National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Dr. Wajahat Hussain Dated 9-5-2107

Course/Section: BEE6-B Semester: 6th Semester

EE-330 Digital Signal Processing
Lab 11 # Frequency Delays and FIR Filtering

Name	Reg. no.	Report Marks / 10	Lab Quiz-Viva Marks / 5	Total / 15
Saad Iqbal	111394			
Usman Iqbal	111393			
Abdullah Bin Asif	111596			

Introduction:

In this lab you will study sample by sample processing methods for FIR filtering and implement them on the TMS320C6713 processor. We will also implement multiple delays using linear circular display and digital audio effects using delays.

Lab Procedure 1

A complete C program that implements the above d-fold delay example on the TMS320C6713 processor is given below:

Solution:

Code:

```
// delay1.c - multiple delay example using circular buffer pointers (pwrap version)
// ---------------------------------------------------------------------------
#include "dsplab.h" // init parameters and function prototypes
short xL, xR, yL, yR; // input and output samples from/to codec
#define D 8000 // max delay in samples (TD = D/fs = 8000/8000 = 1 sec)
short fs = 8; // sampling rate in kHz
float w[D+1], *p, x, y; // circular delay-line buffer, circular pointer, input, output
int d = 4000; // must be d <= D
// ---------------------------------------------------------------------------
void main() // main program executed first
{
int n;

for (n=0; n<=D; n++) // initialize circular buffer to zero
w[n] = 0;
p = w; // initialize pointer
initialize(); // initialize DSK board and codec, define interrupts
sampling_rate(fs); // possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
audio_source(MIC); // use LINE or MIC for line or microphone input
while(1); // keep waiting for interrupt, then jump to isr()
}
// ---------------------------------------------------------------------------
interrupt void isr() // sample processing algorithm - interrupt service routine
```

```
{
read_inputs(&xL, &xR); // read left and right input samples from codec
x = (float) xL; // work with left input only
y = *pwrap(D,w,p+d); // delayed output - pwrap defined in dsplab.c
*p = x; // delay-line input
p = pwrap(D,w,--p); // backshift pointer
yL = yR = (short) y;
write_outputs(yL,yR); // write left and right output samples to codec
return;
}
//
```

---

a. Import and build provided a project on LMS for this program. Then, run it. Give the system an impulse by lightly tapping the table with the mike, and listen to the impulse response. Then, speak into the mike. You should hear repeated echoes.

b. **Change the sampling rate to 16 kHz, recompile and reload keeping the value of d the same, that is, d = 4000. Listen to the impulse response. What is the duration of the delay in seconds now?**

Solution:

The delay duration is:-

4000/16000=0.25s

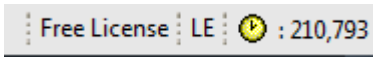
c. **Reset the sampling rate back to 8 kHz, and this time change d to its maximum value d = D = 8000.Recompile, reload, and listen to the impulse response. Experiment with lower and lower values of d and listen to your delayed voice until you can no longer distinguish a separate echo. How many milliseconds of delay does this correspond to?**

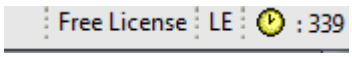
Solution:

The delay duration is:-

8000/8000=1s

d. **In this part you will profile the computational cost of the sample processing algorithm. Open the source file template.c in a CCS window. Locate the read_inputs line in the isr(), then right-click on that line and choose Breakpoint; a dot will appear in the margin. Do the same for the write_outputs line. From the top menu of the CCS window, choose Run -> Clock ; a little yellow clock will appear on the right bottom status line of CCS. When you compile, load, and run your program, it will stop at the first breakpoint, with a arrow pointing to it. Reset the clock by chossing Run -> Clock→reset to clear the number of cycles, then click resume to continue running the program and it will stop at the second breakpoint. Read and record the number of cycles shown next to the profile clock.**





Solution:

The number of cycles are equal to 339.

e. **Change the code of tamplate.c to makes use of the function qwrap instead of pwrap. Repeat parts (a) and (d).**

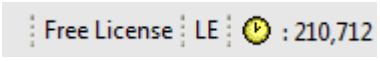
Code

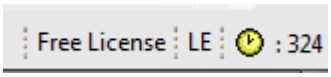
```
// template.c - to be used as starting point for interrupt-based programs
//
// 332:348 DSP Lab - Spring 2011 - S. J. Orfanidis
//
// ---------------------------------------------------------------------------

#include "dsplab.h" // init parameters and function prototypes
short xL, xR, yL, yR; // input and output samples from/to codec
#define D 8000 // max delay in samples (TD = D/fs = 8000/8000 = 1 sec)
short fs = 8; // sampling rate in kHz
float w[D+1], *p, x, y; // circular delay-line buffer, circular pointer, input, output
int d = 8000; // must be d <= D
int q;
// ---------------------------------------------------------------------------
void main() // main program executed first
{
int n;
```

```
for (n=0; n<=D; n++) // initialize circular buffer to zero
w[n] = 0;
p = w; // initialize pointer
initialize(); // initialize DSK board and codec, define interrupts
sampling_rate(fs); // possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
audio_source(MIC); // use LINE or MIC for line or microphone input
while(1); // keep waiting for interrupt, then jump to isr()
}
// ------------------------------------------------------------------------------
interrupt void isr() // sample processing algorithm - interrupt service routine
{
read_inputs(&xL, &xR); // read left and right input samples from codec
x = (float) xL; // work with left input only
y = w[qwrap(D,q+d)]; // delayed output - pwrap defined in dsplab.c
w[q] = x; // delay-line input
q = qwrap(D,--q); // backshift pointer
yL = yR = (short) y;
write_outputs(yL,yR); // write left and right output samples to codec
return;}
```

part D:





Solution:

The number of cycles are equal to 324.

f. Next, Change the code of tamplate.c so that it uses linear buffers. Its isr() will be as follows:

Build the project. You will find that it may not run (because the data shifts require too many cycles that over-run the sampling rate). Change the program parameters D and d to the following values D = 2000 and d = 1000. Rebuild and run the program. Repeat part (d) and record the number of cycles. Change the parameters D, d of the program delay1.c to the same values, and repeat part (d) for that. Comment on the required number of samples using the linear vs. the circular buffer implementation.

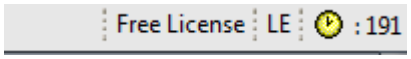
Code:

```
#include "dsplab.h" // init parameters and function prototypes
short xL, xR, yL, yR; // input and output samples from/to codec
#define D 2000 // max delay in samples (TD = D/fs = 8000/8000 = 1 sec)
short fs = 8; // sampling rate in kHz
float w[D+1], *p, x, y; // circular delay-line buffer, circular pointer, input, output
int q;
int d = 1000; // must be d <= D
// ----------------------------------------------------------------------------
void main() // main program executed first
{
int n;
for (n=0; n<=D; n++) // initialize circular buffer to zero
w[n] = 0;
p = w; // initialize pointer
initialize(); // initialize DSK board and codec, define interrupts
sampling_rate(fs); // possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
audio_source(MIC); // use LINE or MIC for line or microphone input
while(1); // keep waiting for interrupt, then jump to isr()
}
// ----------------------------------------------------------------------------
interrupt void isr()
{
int i;
read_inputs(&xL, &xR);
x = (float) xL;
w[0] = x; // delay-line input

y = w[d]; // delay output
for (i=D; i>=0; i--) // update linear buffer
w[i] = w[i-1];
yL = yR = (short) y;
write_outputs(yL,yR);
return;
}
```

Solution:

The number of *Circular cycles* are equal to 191.



The number of *Linear cycles* are equal to 55145.

Comments:

The cycles for circular buffers are much lower than that of linear ones. For a larger D linear buffers are not efficient as large amounts of data are to be moved around in the memory. But in circular one no shifting occurs except in the beginning thus the lesser number of cycles.

Lab Procedure 2

Set the sampling rate to 8 kHz and the audio source to microphone. Choose the delay to be D = 4000,corresponding to TD = 0.5 sec, so that the total duration of the filter is 3TD = 1.5 sec, and set a = 0.5.

a. **Write a C program called comb.c that incorporates the above interrupt service routine. You will need to globally declare/define the parameters D, a, p, as well as the circular buffer w to be a 3D+1 dimensional float array. Make sure you initialize the buffer to zero inside main(), as was done in the previous example, and also initialize p = w.Build and run this project. Listen to the impulse response of the filter by tapping the table with the mike. Speak into the mike. Bring the mike**

Solution:

Code:

```
#include "dsplab.h" // init parameters and function prototypes
short xL, xR, yL, yR; // input and output samples from/to codec
#define D 4000 // max delay in samples (TD = D/fs = 8000/8000 = 1 sec)
short fs = 8; // sampling rate in kHz
float w[3*D+1], *p, x, y; // circular delay-line buffer, circular pointer, input, output
int d = 4000; // must be d <= D
float a=0.5;

// ----------------------------------------------------------------------------
```

```
void main() // main program executed first
{

int n;
for (n=0; n<=D; n++) // initialize circular buffer to zero
{w[n] = 0;}
p = w; // initialize pointer
initialize(); // initialize DSK board and codec, define interrupts
sampling_rate(fs); // possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
audio_source(MIC); // use LINE or MIC for line or microphone input
while(1); // keep waiting for interrupt, then jump to isr()
}
// ----------------------------------------------------------------------------
interrupt void isr()
{
float s0, s1, s2, s3, y; // states & output
read_inputs(&xL, &xR); // read inputs from codec
s0 = (float) xL; // work with left input only
s1 = *pwrap(3*D,w,p+D); // extract states relative to p
s2 = *pwrap(3*D,w,p+2*D); // note, buffer length is 3D+1
s3 = *pwrap(3*D,w,p+3*D);
y = s0 + a*s1 + a*a*s2 + a*a*a*s3; // output sample
*p = s0; // delay-line input
p = pwrap(3*D,w,--p); // backshift pointer
yL = yR = (short) y;
write_outputs(yL,yR); // write outputs to codec
return;
}
```

b. **Keeping the delay D the same, choose a = 0.2 and run the program again. What effect do you hear? Repeat for a = 0.1. Repeat with a = 1.**

Solution:

We noticed a change in intensity. As the value of ***'a'*** increased a more intense echo was heard.

c. **Set the audio input to LINE and play your favorite wave or MP3 song into the input. Experiment with reducing the value of D in order to match your song's tempo to the repeated echoes.**

Solution:

Code

```
#include "dsplab.h" // init parameters and function prototypes
short xL, xR, yL, yR; // input and output samples from/to codec
#define D 100
short fs = 8; // sampling rate in kHz
float w[3*D+1], *p, x, y; // circular delay-line buffer, circular pointer, input, output
int q;
float a=1;
int d = 100; // must be d <= D
// ---------------------------------------------------------------------------
void main() // main program executed first
{
int n;
for (n=0; n<=D; n++) // initialize circular buffer to zero
w[n] = 0;
p = w; // initialize pointer
initialize(); // initialize DSK board and codec, define interrupts
sampling_rate(fs); // possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
audio_source(LINE); // use LINE or MIC for line or microphone input
while(1); // keep waiting for interrupt, then jump to isr()
}
// ---------------------------------------------------------------------------
interrupt void isr()
{
float s0, s1, s2, s3, y; // states & output
read_inputs(&xL, &xR); // read inputs from codec
s0 = (float) xL; // work with left input only
s1 = *pwrap(3*D,w,p+D); // extract states relative to p
s2 = *pwrap(3*D,w,p+2*D); // note, buffer length is 3D+1
s3 = *pwrap(3*D,w,p+3*D);
y = s0 + a*s1 + a*a*s2 + a*a*a*s3; // output sample

*p = s0; // delay-line input
p = pwrap(3*D,w,--p); // backshift pointer
yL = yR = (short) y;
write_outputs(yL,yR); // write outputs to codec
return;
}
```

Comments:

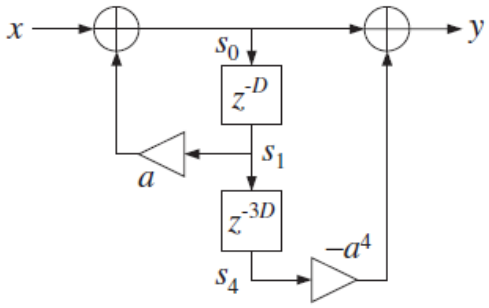
The delay decreases as the value of D decreases. (direct relation)

d. The FIR comb can also be implemented recursively using the geometric series formula to rewrite its transfer function in the recursive form as shown in Eq. (8.2.9) of the text:

$$H(z) = 1 + az^{-D} + a^2z^{-2D} + a^3z^{-3D} = \frac{1 - a^4z^{-4D}}{1 - az^{-D}}$$

This requires a (4D+1)-dimensional delay-line buffer w. The canonical realization and the corresponding

sample processing algorithm are shown below:



for each input x do:
$s_1 = *(p + D)$
$s_4 = *(p + 4D)$
$s_0 = x + as_1$
$y = s_0 - a^4s_4$
$*p = s_0$
$--p$

Write a new program, comb2.c, that implements this algorithm. Remember to define the buffer to be a (4D+1)-dimensional float array. Using the values D = 1600 (corresponding to a 0.2 sec delay) and a = 0.5, recompile and run both the comb.c and comb2.c programs and listen to their outputs. In general, such recursive implementations of FIR filters are more prone to the accumulation of roundoff errors than the non-recursive versions. You may want to run these programs with a = 1 to observe this sensitivity.

Solution:

Code

```
// comb2.c - multiple delay example using circular buffer pointers (pwrap version)
// ------------------------------------------------------------------------------
#include "dsplab.h" // init parameters and function prototypes
short xL, xR, yL, yR; // input and output samples from/to codec
#define D 16000 // max delay in samples
short fs = 8; // sampling rate in kHz
float w[4*D+1], *p, x, y; // circular delay-line buffer, circular pointer, input, output
int d = 4000;
// ------------------------------------------------------------------------------
void main() // main program executed first
{
```

```
int n;
for (n=0; n<=D; n++) // initialize circular buffer to zero
w[n] = 0;
p = w; // initialize pointer
initialize(); // initialize DSK board and codec, define interrupts
sampling_rate(fs); // possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
audio_source(MIC); // use LINE or MIC for line or microphone input
while(1); // keep waiting for interrupt, then jump to isr()
}
// ----------------------------------------------------------------------------
interrupt void isr()
{
float s0, s1, s2, s3,s4, y,a=1; // states & output
read_inputs(&xL, &xR); // read inputs from codec
// work with left input only
s1 = *pwrap(4*D,w,p+D); // extract states relative to p
//s2 = *pwrap(4*D,w,p+2*D); // note, buffer length is 3D+1
//s3 = *pwrap(4*D,w,p+3*D);
s4 = *pwrap(4*D,w,p+4*D);
s0 = a*s1+(float) xL;
y = s0 - a*a*a*a*s4; // output sample
*p = s0; // delay-line input
p = pwrap(4*D,w,--p); // backshift pointer
yL = yR = (short)10*y;
write_outputs(yL,yR); // write outputs to codec
return;
}
```

Comments:

The error increases as the dimensional float array becomes larger.

Conclusion:

- From this lab we got to know about the practical understanding of an audio signal.
- We analyzed the signal by varying delay parameters
- Echoes were generated