## Background

The goal of this project was to use our knowledge of signals and systems to design, develop, and analyze the performance of two different filters to obtain a 10 Hz sine wave from a 10 Hz pulse train. We accomplished this by building an analog 4th order Butterworth filter using electrical components including two op amp stages and implementing a 10th order modified comb filter digitally via Arduino code.

We began by determining the transfer equations for our filters. For the Butterworth filter, we were able to find a transfer function by isolating the left-most poles of the equation

$$H(s) * H(-s) = \frac{1}{1+(\frac{s}{j\omega_c})^{2N}},$$

where N = 4 and $\omega_c = 20\pi$.

Once we had found those poles, we reconstructed the denominator of the FRF via its roots. Finally, we compared the reconstructed equation for H(s) to the transfer function of a two-stage Sallen-Key circuit in order to solve for the values of the four resistors we would need to physically implement the filter based on given capacitors. The required resistor values were:

| | $R_1^I$ | $R_2^I$ | $R_1^{II}$ | $R_2^{II}$ |
|---|---|---|---|---|
| Calculated Value (Ω) | 46506.5 | 247574 | 26605.7 | 95206.2 |
| Standard Resistors Value (kΩ) | 47 | 270 | 27 | 100 |

*Table 1.1 - Calculated and standard resistor values for Sallen-Key circuit*

With these values, we were able to set up our own two-stage Sallen-Key circuit on a breadboard, thereby creating an analog system taking in samples from a 10.3 Hz pulse train generator and outputting data transformed by the Butterworth filter. We recorded the filter's output to be used as input for our modified comb filter.

Our next step was to modify the transfer function for a 10th order comb filter such that it would attenuate all higher-frequency parts of the signal except for the first harmonic, which will help us recreate the sinusoid, and anything at a frequency of 0 (the DC voltage of the system). We took the initial transfer function equation:

$$H_{comb}(z) = b_0(1 - z^{-10})$$

and multiplied it by

$$\frac{z^3}{(z-1)(z-e^{\frac{j\pi}{5}})(z-e^{\frac{-j\pi}{5}})}$$

to remove the zeros at ω = 0 as well as at ω = ± π/5 (the system's first harmonic), and three of the ten poles. This gave us a modified comb filter that passed the desired frequencies.

Once we had our final modified transfer function, we solved for a $b_0$ value that would make the gain at 0 equal to 1 (preserving the DC input) and turned our transfer function into an impulse response in the discrete time domain. From the impulse response $h_{modified\_comb}[n]$, we reconstructed the filter's difference equation, which we could implement digitally on Arduino (for code, see Appendix A).

$$y[n] = b_0x[n-7] + b_1x[n-6] + b_2x[n-5] + b_3x[n-4]$$
$$+ b_4x[n-3] + b_5x[n-2] + b_6x[n-1] + b_7x[n]$$

| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $b_k$ | 0.0382 | 0.1000 | 0.1618 | 0.2000 | 0.2000 | 0.1618 | 0.1000 | 0.0382 |

*Table 1.2 - Modified comb filter difference equation coefficients*

The output from the Butterworth filter was used as the input for this digital modified comb filter, which ultimately aimed to produce a sinusoidal wave output.

## Experimental Results

We applied an analog 4th order Butterworth low pass filter and a digital modified comb low pass filter in series to the pulse train in order to obtain a sinusoidal signal. The contributions of each stage to the overall filtering are shown in these plots of the respective inputs and outputs to each filter that were sampled simultaneously at 100 Hz (sampling period  = 0.01 s).

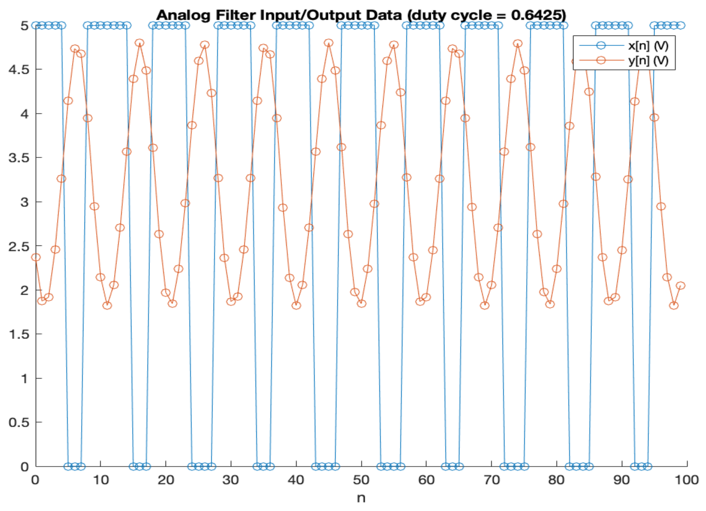

*Figure 2.1. Sampled ~10.3 Hz pulse train (blue) and output (orange) from analog 4th order Butterworth filter.*

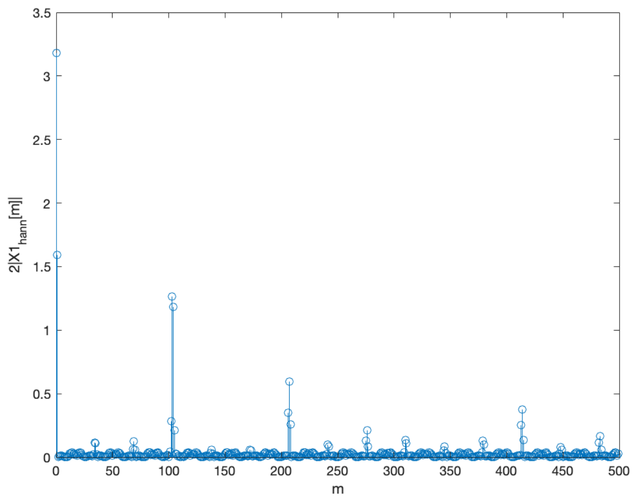

*Figure 2.2. Spectrum of sampled pulse train. The first harmonic peak is at m=103 with a secondary peak at m=104, which indicates that the fundamental frequency of the pulse train is between 10.3 and 10.4 Hz.*

| $k$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $a_k$ | 3.2127 | 1.4600 | 0.6667 | 0.1407 | 0.5224 |

*Table 2.1. Fourier series coefficients for sampled pulse train input data.*

After normalizing our unprocessed data to the actual 0-5 V range of the Arduino, the heights of the peaks in our spectrum plots correspond directly to the amplitudes of the frequency components in voltage. We see that the DC peak matches the mean voltage of the pulse train, 3.21 V. Combined with the amplitude, we find that the duty cycle of the pulse train is 0.6425.

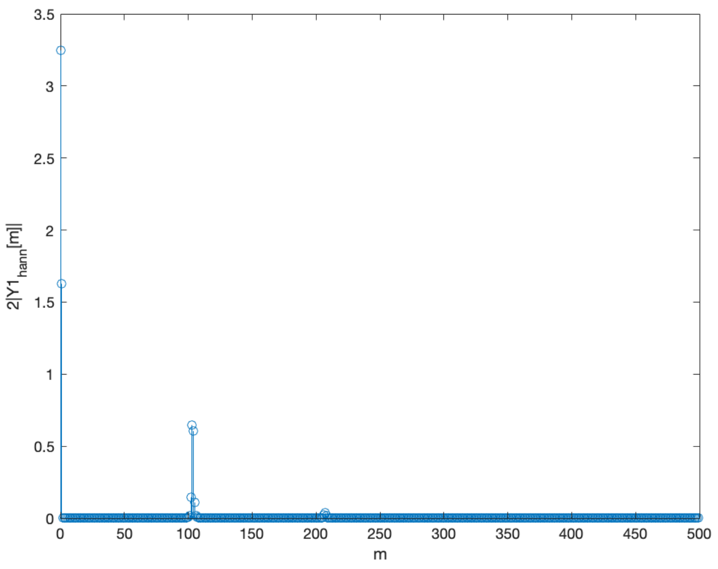

*Figure 2.3. Spectrum of analog low pass filter output, showing attenuation of the 2nd and higher harmonics of the original pulse train.*

We then performed the same analysis for the digital filter.

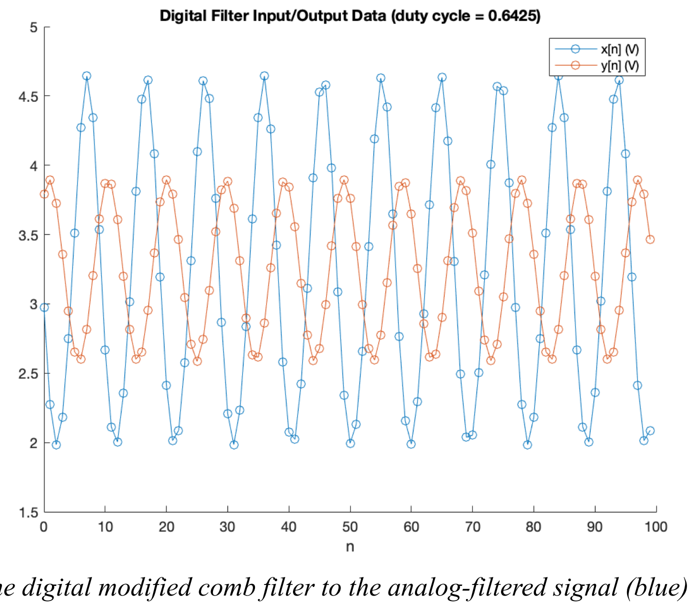

*Figure 2.4. Applying the digital modified comb filter to the analog-filtered signal (blue) produces a ~10.4 Hz sinusoid (orange).*

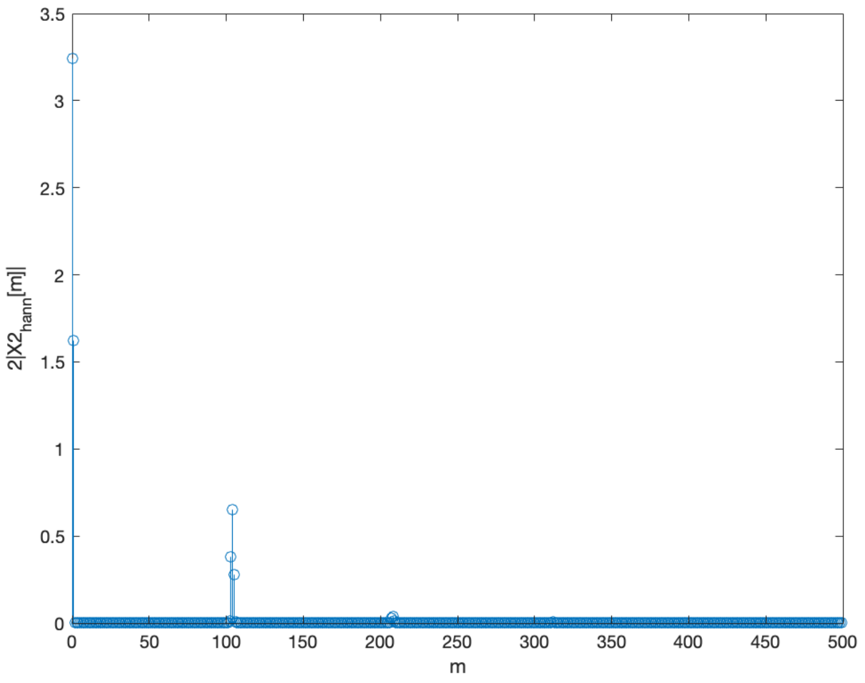

*Figure 2.5. Spectrum of digital low pass filter input (same as analog filter output, but from a different data series).*

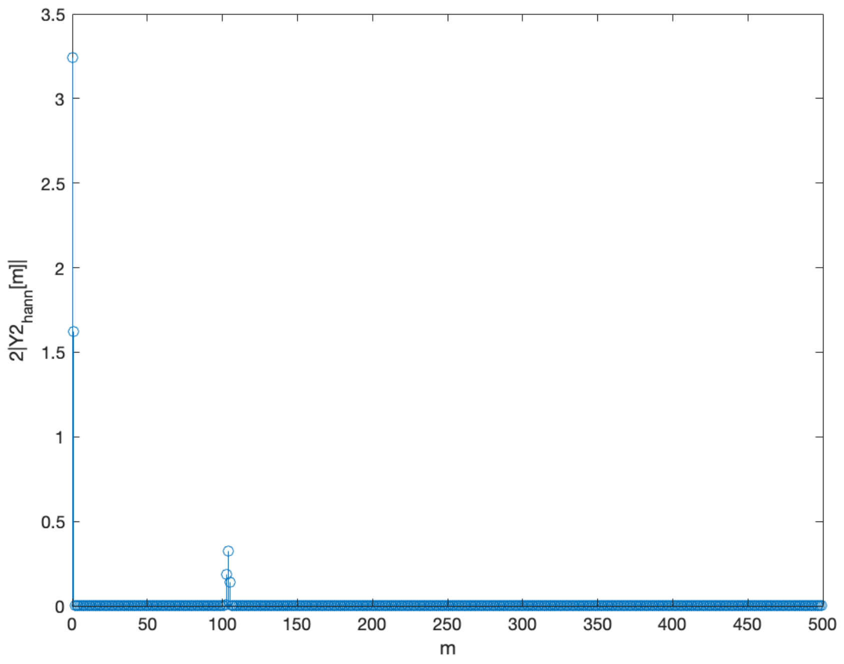

*Figure 2.6. Spectrum of digital low pass filter output. The 2nd and higher harmonics are fully attenuated, leaving only the original DC voltage and the 1st harmonic at about 10.4 Hz that composes our resulting sinusoid.*

## Analysis of Results

Since taking the Fourier Transform of a signal includes an assumption that the data is periodic, we implemented a Hanning window on all of our data to ensure that the periodic boundary condition is satisfied. MATLAB's Hanning window function halves the amplitude of the signal in the middle of the window, so we consider double the Fourier coefficients that we obtained from windowed data.

A pulse train can be represented as a linear combination of sine waves with different frequencies. When we implemented a Butterworth low pass filter, we expected it to attenuate most of the higher frequencies present in our signal. This can be clearly seen in Figure 2.3 of our analysis section, as the magnitudes of the coefficients for all harmonics except the first have been reduced significantly, and the first harmonic itself has also been attenuated. We can also look at the expected and experimental gain caused by the filter to determine the accuracy of our results.

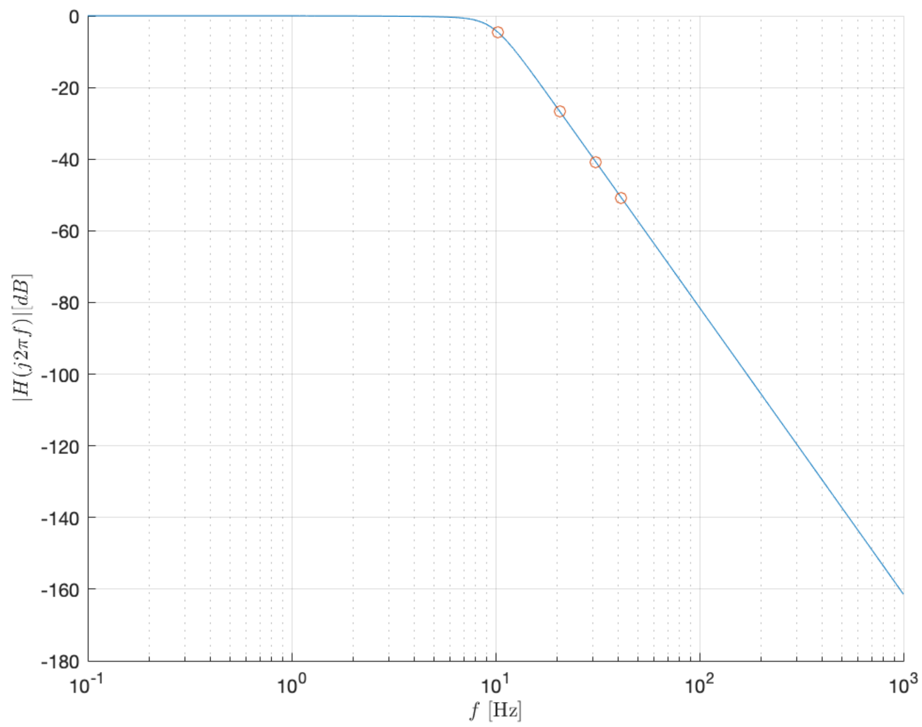

*Figure 3.1. Bode plot for analog 4th order Butterworth low pass filter. Experimentally determined frequencies of the 1st-4th harmonics of our pulse train are marked in orange.*

| k | $2\|X1_{hann}[kN]\|$ | $2\|Y1_{hann}[kN]\|$ | $\|Y1_{hann}[kN]\|/\|X1_{hann}[kN]\|$ | $\|H_{analog}(e^{j2\pi f})\|$ |
|---|---|---|---|---|
| 0 | 3.1825 | 3.2475 | 1.0204 | 1.0000 |
| 1 | 1.2662 | 0.6479 | 0.5117 | 0.5837 |
| 2 | 0.5968 | 0.0383 | 0.0642 | 0.0458 |
| 3 | 0.1342 | 0.0042 | 0.0314 | 0.0092 |
| 4 | 0.3760 | 0.0009 | 0.0024 | 0.0029 |

*Table 3.1. Coefficients of spectrum plots for analog input and output.*

Table 3.1 shows the magnitude of the harmonic peaks for the input and output data of our physical Butterworth filter, as represented by twice the magnitude of the Hanning windowed data. It then compares the ratio of those values to the expected attenuation (no longer in log magnitude) indicated by the Butterworth filter Bode diagram at the harmonic frequencies. At each harmonic, we can see that the reduction in coefficient magnitude caused by our filter is relatively consistent with the expected gain reduction given by the Butterworth transfer function.

When we implemented the modified comb filter, we expected it to finish attenuating sinusoids of all frequencies except the first harmonic. Since the first harmonic represents the sinusoid that guides the shape and frequency of the pulse train, knocking out all frequencies except this one should have ultimately turned our pulse train into a single sinusoid of comparable frequency. In our analysis section, Figure 2.6, the frequency plot of the data after the digital filter has been applied, demonstrates attenuation of the coefficients at all harmonics higher than the first to negligible values, which is exactly what we expected to see. Additionally, Figure 2.4 shows the shape of our final data to be that of an approximately 10.4 Hz sinusoid. We will also compare expected and experimental gain values for the digital filter.

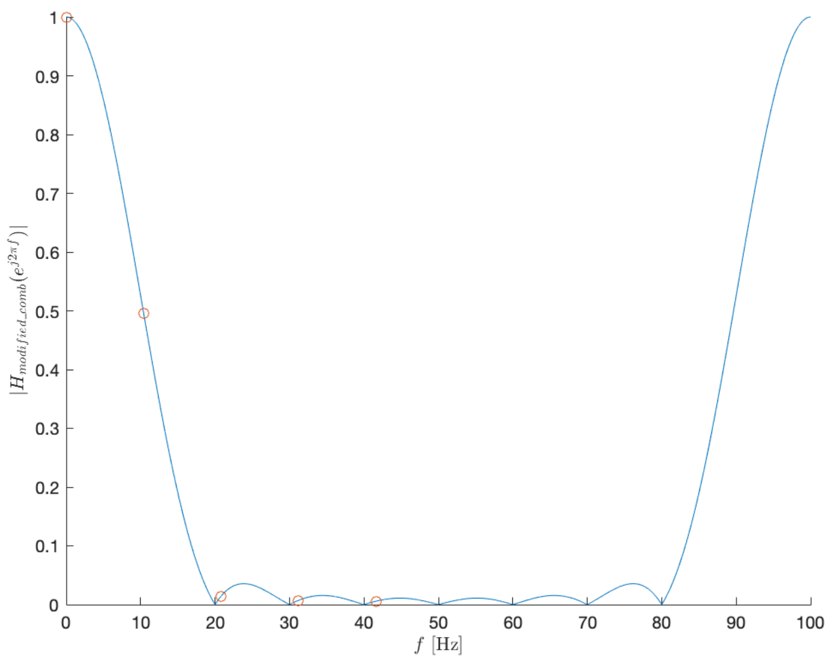

*Figure 3.2. Frequency response of the digital modified comb low pass filter. The 1st-4th harmonics as well as the DC voltage (f=0) are marked in orange.*

| k | $2\|X2_{hann}[kN]\|$ | $2\|Y2_{hann}[kN]\|$ | $\|Y2_{hann}[kN]\|/\|X2_{hann}[kN]\|$ | $\|H_{digital}(e^{j2\pi f})\|$ |
|---|---|---|---|---|
| 0 | 3.2428 | 3.2405 | 0.9993 | 1.0000 |
| 1 | 0.6534 | 0.3248 | 0.4970 | 0.4963 |
| 2 | 0.0389 | 0.0005 | 0.0119 | 0.0143 |
| 3 | 0.0053 | 0.0001 | 0.0130 | 0.0071 |
| 4 | 0.0008 | 0.0001 | 0.1314 | 0.0057 |

*Table 3.2. Coefficients of spectrum plots for digital input and output*

Much like Table 3.1, Table 3.2 compares the ratio of the digital filter input/output Fourier coefficients to the expected attenuation based on our modified comb filter frequency response function (Figure 3.2). Again, we have a strong correspondence between the experimental and expected values, especially at the lower frequency harmonics, which suggests our low pass filtration has been successful.

Although we constructed our analog filter with the higher damping Sallen-Key circuit *before* the lower damping Sallen-Key circuit, if we consider the circuit as a whole we see it doesn't really matter what order the op amps go in. Ultimately, two op amps in series will have a transfer function equal to the product of the transfer function for each op amp, and since multiplication is commutative, it doesn't make a difference which circuit gets placed first. In fact, there is actually a large range of possible circuit configurations that could make a working 4th order Butterworth low pass filter out of two op amps. We were given values for the capacitors to help us solve for workable resistor values, but a variety of capacitor-resistor combinations could potentially create the same effect as the analog filter we chose to implement. The most important job of the physical filter—which ours successfully achieved—is that it successfully filters out higher frequencies from the incoming data. As long as it does this, order of the op amps should not impact the performance of the filter.

## Conclusions

Based on the data we collected, the fundamental frequency of our kit's pulse train generator was ~10.3 Hz. The fundamental frequency of our resultant sinusoid was ~10.4 Hz. This indicates that we were successful in using an analog Butterworth filter and a digital comb filter to construct a sinusoid from our original pulse train.

Using a 1000-point window of sampled data gave us a frequency resolution of 0.1 Hz, meaning that we can only report frequencies to the tenths place. We know that our fundamental frequency is between 10.3 Hz and 10.4 Hz, but without a higher frequency resolution, we can't know the exact values of the harmonic frequencies. This discrepancy may be responsible for some level of error, or at least lack of precision, in our final answers. As shown in our frequency plots, our harmonic peaks did not always line up exactly from one filter to the next, implying that the true peak is somewhere between our two appearing values. Without exact peak frequencies, our coefficient ratios may be slightly off. Additionally, our comb filter was designed for a 10 Hz signal, so its performance was not perfect at 10.4 Hz but was largely as expected.

We did not run into aliasing issues because we collected all of our data using a sampling frequency of 100 Hz, which is more than double the fundamental frequency of the given pulse train signal (10 Hz). The amplitudes of the higher harmonics (at which aliasing could occur) are so much lower than that of the first harmonic that we are not concerned about losing the overall signal. The main differences between our expected and experimental values likely originate in the relatively large resolution frequency at which we analyzed our data. Another small source of potential error could be the existence of general analog noise.

Overall, we were able to successfully apply signal processing techniques to filter out unwanted frequencies from a signal in order to obtain a desired signal.

## Appendix A: Arduino code

AnalogInput2.ino

Modified from template. Reads two analog input pins, in this case the pulse train that is the input to the analog 4th order Butterworth low pass filter and its output.

```
/*
  Analog input 2

 Reads 2 analog input pins (range 0-1023) and
 prints the results to the serial monitor.

 The circuit:
 * analog signal connected to analog pin A0.
 * analog signal connected to analog pin A1.

*/

// These constants won't change.  They're used to give names
// to the pins used:
const int inputPin0 = A0;  // Analog input pin 0
const int inputPin1 = A1;  // Analog input pin 1

void setup() {
      Serial.begin(9600);
}

void loop() {

  // analogRead values from 0 to 1023 (10 bit ADC)
  int sensorVal0 = analogRead(inputPin0);
  int sensorVal1 = analogRead(inputPin1);


  // print the results to the serial monitor:
  Serial.print(sensorVal0);
  Serial.print("  ");
  Serial.println(sensorVal1);

  delay(10);  // sample time ms
}
```

Digital_Filter.ino

Modified from template. Reads an analog input pin and implements a digital modified comb low pass filter using the difference equation. In this case, the input to the digital filter is the output of the analog filter.

```
/*
  Digital FIR Filter

 Reads analog input pin (range 0-1023) and
 prints the filtered output to the serial monitor

 The circuit:
 * analog signal connected to analog pin A1

*/
const int inputPin = A1;  // Analog input pin

// filter coefficients
const float b0=0.0382;
const float b1=0.1000;
const float b2=0.1618;
const float b3=0.2000;
const float b4=0.2000;
const float b5=0.1618;
const float b6=0.1000;
const float b7=0.0382;

// initial values for internal signals
int xstore1 = 0;
int xstore2 = 0;
int xstore3 = 0;
int xstore4 = 0;
int xstore5 = 0;
int xstore6 = 0;
int xstore7 = 0;

int yn = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {

  // analogRead value from pin A1 (0-1023)
  int xn = analogRead(inputPin);


  // difference equation for filter
```

```
   yn= b0*xstore7  + b1*xstore6 + b2*xstore5 + b3*xstore4 + b4*xstore3 + b5*xstore2 +
b6*xstore1 + b7*xn;

  // print to output
 Serial.print(xn);
 Serial.print("  ");
 Serial.println(yn);

  // update stored values
xstore7 = xstore6;
xstore6 = xstore5;
xstore5 = xstore4;
xstore4 = xstore3;
xstore3 = xstore2;
xstore2 = xstore1;
xstore1 = xn;

  delay(10);  // sample time ms
}
```

## Appendix B: MATLAB code

```
% normalize experimental data to 0-5 V range (from 0-1023)
x1 = 5/1023 .* x;         % analog input (pulse train)
y1 = 5/1023 .* x_f;       % analog output
x2 = 5/1023 .* x_f2;      % digital input
y2 = 5/1023 .* x_mc;      % digital output (final ~10 Hz sinusoid)

A = (max(x1)-min(x1));    % amplitude         5      V
a0 = mean(x1);             % average voltage   3.2127    V
d = a0/A;                 % duty cycle        0.6425

Ts = 10e-3;  % sampling period [s]
Fs = 1/Ts;   % sampling frequency [Hz] = 100 Hz

% fundamental frequency peaks at m = 103 --> N = 103
N = 103;

% signal duration, D = fundamental period
D = 1000;
% frequency resolution, w_r = 2pi/N or f_r = 1/D
w_r = 2*pi/D;
f_r = 1/D;

% fundamental period of original signal, T0 = 1/(106 * 1/1000 * 100)
% T0 = 1/(N * f_r * f_s)
T0 = 1/(N * f_r * Fs);  % 0.0971 [s]
F0 = 1/T0;              % 10.3 [Hz]
% fundamental period of original signal in discrete sampled pulse train
% N0 = T0 * Fs = T0 * 1/Ts
N0 = T0/Ts;             % 9.7087 --> 10 [dimensionless]

%% Analog Filter In/Out

figure
hold on
title('Analog Filter Input/Output Data (duty cycle = 0.6425)')
plot(0:99, x1(1:100), '-o')
plot(0:99, y1(1:100), '-o')
xlabel('n')
legend('x1[n] (V)', 'y1[n] (V)')

%% Digital Filter In/Out

figure
hold on
title('Digital Filter Input/Output Data (duty cycle = 0.6425)')
plot(0:99, x2(1:100), '-o')
plot(0:99, y2(1:100), '-o')
xlabel('n')
legend('x2[n] (V)', 'y2[n] (V)')

%% SPECTRUM PLOTS
```

```
% 1000-pt Hanning window
w = hann(D);

%% Spectrum Analog In

% apply 1000-pt Hanning window to sampled signal
x1_hann = x1(1:1000) .* w;
[X1_hann, ~] = fdomain(x1_hann, Fs);

% plot one-sided DFT magnitude vs index, m = 0,...,499
% Spectrum plot for Butterworth input (pulse train) using Hanning window
figure
stem(0:499, 2 .* abs(X1_hann(501:1000)))
ylabel('2|X1_{hann}[m]|')
xlabel('m')

%% Spectrum Analog Out

% apply 1000-pt Hanning window to sampled signal
y1_hann = y1(1:1000) .* w;
[Y1_hann, ~] = fdomain(y1_hann, Fs);

% plot one-sided DFT magnitude vs index, m = 0,...,499
% Spectrum plot for Butterworth output using Hanning window
figure
stem(0:499, 2 .* abs(Y1_hann(501:1000)))
ylabel('2|Y1_{hann}[m]|')
xlabel('m')

%% Spectrum Digital In

% apply 1000-pt Hanning window to sampled signal
x2_hann = x2(1:1000) .* w;
[X2_hann, ~] = fdomain(x2_hann, Fs);

% plot one-sided DFT magnitude vs index, m = 0,...,499
% Spectrum plot for Butterworth output using Hanning window
figure
stem(0:499, 2 .* abs(X2_hann(501:1000)))
ylabel('2|X2_{hann}[m]|')
xlabel('m')

%% Spectrum Digital Out

% apply 1000-pt Hanning window to sampled signal
y2_hann = y2(1:1000) .* w;
[Y2_hann, ~] = fdomain(y2_hann, Fs);

% plot one-sided DFT magnitude vs index, m = 0,...,499
% Spectrum plot for Butterworth output using Hanning window
figure
stem(0:499, 2 .* abs(Y2_hann(501:1000)))
ylabel('2|Y2_{hann}[m]|')
xlabel('m')
```

```
%% FILTER FRF PLOTS

% log magnitude [dB]
LM = @(G) 20 .* log10(G);

%% Analog Filter FRF

C1a = 0.22e-06; % 0.22 μF
C2a = 0.10e-06; % 0.1  μF
C1b = 1.00e-06; % 1 μF
C2b = 0.10e-06; % 0.1  μF

R1a =  47e+03;  %  47  kΩ
R2a = 270e+03;  % 270  kΩ
R1b =  27e+03;  %  27  kΩ
R2b = 100e+03;  % 100  kΩ

Ha = @(s) 1./(s.^2 * R1a*R2a*C1a*C2a + s * (R1a+R2a)*C2a + 1);  % stage 1
Hb = @(s) 1./(s.^2 * R1b*R2b*C1b*C2b + s * (R1b+R2b)*C2b + 1);  % stage 2
H_analog = @(s) Ha(s) .* Hb(s);               % overall Butterworth filter

% plot FRF |H_analog(j*2*pi*f)| vs f
figure; hold on
fvec = 0.1:0.001:1000;
semilogx(fvec, LM(abs(H_analog(1j* 2*pi.*fvec))))
f1vec = 0.1 .* [0 103 207 310 414];
semilogx(f1vec, LM(abs(H_analog(1j *2*pi*f1vec))), 'o') % dots at harmonics
set(gca, 'XScale', 'log')
grid on
xlabel('$f$ [Hz]', 'Interpreter', 'latex')
ylabel('$|H(j2{\pi}f)| [dB]$', 'Interpreter', 'latex')

%% Digital Filter FRF

% modified comb filter
b0 = 0.038196601125011;
H_modified = @(z) b0 .* ((z - (exp(1j*2*pi/5))) .* (z - (exp(1j*3*pi/5))) ...
       .* (z - (exp(1j*4*pi/5))) .* (z - (exp(1j*5*pi/5))) .* (z - (exp(1j*6*pi/5)))
...
       .* (z - (exp(1j*7*pi/5))) .* (z - (exp(1j*8*pi/5)))) ...
       ./ z.^7;

f2vec = 0.1 .* [0 104 208 312 416];

% plot FRF |H_digital(exp(1j*2*pi*f))| vs f
fvec = 0:0.001:100;
figure; hold on
plot(fvec, abs(H_modified(exp(1j*0.01*2*pi*fvec))))
plot(f2vec, abs(H_modified(exp(1j*0.01*2*pi*f2vec))), 'o')
xlim([0 100])
ylim([0 1])
xlabel('$f$ [Hz]', 'Interpreter', 'latex')
ylabel('$|H_{modified\_comb}(e^{j2{\pi}f})|$', 'Interpreter', 'latex')

%% Analog Coefficients/Ratios
```

```
% calculate a_k coefficients for original pulse train
a = zeros(1,5);
a(1) = abs( (A*cos(0*pi*d) *pi*d) / (N0*cos(0*pi/N0) *pi/N0) ); % k=0
for k=1:4
      a(k+1) = abs( (A*sin(k*pi*d)) / (N0*sin(k*pi/N0)) );
end

kN1vec = 501 + [0 103 207 310 414];    % values are peaks from spectrum vs m
coeff_array_1 = zeros(5,6);
coeff_array_1(:,1) = 0:4;                               % k
coeff_array_1(:,2) = a;                                 % a_k (pulse train)
coeff_array_1(:,3) = 2 .* abs(X1_hann(kN1vec));      % 2|X1_hann[kN]|
coeff_array_1(:,4) = 2 .* abs(Y1_hann(kN1vec));      % 2|Y1_hann[kN]|
coeff_array_1(:,5) = abs(Y1_hann(kN1vec)) ./ abs(X1_hann(kN1vec));
coeff_array_1(:,6) = abs(H_analog(1j *2*pi*f1vec));

coeff_table_1 = array2table(coeff_array_1);
coeff_table_1.Properties.VariableNames = {'k','a_k (for
x1)','2|X1_hann[kN]|','2|Y1_hann[kN]|','|Y1_hann[kN]|/|X1_hann[kN]|', '|H(kN*f_r)|'};

%% Digital Coefficients/Ratios

kN2vec = 501 + [0 104 208 312 416];    % values are peaks from spectrum vs m
coeff_array_2 = zeros(5,5);
coeff_array_2(:,1) = 0:4;                               % k
coeff_array_2(:,2) = 2 .* abs(X2_hann(kN2vec));      % 2|X2_hann[kN]|
coeff_array_2(:,3) = 2 .* abs(Y2_hann(kN2vec));      % 2|Y2_hann[kN]|
coeff_array_2(:,4) = abs(Y2_hann(kN2vec)) ./ abs(X2_hann(kN2vec));
coeff_array_2(:,5) = abs(H_modified(exp(1j*0.01*2*pi*f2vec)));

coeff_table_2 = array2table(coeff_array_2);
coeff_table_2.Properties.VariableNames = {'k',
'2|X2_hann[kN]|','2|Y2_hann[kN]|','|Y2_hann[kN]|/|X2_hann[kN]|', '|H(kN*f_r)|'};
```