Lab 4 Report

ECSE 324 Nada Marawan (260720514)

Prof. Davis Imad Dodin (260713381)

Demo Date: 22/03/19 **Due Date: 29/03/19**

1 VGA

This section requires us to implement functionality for writing to VGA output by using the FPGA boards Pixel and Character Buffer Interface addresses. We implement subroutines for clearing the character and pixel buffer addresses (`VGA_clear_charbuff_ASM` and `VGA_clear_pixelbuff_ASM`, respectively). We also implement subroutines for writing characters and bytes to the screen through use of the VGA character buffer, and writing pixels (of varying colours) through use of the VGA pixel buffer.

We clear the character and pixel buffers by looping "column by column" through our addresses. We increment our x coordinate through use of a counter, essentially setting a base address that we use when looping down the column. A major difficulty that we faced in this point was due to an incorrect assumption that the address for the first pixel / character on the second row immediately follows the address for hte last pixel / character on the first row. After noticing the error in this assumption, implementing this logic was not difficult.

Writing characters and bytes to the Character buffer was an interesting task as it demonstrated the importance in constructing proper subroutines for code modularity and thus minimization of effort. In essence these subroutines take in the x and y positions for the desired character / byte output in the first two arguments, followed by the desired character or byte as the final argument. We note that the latter performs shifts to write each character within the byte through use of the former subroutine (obviously storing the appropriate arguments in R0, R1 and R2 as the calling convention dictates.)

We note that this task was the most time consuming of the tasks in the lab, due to the sheer amount of looping and shifting required (whereby many errors were made during development).

2 PS/2 Keyboard

The objective of this section was to read and display input from a PS/2 keyboard. An important note to make is tat the characters read in from the PS/2 interface address are actually read in from a FIFO queue, and not directly from the hardware circuit interfacing with the keyboard. (As a sidenote, the number of contained entries here is stored in the `RAVAIL` field.

We begin the assembly subroutine by determining the validity of the data read in by the keyboard via the `RVALID` bit of the bit-sequence in the PS/2 Interface Address. If this flag is set to 1, we continue to store the received data into the passed in *memory address* of our character variable, and exit otherwise. This was an interesting task as it demonstrated the ability to return values via memory addresses as opposed to in the R0 register.

We utilize this subroutine in a C program which clear's the screen (character buffer) and continuously reads the data from the keyboard in the provided data character pointer, writing the read-in value by use of the previously implemented `VGA_write_byte_ASM` subroutine.

This section was actually implemented fairly quickly but was the most interesting part of the lab for us, as interfacing with hardware that one uses on a daily basis was exciting!

3 Audio

In this section, we successfully generate a 100 Hz square wave on the output audio port of the FPGA board.

Two addresses exist for the audio output of the board, corresponding to the left and right channels of the output devices. We begin our subroutine by verifying that the FIFO queues of both of these addresses allow more output (to avoid an overflow). We then write the sound that we desire onto the left and right Interface Addresses.

We have to, however, determine the rate of alternation to output the 100 Hz frequency wave. We use the relation that $AlternatingRate = \frac{SampleRate}{2 \times Frequency}$. The board's sampling rate is found in the manual and is given as 48000 samples per second. Our above relations thus gives us an Alternating Rate of 240 samples.

We implement this via a C program that passes a signal of `0x00FFFFFF` 240 times, followed by a signal of `0x00000000` 240 times. We repeat this process to output our desired

audio.

References

[1] Altera Corporation (2014) *DE1-SoC Computer System with ARM Cortex-A9 Manual.* San Jose, CA: Author.