ECSE 324 Lab 4 Report Group 11

Authors: Aleksas Murauskas, 260718389 and Brendan Kellam, 260718806

VGA:

For testing the VGA the method VGA_Test is called. Within in the method an infinite loop iterates waiting for a command from the VGA. It reads the data using the read_PB_data_ASM() and read_slider_switches_ASM() methods used in the previous lab. If the button 0 is pressed the and none of the slider switches are on, the method test_char() is called. Test char was given, which loops through through the char buffer writing a char of increasing value at each point. VGA_write_char_ASM writes a char to a specified x and y position in the buffer. First it checks if the x and y are valid, and then sets the point to zero. If button 0 is pressed while a slider is on, the method test_byte is called. Test_byte() operates similarly to test_char with two key changes. X iterates by 3[the size of the byte and a space] and uses the Assembly code VGA_write_byte_ASM. VGA_write_byte_ASM finds the ascii character of the first 4 bits and the last 4 bits and branches to VGA_write_char_ASM to write the chosen ascii.

If the second button is pressed, the method test_pixel() is called. test_pixel() loops through the resolution using to nested loops with the limits 240 and 320. At each nested iteration the the assembly code VGA_draw_point_ASM. X and y are Left shifted to find their offset in the pixel_buffer. The given unsigned int color is then stored at that location in memory.

If the third button is pressed, the method VGA_clear_charbuff_ASM() is called. This goes to the assembly code labelled the same. Within this method two nested loops occur. The outer loop increments through row to row, while the inner loop iterates through each row. The outer loop limits at 80, and the internal limits at 60. At each iteration of the inner loop, the value is set to zero to clear the buffer.

If the fourth button is pressed, VGA_clear_pixelbuff is called. This method clears in a similar way to VGA_clear_charbuff_ASM. However, now the the limits of the outer and inner loops are 320 and 240 respectively. At each iteration of the inner loop, the value is set to zero to clear the buffer.

PS2Keyboard

The ps2 keyboard code is tested in the ps2_test method. Within the method, the pixel and char buffs are cleared first. Then an infinite loop iterates waiting for a response from the keyboard. The keyboards inputs are read using the read_PS2_data_ASM assembly code. First the code loads the data register. The then compares the value of the data register with 16th to determine if the the data register is valid. If not, zero is returned and the code branches back to the main. Otherwise, the ps2 data is stored in the location passed to the function and the code

branches back to the main program. Then the character is written to the character buffer using the VGA_write_byte_ASM assembly code described earlier. Then the x and y locations are incremented in two nested loops, similar to the test_byte() method. If the y value becomes larger than the maximum, then the VGA_clear_charbuff method is called as new characters would not be able to be added to a full screen.

AUDIO

The audio code is tested in the audio_test() method. The audio_Test method contains an infinite loop. Within the infinite loop there are two for loops. Since we know that we want a 100hz wave, and we know that the hardware samples at 48,000 samples a second. So to create the wave, we need to divide our sample rate by the desired rate. So 48,000 divided by 100 hz is 480 samples per period. To create the wave, we need both a top and a bottom to the wave, so our samples per period must be divided by two. So each for loop is bounded by 240. The first loop writes the audio using the assembly code write_audio_FIFO_ASM. If at any point write_audio_FIFO_ASM returns 0, it means the write method has failed, so the loop counter decrements to attempt to write again. The audio is stored First In First Out. The assembly code write_audio_FIFO_ASM begins by loading the audio fifo location and loading a mask value that holds 0xFFFF0000. The value currently held in the FIFO location is ANDed with the mask to retrieve the top bits of the fifospace. If the result of the and is 0, then there is no capacity left and must branch out as the program is unable to return to the buffer. The program also returns 0 to notify that it failed. Otherwise the the location of the left and right speaker is loaded into registers. The parameter stored in R0 is then stored in the in both left and right locations. Afterwards the program sets R0 to 1 to notify that the audio was written successfully.