Technical Note **LIGO-T1600453–v1** 2016/10/05

Time delay between IOP and user models

Kiwamu Izumi for the aLIGO calibration group

Distribution of this document:

Public

Contents

1 Overview of this document **2**

2 Synopsys **2**

3 Simplified data flow **2**

4 Delay analysis **3**

4.1 From ADC to output of digital AA 3

4.2 From output of digital AA to output of downsampling filter 4

5 Conclusion **5**

A Time domain simulation of AA filter **6**

B User model code **8**

1 Overview of this document

Understanding time delay in the aLIGO digital system is critical for providing accurate sky localization of gravitational wave events. This document summarizes theoretical study of time delays in a part of the aLIGO digital system.

This document is not meant for accurately describing the digital system and therefore for those who look for details of the digital system should read other CDS materials instead.

2 Synopsys

- There should be NO time delay for signals propagating from an IOP (input output processor) to a user model except for the phase lag due to the digital AA (anti aliasing) filter.
- This statement is consistent with a recent duo tone measurement [1] as well as measurements done in the past [2].

3 Simplified data flow

In this section, we briefly review the sequence of the data flow in a part of the digital system including ADC, IOP and user models. A concise summary can be found in [3] and the actual code can be found at `/src/fe/controller.c` [4].

1. ADCs are synchronized to timing distribution system.
2. Execution of realtime task (i.e. IOP) is triggered by arrival of ADC sample.
3. ADC data is read and then written into a shared memory together with the time stamp by IOP at sampling rate of 65536 Hz.
4. User model reads the data in the shared memory and apply AA filter to it at a rate of 65536 Hz.

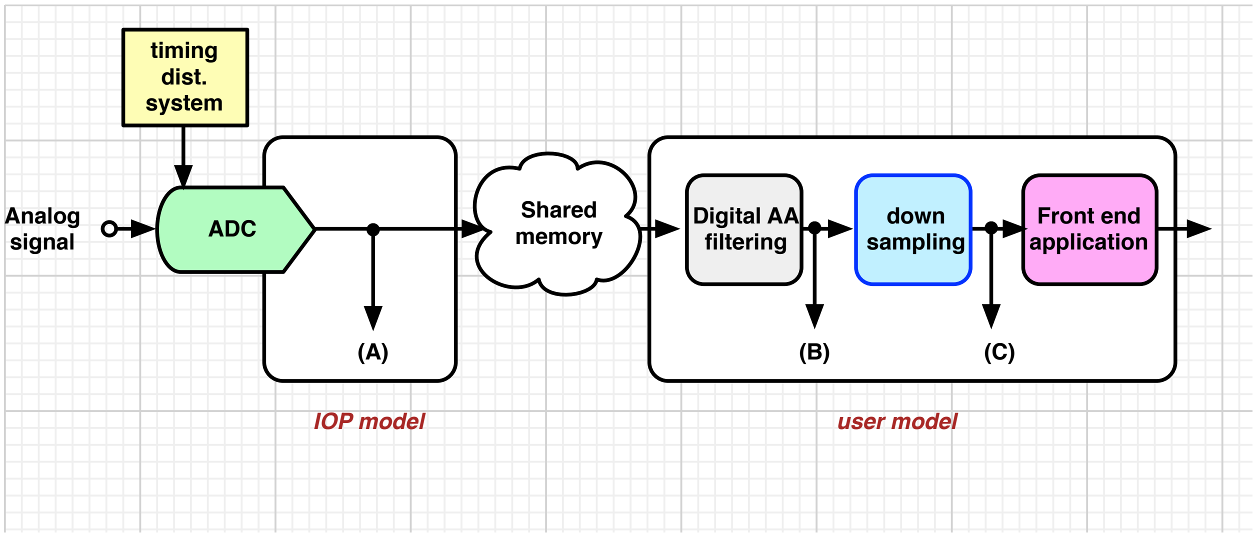

Figure 1: Diagram of the signal flow.

5. User model grabs the filtered data at every 4th cycle of 65536 Hz (in case of 16384 Hz model) and executes the user model application.

This series of steps is illustrated in figure 1.

4 Delay analysis

We now study delays in the system. For convenience, we divid the system into two pieces – one from ADC to output of the digital AA filter [from (A) to (B) in figure 1], and the other from output of digital AA to output of downsampling [from (B) to (C)].

4.1 From ADC to output of digital AA

Here, we analyze the propagation of signal from (A) to (B) in figure 1. Note that both the data transfer through the shared memory and the digital AA filtering finish the process/calculation much faster than the rate of 65536 Hz. The digital AA gives a phase lag but does not introduce a time delay. Figure 2 shows the expected and simulated transfer functions of this particular portion of the system. The simulation was done in time domain and the transfer function was extracted by a swept sine measurement (the simulation code

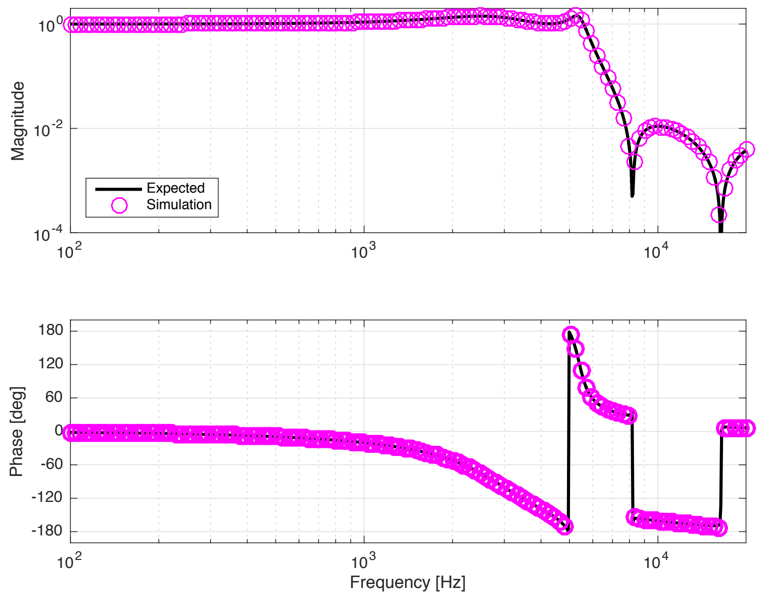

Figure 2: Digital AA filter. The black curve is the expected transfer function without no time delay. The pink circles are the measured transfer function of the time-domain-simulated AA filter.

is attached in Appendix A). The fact that the expected and simulated curves match indicate that there is no time delay as expected, except for the known phase lag due to the AA filter.

4.2 From output of digital AA to output of downsampling filter

We now analyze the propagation of signal from (B) to (C) in figure 1. People often think that there must be a delay by four IOP clock cycles because the decimation function has to wait for four 65536 Hz data points to obtain a single 16384 Hz data point. This picture is NOT correct.

The functionality of the decimation is merely to reduce the number of data points and it does not introduce a time delay. The below shows a highly simplified version of the actual user model code to illuminate what they actually do.

```
for(jj = 0; jj < 4; jj++) // in case of 16kHz user model
{
        // digital AA filtering at 65538 Hz
        out = IIR(input);
}
```

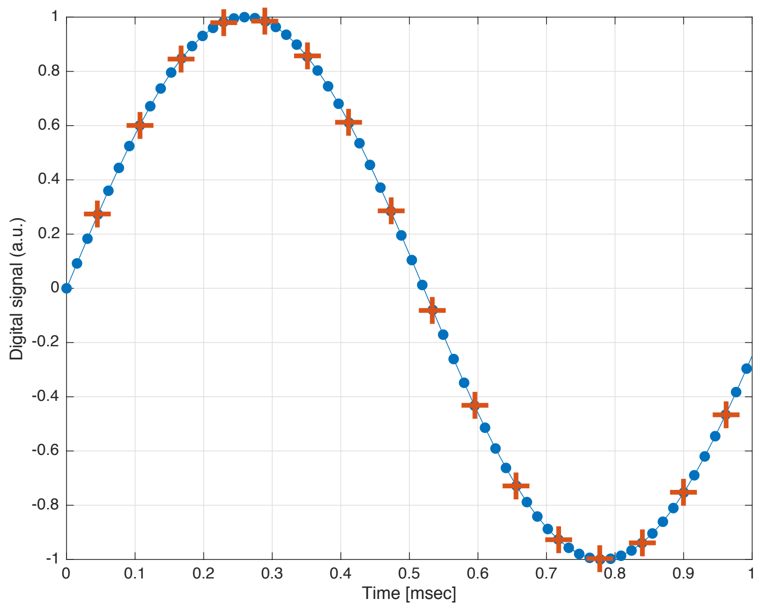

Figure 3: An example of decimation. A 960 Hz sine wave is injected to the system. See the main body for explanation.

```
        // execute the user model application
        feCode(out);
```

As shown in the simplified code, the digital AA filter is executed for every 65538 Hz data sample. Once the digital filtering is done four times (in the case of 16384 Hz user model), it then executes the user model application by handing the latest output from the AA filter to the application.

Figure 3 provides a visualization of the decimation. The blue dots in the plot are the data coming out from the digital AA filter. The red plus marks are the ones after the decimation function which obviously don't show any delay relative to the blue dots.

5 Conclusion

According to the actual C code, transferring data from IOP to user model should not introduce a time delay. This is consistent with measurements.

A Time domain simulation of AA filter

This is a matlab script to generate figure 2 by running the IIR filter in time domain.

```
% IOPdownsamp.m
%
% this is a script to check the effects of the IOP downsamping.
%
% created: 2016-10-04
%

fm = 960; % excitation frequenyc in [Hz]
fs = 2^16;
Ts = 1/fs;
N = 256;
t = linspace(0, (N-1)/fs, N);

%% The below is a copy from controller.c
%/* Coeffs for the 4x downsampling (16K system) filter */
%static double __attribute__ ((unused)) feCoeff4x[9] =
%        {0.014805052402446,
%        -1.71662585474518,     0.78495484219691,    -1.41346289716898,    0.99893884152400,
%        -1.68385964238855,     0.93734519457266,     0.00000127375260,    0.99819981588176};

g = 0.014805052402446;
a = [-1.71662585474518     0.78495484219691;...
     -1.68385964238855    0.93734519457266];
b = [-1.41346289716898    0.99893884152400;...
     0.00000127375260    0.99819981588176];

f = logspace(2, 5, 1024);
zinv = exp(-i*2*pi*f*Ts);
%% double check the shape of the down samping filter in freq domain.
h = g*ones(1, length(f));

for jj = 1:2
    h = h .* (1+b(jj, 1)*zinv + b(jj, 2)*zinv.^2) ./ (1 + a(jj, 1)*zinv + a(jj,2) *zinv.^2);
end

if false
    figure(4)
    subplot(211)
    loglog(f, abs(h))
    grid on;
    ylabel('Magnitude')
    ylim([1e-4 2])
    subplot(212)
    semilogx(f, rad2deg( angle(h) ))
    grid on;
    xlabel('Frequency [Hz]')
    ylabel('Phase [deg]')
end
%% (swept sine) transfer function measurement
ncycle = 100; % number of integration cycle.
nsettle = 5; % number of cycle to settle.
tf =[]; % empty transfer function.

% frequency list
fm_list = logspace(2, 4.3, 128);

% do the swept sine.
for fm = fm_list
```

```
    % compute the number of data points required to
    % meet the specified ncycle and nsettle.
    N = ceil(1/fm * (ncycle + nsettle)/Ts);
    Nsettle = ceil(1/fm*nsettle/Ts);

    t = linspace(0, (N-1)/fs, N); % generate time vector accordingly

    exc = sin(2*pi*fm*t); % excitation signal
    buf = zeros( size(a) ); % IIR buffer or history
    nFft = 0; % numerator in the final TF
    dFft = 0; % denominator in the final TF

    % run through the number of time series
    for kk = 1:N
        in = exc(kk); % give the excitation as an input.

        % do the IIR filtering using the ancient sos II form.
        % output should be multiplied by the gain i.e. out = in *g;
        for jj = 1:length(a)
            y = in - a(jj, 1) * buf(jj,1)  - a(jj, 2) * buf(jj,2);
            in = y + b(jj,1) * buf(jj,1) + b(jj,2)*buf(jj,2);
            buf(jj, 2) = buf(jj,1);
            buf(jj,1) = y;
        end

        if kk >= Nsettle
            % do the instantaneous FFT;
            nFft = nFft + in*g * exp(-i*2*pi*fm*t(kk));
            dFft = dFft + exc(kk) * exp(-i*2*pi*fm*t(kk));
        end
    end


    tf = [tf; nFft/dFft];
    fprintf('f␣=␣%f␣[Hz]\n', fm);
end


%%
figure(101)
subplot(211)
loglog(f, abs(h), 'LineWidth', 2, 'Color', 'k')
hold on;
loglog(fm_list, abs(tf), 'o',...
    'MarkerSize', 10, 'LineWidth', 1, 'Color', 'm')
grid on;
hold off;
xlim([fm_list(1) fm_list(end)])
ylim([1e-4 2])
legend('Expected', 'Simulation', 'Location', 'SouthWest')
ylabel('Magnitude')
title('␣')

subplot(212)
semilogx(f, rad2deg( angle(h)), 'LineWidth', 2, 'Color', 'k')
hold on;
semilogx(fm_list, rad2deg( angle(tf) ), 'o',...
        'MarkerSize', 10,'LineWidth', 2, 'Color', 'm')
grid on;
hold off;
xlim([fm_list(1) fm_list(end)])
xlabel('Frequency␣[Hz]')
```

```
set(gca, 'YTick', [linspace(-180, 180, 7)])
ylabel('Phase␣[deg]')


set(gcf,'PaperPositionMode','auto')
set(gcf, 'PaperOrientation', 'landscape')
set(gcf, 'PaperType', 'usletter')
%set(gcf, 'PaperSize', [10 10])
print('-depsc', '-r600', './figures/downsamp.eps')
```

B User model code

The below shows parts of the actual code (i.e. controller.c [4]) corresponding to the snippet shown in section 4.2.

```
.
.
l.949            for(ll=0;ll<sampleCount;ll++)
l.950            {
.
.
l.1189  #ifdef OVERSAMPLE
l.1190           /// - ---- Downsample ADC data from 64K to rate of user application
l.1191           if (dWordUsed[jj][ii]) {
l.1192  #ifdef CORE_BIQUAD
l.1193           dWord[jj][ii] = iir_filter_biquad(dWord[jj][ii],FE_OVERSAMPLE_COEFF,2,&dHistory[ii+jj*32][0]);
l.1194  #else
l.1195           dWord[jj][ii] = iir_filter(dWord[jj][ii],FE_OVERSAMPLE_COEFF,2,&dHistory[ii+jj*32][0]);
l.1196  #endif
.
.
l.1207       #endif
.
.
l.1217           }
.
.
l.1249 /// \> Call the front end specific application  ******************\n
l.1250 /// - -- This is where the user application produced by RCG gets called and executed. \n\n
l.1251  rdtscl(cpuClock[CPU_TIME_USR_START]);
l.1252  iopDacEnable = feCode(cycleNum,dWord,dacOut,dspPtr[0],&dspCoeff[0],(struct CDS_EPICS *)pLocalEpics,0);
l.1253  rdtscl(cpuClock[CPU_TIME_USR_END]);
.
.
```

The `for` sentence at the very top runs until it satisfies the decimation factor of `ll >= 4`. In the middle of the snippet, `dWard` is a double precision array representing both input and output of the AA filter. The AA filtering is executed at either line 1193 or line 1195. Finally the user model application is executed at line 1252.

References

[1] E. Goetz, "Better Understanding Pcal timing signals" (2016)
https://alog.ligo-wa.caltech.edu/aLOG/index.php?callRep=29259

[2] S. Countryman, Z. Marka and K. Kawabe, "Timing witness signals indicate trustworthy timing for G184098, a BBH Event candidate" T1500516-v2 (2015)
https://dcc.ligo.org/LIGO-T1500516

[3] R. Bork and A. Ivanov, "aLIGO CDS Real-time Sequencer Software" (2012)
https://dcc.ligo.org/LIGO-T0900607

[4] R. Bork and A. Ivanov, controller.c rev4195 (Apr. 2016)
https://redoubt.ligo-wa.caltech.edu/websvn/trunk/

[5] P. Fritschel, "New RCG 3.0 Decimation (IOP Upsampling / Downsampling) Filter for 16 kHz models" (2016)
https://dcc.ligo.org/LIGO-T1600066