ECSE 324 Lab 4 Report
Group 66

Gengyi Sun 260768270
Liang Zhao 260781081

1.VGA

1.1.1.VGA Clear Char and VGA Clear Pixel

The goal for this part was to write two subroutines which would set all the memory locations in the Character Buffer and the Pixel Buffer to zero. For doing this, we implemented two loops to iterate the x-axis and y-axis for each subroutine. For clearing Pixel Buffer, we iterated in the x-axis from 0-319 and y-axis from 0-239. For clearing Character Buffer, we iterate in the x-axis from 0-79 and the y-axis from 0-59. For every Character or Pixel, there is a unique corresponding address. We reach these addresses by the BASE ADDRESSES of Character and Pixel Buffers, then adding its x-axis and y-axis offsets.

For clearing the Character Buffer, the y-coordinate was first shifted to the left by 7 bits to make room for the x-coordinate, 'ORRed' with the x-coordinate and finally added the BASE ADDRESS. For clearing the Pixel Buffer, the y-coordinate was shifted left by 10 bits for x-coordinate, the x-coordinate was shifted left by 1 bit for the '0'.

For clearing Character and Pixel Buffer, both subroutines were very similar to each other and straightforward.

1.1.2.VGA Write Character

To implement this part, we first used two registers to track the length and width of the screen in pixels (x=80 and y=60). Then we used the CMP operation to check if the entered x and y were both lies between 0-80 in x and 0-60 in y. If they were both in this domain, it meant they were valid values. We would offset y by 7-bits to make room for x and added them together. Finally, we would store the input value into the address we calculated above. If they were either in this domain, we directly used branch to end this subroutine.

1.1.3.VGA Write Byte

This part is very similar to VGA Write Character. However, Byte contains 2 characters in ARM. We used the write character subroutine twice for the write byte subroutine.

1.1.4.VGA Write Byte

To implement this part, we first store the color into a register. Then, the Y was shifted left by 10 to make room for X, and the X is shifted left by one bit to make space for '0'. Then, we added the shifted X and Y together with the BASE ADDRES. Finally, we stored the color into this address.

1.2.Simple VGA Application

Finally, we wrote a simple C program which will test the functionality of our VGA driver. This code consisted of 4 if statements, one for each button pressed (1,2,3,4). The code would simply do that if read Push Button Data was one of these numbers, and the code should do corresponding mission as stated in the manual. This C program was very simple and straightforward, we didn't encounter any challenges here.

2.Keyboard

The Keyboard part used the PS/2 port of the FPGA. In this lab, we used the PS2_Data register. When a key was pressed on the keyboard, a make code was created and stored to a FIFO register and When the key was released, a break code was stored.

Our ARM code first checked whether the 15th bit(RVALID) was set to 1. If it is 1, it indicated that there was data to be read from the head of the FIFO register. The head of the FIFO register was pointed by the last 8 bits of the PS2_Data register. We stored the last 8 bits data of this register as a byte and passed to the subroutine. The ARM code returned 1 when the PS2_Data was successfully read. Otherwise, if the 15th (RVALID) was set to 0, the ARM code returned 0.

The C code for the keyboard was implemented by a while loop which allowed the read_PS2_Data_ASM to continuously read from the keyboard. If the data was successfully read, it called the VGA_write_byte_ASM with the value read and the next x and y coordinates. Notice that we were incrementing the x coordinate by 3 because we were writing bytes.

This part of the lab was pretty straightforward too, because we could use our codes from previous lab and we could also use VGA_Write and VGA_Read functions from section1.