ECSE 324 , Group 17

Lab 4 Report

Part 1 : VGA

Firstly, we had to write 5 subroutines

VGA_clear_charbuff_ASM:

- This subroutine cleared (set to 0) all the valid memory locations in the character buffer. The subroutine iterated through the valid memory locations in the character buffer and then stored a full byte of zeros to the location.

VGA_clear_pixelbuff_ASM:

- This subroutine cleared (set to 0) all the valid memory locations in the pixel buffer. The subroutine iterated through the valid memory locations in the pixel buffer and then stored a full half-word of zeros to the location.

There were some differences between the two clear methods. In ***VGA_clear_charbuff_ASM*,** the subroutine iterated through an 80 x 60 grid whereas in the ***VGA_clear_pixelbuff_ASM***, the subroutine iterated through a 320 x 240 grid.

VGA_write_char_ASM:

- This subroutine wrote the ASCII code passed in the third argument to the screen at the (x,y) coordinates given in the first two arguments.

VGA_write_byte_ASM:

- The subroutine wrote the hexadecimal representation of the value passed in the third argument to the screen. In essence, two characters are printed to the screen per iteration. For example, passing a value of 0xAB in byte resulted in the characters 'AB" being displayed on the screen starting at the x,y coordinates passed in the first two arguments.

- The main challenge was when approaching ***VGA_write_byte*** as we somehow had to convert the input of hexadecimal numbers into ASCII code such that the output would be a display of hexadecimal numbers.

    - We searched up an ASCII table and realised that ASCII code difference between character 9 and character A was 7 and the ASCII code for character 0 was 48. We used this to convert our input of hexadecimal numbers into ASCII code such that the output would be a display of hexadecimal numbers.

VGA_draw_point_ASM:

- The subroutine drew a point on the screen with the colour as indicated in the third argument, by accessing only the pixel buffer memory.

Main.c:

- Here we used the ***pushbuttons.s***, ***pushbuttons.h***, ***slider_switches.s*** and ***slider_switches.h*** files from the previous lab (Lab 3).
- We wrote a main method such that
    - PB0 pressed: if any of the slider switches is on, call the test_byte() function, otherwise, call the test_char() function
    - PB1 pressed: call the test_pixel() function
    - PB2 pressed: clear the character buffer
    - PB3 pressed: clear the pixel buffer

Part 2 : P/2Keyboard

In this part, we first created a subroutine **read_PS2_data_ASM** that checked the RVALID bit in the PS/2 data register. If it was valid, then the data from the same register was stored at the address in the char pointer argument, and the subroutine returned 1 to denote valid data. If the RVALID bit was not set, then the subroutine simply returned 0.

We then wrote a main method that used the PS/2 keyboard and VGA monitor. The method read raw data from the keyboard and displayed it to the screen if the data was valid. We only used the ***VGA_write_byte_ASM*** subroutine from the VGA driver, and the input byte was simply the data read from the keyboard. The corresponding display is associated with the PS/2 keyboard scan code for each single letter, there are 3 pairs of 2-digit hexadecimal numbers for one letter on keyboard, with the left 2-digit as the make code and the rest 4 digits act as break code as indicated in the lab description.