Contents

Introduction

In this lab experiment, we will explore the concept of discrete time series. Specifically, we will be implementing moving average filters. Consider the periodic time series $y[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-1]$ as described in table 1 below. The output of *y* depends on the current and previous value of *x* for some index *n*. For example, to compute $y[0]$, we need to know $x[0]$ and $x[0-1] \equiv x[-1]$ and then compute the average. Thus, $y[0] = \frac{1}{2}(x[0] + x[-1]) = \frac{1}{2}(50 + 100) = 75$ as confirmed by the table.

n	-1	0	1	2	3	4	5	6	7	8	9	10	11	...
x[n]	100	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	...

Table 1: Periodic time series of $y[n]$

In a similar matter, the time series $z[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-2]$ is described in table 2. This time, output of *z* depends on the current and previous value of *x* for some index *n* and *n-2*. For example, to compute $z[4]$, we need to know $x[4]$ and $x[4-2] \equiv x[2]$ and then compute the average. Thus, $z[4] = \frac{1}{2}(x[4] + x[2]) = \frac{1}{2}(150 + 50) = 100$ as confirmed by the table.

n	-1	0	1	2	3	4	5	6	7	8	9	10	11	...
x[n]	100	50	0	50	100	150	200	250	200	150	100	50	0	...
z[n]	...	100	50	50	50	100	150	200	200	200	150	100	50	...

Table 2: Periodic time series of $z[n]$

We can also consider the time series $a[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-3]$, $b[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-4]$, and $c[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-5]$. In more general terms, these time series can be expressed by the single time series $d[n] = \frac{1}{2}x[n] + \frac{1}{2}[n-k]$ where $k \geq 1$.

Generating $y[n]$

In each task we need to first implement a *memory buffer* to store the previous and current values. The template code already provides the current value $x[n]$ as shown below.

```
TBLRD*+
movf TABLAT, W
movwf valueL ; get current value
```

Figure 1a: Obtaining the current value from TABLAT

Next, we need a mechanism to yield the value of $x[n-1]$, which is necessary to compute $y[n]$. This can be done by simply moving the table pointer back the necessary steps. Below shows the implementation of the memory buffer. Note that in addition to moving the table pointer back, we must save the current status of the pointer and restore it later so that its location is properly set throughout each iteration.

```
; Determine x[n-1] by changing
; pointer to 3 steps back to get x[n-1]
movff TBLPTRL,ProgMem

tblrd*-
tblrd*-
tblrd*-
```

Figure 1b: Memory buffer for $y[n]$

Next, we require an adder and divider to obtain the correct results as shown in table 1. Note, we may need to use more than 1 register to store the summations if the summation is greater than 255 (beyond the storage capacity of an 8-bit register). For this reason, the summation result will span across two registers to store the lower and upper 8 bits: `SUMU` and `SUML`.

```
; Adder
movf TABLAT, W ;W = x[n-1]
movwf TEMPL
addwf valueL,W ; x[n]+x[n-1]
movwf SUML ; Store lower result into SUML
movf TEMPU,W
addwfc valueU,w ;x[n]+x[n-1] with bit carry
movwf SUMU ;Store upper result into SUMU
```

Figure 1b: Adder

To divide by 2 requires a shift or rotation of all bits 1 space to the right which is accomplished by utilizing the `rrcf` instruction to on the upper and lower summation register results. Again, as mentioned before, we restore the pointer status to ensure proper results of each iteration.

```
; Divider
rrcf SUMU, W ; ResU/2
movwf SUMU
rrcf SUML, W ; ResL/2
movwf SUML

movff ProgMem, TBLPTRL ; restore pointer location
```

Figure 1c: Divider

To verify the correctness of the code above, figures 2a to 2d show the values of $y[1]$, y[3], and y[8] respectively.

Symbol Name	Value	Decimal	Binary
counter	0x09	9	00001001
WREG	0x19	25	00011001
TABLAT	0x00	0	00000000
valueU	0x00	0	00000000
valueL	0x32	50	00110010
SUMU	0x00	0	00000000
SUML	0x19	25	00011001

Figure 2a: Watch window showing the value of $y[1]$

Symbol Name	Value	Decimal	Binary
WREG	0x4B	75	01001011
TABLAT	0x32	50	00110010
valueU	0x00	0	00000000
valueL	0x64	100	01100100
SUMU	0x00	0	00000000
SUML	0x4B	75	01001011

Figure 2b: Watch window showing the value of $y[3]$

Symbol Name	Value	Decimal	Binary
WREG	0xE1	225	11100001
TABLAT	0xC8	200	11001000
valueU	0x00	0	00000000
valueL	0xFA	250	11111010
SUMU	0x00	0	00000000
SUML	0xE1	225	11100001

Figure 2c: Watch window showing the value of $y[6]$

Generating $z[n]$

The *memory buffer* is similar to the previous task. However, to generate $z[n]$, we need to compute $x[n-2]$. Again, this can be done by moving the table pointer back the necessary steps. The memory buffer needed to generate $z[n]$ is shown in figure $z[n]$ below. The adder and divider code remains the same as in the previous implementation.

```
; ---------------------------------
; (1) WRITE CODE FOR MEMORY BUFFER HERE
;       you may write the full code
;       here or call a subroutine

; Determine x[n-2] by changing
; pointer to 4 steps back to get x[n-2]
movff TBLPTRL,ProgMem

tblrd*-
tblrd*-
tblrd*-
tblrd*-
movf TABLAT, W ;W = x[n-2]
movwf TEMPL
```

Figure 3a: Memory buffer for $z[n]$

To verify the correctness of the code above, figures 4a to 4d show the values of $z[2]$, $z[4]$ and $z[8]$ respectively.

Symbol Name	Value	Decimal	Binary
WREG	0x32	50	00110010
TABLAT	0x00	0	00000000
valueU	0x00	0	00000000
valueL	0x64	100	01100100
SUMU	0x00	0	00000000
SUML	0x32	50	00110010

Figure 4a: Watch window showing the value of $z[2]$

Symbol Name	Value	Decimal	Binary
WREG	0x64	100	01100100
TABLAT	0x32	50	00110010
valueU	0x00	0	00000000
valueL	0x96	150	10010110
SUMU	0x00	0	00000000
SUML	0x64	100	01100100

Figure 4b: Watch window showing the value of $z[4]$

Symbol Name	Value	Decimal	Binary
WREG	0xC8	200	11001000
TABLAT	0xC8	200	11001000
valueU	0x00	0	00000000
valueL	0xC8	200	11001000
SUMU	0x00	0	00000000
SUML	0xC8	200	11001000

Figure 4c: Watch window showing the value of $z[8]$

Generating $a[n]$, $b[n]$, and $c[n]$

At this point, we have realized a general pattern for generating a moving average filter. Upon moving the table pointer forward to the appropriate locations, we can obtain the correct results so long as we save and restore the table pointer status before and after each iteration. Using these methods, we can implement the filters for the following time series shown in table 3.

n	-1	0	1	2	3	4	5	6	7	8	9	10	11	…
x[n]	100	50	0	50	100	150	200	250	200	150	100	50	0	…
a[n]	…					75	125	175	175	175	175	125	75	…
b[n]	…						100	150	150	150	150	150	100	…
c[n]	…							125	125	125	125	125	125	…

Table 3: Periodic time series of $a[n]$, $b[n]$, $c[n]$

The following memory buffers are implemented for $a[n]$, $b[n]$, and $c[n]$ respectively. The adder and divider code remains unchanged.

```
; ---------------------------------
; (1) WRITE CODE FOR MEMORY BUFFER HERE
;       you may write the full code
;       here or call a subroutine

; Determine x[n-3] by changing
; pointer to 5 steps back to get x[n-3]
movff TBLPTRL,ProgMem

tblrd*-
tblrd*-
tblrd*-
tblrd*-
tblrd*-
movf TABLAT, W ;W = x[n-3]
movwf TEMPL

; ---------------------------------
; (1) WRITE CODE FOR MEMORY BUFFER HERE
;       you may write the full code
;       here or call a subroutine

; Determine x[n-4] by changing
; pointer to 5 steps back to get x[n-4]
movff TBLPTRL,ProgMem

tblrd*-
tblrd*-
tblrd*-
tblrd*-
tblrd*-
tblrd*-
movf TABLAT, W ;W = x[n-4]
movwf TEMPL
```

```
; ---------------------------------
; (1) WRITE CODE FOR MEMORY BUFFER HERE
;       you may write the full code
;       here or call a subroutine

; Determine x[n-5] by changing
; pointer to 5 steps back to get x[n-5]
movff TBLPTRL,ProgMem

tblrd*-
tblrd*-
tblrd*-
tblrd*-
tblrd*-
tblrd*-
tblrd*-
movf TABLAT, W ;W = x[n-5]
movwf TEMPL
```

Figure 5: Memory Buffers of $a[n]$, $b[n]$, and $c[n]$ (Top to bottom)

The following figures show several values for each average moving filter to verify the correctness of the code.

Symbol Name	Value	Decimal	Binary
WREG	0x4B	75	01001011
TABLAT	0x00	0	00000000
valueU	0x00	0	00000000
valueL	0x96	150	10010110
SUMU	0x00	0	00000000
SUML	0x4B	75	01001011

Figure 6a: Watch window showing the value of $a[4]$

Symbol Name	Value	Decimal	Binary
WREG	0x7D	125	01111101
TABLAT	0x32	50	00110010
valueU	0x00	0	00000000
valueL	0xC8	200	11001000
SUMU	0x00	0	00000000
SUML	0x7D	125	01111101

Figure 6b: Watch window showing the value of $a[5]$

Symbol Name	Value	Decimal	Binary
WREG	0x4B	75	01001011
TABLAT	0x00	0	00000000
valueU	0x00	0	00000000
valueL	0x96	150	10010110
SUMU	0x00	0	00000000
SUML	0x4B	75	01001011

Figure 6c: Watch window showing the value of $a[11]$

Symbol Name	Value	Decimal	Binary
WREG	0x64	100	01100100
TABLAT	0x00	0	00000000
valueU	0x00	0	00000000
valueL	0xC8	200	11001000
SUMU	0x00	0	00000000
SUML	0x64	100	01100100

Figure 7a: Watch window showing the value of $b[5]$

Symbol Name	Value	Decimal	Binary
WREG	0x96	150	10010110
TABLAT	0x32	50	00110010
valueU	0x00	0	00000000
valueL	0xFA	250	11111010
SUMU	0x00	0	00000000
SUML	0x96	150	10010110

Figure 7b: Watch window showing the value of $b[6]$

Symbol Name	Value	Decimal	Binary
WREG	0x64	100	01100100
TABLAT	0x00	0	00000000
valueU	0x00	0	00000000
valueL	0xC8	200	11001000
SUMU	0x00	0	00000000
SUML	0x64	100	01100100

Figure 7c: Watch window showing the value of $b[11]$

Symbol Name	Value	Decimal	Binary
WREG	0x7D	125	01111101
TABLAT	0x32	50	00110010
valueU	0x00	0	00000000
valueL	0xC8	200	11001000
SUMU	0x00	0	00000000
SUML	0x7D	125	01111101

Figure 8a: Watch window showing the value of $c[6]$

Symbol Name	Value	Decimal	Binary
WREG	0x7D	125	01111101
TABLAT	0x32	50	00110010
valueU	0x00	0	00000000
valueL	0xC8	200	11001000
SUMU	0x00	0	00000000
SUML	0x7D	125	01111101

Figure 8b: Watch window showing the value of $c[11]$

Conclusion

Overall, we refreshed and expanded our knowledge of discrete time signals by implementing them via assembly code. The methodology used is worked to a certain degree but in essence, it not practical if say, the filter to be implemented had a time series of $\frac{1}{2}x[n] + \frac{1}{2}x[n-10]$. Hence, a better and optimized solution should be realized as further exercise.