```
    f = $fopen("output.txt","w");
    @(posedge clk);
    for (i = 0; i<360; i=i+1) begin
      @(posedge clk);
      $fwrite(f,"%b\n",  Frameout);
    end
 $fclose(f);
  end
endmodule
```

Controller Module:

/*Module for the controller - implemented a Moore state machine using Case statements*/

```
module
statemachine(RE0,WE0,RE1,WE1,SelR0,SelG0,SelB0,SelR1,SelG1,SelB1,SelBuf0,SelBlank,SelBuf1,IncPx,ResetPx,IncLine,
ResetLine,SyncVB,Buf0EMpty,Buf1Empty,IncAddr0,ResetAddr0,IncAddr1,ResetAddr1,Pxout,Lineout,CSDisplay,clk,reset);
output
RE0,WE0,RE1,WE1,SelR0,SelG0,SelB0,SelR1,SelG1,SelB1,SelBuf0,SelBlank,SelBuf1,IncPx,ResetPx,IncLine,ResetLine,Syn
cVB,Buf0EMpty,Buf1Empty,IncAddr0,ResetAddr0,IncAddr1,ResetAddr1;
reg RE0,
WE0,RE1,WE1,SelR0,SelG0,SelB0,SelR1,SelG1,SelB1,SelBuf0,SelBlank,SelBuf1,IncPx,ResetPx,IncLine,ResetLine,SyncVB,
Buf0EMpty,Buf1Empty,IncAddr0,ResetAddr0,IncAddr1,ResetAddr1;
input [9:0]Pxout;
input [9:0]Lineout;
reg [9:0]VBout;
reg [9:0] HBout;
reg [9:0]AIPout;
reg [9:0]AILout;
input CSDisplay;
input clk;
input reset;
wire [9:0]Pxout;
wire[9:0]Lineout;
```

/*Instantiating the Address, Pixel and Line counters*/

```
linecounter l(.clock(clk),.ResetLine(ResetLine),.IncLine(IncLine),.LineOut(Lineout));
pixelcounter p(.clock(clk), .ResetPx(ResetPx), .IncPx(IncPx),.PxOut(Pxout));
Addrcounter a0(.clock(clk),.ResetAddr(ResetAddr0),.IncAddr(IncAddr0));
Addrcounter a1(.clock(clk),.ResetAddr(ResetAddr1),.IncAddr(IncAddr1));
```

/*Defining the local parameters for all the states*/

```
localparam [4:0] // for 39 states : size_state = 4:0
    Start0 = 5'd0, VB0R = 5'd1, VB0G= 5'd2, VB0B = 5'd3, RVB0R = 5'd4, RVB0G = 5'd5, RVB0B = 5'd6, SWTCH0= 5'd7,
R0 = 5'd8, G0 = 5'd9, B0 = 5'd10,
 RR0 = 5'd11, RG0 = 5'd12, RB0 = 5'd13, LR0 = 5'd14, LG0 = 5'd15, LB0 = 5'd16, RLR0 = 5'd17, RLG0 = 5'd18, RLB0 =
5'd19,
Start1 = 5'd20, VB1R = 5'd21, VB1G = 5'd22, VB1B = 5'd23, RVB1R  = 5'd24, RVB1G = 5'd25, RVB1B = 5'd26, SWTCH1 =
5'd27, R1= 5'd28, G1=5'd29, B1= 5'd30,
RR1 = 5'd31, RG1 = 5'd32, RB1 = 5'd33, LR1 = 5'd34, LG1 = 5'd35, LB1 = 5'd36, RLR1 = 5'd37, RLG1 = 5'd38, RLB1=5'd39;
reg[4:0] state_reg, state_next;
```

/*Assigning initial value for different parameters*/

```
initial
begin
state_reg <= Start0;
assign VBout = 10'b10;
assign AIPout = 10'b1010;//currently 10, change to 100 for 100 pixel
assign AILout = 10'b1010; //currently 10, change to 100 for 100 pixel
assign HBout = 10'b0;
RE1=0;
```

```
RE0=0;
IncPx = 0;
IncLine=0;
SelBuf0=0;
SelBuf1=0;
SelR0=0;
SelG0=0;
SelB0=0;
SelR1=0;
SelG1=0;
SelB1=0;
SelBlank=0;
IncPx=0;
SyncVB=0;
ResetPx=0;
IncAddr0=0;
IncAddr1=0;
ResetAddr0=0;
ResetAddr1=0;
IncLine=0;
ResetLine=0;
Buf0EMpty=0;
Buf1Empty = 0;
end
/*Defining the conditions for reset and state transition*/
always @(posedge clk,posedge reset)
begin
   if (reset) begin
      state_reg <= Start0;
   end
   else begin
      state_reg <= state_next;
   end
end
/*Defining the Case statement for different states and their parameters*/
always @(state_reg, CSDisplay)
begin
  state_next = state_reg; // default state_next if(CSDiplay==1)
case({state_reg,CSDisplay})
/*The Start state for Buffer 0*/
      {Start0,1'b1} : begin
                        RE1=0;
                        RE0=0;
                        IncPx = 0;
                        IncLine=0;
                        SelBuf0=0;
                        SelBuf1=0;
                        SelR0=0;
                        SelG0=0;
                        SelB0=0;
                        SelR1=0;
                        SelG1=0;
                        SelB1=0;
                        SelBlank=0;
                        WE0=1;
                        WE1=1;
                        @(posedge clk);
```

```
                begin
                SelBlank=1;
                Buf1Empty=1;
                SyncVB=1;
                end
         state_next = VB0G;
                end
/*Start of Vertical Blanking*/
        {VB0R, 1'b1}:begin
                IncPx = 0;
                        IncLine=0;
                        ResetPx=0;
                        ResetLine=0;
                        SyncVB=0;
                        SelBlank=1;
                        Buf1Empty=1;
                        state_next = VB0G;
                        end
      {VB0G, 1'b1} : begin
                SelBlank=1;
                Buf1Empty=1;
                SyncVB=0;
                IncPx = 0;
         state_next = VB0B;
        end
      {VB0B, 1'b1} : begin
                        SelBlank=1;
                        ResetPx=0;
                        IncPx = 1; //increment pixel counter by 1
                        if(Pxout<AIPout-2) //HBout + AIPout-2
                        state_next= VB0R;
                        else
                        state_next = RVB0R;
                        end
/*Last Pixel of the Blanking line  */
        {RVB0R, 1'b1} : begin
                        SelBlank=1;
                        state_next=RVB0G;
                        IncPx = 0;
                        end

        {RVB0G, 1'b1} : begin
                        SelBlank = 1;
                        IncPx = 0;
                        if(Lineout<VBout-1)
                        state_next=RVB0B;
                        else
                        state_next=SWTCH0;
                        end

        {RVB0B, 1'b1} : begin
                        SelBlank=1;
                        ResetPx=1;
                        ResetLine=0;
                        IncLine=1;
                        state_next = VB0R;
                        end
```

```
/*Switch to active image once vertical blanking is complete*/
        {SWTCH0,1'b1}: begin
                        ResetPx=1;
                        ResetLine=1;
                        SelBlank=1;
                        WE0=0;
                        RE0=1;
                        state_next=R0;
                        end
/*Reading the active image starts */
        {R0,1'b1}:      begin
                        IncPx=0;
                        IncLine=0;
                        ResetPx=0;
                        ResetLine=0;
                        SelR0=1;
                        SelB0=0;
                        SelBlank=0;
                        SelBuf0=1;
                        RE0=1;
                        state_next=G0;
                        end
        {G0,1'b1}:      begin
                        SelR0=0;
                        SelG0=1;
                        SelBuf0=1;
                        RE0=1;
                        ResetAddr0=0;
                        IncAddr0=1;
                        state_next=B0;
                        end

        {B0,1'b1}:      begin
                        ResetPx=0;
                        IncPx=1;
                        IncAddr0=0;
                        SelG0=0;
                        SelB0=1;
                        SelBuf0=1;
                        RE0=1;
                        if(Pxout<AIPout-2)
                        state_next=R0;
                        else
                        state_next=RR0;
                        end
/*Last pixel of the line*/
        {RR0,1'b1}:     begin
                        IncPx=0;
                        SelB0=0;
                        SelR0=1;
                        SelBuf0=1;
                        RE0=1;
                        state_next=RG0;
                        end
        {RG0,1'b1}:     begin
                        SelR0=0;
                        SelG0=1;
```

```
                    SelBuf0=1;
                    IncAddr0=1;
                    RE0=1;
                    state_next=RB0;
                    end
        {RB0,1'b1}:     begin
                    ResetPx=1;
                    ResetLine=0;
                    IncLine=1;
                    IncAddr0=0;
                    SelG0=0;
                    SelB0=1;
                    SelBuf0=1;
                    if(Lineout<AILout-2)
                    state_next=R0;
                    else
                    state_next=LR0;
                    end
        {LR0,1'b1}: begin
                    SelB0=0;
                    SelR0=1;
                    ResetPx=0;
                    IncPx=0;
                    IncLine=0;
                    SelBuf0=1;
                    RE0=1;
                    state_next=LG0;
                    end
        {LG0,1'b1}:begin
                    SelR0=0;
                    SelG0=1;
                    SelBuf0=1;
                    RE0=1;
                    ResetAddr0=0;
                    IncAddr0=1;
                    state_next=LB0;
                            end
        {LB0,1'b1}:begin
                    ResetPx=0;
                    IncPx=1;
                    SelG0=0;
                    SelB0=1;
                    IncAddr0=0;
                    SelBuf0=1;
                    RE0=1;
                    if(Pxout<AIPout-2)
                    state_next=LR0;
                    else
                    state_next=RLR0;
                    end
/*Last Pixel of the last line in active image*/
        {RLR0,1'b1}:    begin
                    IncPx=0;
                    SelB0=0;
                    SelR0=1;
                    SelBuf0=1;
                    RE0=1;
```

```
                        state_next=RLG0;
                        end
        {RLG0,1'b1}:    begin
                        SelR0=0;
                        SelG0=1;
                        SelBuf0=1;
                        RE0=1;
                        ResetAddr0=1;
                        state_next=RLB0;
                        end
        {RLB0,1'b1}:    begin
                        ResetLine=1;
                        ResetPx=1;
                        ResetAddr0=0;
                        SelG0=0;
                        SelB0=1;
                        SelBuf0=1;
                        state_next = Start1;
                        end
/*Buffer0 execution complete*/
/*Buffer1 execution starts*/
        {Start1,1'b1} : begin
                        IncPx = 0;
                        IncLine=0;
                        ResetPx=0;
                        ResetLine=0;
                        SelBuf0=0;
                        SelBuf1=0;
                        SelR0=0;
                        SelG0=0;
                        SelB0=0;
                        SelR1=0;
                        SelG1=0;
                        SelB1=0;
                        RE0=0;
                        SelBlank=1;
                        Buf0EMpty=1;
                        SyncVB=1;
                        state_next = VB1G;
                        end
/*Vertical Blanking starts*/
        {VB1R, 1'b1}:   begin
                IncPx = 0;
                        IncLine=0;
                        ResetPx=0;
                        ResetLine=0;
                        SyncVB=0;
                        SelBlank=1;
                        Buf0EMpty=1;
                        state_next = VB1G;
                        end
      {VB1G, 1'b1} : begin
                SelBlank=1;
                        Buf0EMpty=1;
                        SyncVB=0;
                        IncPx = 0;
                state_next = VB1B;
```

```
                end
        {VB1B, 1'b1} : begin
                                SelBlank=1;
                                ResetPx=0;
                                IncPx = 1; //increment pixel counter by 1
                                if(Pxout<AIPout-2) //HBout + AIPout-2
                                state_next= VB1R;
                                else
                                state_next = RVB1R;
                                end
                {RVB1R, 1'b1} : begin
                                SelBlank=1;
                                state_next=RVB1G;
                                IncPx = 0;
                                end
                {RVB1G, 1'b1} : begin
                                SelBlank = 1;
                                IncPx = 0;
                                if(Lineout<VBout-1)
                                state_next=RVB1B;
                                else
                                state_next=SWTCH1;
                                end
                {RVB1B, 1'b1} : begin
                                SelBlank=1;
                                ResetPx=1;
                                ResetLine=0;
                                IncLine=1;
                                state_next = VB1R;
                                end


/*Vertical blanking ends and switch to active image*/
                {SWTCH1,1'b1}: begin
                                ResetPx=1;
                                ResetLine=1;
                                SelBlank=1;
                                WE1=0;
                                RE1=1;
                                state_next=R1;
                                end
/*Active image starts*/
                {R1,1'b1}:      begin
                                IncPx=0;
                                IncLine=0;
                                ResetPx=0;
                                ResetLine=0;
                                SelB1=0;
                                SelR1=1;
                                SelBlank=0;
                                SelBuf1=1;
                                RE1=1;
                                state_next=G1;
                                end
                {G1,1'b1}:      begin
                                SelR1=0;
                                SelG1=1;
```

```
                SelBuf1=1;
                RE1=1;
                ResetAddr1=0;
                IncAddr1=1;
                state_next=B1;
                end
{B1,1'b1}:      begin
                ResetPx=0;
                IncPx=1;
                IncAddr1=0;
                SelG1=0;
                SelB1=1;
                SelBuf1=1;
                RE1=1;
                if(Pxout<AIPout-2)
                state_next=R1;
                else
                state_next=RR1;
                end
{RR1,1'b1}:     begin
                IncPx=0;
                SelB1=0;
                SelR1=1;
                SelBuf1=1;
                RE1=1;
                state_next=RG1;
                end
{RG1,1'b1}:     begin
                SelR1=0;
                SelG1=1;
                IncAddr1=1;
                SelBuf1=1;
                RE1=1;
                state_next=RB1;
                end
{RB1,1'b1}:     begin
                RE1=1;
                ResetPx=1;
                ResetLine=0;
                IncLine=1;
                IncAddr1=0;
                SelG1=0;
                SelB1=1;
                SelBuf1=1;
                if(Lineout<AILout-2)
                state_next=R1;
                else
                state_next=LR1;
                        end
{LR1,1'b1}: begin
                SelB1=0;
                SelR1=1;
                ResetPx=0;
                IncPx=0;
                IncLine=0;
                SelBuf1=1;
                RE1=1;
```

```
                    state_next=LG1;
                    end
        {LG1,1'b1}:begin
                    SelR1=0;
                    SelG1=1;
                    SelBuf1=1;
                    RE1=1;
                    ResetAddr1=0;
                    IncAddr1=1;
                    state_next=LB1;
                    end
        {LB1,1'b1}:begin
                    ResetPx=0;
                    IncPx=1;
                    SelG1=0;
                    SelB1=1;
                    IncAddr1=0;
                    SelBuf1=1;
                    RE1=1;
                    if(Pxout<AIPout-2)
                    state_next=LR1;
                    else
                    state_next=RLR1;
                    end
/*Last pixel of the last line*/
        {RLR1,1'b1}:    begin
                    IncPx=0;
                    SelB1=0;
                    SelR1=1;
                    SelBuf1=1;
                    RE1=1;
                    state_next=RLG1;
                    end
        {RLG1,1'b1}:    begin
                    SelR1=0;
                    SelG1=1;
                    SelBuf1=1;
                    RE1=1;
                    ResetAddr1=1;
                    state_next=RLB1;
                    end
        {RLB1,1'b1}:    begin
                    ResetLine=1;
                    ResetPx=1;
                    ResetAddr1=0;
                    SelG1=0;
                    SelB1=1;
                    SelBuf1=1;
                    state_next = Start0;    /*Buffer1 execution is complete and routed to the start of Buffer0.*/
                    end
   endcase
end
endmodule
```

Buffer 0 Multiplexer:

```
 /*Multiplexer module for Buffer0*/
```

```
module Buf0mux(MuxBuf0,SelR0,SelG0,SelB0,IN0,IN1,IN2);
input SelR0,SelG0,SelB0;
input [7:0]IN0;
input [7:0]IN1;
input [7:0]IN2;
output [7:0]MuxBuf0;
reg [7:0]MuxBuf0;
/*The select signals are received from the controller and accordingly the multiplexer gives output*/
always @(MuxBuf0 or SelR0 or SelG0 or SelB0 or IN0 or IN1 or IN2)
begin
  if(SelR0==1 && SelG0==0 && SelB0==0)
    MuxBuf0=IN0;
  else if( SelR0==0 && SelG0==1 && SelB0==0)
    MuxBuf0=IN1;
else if(SelR0==0 && SelG0==0 && SelB0==1)
MuxBuf0 = IN2;
end
endmodule
```

Buffer 1 Multiplexer:

```
 /*Mutliplexer module for Buffer1*/
module Buf1mux(MuxBuf1,SelR1,SelG1,SelB1,IN0,IN1,IN2);
input SelR1,SelG1,SelB1;
input [7:0]IN0;
input [7:0]IN1;
input [7:0]IN2;
output [7:0]MuxBuf1;
reg [7:0]MuxBuf1;
/*The select signals are received from the controller and accordingly the multiplexer gives output*/
always @(MuxBuf1 or SelR1 or SelG1 or SelB1 or IN0 or IN1 or IN2)
begin
  if(SelR1==1 && SelG1==0 && SelB1==0)
    MuxBuf1=IN0;
  else if( SelR1==0 && SelG1==1 && SelB1==0)
    MuxBuf1=IN1;
else if(SelR1==0 && SelG1==0 && SelB1==1)
MuxBuf1 = IN2;
end
endmodule
```

Line Counter Module:

```
/*Module for line counter*/
module linecounter(clock,ResetLine,IncLine,LineOut);
input clock;
input ResetLine;
input IncLine;
output [9:0] LineOut;
reg [9:0] LineOut;
initial
begin
 LineOut=10'b0;
end
always@(posedge clock)
begin
if(ResetLine==1'b1)
```

```
LineOut <= 10'b0;
/*Line counter to increment only when IncLine signal received from the controller */
else if (ResetLine==0 && IncLine==1)
LineOut <= LineOut + 10'b1;
end
endmodule
```

File Read Module:

```
module file_read(WData, clock);
/*Module to read the data from file and load it into memory*/
input clock;
reg [100:1] file_name;
output reg [31:0] WData;
reg [31:0] RAM [199:0];
reg[7:0] counter;
integer file;
integer i = 0;
initial begin
        $readmemb("image.txt", RAM);
        counter <= 8'b11111111;
end
always@(clock)
begin
if(counter == 8'b11001000) counter <= 8'b11111111;
else
begin
        counter = counter + 1;
        WData = RAM[counter];
end
end
endmodule
```

Counter module for to write into Buffer1:

```
/*Address counter module to write the data into the Buffer1*/
module buf1write(clock,WE1,reset,addrB);
input clock;
input reset;
input WE1;
output [19:0] addrB;
reg [19:0] addrB;
initial
begin
addrB = 20'b0;
end
always@(posedge clock or negedge clock)
begin
if(reset == 1'b1)
begin
addrB <= 20'b000;
end
/*Write into the buffer only if write enable signal is high*/
else if(WE1 == 1'b1 && reset ==0)
begin
addrB <= addrB + 20'b001;
end
end
endmodule
```

Pixel Counter Module:

```
/*Module for Pixel Counter*/
module pixelcounter(clock, ResetPx, IncPx,PxOut);
input clock;
input ResetPx;
input IncPx;
output [9:0] PxOut;
reg [9:0] PxOut;
initial
begin
 PxOut=10'b0;
end
always@(posedge clock)
begin
if(ResetPx==1'b1)
PxOut <= 10'b0;
/*Pixelcounter to increment only when IncPx signal recevied from controller */
else if (ResetPx==0 && IncPx==1)
PxOut <= PxOut + 10'b1;
end
endmodule
```

Counter module for to write into Buffer0:

```
/*Address counter module to write the data into the Buffer0*/
module counter(clock,WE0,reset,addrA);
input clock;
input reset;
input WE0;
output [19:0] addrA;
reg [19:0] addrA;

initial
begin
addrA = 20'b0;
end
always@(posedge clock or negedge clock)
begin
if(reset == 1'b1)
begin
addrA <= 20'b000;
end
/*Write into the buffer only if write enable signal is high*/
else if(WE0 == 1'b1 && reset ==0)
begin
addrA <= addrA + 20'b001;
end
end
endmodule
```

Buffer 1 Module:

```
/*Buffer 1 module*/
module Buf1(dataout1,dout00,dout01,dout02,WData,addr1,addr_read1,WE1,RE1,clock);
output [7:0]dout00;
output [7:0]dout01;
output [7:0]dout02;
output [23:0]dataout1;
```

```
input [31:0]WData;
input [19:0]addr1;
input [19:0]addr_read1;
input WE1,RE1,clock;
reg [23:0]SRAM[99:0];//9999 for 100, 99 for 10 pixel
reg [7:0]dout00;
reg [7:0]dout01;
reg [7:0]dout02;
reg [23:0]dataout1;
/*Writing into the Buffer1 */
always@(posedge clock or negedge clock)
begin
if(WE1==1)
SRAM[addr1]<=WData;
end
/*Reading from Buffer1 */
always@ (posedge clock)
begin
if(RE1==1)
begin
dout00=SRAM[addr_read1][7:0];
dout01=SRAM[addr_read1][15:8];
dout02=SRAM[addr_read1][23:16];
end
end
endmodule
```

Buffer 0 Module:

```
/*Buffer 0 module*/
module Buf0(dataout,dataout00,dataout01,dataout02,WData,addr0,addr_read0,WE0,RE0,clock);
output [7:0]dataout00;
output [7:0]dataout01;
output [7:0]dataout02;
output [23:0]dataout;
input [31:0]WData;
input [19:0]addr0;
input [19:0]addr_read0;
input WE0,RE0,clock;
reg [23:0]SRAM[99:0;
reg [7:0]dataout00;
reg [7:0]dataout01;
reg [7:0]dataout02;
reg [23:0]dataout;
/*Writing the data into Buffer0 */
always@(posedge clock or negedge clock)
begin
if(WE0==1)
SRAM[addr0]<=WData;
end
/*Reading the data from Buffer0 */
always@ (posedge clock)
begin
if(RE0==1)
begin
dataout00=SRAM[addr_read0][7:0];
dataout01=SRAM[addr_read0][15:8];
dataout02=SRAM[addr_read0][23:16];
```

```
end
end
endmodule
```

3-1 Frame Multiplexer:

```
/*Multiplexer module to select between the buffers 0 & 1 and blanking*/
module Bigmux(FrameIn,SelBuf0,SelBlank,SelBuf1,MuxBuf0,MuxBuf1);
input SelBuf0,SelBlank,SelBuf1;
input [7:0]MuxBuf0;
input [7:0]MuxBuf1;
output [7:0]FrameIn;
reg [7:0]FrameIn;
/*The select signals are recieved from the controller and accordingly the multiplexer gives output */
always @( SelBuf0 or SelBlank or SelBuf1 or MuxBuf0 or MuxBuf1 )
begin
  if(SelBuf0==1 && SelBuf1==0 && SelBlank==0)
    FrameIn = MuxBuf0;
  else if( SelBuf0==0 && SelBuf1==1 && SelBlank==0)
    FrameIn = MuxBuf1;
else if(SelBuf0==0 && SelBuf1==0 && SelBlank==1)
FrameIn = 8'b0;
end
endmodule
```

Address Counter:

```
/*Address counter module for providing addresses to read from Buffer 0 & Buffer 1.*/
module Addrcounter(clock, ResetAddr,IncAddr,Addr0);
input clock;
input ResetAddr;
input IncAddr;
output [19:0] Addr0;
reg [19:0] Addr0;
initial
begin
 Addr0=20'b0;
end
always@(posedge clock)
begin
if(ResetAddr==1'b1)
Addr0 <= 10'b0;
/*Addrress counter to be incremented only when IncAddr signal received from the controller*/
else if (ResetAddr==0 && IncAddr==1)
Addr0 <= Addr0 + 10'b1;
end
endmodule
```

Appendix- II

Waveforms (attached)
Relevant explanation for the attached waveforms:

1) **Vertical blanking for Buffer0** – The waveform shows the vertical blanking of one line for Buffer0. The CSDisplay becomes high and the blanking pixels are visible in the FrameIn parameter.
   Also the SelBlank goes high while in blanking state. The pixel counter (Pxout) counts from 0-9 for 10 pixels.

2) **Active Image for Buffer0** – The waveform shoes the active image data being written in the frame buffer (FrameIn). The RE0(read enable for Buffer0) is high and according to the pixel coming in the SelR0/G0/B0 lines are activated. The SelBlank goes low once reading starts and SelBuf0 signal is high indication data is being read from Buffer0.

3) **Vertical blanking for Buffer1** – The waveform shows the vertical blanking of one line for Buffer1. The CSDisplay becomes high and the blanking pixels are visible in the FrameIn parameter.
   Also the SelBlank goes high while in blanking state. The pixel counter (Pxout) counts from 0-9 for 10 pixels.

4) **Active Image for Buffer1** – The waveform shoes the active image data being written in the frame buffer (FrameIn). The RE1 (Read enable for Buffer1) is high and according to the pixels coming in, the SelR0/G0/B0 lines are activated. The SelBlank goes low once reading starts and SelBuf0 signal is high indication data is being read from Buffer0.