

The City College of New York

Grove School of Engineering - Electrical Engineering Dept.

EE 42500 – Computer Engineering Lab

Professor Jinglun Feng

Spring 2020

**Experiment 5**

Report due: 4/27/2020

**Prepared By**: Ben Variano

**Objective**

The objective of this experiment is to implement moving average filters with application to discrete-time series. We take a signal x[n] and implement a variety of filters to get outputs y[n], z[n], a[n], b[n], and c[n].

**Task 1**

In Task 1, our main focus was to familiarize ourselves with the new code and ensure that we knew how and why it was operating the way that it was. One particular point of interest for us was the value register. We observed it via the *Watch* window where we could monitor what is stored in it, as well as the WREG. In addition, we familiarized ourselves with the TABLAT pointer, just understanding it’s function and what it does without going in depth about how it works since that was not necessary for the experiment.

**Task 2**

For Task 2, we were asked to truly understand the idea behind a moving average filter, and how it manipulates the signal x[n] to provide an output ( y[n], z[n], a[n], b[n], c[n]). In order to get a complete understanding of the filters, and to determine what the output for our program should be, we completed the table below. With each output corresponding the index n. We can assume that the pattern for x[n] repeats continually onwards both before and after the indices shown. Some of the earlier values of the output are based on prior values of x[n] not shown on this table. The formulas for x[n], y[n], z[n], a[n], b[n], and c[n] are shown below.

x[n] = {… 0, 50, 100, 150, 200, 250, 200, 150, 100, 50, 0, …}

y[n] = ½\*x[n] + ½ x[n-1]

z[n] = ½\*x[n] + ½ x[n-2]

a[n] = ½\*x[n] + ½ x[n-3]

b[n] = ½\*x[n] + ½ x[n-4]

c[n] = ½\*x[n] + ½ x[n-5]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | … |
| x[n] | 50 | 0 | 50 | 100 | 150 | 200 | 250 | 200 | 150 | 100 | 50 | 0 | … |
| y[n] | 75 | 25 | 25 | 75 | 125 | 175 | 225 | 225 | 175 | 125 | 75 | 25 | … |
| z[n] | 100 | 50 | 50 | 50 | 100 | 150 | 200 | 200 | 200 | 150 | 100 | 50 | … |
| a[n] | 125 | 75 | 75 | 75 | 75 | 125 | 175 | 175 | 175 | 175 | 125 | 75 | … |
| b[n] | 150 | 100 | 100 | 100 | 100 | 100 | 150 | 150 | 150 | 150 | 150 | 100 | … |
| c[n] | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | 125 | … |

*Table 1: Outputs corresponding to signal x[n]*

**Task 3**

In Task 3, we finally begin to implement code. We are asked to write the code needed in order to implement y[n] = ½\*x[n] + ½\*x[n-1]. I wrote two subroutines, one which stored a 6-value linear buffer of x[n] to x[n-5], and one that performed the required arithmetic for the calculations and stored the output for that index in the variable, output. The latter subroutine needed to add two numbers and then divide the sum by 2.

A screenshot of a cell phone

Description automatically generated

*Figure 1: Subroutines to Store Linear Buffer and Compute Arithmetic*

The above image is taken from the assembly file used to compute c[n]. We will see this code in Task 4. The only difference between this and the subroutine for y[n] in Task 3 is that v5 in the subroutine is replaced by v1 for y[n].

**Task 4**

In Task 4 we are asked to modify our code for Task 3 in order to computer z[n], a[n], b[n] and c[n] instead of y[n]. This is a very simple conversion since in Task 3, we had a 6-variable long buffer, which is more than enough to calculate each of these outputs. The code is identical for each of these new assembly files, except that the variable in the subroutine to compute the arithmetic. It will be v1-v5 depending on which output you would like. This variable corresponds to the offset of x[n] needed for the computation. For z[n], we need x[n-2] so we use variable v2. For a[n], we need x[n-3] so we use variable v3. For b[n], we need x[n-4] so we use variable v4. Lastly, for c[n], we need x[n-5] so we use variable v5. In Task 3 above, y[n] needed x[n-1] so v1 was used. Aside from this one line, the code remains identical throughout the rest of the assembly file.

**Images**

**A screenshot of a social media post

Description automatically generated**

*Figure 2: Mainline Function for all Assembly Files*

*A screenshot of a social media post

Description automatically generated*

*Figure 3: Variables I Implemented*

*A screenshot of a cell phone

Description automatically generated*

*Figure 4: Memory and Arithmetic Subroutines Used, Variable v5 Changed Depending on Desired Output*

**Conclusion**

Throughout this lab we were able to implement a variety of filters to get our desired outputs. We implemented two subroutines to create a linear memory buffer and to compute the output for each filter. The code remained the same throughout each assembly file for the varying outputs, aside from one variable (x[n-k]) in our subroutine for computing the arithmetic.

The full assembly files for each of the 5 different outputs will be linked separately from this report.