# LAB 4 REPORT

Muhammad Anza Khan

260618490

Ayden Aamir Malik

260627064

# VGA

Main.c is the main function of the vga drivers which includes vga.h and vga.s.

## Main.c

We begin by including driver files for the vga, pushbuttons and slider switches. Following that we check to see whether any of the 4 push button is pressed or not using a while loop with four separate 'if' conditions which are discussed later on.

In each of the 'if' condition we check which specific pushbutton was pressed. We do this because each pushbutton is mapped to a certain function call which in turn calls a specific subroutine.

Pushbutton 0

The if condition for Pushbutton 0 works by calling PB_data_is_pressed_ASM and passing it PB0.
Then it enters another if block which checks the states of the slider switches by calling read_slider_switches_ASM and checking whether its not equal to 0.

If it is true(i.e. not equal to 0) we call test_byte which in turn calls VGA_write_byte_ASM which writes the hexadecimal representation of the value passed in the third argument i.e. the character passed, to the screen.

If the slider switch is 0 we call test_char() which inturns call VGA_write_char _ASM() which writes the ASCII code of the value passed in the third argument i.e. the character passed, to the screen.

Pushbutton 1

Here we check whether PB 1 is pressed. If it is indeed pressed we call text_pixel() which in turn calls VGA_draw_point_ASM. This subroutine simply draws a point on the screen with the color as indicated in the third argument by accessing only the pixel buffer memory.
This subroutine was already defined for us.

Pushbutton 2
Here we check whether PB 2 is pressed. If it is pressed, we call the VGA_clear_charbuff_ASM subroutine which simply clears the character buffer.

Pushbutton 3

Here we check whether PB 3 is pressed. If it is pressed, we call the VGA_clear_pixelbuff_ASM subroutine which simply clears the pixel buffer.

test_char()

First of all we start with the test_char() function in which we initialize two variable x & y which are loop counters for 2 'for' loops and a character c. c is initialized at 70. Each time we pass a new value for each argument x, y & c to the VGA_write_char_ASM() subroutine. It is used to display the character passed using c at coordinates x and y which are initialized at 0. With each cycle char c is the third argument which is the ASCII code for all characters cycled through and each character is displayed on the screen.

test_byte ()

It also starts by initializing x & y with 0 but here x is incremented by 3 in each iteration to leave some space between each column of hexs' that are passed in the third argument of the VGA_write_byte_ASM().

test_pixel()

This subroutine starts by initializing x & y which are loop counter for the 2 nested 'for' loops which iterate through the 320 by 240 pixel screen. It draws a point on the screen with the color as indicated in the third argument, by accessing and varying the pixel buffer memory.

## Keyboard application

### Main_keyboard

The main function starts by clearing the character buffer memory then proceeds to a while loop that reads the character entered from the keyboard and displays it while keeping 3 columns vacant between each column of hex representation of the character entered in the keyboard.
When we reach the end of the screen reset x &y to 0 using an 'if' condition that has to is controlled by the boundary condition i.e. y has to be less than equal to 60. Since, there are 59 rows only, x & y are set to zero when y hits 60.

Ps2_keyboard

This file written in assembly code accesses the ps/2 data register that reads the register's $15^{th}$ bit which is the RVALID bit and 'bitwise-and' it to check whether that bit is 1 or not. If it is 1, it means we can a pass character onto the screen, we return 1 and exit, which indicates successful execution of the subroutine.
If the RVALID is not 1, we return 0 and exit the subroutine. We do not write anything and this indicates an unsuccessful execution of the subroutine.

## Audio

### Audio_main

In this file we generate a string of 0's & 1's at a frequency of 100Hz. Using nested while loops within one major while loop
The sampling rate of our board is 48000Hz so we divide it by 2 then by 100(required freq.) and obtain the number 240.

240 is the number of times the first while loop runs to pass the number 1 240 times to the audio subroutine in the audio.s file.

After that we exit the loop and enter a new while loop that also runs 240 times to produce a strings of 0's by passing 0 to the audio subroutine in audio.s.

After exiting the second while loop we reset and the above loop counters i & j to 0 and return 0 to the outer most while loop and set run it again.

This process occurs repetitively to pass a string of 0's and 1's to the audio subroutine which will ultimately produce a square wave of 100Hz.

### Audio.s

In this subroutine, from the FIFO space register we extract the values for WSLC and WSRC. If both of their value is less than 1, we cannot store anything into the register and we exit the subroutine returning a value of 0 indicating the subroutine failed.

If there is space, we store data into the Leftdata and Rightdata and exit the subroutine returning 1 indicating successful execution of the subroutine.