Patrick Ghazal – 260672468
Kartik Misra – 260663577

ECSE324: Computer Organization
Lab 4 Report
Group 7

I. VGA

Several subroutines were developed for the VGA demonstration.
The **VGA_clear_charbuff_ASM** and **VGA_clear_pixelbuff_ASM** subroutines clear the valid memory locations in the character and pixel buffer respectively, by looping through the x and y coordinates and storing a value of 0 at each address. In the VGA demonstration, they are triggered through the third and fourth push-buttons (counting from the right) to clear the screen of any characters or pixels.

VGA_write_char_ASM receives three arguments: the x coordinate, the y coordinate, and the character to be printed to the screen at coordinate (x, y). It checks that the coordinates are valid, in this case that x is a positive number lower than 80, and y a positive number lower than 60.

VGA_write_byte_ASM receives the same arguments. It separates the byte argument into two nibbles, then finds the character representation of each in the hexadecimal system and prints it. It calls the **write_char** subroutine to print each character, which allows the verification of coordinates to be performed as well.

VGA_draw_point_ASM works quite similarly to the **write_char** subroutine, by computing the shift in coordinates and offset, and adding them to the address upon which to store the colour.

The VGA application that brings those subroutines together is the vga() method of the main.c file. It reads the data from the push-buttons, and depending on which push-button was pressed (and whether any slider switch is on), it executes a certain behaviour.

II. PS/2 Keyboard

read_PS2_data_ASM is the only subroutine that was developed specifically for this part. Its role is to read the data from the PS/2 keyboard that is connected to the DE1 computer. The RVALID bit is isolated by ANDing the 4 input bytes with 0x8000. If it is 1, the program continues; if it is 0, the program returns 0 and ends. The input is then ANDed with 0xFF to isolate the last byte which corresponds to the actual data being transmitted. This data is stored at the char pointer, and a value of 1 is returned to denote success.

The application is the ps2keyboard() which continuously waits for data. Upon reception, the data is checked using the **read_PS2** subroutine. If it returns 1, the data is passed to the **write_byte**

subroutine. Coordinate checks are then performed. If x exceeds its max value (set in the method), it is brought back to 0 and y is incremented by 1. If y exceeds its max value (also set), it is brought back to 0 and the entire display is cleared due to lack of remaining space. Printing the data can then resume at coordinates 0, 0.

III. Audio

The audio demonstration was quite simple. The **write_audio_data_ASM** subroutine checks the both FIFOs for availability (i.e. that they contain 0), returns 0 to indicate failure if one or both FIFOs are full, and otherwise stores the input into each channel.

The goal of the audio method is to obtain a wave of 100 Hz frequency at a sampling rate of 48K samples/sec, i.e. 100 full complete wave cycles in 48000 samples. This represents (48000/100 =) 480 samples per cycle. Given that each cycle is a low and high value half of that number is to be used for each value. There are therefore 240 samples with a “1” (here, 0x00FFFFFF) and 240 with a “0” (here, 0x00000000).