Karl Doumar 260733160 March 30th, 2018

He Qian Wang 260688073

Group 32

ECSE 324 Lab 4 Report

**Introduction**

The objective of this lab was to implement and create some basic drivers to interact with I/O peripherals such as the VGA, keyboard and audio ports. We created drivers with subroutines that allowed to either read from or write to the addresses of I/O ports on the board and a main C program was implemented to test the functionality of our drivers.

**Part 1 – VGA**

The main purpose of this section, was to display ASCII characters and blocks of colored pixels by using a VGA cord between our monitor and the Cyclone 5 FPGA. We had three subroutines to implement which were to clear character buffers, write character and write byte. The initial task was to clear the memory buffers of both character and pixel buffers located in the memory of the board. For the character buffer, we read the documentation related to this character buffers and we approached the clear charbuff ASM which a similar method to clear pixelbuff ASM. We looped the each Y value through its X values and we simply reset the data inside. The method is similar to visiting every entry of a 2D array.

Concerning the write char ASM, the subroutine takes three arguments as inputs which are the x,y coordinates and which char to enter. We first check whether the x, y coordinates are valid then we simply store the corresponding character’s ascii code in the corresponding location in memory.

The write byte ASM subroutine, it was done similarly, except, we have to split the the most significant HEX and the least significant HEX from the 8 bit ASCII code. We transform the 4 bits to its corresponding hexadecimal value and we convert that hexadecimal to its ascii value by adding 48 offset and if it is bigger than 9, we would add an extra 7 offset. The result will be store in its location in memory.

**Part 2 – Keyboard**

This part of the lab consisted of creating an application that would read the raw data input from a keyboard, more specifically the scan codes. A VGA port would be used in conjecture to display them to a screen. To accomplish this, a subroutine from the VGA driver made in part 1 was needed. Another subroutine was needed which would read data from the keyboard when there is valid data. This part of the lab was a little more troublesome because of one little oversight. We would go over it at the end of the section.

The use of the \_PS2\_data\_ASM subroutine is to take the input form the keyboard and store it in a specific location where the write byte subroutine would then write it to the screen. The write byte subroutine uses as input a data pointer where the data from the keyboard will be stored if it is valid. To verify if it is valid, the subroutine checks a bit RVALID within the PS2\_Data word address. We decided to use a AND ARM operator to determine whether it is true. To check inside RVALID, the subroutine loads the byte containing it into a register and clears the other bits if they are unused. There is valid data to store if RVALID is 1, and the data is read by the data byte and stored in an address indicated by a point. Once we load the contents of the register, it will decrement its stored values. The subroutine returns a value of 1 if the data was read and a 0 if no read happened. At some point, we decided to check Load from the keyboard twice by using the LDRB, but this action decrements the value inside the register twice. As a result, our method was really clunky to use and wouldn’t properly display the F0 byte on the screen.

**Part 3 Audio**

The last part of the lab is to make students familiarize themselves with the audio output function of Cyclone 5 board. We needed to write a driver for the audio port of the lab to emit a 100Hz square wave on the output. We know that the FPGA has 48k samples/s which means that in order to reach a 100Hz wave, we need to have 240 samples per half wave. The audio acts on a first in first out manner like a queue which means that the signals will each wait in line to be output.

We know that the audio has four registers. We are using the Fifospace, lefdata and rightdata registers. On our audio\_write subroutine, we read the values of the WSRC and the WSLC, then we check if the FIFO is full. If it is, we exit the subroutine as we can’t add to the queue. If not, we store the data into the left and right channels and we branch back to our program.

In the main program, we simply loop until we reach 240 samples. We only increment if audio\_write\_ASM returns 1. If x is hi, then we set it to low, if x is low, we set it back to high. As a result, the signal is a 100Hz audio output.

One issue we ran into was that we didn’t check whether the audio\_write\_ASM would return 1, the only way we would get a signal of 100Hz was by changing the sample size to 2400 instead of 240. We believe that this is caused by overloading the FIFO.