# VGA Assignment

Design a Verilog module using state for a display adapter to retrieve and display an image following the XGA standard. The adapter must support a resolution of 1024x768 pixels at 60Hz and a color depth of 24 bits.

Submitted by: **Muhammad Tariq Waseem**
Roll No: **EL-21056**
Department: **Electronics Engineering**
IC Design Class Section: **A**

The design and implementation of a VGA (Video Graphics Array) adapter is a crucial component in developing display systems for a variety of applications, from computer monitors to embedded displays. This report provides a detailed analysis of the VGA adapter module presented, highlighting the key design choices, functionality, and compliance with the XGA (Extended Graphics Array) standard.

The VGA adapter module is responsible for generating the necessary timing signals, retrieving pixel data from memory, and outputting the video signal to drive a display with 1024x768 resolution at a 60Hz refresh rate. The report delves into the architectural decisions made to achieve this functionality, ensuring the module meets the specific requirements outlined in the project specifications.

By examining the logic, state machine, and interfacing components of the VGA adapter, this report aims to provide a comprehensive understanding of the design and its implementation. The report will discuss the rationale behind the chosen memory interface, the synchronization of horizontal and vertical timing, and the strategies employed to retrieve and output the pixel data in a seamless manner.

## VGA Adaptor:

The VGA module is responsible for generating the necessary signals and timing to drive a VGA display with 1024x768 resolution at 60Hz refresh rate. The key aspects of the design are:

### Required Frequency

The VGA standard for 1024x768 resolution at 60Hz refresh rate has the following timing parameters:

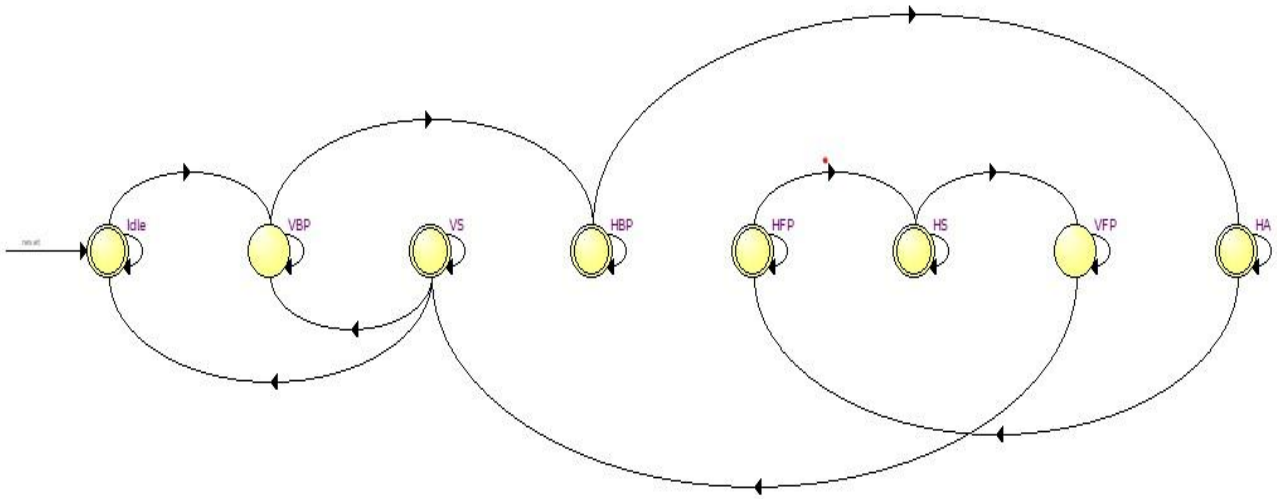
- Horizontal Active Pixels (H_ACTIVE): 1024
- Horizontal Front Porch (H_FRONT_PORCH): 24
- Horizontal Sync Pulse (H_SYNC): 136

- Horizontal Back Porch (H_BACK_PORCH): 160
- Total Horizontal Pixels (H_TOTAL): 1024 + 24 + 136 + 160 = 1344
- Vertical Active Lines (V_ACTIVE): 768
- Vertical Front Porch (V_FRONT_PORCH): 3
- Vertical Sync Pulse (V_SYNC): 6
- Vertical Back Porch (V_BACK_PORCH): 29
- Total Vertical Lines (V_TOTAL): 768 + 3 + 6 + 29 = 806

The required frequency can be calculated as:

Required Frequency = H_TOTAL × V_TOTAL × Refresh Rate Required Frequency = 1344 × 806 × 60 Hz **Required Frequency = 65 MHz**

## State Diagram




## Explantion

The state will remain at Idle until a start signal is given after the start signal it jumps to the Vertical Back Porch(**VBP**) and remain there for next 29 clk positive edge then it jump to the Horizontal Back Porch (**HBP**) where it stay for 160 positive edge then jump to the Horizontal Active (**HA**) at this states it start fetching the data from the memory 24 bits in every cycle to present a pixel. This 24 bits include 8 bit represent Red colour 8 representing Green and 8 Blue. After 1024x3 bytes of data fetching it jumps to the Horizontal front Porch state (**HFP**) here it stay for 24 clock cycle then jumps to Horizontal Sync (**HS**) here it generates a horizontal sync pulse for the VGA port for 134 cycles then after completing the Horizontal line it jumps to the vertical Front porch(**VFP)** after 3 cycles it goes to Vertical Sync (**VS**) to generate a sync pulse for VGA port for 6 cycles then this whole cycle will repeat again this will all repeat for 768 time for every horizontal line and after all that it comes back to its ideal state.
The top module code and the memory interface code and the RTL view and simulation result is attached to the end of the document.

## Explanation for selecting SRAM over BRAM:

1. **Storage Capacity:** The design requires storing an image with a resolution of 1024x768 pixels and 24-bit color depth. This amounts to a total of 1,966,080 bytes of pixel data. SRAM provides a larger storage capacity compared to on-chip BRAM, allowing the entire image to be stored without the need for external memory.
2. **Scalability:** As the resolution or color depth of the image increases, the storage requirements also grow. SRAM can be easily scaled up by using wider and/or deeper memory configurations, making it more suitable for handling higher-resolution or higher-bit-depth images in the future.

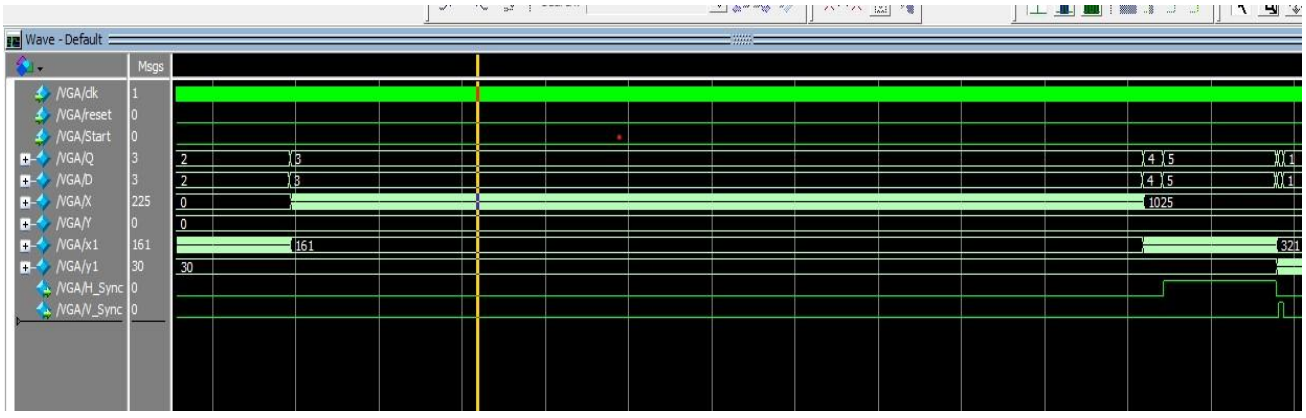
### SRAM Selection:

For the given design specifications of a 65MHz operating frequency and a minimum SRAM access time of 10-15ns, a 32-bit wide SRAM would be the optimal choice. This allows the VGA adapter to fetch 4 bytes of pixel data per memory access, reducing the overall memory bandwidth requirements and simplifying the memory interface logic. There is one more way a normal way to do this using a byte wide memory and connect three memory in parallel to reduce the fetching time and run it on 65Mhz.

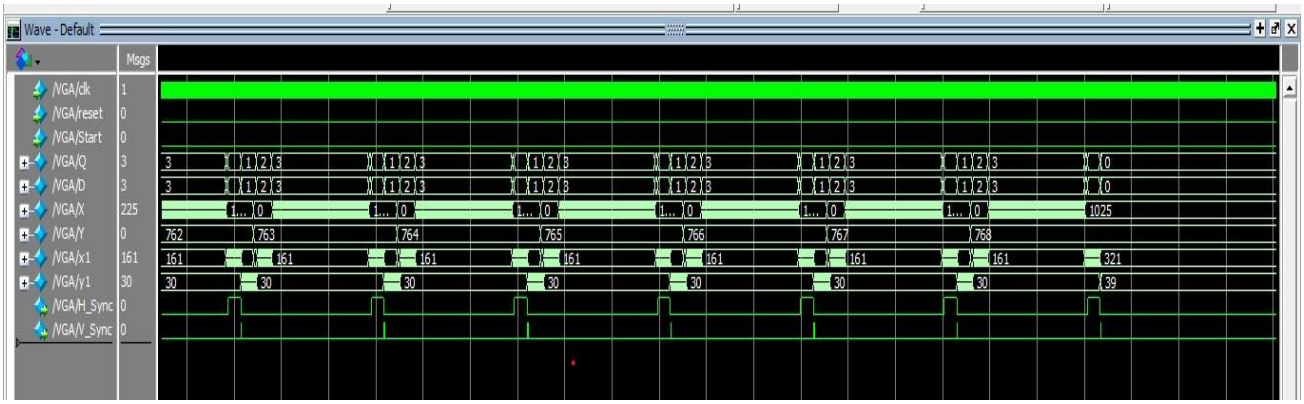
One suitable SRAM option for this design is the IS61LV25616 from ISSI. This is a 32-bit wide, 16Mb SRAM with a maximum access time of 10ns, which meets the design's timing requirements. The 16Mb capacity is more than sufficient to store the 1024x768x24-bit image, with additional space for potential future expansions.

By using a 32-bit wide SRAM, the design avoids the need for parallel SRAM chips, which would have increased the overall complexity of the memory interface and control logic. The 32-bit data width provides a good balance between memory access speed, storage capacity, and implementation complexity for this VGA adapter design.

### Simulation Result




Simulation of a Single Horizontal line




Simulation of Multiple Horizontal line ending at 768 decimal value

## SRAM Interface module

```
module SRAM #(
    parameter ADDR_WIDTH = 18,
    parameter DATA_WIDTH = 16
) (
    input clk,
    input reset,

    // SRAM interface
    output reg [ADDR_WIDTH-1:0] sram_addr,
    input wire [DATA_WIDTH-1:0] sram_data,
    output reg sram_ce_n,
    output reg sram_we_n,
    output reg sram_oe_n,

    // User interface
    input read_enable,
    input write_enable,
    input [ADDR_WIDTH-1:0] user_addr,
    output reg [DATA_WIDTH-1:0] user_read_data
);
    reg [DATA_WIDTH-1:0] sram_read_data;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            sram_addr <= 0;
            sram_ce_n <= 1;
            sram_we_n <= 0;
            sram_oe_n <= 1;
            user_read_data <= 0;
        end else begin
            if (read_enable) begin
                sram_addr <= user_addr;
                sram_ce_n <= 0;
                sram_we_n <= 0;
                sram_oe_n <= 0;
                user_read_data <= sram_read_data;
            end else begin
                sram_ce_n <= 1;
                sram_we_n <= 0;
                sram_oe_n <= 1;
            end
        end
    end
    // SRAM read data capture
    always @(posedge clk) begin
        sram_read_data <= sram_data;
    end
Endmodule
```

## VGA Code:

```
module VGA(clk,reset,Start,VR,VG,VB,GPins,H_Sync,V_Sync);
input clk, reset,Start;
output [7:0]VR,VG,VB;
output reg GPins,H_Sync,V_Sync;

parameter H_ACTIVE = 1024 , H_SYNC = 136 , H_BACK_PORCH = 160 , H_FRONT_PORCH = 24 , V_ACTIVE =
768,
        V_SYNC = 6, V_BACK_PORCH = 29, V_FRONT_PORCH = 3;
parameter Idle = 3'd0 , VBP = 3'd1 , HBP = 3'd2 , HA = 3'd3 , HFP = 3'd4 ,
        HS = 3'd5 , VFP = 3'd6 , VS = 3'd7;

reg [2:0]D,Q;
reg [10:0]X,Y;
reg [8:0]x1,y1;
reg [10:0]A;
reg a,b,c,d,RD;
reg [19:0]ADD;
wire [19:0]Addr;
wire [31:0]Data;
wire CE,OE,W;
wire [31:0]DATA;

always@(posedge clk or posedge reset) begin
          if((reset == 1'b1)) begin
                    X <= 11'd0;
                    Y <= 11'd0;
                    x1 <= 9'd0;
                    y1 <= 9'd0;
    ADD <= 20'd0;
          end
          else if(y1 == V_SYNC + V_FRONT_PORCH + H_BACK_PORCH) begin
            X <= 11'd0;
                    x1 <= 9'd0;
                    y1 <= 9'd0;
                    Y <= Y + 1;
                    ADD <= ADD + 20'd1024;
          end
          else begin
            if(a) y1 <= y1 + 1;
            if(b) x1 <= x1 + 1;
            if(c) X <= X + 1;
            if(d) Y <= Y + 1;
          end
end

always@(*) begin
  case(Q)
            Idle : if(Start) D <= VBP;
                        else D <= Idle;
                    VBP  : if(y1 == V_BACK_PORCH) D <= HBP;
                        else D <= VBP;
                    HBP  : if(x1 == H_BACK_PORCH) D <= HA;
                        else D <= HBP;
                    HA   : if(X == H_ACTIVE) D <= HFP;
                        else D <= HA;
                    HFP  : if(x1 == H_FRONT_PORCH + H_BACK_PORCH) D <= HS;
                        else D <= HFP;
                    HS   : if(x1 == H_SYNC + H_FRONT_PORCH + H_BACK_PORCH) D <= VFP;
                        else D <= HS;
                    VFP  : if(y1 == V_FRONT_PORCH + V_BACK_PORCH) D <= VS;
                        else D <= VFP;
                    VS   : if(y1 == V_SYNC + V_FRONT_PORCH + V_BACK_PORCH) begin
                             if(Y == V_ACTIVE) D <=Idle;
                                                 else D <= VBP;
                                    end
                        else D <= VS;
                    default : D <= Idle;
          endcase
end
```

```
always@(posedge clk or posedge reset) begin
  if(reset) begin
            Q<=3'b000;
          end
          else Q <= D;
end

always@(Q) begin
  case(Q)
            Idle :  begin
                        a <= 0;
                                    b <= 0;
                                    c <= 0;
                                    d <= 0;
                                    H_Sync <= 1'b0;
                                    V_Sync <= 1'b0;
                    end
            VBP  :  begin
                        a <= 1;
                                    b <= 0;
                                    c <= 0;
                                    d <= 0;
                                    H_Sync <= 1'b0;
                                    V_Sync <= 1'b0;
                                end
                HBP  :  begin
                        a <= 0;
                                    b <= 1;
                                    c <= 0;
                                    d <= 0;
                                    H_Sync <= 1'b0;
                                    V_Sync <= 1'b0;
                                end
                HA   :  begin
                        a <= 0;
                        b <= 0;
                                    c <= 1;
                                    d <= 0;
                                    H_Sync <= 1'b0;
                                    V_Sync <= 1'b0;
                                end
                HFP  :  begin
                        a <= 0;
                        c <= 0;
                        b <= 1;
                                    d <= 0;
                                    V_Sync <= 1'b0;
                                    H_Sync <= 1'b0;
                                end
                HS   :  begin
                                    H_Sync <= 1'b1;
                                    V_Sync <= 1'b0;
                                    b <= 1;
                                    a <= 0;
                                    c <= 0;
                                    d <= 0;
                                end
                VFP  :  begin
                        b <= 0;
                        a <= 1;
                                    c <= 0;
                                    d <= 0;
                        H_Sync <= 1'b0;
                                    V_Sync <= 1'b0;
                                end
                VS   :  begin
                        b <= 0;
                                    c <= 0;
                        V_Sync <= 1'b1;
                                    H_Sync <= 1'b0;
                                    if(1 == V_SYNC + V_FRONT_PORCH + V_BACK_PORCH) d <= 1;
            a <= 1;
                                end
                default : {H_Sync,V_Sync,A,GPins} = 23'd0;
```

```
  endcase
end

SRAM #(20,32) S1(
    .clk(clk),
    .reset(reset),
    .sram_addr(Addr),
    .sram_data(Data),
    .sram_ce_n(CE),
    .sram_we_n(W),
    .sram_oe_n(OE),
    .read_enable(RD),
    .write_enable(1'b0),
    .user_addr(ADD + X),
    .user_read_data(DATA)
);

Memory M1(Addr,Data,CE,OE,W);
assign {VR,VG,VB} = DATA[23:0];

endmodule
```

## RTL Diagram