ECSE 324 LAB REPORT #1

Name: Ameer Ibrahim Osman

ID: 260682723

Name: Ahmed Abdel-Magied

ID: 260653515

Throughout this lab we were able to successfully deliver 4 different algorithms, Largest integer program, Standard deviation program, Centering program, and Sorting program. In order to develop these 4 programs, we worked with the DE1-SoC Computer System, which is composed of an ARM Cortex-A9 processor and peripheral components located on the FPGA found on the DE1-SoC board. We have also used an IDE, Intel FPGA Monitor Program 16.1 which was used to ‘compile’ the code and load it on the board. The FPGA Monitor Program also has a debugging tool that has allowed us to step through the code through several stages and verify that our code works. It also allowed us to check the memory so we can verify that our registers had the correct values in any given point.

2.0 The largest Integer Program:

The largest Integer Program was a walk through program that helped us to familiarize ourselves with the assembly language as well as with the FPGA assembly kit. So we were given a list of numbers through an array, and then coded a program to find the largest number in the list and return it as the output. So we constructed a loop that iterated through the array and compared each value to its ascending value and it found the largest integer till this point. Then the loop keeps on iterating through the array and if it finds a larger integer than the current largest it swaps them. The number of iterations of the loop is determined by the size of the array and the number “N” of elements present in the array.

2.1 Fast standard deviation computation:

Unfortunately, implementing the actual Standard devotion formula (1) through assembly isn’t functional as it requires multiplication, division, and square root operations, which are not available as instructions on all processors and are slow to emulate using other instructions. Thus, we will use an a approximation method using the formula so called the “Range Rule” (2), where we subtract the maximum vale (xmax) and the minimum value (xmin) of the signal and then divide it by 4.

Formula: 1

Formula: 2

1. We declared two variables called “MAX,” and “MIN” in memory initializing their values both to 0. These two variables have the functions of storing the values of the maximum and minimum numbers in memory.
2. Then we initialized a loop to find the maximum number “MAX.” Basically what this code does is that it finds the maximum number in the array by comparing it to the other numbers in the array, similar to the largest integer program. So, it stores the first number in the list as the maximum number, and then it checks if the next number in the list is greater than the maximum number. If not then branches back to “MAX,” if yes then it updates the current number to be the maximum. Then it branches back to “MAX” and the loop keeps on running until the counter reaches “0,” this marks the end of the loop thus the algorithm now branches to store and stores the maximum number in the memory location.
3. Then we initialize the loop to find the minimum number in the list “min.” It uses the same concept as the maximum number loop but instead of finding the maximum number it finds the minimum number.
4. After that we utilized the “SUB” method which basically subtracts the maximum number found by the minimum number. Finally, we perform an arithmetic right shift by 2 in order to divide the value by 4.

Improvements:

We could have implemented the “MAX” and “MIN” counters into one counter and used one loop to find both values. This would have made our code twice as faster and would have decreased our overall code execution time.

2.2 Centering an array

It is often necessary to ensure that a signal is centered, which basically is having an average of 0. It is usually implemented within DC signals, as DC signal can damage a loudspeaker, so it is important to center an audio signal to remove DC. Basically, you can center a signal by calculating the average value of the signal and subtracting the average from every sample of the signal.

1. First of all, we calculated the average of the values in the list, we achieved this by calculating the sum of the all the values in the array by incrementing through them and then dividing it by the number of elements in the list. In order to implement this method, we initialized a method “SUM,” which basically adds the first element in the list to sum, after that we subtract our counter so we can shift to the next number in the list and then we add it to “SUM”. Then as soon as the counter reaches zero then the algorithm then branches to “SETUP,” and if not then then it should be branched back to “SUM.”
2. Then we initialized the “SETUP” counter where we load the divisor and then we dereference Number of elements for “CENTER” subtraction.
3. Then we initialized the “DIVIDE” counter in order to divide the sum by the divisor. So basically, what happens is that we subtract the Sum by divisor, then we compare the subtracted sum by divisor if its larger we branch back to “DIVIDE” if not then we branch to center. Thus, the number of subtraction iterations would provide us with the result of the division.
4. Then we initialized the “CENTER” counter, through it we center the array out. We do that by subtracting the current number by the average and then storing it in the average list, after that we point to the next element in the list and then it keeps on looping until our counter reaches 0.

Improvements:

For the division we could have just divided by the powers of 2 instead of the subtraction method that we implemented. As through the lab manual it gave us the permission to use the powers of 2 as an approximation for the division method. This would’ve made our code way faster, however less reliable.

2.3 Sorting

Through sorting we implemented the simple bubble sort algorithm below:

*// Given an array A of length N*

sorted = false;

**while** not sorted:

sorted = true;

**for** i = 2 to N:

**if** A[ i ] < A[ i - 1] , swap A[ i ] with A[ i - 1] and set sorted = false;

1. Basically, our code consists of two loops. The first loop “BUBBLESORT” is an outside loop that loop checks if it has been sorted or not. If sorted then the program is complete thus it “END”s the execution of the code. If not then it branches to the inner loop “WHILE” so it can sort out the unsorted numbers.
2. Through the inner loop the “WHILE” loop what basically happens is that we use ‘CMP’ to compare 1st and 2nd number, if the 1st number is greater than the 2nd number then we use “BGT SWAP” to branch to the “SWAP” counter. What the “SWAP” counter basically does is that it swaps these 2 number such that they are in the ascending order.
3. As soon as all of the elements in the array are in the right order then the program comes to an end.

Improvements:

Instead of implementing 2 loops we could have implemented it only in one loop this would have made our code as twice as faster.