ECE 901 Digital Systems Prototyping

Mini-Project 2

Display unit using DVI port

THE UNIVERSITY of WISCONSIN MADISON

Team Name: Untitled Design

Kushagra Garg	907 007 5420
Rohit Shukla	906 719 3145
Vignesh Chandrasekaran	906 982 9563

Table of Contents

Table of Figures

1. Block Diagram

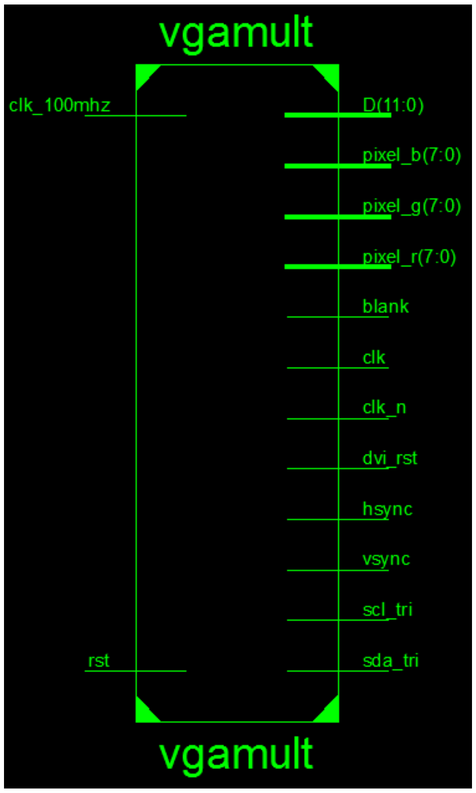

Figure 1: Schematic of Top_level module

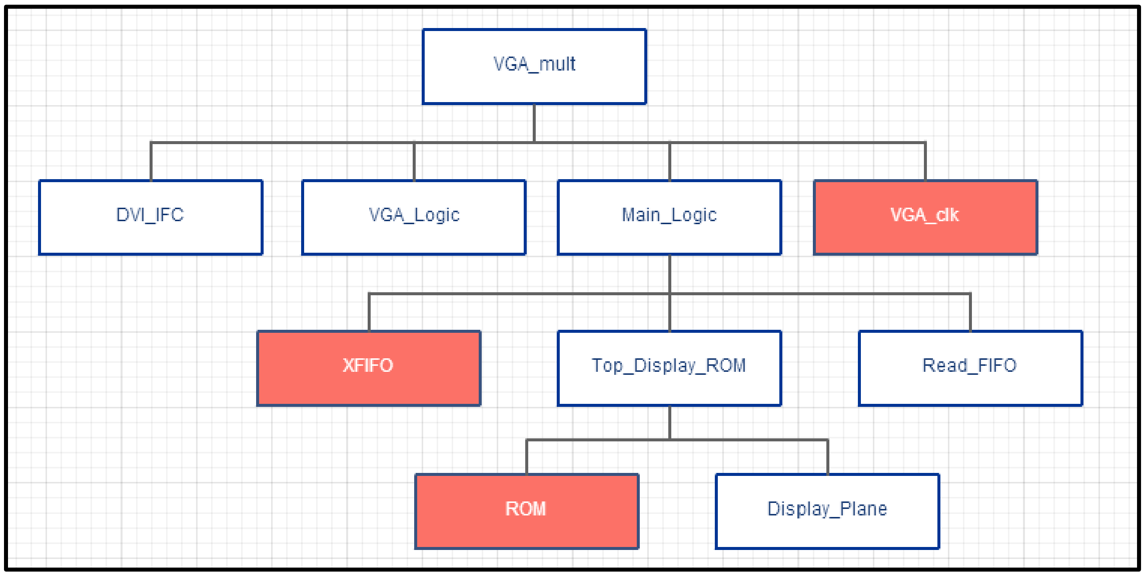

Figure 2: Module Hierarchy

2. Blocks

The display unit is capable of displaying the given 80x60 image as 16 titles on a 640x480 screen.

2.1 VGA_mult
This is the top level module which integrates all the other modules. It is also multiplexes VGA output to DVI output.

2.2 DVI_ifc
This module implements the IIC controller.

2.3 VGA_Logic
This module implements the timing generator required for VGA display and produces read enable signal for FIFO.

2.4 Main_Logic
It is responsible for integrating the Display panel, FIFO and Read_FIFO module.

2.5 Clock Generator
Our Verilog code uses the Xilinx ISE coregen to generate a divide-by-4 clock divider. Clock generator takes the 100 MHz clock as input and generates a 25 MHz at the output.

2.6 XFIFO
We generated the cross clock FIFO module using Xilinx coregen which writes the data from ROM at 100 MHz and reads the data at 25 MHz to display on screen. The write enable signal is produced from signals received from display panel and read enable signal is enabled from VGA_logic module.

The logic for generating write enable is:

assign wr_en_fifo = (~full_fifo)&(~rst);

Write enable is set only when FIFO is not full and reset is not set. Otherwise, we will be just trying to write the data into the FIFO when it is full as a result dropping the pixel values.

The logic for generating read enable signal is:

assign rd_fifo = ((next_pixel_x < 10'd640) && (next_pixel_y < 10'd480) && ~fifo_empty && done);

Our top module does not try to read the data from the FIFO if it is empty. Top module also keeps in check that it does not try to increment the x and y pixel counters for the monitor screen if the FIFO is empty.

2.7 RD_FIFO

The read FIFO module reads the 24-bit data from XFIFO and splits it up into 8-bit pixel_r, pixel_b, pixel_g values. These 8-bit values are sent to the top module vgamult where it is multiplexed for DVI controller.

The Verilog code for splitting up the 24-bit data into 8-bit RGB values is:

```
assign pixel_r = fifo_data_rd[23:16];

assign pixel_g = fifo_data_rd[15:8];

assign pixel_b = fifo_data_rd[7:0];
```

2.8 ROM

ROM module is generated using coregen feature. Each entry is 24bit and ROM contains 4800 entries i.e the pixel values provided by .coe file.

2.9 Display Plane

Display Plane module does the following tasks:

- Provides the address to read pixels from the ROM.
- Writes those pixel to FIFO, if the FIFO is not full

We implemented tiling to scale 80x60 resolution image to 640x480 resolution.

For a given image, our output would approximately look like this:

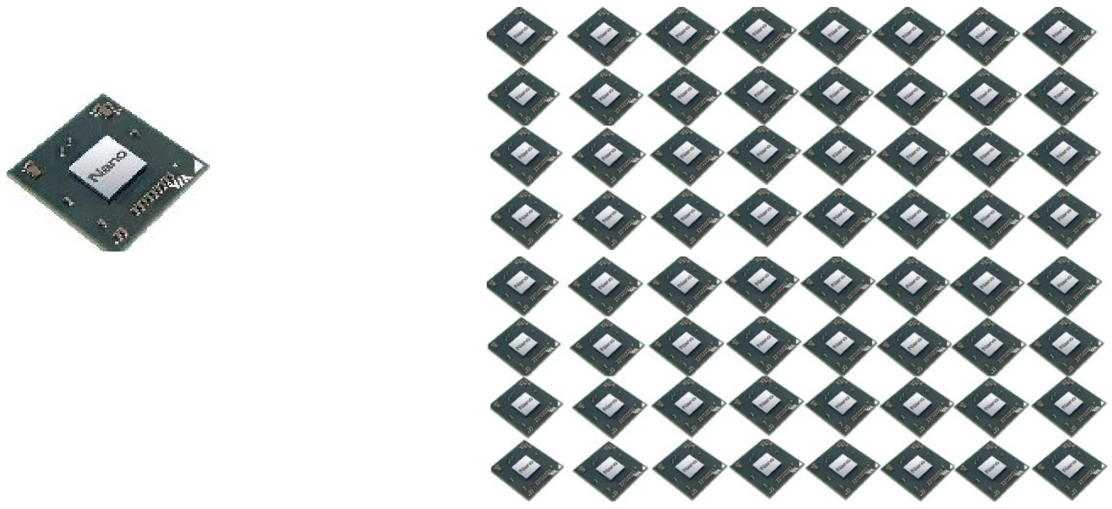

Figure 3: Tiling illustration

Logic for Display Plane

- For tiling, we read the first 80 ROM pixels i.e first row of the image. We then send these pixels to FIFO 8 times. This constitutes scaling 80 pixels to 640 pixels.
- Next, we shift the reading base to address 80 to 160 which implies we read second row of the image. We again send these 80 pixels to FIFO 8 times.
- We repeat the above process 60 times so that reading base address shift by 80 to reach 4800 thereby covering the whole image.
- We repeat this process 8 times so that scaling to 480 pixels is done vertically.
- To achieve this we have 4 counters in our display plane design. Below mentioned code describes the counter synchronization. Four counters count till 80, 8, 60 and 8 respectively to generate appropriate address.

```
if(hp_flag_80)
        begin
                hp_count_80 <= 0;
                h_count_8 <= h_count_8 + 1;
                addr <= baseaddr;
        end
if(h_flag_8 & hp_flag_80)
        begin
                h_count_8 <= 0;
                vp_count_60 <= vp_count_60 + 1;
                baseaddr <= baseaddr + 80;
                addr <= baseaddr+80;
        end

if(vp_flag_60 & h_flag_8 & hp_flag_80)
        begin
                vp_count_60 <= 0;
                v_count_8 <= v_count_8 + 1;
```

```
                    baseaddr <= 0;

                    addr <= 0;

                end


        if(v_flag_8 & vp_flag_60 & h_flag_8 & hp_flag_80)

                begin

                    v_count_8 <= 0;

                end


        if(!hp_flag_80)

                begin

                    addr <= addr + 1;

                    hp_count_80 <= hp_count_80 + 1;

                end
```

Synchronization with FIFO

- If FIFO is full, we stop the counters from incrementing which would prevent the read address from being updated. We also disable write_en to the FIFO.
- Once FIFO is not full, we read the data from ROM by specifying the address and correspondingly writing the same data to FIFO by enabling write_en.

2.10 Top Display ROM

This module contains instances of ROM and Display plane.

3. Problems encountered

- The output picture that was displayed was initially tiled 16 times horizontally and vertically instead of the expected 8x8 tiling. This occurred because we assumed that the Full signal from the FIFO was Synchronous but it was actually Asynchronous.

- The displayed picture was shifted by a few pixels. This occurred because the counter was incrementing even when the FIFO was empty i.e. the counter was incrementing even when there was no data being read.

- Initially we took the blank signal as the read enable signal for FIFO but we later modified this to another valid logic.

- The tiled picture was moving in a marquee fashion and then it was later fixed by modifying the FIFO read enable.

- The clock that was used as 100MHz clock is clk_100mhz_buf as the buffered clock is the only one that synthesizes.

4. Result

The given picture was displayed as 16 tiles on the screen

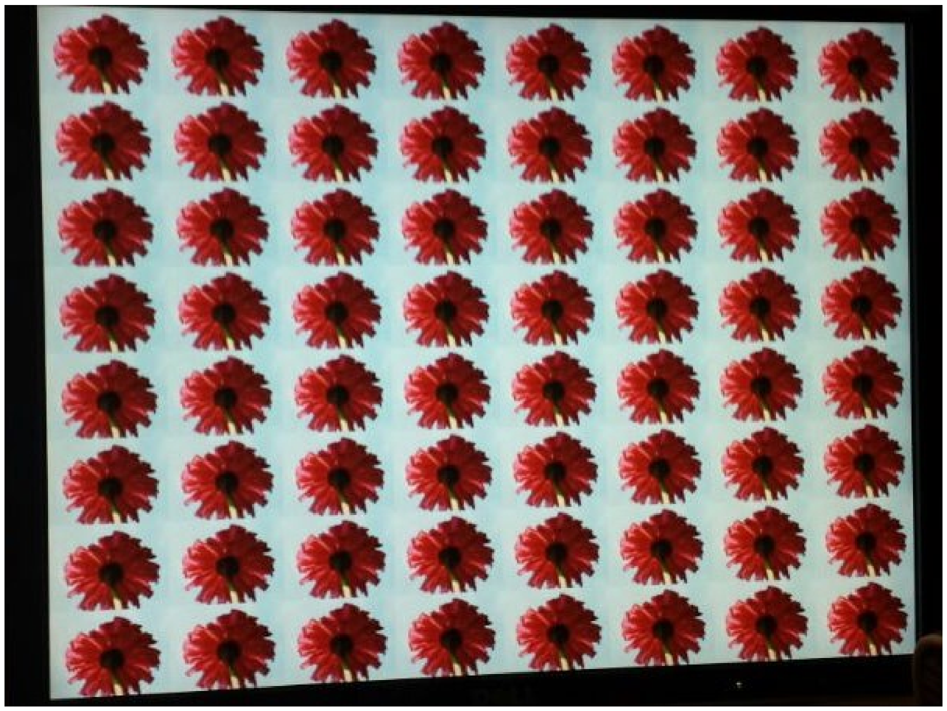

Figure 4: Output display screen