ECSE 324
Lab 4 Report

VGA

This part constituted a few parts. The first part was the VGA_clear_charbuff_ASM, this was used to clear the pixel buffer. First the base address of the pixel buffer was loaded to a register. The each pixel was iterated, first clearing a column then moving to the next row by one and then clearing the corresponding column. The pixels were cleared by storing 0 to the specific address. The specific address of each pixel was found by adding the X and Y offsets to the base location of the pixel buffer as specified in the manual for the board.

The next subroutine was the VGA_clear_charbuff_ASM. This was almost identical to the previous one with the only difference being the the dimensions of the character buffer was 80 by 60 instead of 320 by 240 for the pixel buffer.

The method VGA_write_byte_ASM was implemented to print hexadecimal numbers on the screen. One char is one byte, we dedicate the first 4 bits for the first character and the last 4 bits for the second character. Hex hex input is converted to ASCII.

The next method, VGA_write_char_ASM traverses through the x and y coordinates, we also check to see if we are within the limit. Then the char value is stored in the desired location.

For the VGA_draw_point_ASM method, we again first check that we are within the grid, then we store the inputted half word to the correct memory location.

Keyboard

In this section, we built a driver to read the PS/2 keyboard data and created an application that can read the raw data from the keyboard and write the data on the keyboard if they are valid.

In the ps2_keyboard.s, the subroutine read_PS2_data_ASM takes a char pointer variable as input and return an integer depends on whether the data read is valid or not.

Specifically, the subroutine checks the RVALID bit in the PS/2 Data register. If it is valid, the data from the same register should be stored at the address in the char pointer argument, and return 1. Otherwise, the subroutine should return 0.

We take the PS2_Data address and the value of it into a register. And use AND operation between the result and 1000000000000000. If the result turns out to be equal, it means RVALID bit is 0. Otherwise, RVALID is 1. Then the data is stored in the char pointer and program returns 1 to indicate the validity.

In the main C program, we set two counter x, y to keep track of the address and a

char pointer *c. In the while loop, we use 'read_PS2_data_ASM(char data)' to check whether the input is valid. If it is valid, use VGA_write_byte_ASM(x, y, c); and increase x counter by 3 to update the position on the grid. We also need to set the boundary for the x, y counter.

Audio

The Audio.s subroutine takes one integer argument and writes it to both left and right FIFO only if the register has space.

To be more specific, we load the address of base address of Fifospace address of the audio port register and store the input into that address.

We left shift the content stored in R5 by #16 to check the bit field of WSRC, if it is greater than zero, there is space in the right data, and then it will go to check left data by right shift the R5 by #8, and check the bit field of the WSLC. If the both WSLC and WSRC are not zero, this means the data can be written in left data and right data.

For the main C program, the sampling rate is 48K sample/sec and the frequency is set to be 100Hz.This means that for each period there are 480 samples. In every 240 sample, a ‘1’should be written to the FIFOs and in other 240 samples, a’0’ should be written. So we use a while loop and take a counter starting from 0 and increase the counter (write 1 to FIFOs) until it reaches 240(write 0 to FIFOs).