Jia Wei Sun
Karl Koerich
November 17, 2019

Lab #4 report

Overview

For this lab, we used high level input output capabilities of the De1-SoC computer to write 2 small programs: use the VGA controller to display pixels and characters and use the PS/2 port to accept input from a keyboard.

Approach

VGA_clear_charbuff_ASM and VGA_clear_pixelbuff_ASM

These functions are written to set the content of memory locations in the char buffer and the pixel buffer to 0. Each pixel buffer is represented by a 16 bit halfword, with 10. For their address, the 10 LSB are for its x coordinate and the next 8 are for its y coordinate. We wrote a loop to iterate through all possible y values, and we wrote an inner loop to set all the possible x coordinates to 0 as well. To get the 0x value of the address of each memory location, we first left shift the y by OFFSET to leave space for the x coordinate, and we used the ORR instruction to "merge" both and obtain the complete memory address. For char buffer, we only shift the y coordinate by 7 since there are only 8 x coordinate bits.

VGA_write_char_ASM

This function is written to write char values to memory addresses. We first check if the coordinates are valid by doing 4 comparison instructions, then we offset y by 7 like the previous functions and we ORR with x coordinate to obtain the complete memory address. Then we store the argument char to the address.

VGA_write_byte_ASM

For this function, we want to write the hex representation of the value passed in the argument. To do this, we left shifted the memory address by 4 bits, and we used AND instruction to obtain the last 4 bits of the byte. Then we used the write char function again to perform the write byte operation.

VGA_draw_point

This function is written to draw a point on the screen, whose colour is determined by the third argument of the function. This function is very similar to the write char function, and as mentioned earlier, the pixel buffer address has 10 bits for x and 1 bit for the 0 (LSB). We left

shift y by 10 bits to give space to x, and we left shift x by 1 bit to give space to 0. We obtain the complete memory address and then we store the appropriate colour at the address.

Simple VGA application

We make use of if statements to incorporate sliders and buttons in this program. In the main, we check for which button is pressed, respectively checking for value 1 (test_char or test_byte, 2 (text_pixel), 4 (VGA_clear_charbuff_ASM) and 8 (VGA_clear_pixelbuff_ASM). We made use of pushbottons.s and slider_switches.s from lab 3.

Keyboard

This application consists in reading raw data from the keyboard and display it to the screen. In ps2_keyboard.s, we first check RVALID built in ps2 data register. If it is valid, data from the same register should be stored at this address in the char pointer argument. Upon keypress, a make code is created and stored, and upon release, a break code is created and stored. The subroutine read_PS2_data_ASM returns 1 to denote valid data; when RVALID is 1, we read the data at the last 8 bits of the register. This data is stored as a byte at the pointer passed to the subroutine. As of the c code, we used the VGA_write_byte_ASM to continuously read keyboard data and write them. We use the VGA_clear_charbuff_ASM function to clear the buffer when the x or y exceeds the maximum possible value.

Conclusion

As we advance through the course, we are more and more mixing concepts and the Labs are getting more challenging, was requires us to invest more time on them. The main purpose of this lab was to not only introduce us to the basics of input output application, but also to make us more familiar with hexadecimal display, VGA and PS/2. Although we spent time on debugging some minor bugs, we think this lab is very useful for our learning and we are satisfied with the results.