UNIVERSITY OF LEEDS

ELEC5620M
Mini Project

DE1-SoC Pong

Alexander Bolton - 200938078
Sam Wilcock - 201285260
John Jakobsen -

May 2019

Submitted in accordance with the requirements for the degree of
Master of Science in Embedded Systems Engineering

The University of Leeds
School of Electronic and Electrical Engineering

Contents

1 Introduction

This report will discuss the group project for the Embedded Microprocessor System Design module. The groups member's were Alexander Bolton, Sam Wilcock, and John Jakobsen. The projects aim was to create a game of Pong on the DE1-SoC's microprocessor unit (MPU) which utilised the LT24 LCD Screen, a VGA screen, PS2 keyboard controls, button controls, and have audio output.

This report will be broken down into sections with section 1 being the introduction. Section 2 will discuss the display and graphics side of the project including the VGA driver which controls the monitor, the display driver which controls both LCD and VGA screens with a frame buffer, sprites and text which will go into depth of how the sprites are created, finally game engine graphics which will go into how the game engine uses the sprites including destroying, creating, and moving the sprites.

Section 3 will discuss...

Section 4 will discuss...

Section 5 will be the conclusion which will summarise the report and discuss if we have met the aims of the project. It will discuss what could be improved upon and changed. All code will be placed in the end of the report in the appendices.

2 Display and Graphics

2.1 VGA Driver

This subsection discusses the VGA driver and how it was implemented in the project. The VGA video out supports 640x480 however in this project is set to the default value of 320x240 pixels. The image displays from the VGA controller which is addressed from a pixel buffer. Each pixel value is write addressable using equation 1. An example of the pixel at 0,1 is shown in equation 2. The default base address for the pixel buffer is 0xC8000000 as stated in the manual. [1]

$$VGA_{baseaddress} + (pixelX_{coordinates}\; pixelY_{coordinates}\; 0_2) \tag{1}$$

$$C8000000_{16} + (00000001\;000000000\;0)_2 = C8000400_{16} \tag{2}$$

The pixels are layed out with the y coordinate starting from the top to bottom of the screen. The x coordinate is from right to left of the screen as shown in figure 1.

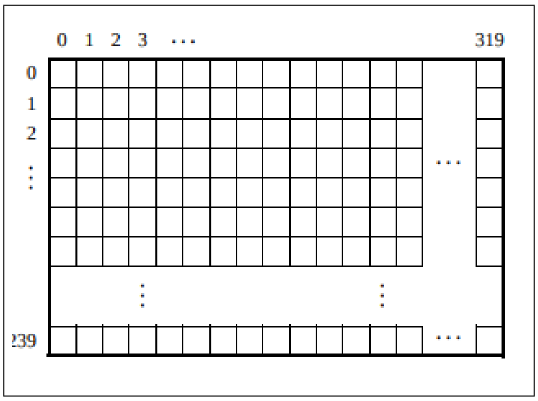

Figure 1: Pixel layout for pixel buffer of VGA controller [1]

Each pixel once addressed can be set to a value of colour with by setting bits for red, green, and blue. Each colour is allocated 5 bits which indicate the strength of colour for the pixel as shown in figure 2.

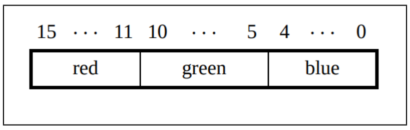

Figure 2: Pixel Colour Layout [1]

This code extracted from the project shows how a pixel is set in C.

```
void VGA_drawPixel(int x, int y, short colour){
  //Call address of pixel
    //Address base + [Pixel Y][Pixel X]0
  volatile short *vga_addr=(volatile short*)(0xC8000000 + (y<<10) + (x<<1));
  *vga_addr=colour; //Set pixel to colour
}
```

Figure 3: Code used to set pixel to a colour.

2.2 Display Driver

This subsection discusses the display driver which allows pixels to be set to a frame buffer and refreshed on command. The frame buffer is made up of 2 arrays which were a front frame buffer (what's currently on screen) and a rear frame buffer (what wants to be put onto the screen). The advantage of this is that it allows pixels to be checked and allows refresh on command.

Once the screen is desired to be refreshed the memory is compared between the front frame buffer and rear frame buffer to check if they are the same. If not the frame buffers are updated by checking each pixel in the frame buffers values. Any differences then update to display and to the front frame buffer. When first creating this frame buffer a problem with articulating occurred which is believed due to a memory overflow. The decision was made to split the frame buffer into 2 halves which reduced the memory used in each array which eliviated the issue.

Checking the pixels in a frame buffer is a very slow process so multiple frame buffers were created to experiment with. This experimentation involved splitting the screen into multiple sections and finding how many sections would be the fastest. This improved performance as multiple frame buffers were checked in memory and making the changes was done by comparing all pixels in a smaller area when there was a change in that area.

2.3 Sprites and Text

-

2.4 Game Engine Graphics

-

3 Controls and Menus

4 Game Physics

5 Conclusion

6 Appendix

References

[1] Altera, *DE1-SoC Computer System with Nios II*. Altera, 2014.