FPGA and Instability due to Sampling

Riley Kenyon

May 7th, 2019

Abstract

This report describes how using a simplified FPGA interface reduces computational overhead and results in more accurate loop duration at high sampling rates. In addition the report describes how at low sampling rates, even with a stable controller, a system can become unstable.

1 Introduction

The RRC circuit (shown in figure 1) previously used as a first-order system in proportional control with the myRIO can be represented in a control loop described in figure 3. The plant G which has an open-loop transfer function described in equation 1 with $R = 5k\Omega$, and $C = 2.2\mu F$. Under realizable proportional control, the step response of the RRC improves with increased proportional gain K. It is important to note that in implementation, the controller cannot output a voltage higher than the max output. In order to

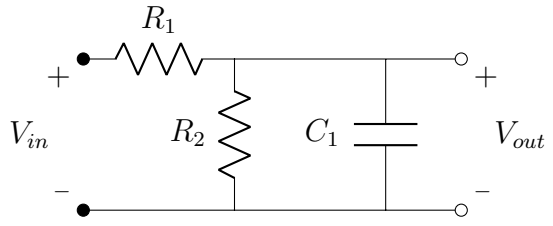

Figure 1: RRC Circuit - a simple first order system representation

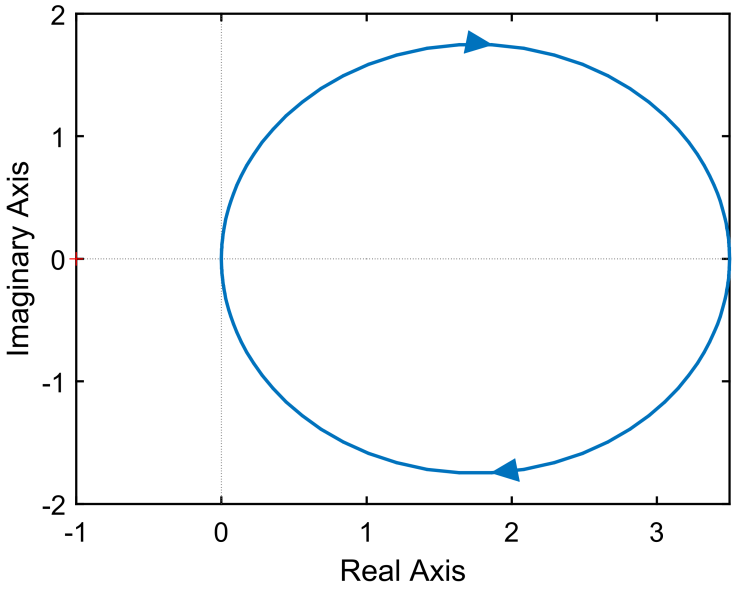

Figure 2: The nyquist plot of the loop transfer function $L(s)$ with gain $K = 7$. Notice there is no encirclement of -1, representative of a stable system. Additionally, the gain margin of the closed loop is infinity meaning that the continuous time system remains stable for any proportional gain $K > 0$.

avoid saturation, the gain of the controller has a physical limit or it becomes a non-linear and the control theory can become misaligned.

$$G(s) = \hat{k}\frac{\sigma}{s+\sigma} = 0.5\frac{2/RC}{s+2/RC} \tag{1}$$

Theoretically, in proportional control the RRC circuit should not be able to go unstable. The continuous time closed loop transfer function $T(s)$, as shown in equation 2, is stable for all $K > 0$. This is more easily shown using a Nyquist plot of the loop transfer function $L(s)$, as seen in figure 2.

$$T(s) = \frac{KG}{1+KG} = \frac{L(s)}{1+L(s)}, \tag{2}$$

The Nyquist plot does not encircle -1, and can be increased infinitely but will still not cross over axis. The gain margin of the closed loop system is infinity

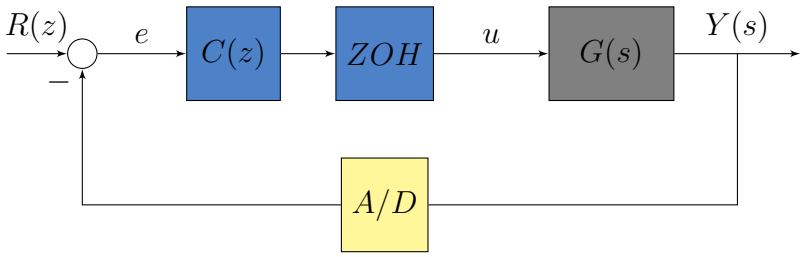

Figure 3: Closed Loop Block Diagram of signal conversion between myRIO(Blue) and input/output of circuit (Gray) using an digital to analog converter(D/A) and an analog to digital converter (A/D).

representative of the statement that any gain will not make it go unstable. However, it is possible to go unstable due to sampling which is rooted in the phase margin of the system.

2 FPGA

Formally a Field Programmable Gate Array, the myRIO comes with a 'shipped personality'. This is the standard FPGA setup that comes stock with any myRIO project. Utilizing this default setup is great due to the access of stock functions such as analog input and analog output. However, there is a downside of the default personality and that is due to the overhead associated with the stock FPGA. In order to speed things up, a custom FPGA template and .dll file will be created and used to perform proportional control of the RRC circuit. The only two functions that are necessary in this situation are the input of analog-in 0 (AIO) and the output of analog-out 0 (AO0).

2.1 LabVIEW setup

To create the setup described in the above section, a custom FPGA template has to be modified. As seen in figure 4, the personality was simplified to an analog input and output on channel C of the myRIO. The analog input (AI0) is set with an indicator, and the output (AO0) is set with a control. It is important to note that these variables do not correspond to voltages, but rather counts. If the full-scale voltage of the myRIO is looked at, it is found to be $\pm 10V$. Additionally, looking at the resolution of the myRIO analog

input,it is a 12 bit DAC meaning it can resolve 2^{12} sections within the full-scale voltage. These two pieces of information give insight into how to read inputs or outputs from the FPGA variables. The voltage resolution of the myRIO is therefore $2^{12}/20$ or 204.8 counts/V. This conversion factor is what is used to interpret the input and adjust the output to be in terms of counts or volts. This is accounted for when using the analog in or analog out blocks from the default personality, but because a custom template is in use the conversion factor needs to be accounted for in the main virtual instrument. The main timed loop performs jitter, reference function generation, and the implementation of a proportional controller. The configuration shown in figure 4 is for the 100 Hz sampling rate, running on a 1 MHz clock cycle.

3 Jitter

Jitter, or the variation between loop periods, is used as a metric to look at how accurately a controller follows a given time interval. Items such as real-time plotting within a loop increase the complexity, and therefore the time needed to perform the task. The same goes for complex controllers and reading/writing data. If a larger interval is given, the more operations can be completed within the loops duration. However, the converse is true; specifically at high sampling rates, not everything can be completed in a single loop and have to either be skipped or performed and exceed the desired loop execution time. The sampling rates used in this analysis are increased in magnitude until 10 kHz. A main metric used to quantify the amount of jitter is the root-mean-square value of the jitter (RMS) which gives insight into how the system is performing with respect to loop timing, defined as

$$T_{RMS} = \sqrt{\frac{1}{n}(T_1^2 + T_2^2 + ... + T_n^2)} \tag{3}$$

where T_i corresponds to the time taken to complete loop iteration i of the code.

3.1 Comparison at 100 Hz

At this rate, the duration of a loop is desired to be 10 ms which is plenty to perform all the tasks within the timed loop. The usage of a custom FPGA is not noticeable at this sampling rate because if the loop is executed within

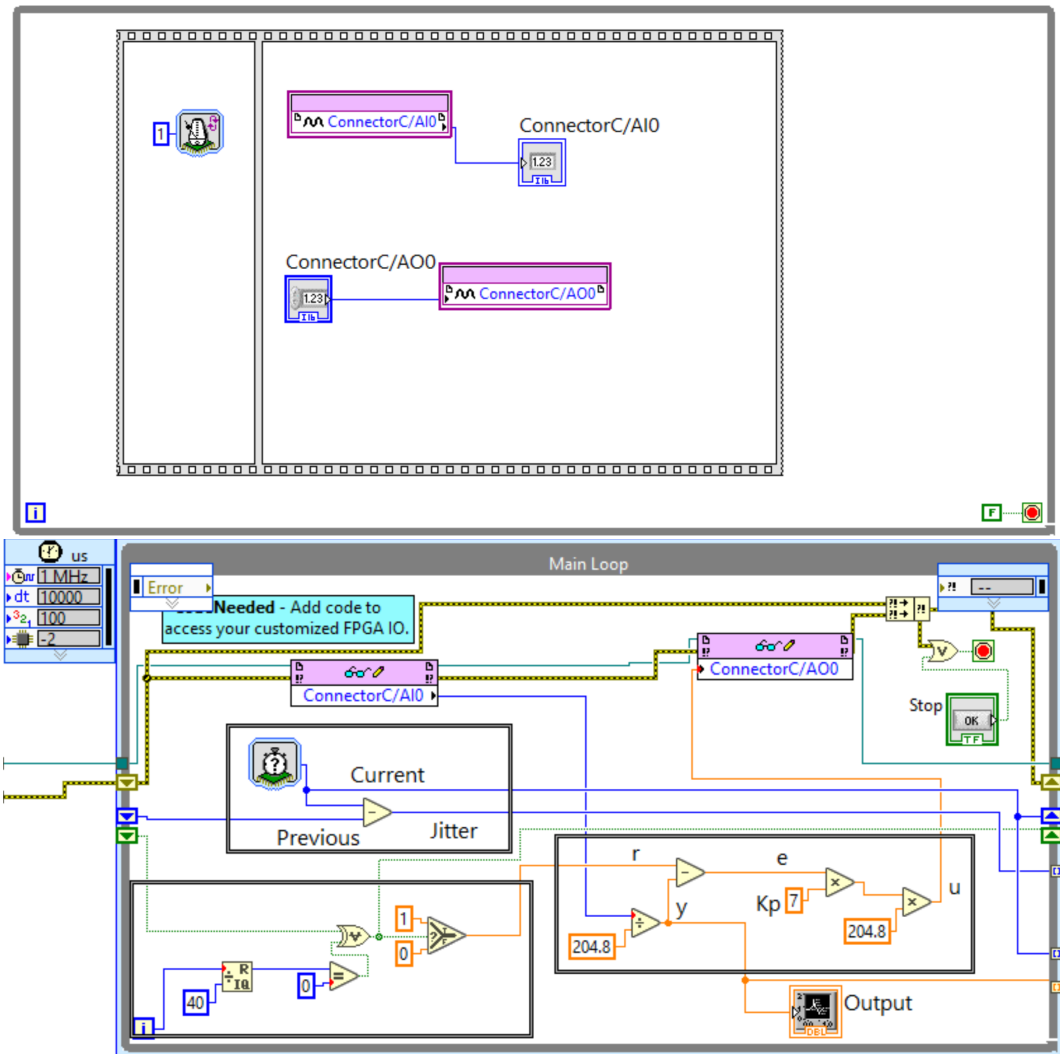

Figure 4: Initiation of the custom FPGA and accessing the variables defined in the FPGA Main Default (top). Bottom left is the function generator for a 1V reference. Above that is the calculation for jitter, using the timing of the current loop and the previous. To the right is the controller, converting from counts to volts and using the proportional constant $K = 7$. The two variables ConnectorC/AI0 and ConnectorC/AO0 are defined within the FPGA.

the desired duration, the timed loop will wait until the clock signals the beginning of the next loop. Thus an increase in execution time is not useful or necessary, as both jitter RMS values are 10000 µs. As seen in figure 5, the

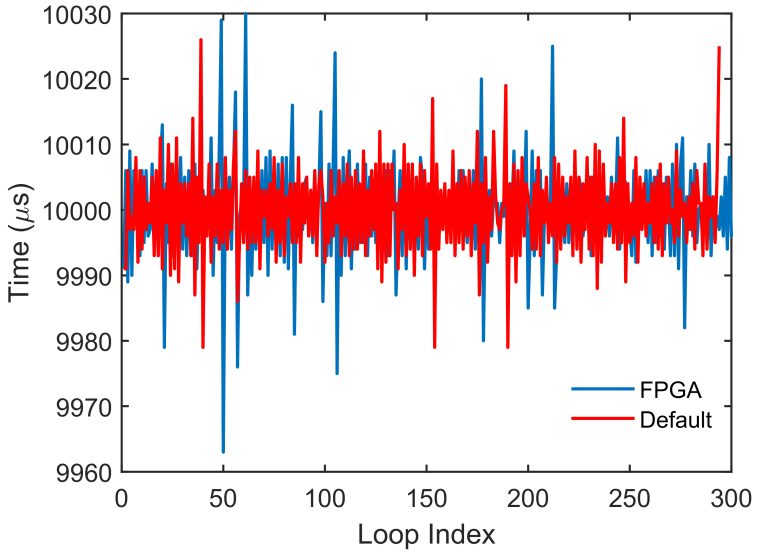

Figure 5: The jitter of the timed loop with the custom FPGA and the default personality at a sampling rate of 100Hz. Both configurations centered at 10000 µs, and have similar deviation.

jitter of both the default personality and the custom FPGA center around the same value and visually have a similar deviation.

3.2 Comparison at 1 kHz

At a sampling rate of 1 kHz, the loop period is 1 ms which is still enough for both the default personality and the custom FPGA to complete the code within the loop. The only intensive process within the loop is plotting the output of the RRC circuit and indexing both jitter and analog input for plotting in Matlab. As seen in the figure 6, both jitter profiles are centered around the same value with an RMS of 1000 µs for both.

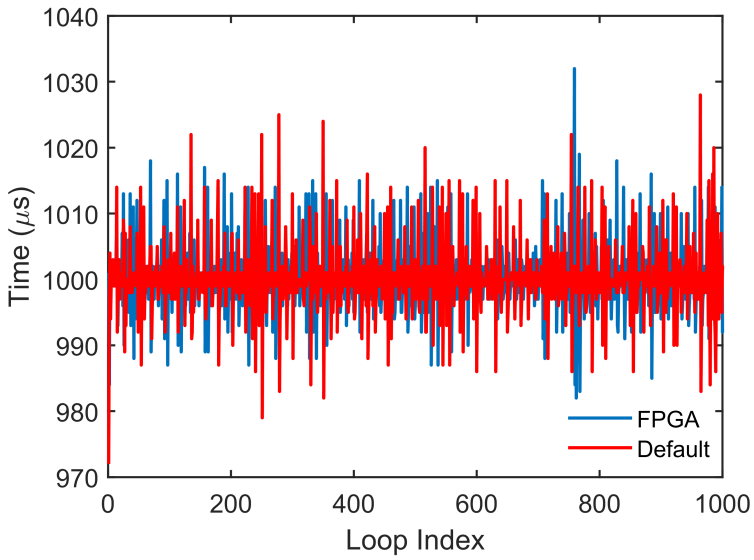

Figure 6: The jitter of the timed loop with the custom FPGA and the default personality at a sampling rate of 1 kHz. Both configurations centered at 1000 µs, and have similar deviation.

3.3 Comparison at 10 kHz

At the 10 kHz sampling rate, this is where there is a noticeable improvement between the two because the default personality cannot sample at 10 kHz with the overhead associated with the default. The simplified FPGA project succeeded in the fact that the loop execution time did not require the amount of other functions shipped on the default personality. Looking at figure 7, the FPGA jitter hovers around the 100 µs mark, where the default is noticeably slower and appears to vary more. The RMS value for the default and custom FPGA are 150.02 µs and 102.05 µs respectively.

4 Step Response

For looking at how the RRC circuit responds at different sampling rates, the controller was set with a proportional constant $K = 7$, and only the sampling frequency was changed. The proportional constant was chosen to be this value to emphasize the instability of the system at low sampling rates.

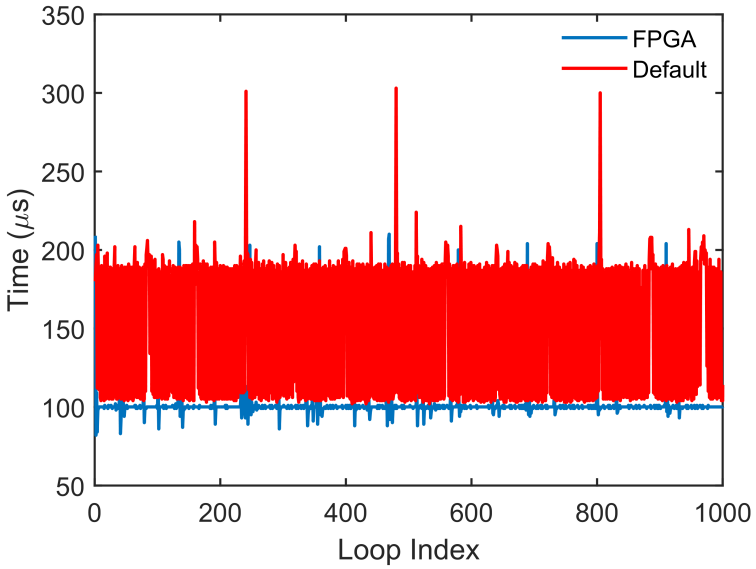

Figure 7: The jitter of the timed loop with the custom FPGA and the default personality at a sampling rate of 10kHz. The default personality is not able to execute the code within the loop before 100 µs., whereas the implementation of the custom FPGA improves the performance of the loop, reducing the amount of jitter.

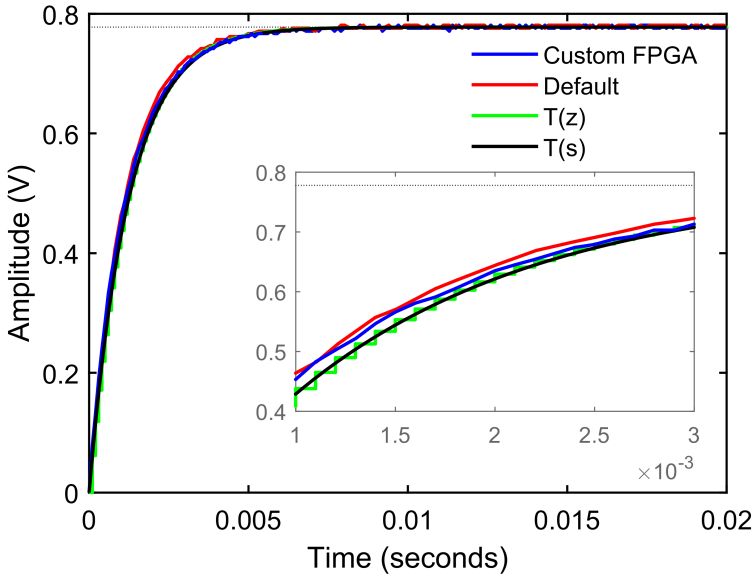

Figure 8: Step response of the RRC circuit under proportional control with $K = 7$ at a sampling rate of 10 kHz for both the custom FGPA and default Personality. At this sampling rate, the experimental data closely follows the theoretical step response.

4.1 Sampling at 10kHz

Although the custom FPGA had less jitter than the default personality, both kept the system stable. This is because at this sampling rate, the digital implementation matches theory. That being said, there is a minor time shift (phase shift) that can be seen between the two methods which is due to the default personality not running the loop exactly at 100 µs.

4.2 Sampling at 1kHz

The 1 kHz sampling rate pretty closely follows the trend of the continuous time model, but is more in line with the theoretical digital model. The custom FPGA is slightly less phase shifted than the default personality, but as seen by their RMS jitter both execute the code within the control loop within the sampling period. The theoretical step response, given by the digital transfer function most closely matches the data collected. This makes sense due to the digital model being a sampled version of the data output by the RRC

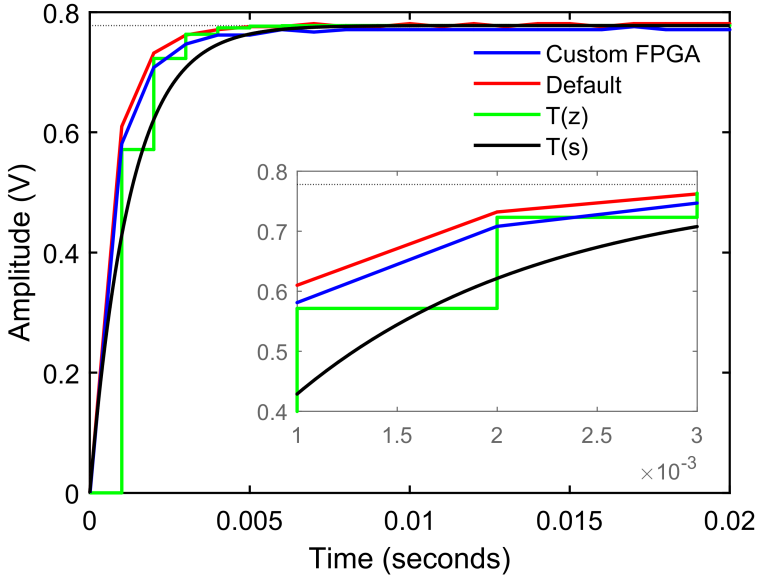

Figure 9: Step response of the RRC circuit under proportional control with $K = 7$ at a sampling rate of 1 kHz for both the custom FGPA and default Personality. At this sampling rate, the experimental data closely follows the theoretical step response.

circuit, the same that is collected by the myRIO.

4.3 Sampling at 100Hz

At the 100 Hz sampling rate, the time between each sample spans 10 ms. Considering the theoretical rise time of the closed loop system is approximately 3 ms, this should provide insight into the information received by the controller (High/Low). The experimental data collected for the 100Hz sampling rate oscillate back and forth between $\pm 3.5V$, never settling on a final value as is expected by the step response. The theoretical model of the closed loop digital system also goes unstable, this is because the infinite output of the ideal controller and does not saturate so the response continuous blowing up. At this sampling rate, the RRC circuit becomes unstable for both the custom FPGA and the default personality.

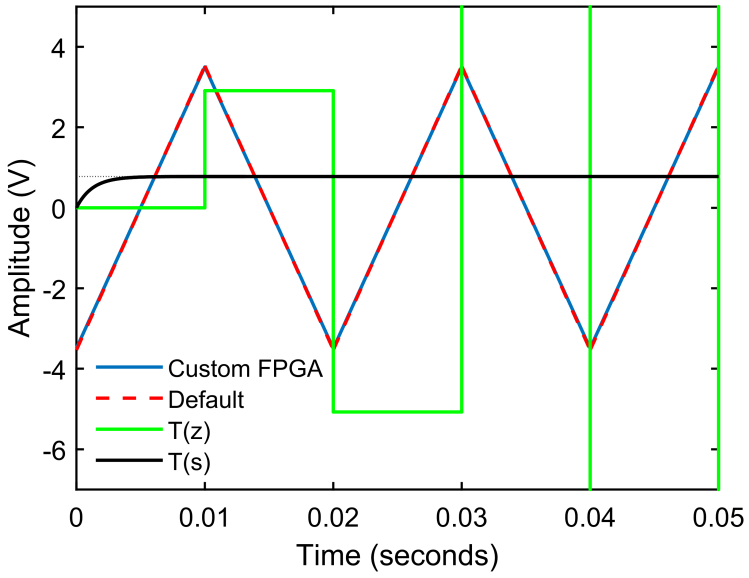

Figure 10: Step response of the RRC circuit under proportional control with $K = 7$ at a sampling rate of 100Hz for both the custom FGPA and default Personality. At this sampling rate, the system becomes unstable due to the phase shift associated with the sampling rate.

5 Instability from Sampling

Instability of the RRC circuit comes from the sampling rate of the digital controller, as verified in the beginning of the report, in the continuous time domain the system is stable for all gain $K > 0$. The discrepancy that causes the ultimate instability of the 100 Hz sampling rate is derived from the phase shift that accompanies sampling. Logically, this can be thought of similarly with respect to the rise time of the RRC circuit. The rise time of the circuit in closed loop is approximately 3 ms, and with a sampling rate of 100 Hz, or 10 ms, results in no resolution of what is going on in the system. This is why the controller saturates and outputs the max voltage either positive or negative depending if the system is being read as high or low, as shown in figure 10.

5.1 Phase Margin

Phase margin is useful in determining the stability of the system, similarly to gain margin. As shown earlier, the gain margin of the RRC circuit closed loop transfer function was infinite. However, the same cannot be said of the phase margin of the system. The definition of phase margin is the point of the cross-over frequency (frequency where the magnitude FRF crosses 0 dB) for the open-loop system. Additionally this frequency is a measure of closed loop bandwidth which provides some insight into the phase shift of a sampled system. Recall that a zero-order hold can be represented as the following transfer function

$$G(s) = \frac{1 - e^{-sT_s}}{s}, \tag{4}$$

for a sampling period of T_s and that exponential values of s are time delays of T_s. The corresponding relation in the frequency domain is achieved by plugging in $s = j\omega$.

$$G(j\omega) = \frac{1 - e^{-j\omega T_s}}{j} = \frac{e^{-j\omega T_s/2}(e^{j\omega T_s/2} - e^{-j\omega T_s/2})}{j\omega} \tag{5}$$

Performing some simplifications with Euler's formula yields the equation of

$$G(j\omega) = \frac{2sin(\omega T_s/2)}{\omega} e^{-j\omega T_s/2}. \tag{6}$$

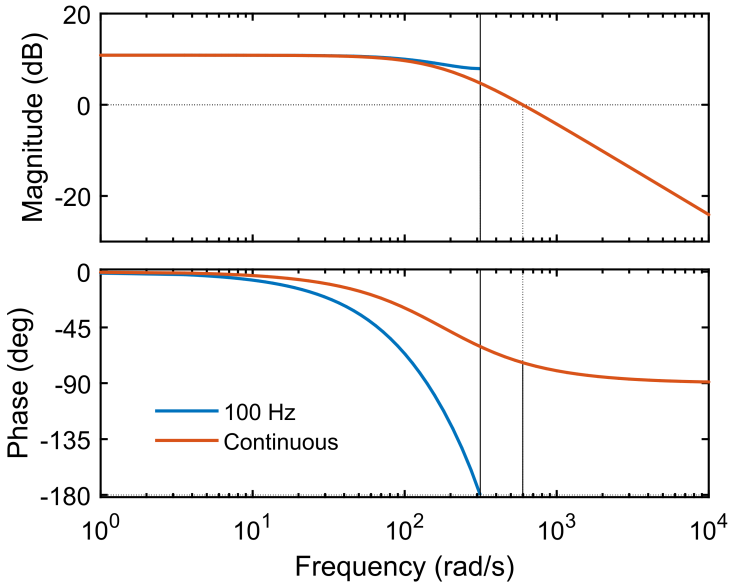

Figure 11: The bode plot of the loop transfer function for an RRC circuit under proportional control with $K = 7$ at a sampling rate of 100 Hz. At this sampling rate, the system becomes unstable due to the phase shift of the system.

This relation proves the phase shift associated with sampling, because the collection of data is a ZOH. The delay is seen in the exponential term, with the $-\omega T_s/2$. On average, the phase shift associated with sampling is on the order of half a sampling period and for the purposes of analysis the system is evaluated at the estimated bandwidth of the system ($\omega_{crossover}$).

In the case of the 100 Hz sampling rate, the system remains above the crossover frequency from the bode plot (figure 11, but the phase of the system crosses -180 degrees at a frequency of 314 rad/s. Phase margin is a measure between -180 degrees and the closest point given by the phase curve. Here there is no phase margin, and the resulting delay is excessive to the point where the system becomes unstable as seen in figure 10. All systems have to drop off at some point, meaning that the $\omega_{crossover}$ is pushed further to the right, resulting in an even larger phase delay leading to the instability.

6 Conclusion

The effects of sampling can be profound, as seen in the case of the 100 Hz sampling rate. Not only can sampling make a system unstable, but the performance is not always as anticipated which is where knowledge of digital control is applicable. The reason for the RRC circuit going unstable at the 100 Hz sampling rate was due to the half phase shift associated with sampling. For the case of the 1 kHz sampling, the results followed the general trend of the continuous estimate but was better aligned with the digital version predicted with a zero-order hold, which makes sense due to the digital estimate being a sampled version of the output from the system (RRC circuit). The 10 kHz sampling rate is fast enough to the point of essentially replicating a continuous time estimate for the circuit's step response. If it is truly important to achieve the correct sampling rate, jitter can be eliminated by the use of a simplified FPGA over the default personality. The default produced loop iterations with a RMS value of 150 µs, while the custom FPGA was able to achieve a RMS of 102 µs.

A Code

A.1 Creating Plots

```
%-------------------------------------------------------------------------
%% Industrial Automation Lab 6 - FPGA
% Riley Kenyon 5/09/2019
%% Experimental Data
clear all; clc; close all;

%% Import Data
% FPGA
FPGAStep100 = csvread('FPGAStep100.csv');
FPGAStep1000 = csvread('FPGAStep1000.csv');
FPGAStep10000 = csvread('FPGAStep10000.csv');
FPGAjitter100 = csvread('jitter100.csv');
FPGAjitter1000 = csvread('jitter1000.csv');
FPGAjitter10000 = csvread('jitter10000.csv');
```

```
% NORMAL
normalStep100 = csvread('normalStep100.csv');
normalStep1000 = csvread('normalStep1000.csv');
normalStep10000 = csvread('normalStep10000.csv');
normalJitter100 = csvread('normalJitter100.csv');
normalJitter1000 = csvread('normalJitter1000.csv');
normalJitter10000 = csvread('normalJitter10000.csv');
%% Theoretical Transfer Functions - RRC in Proportional Control [K]
% CONTINUOUS
R1 = 5100; % ohm
C1 = 2.2e-6; % uF
G = tf(1,[R1*C1 2]); % RRC
K = 7; % gain
CL  = feedback(K*G,1);

% DIGITAL
Ts = [100e-6 1e-3 10e-3]; % 10kHz 1kHz 100Hz
Gz100 = c2d(G,Ts(1),'zoh');
Gz1000 = c2d(G,Ts(2),'zoh');
Gz10000 = c2d(G,Ts(3),'zoh');
CLz100 = feedback(K*Gz100,1);
CLz1000 = feedback(K*Gz1000,1);
CLz10000 = feedback(K*Gz10000,1);

%% Plotting
figure(1)
hold on
plot(FPGAStep100(:,1),FPGAStep100(:,2),'b','LineWidth',1.5);
plot(normalStep100(:,1),normalStep100(:,2),'r','LineWidth',1.5);
step(CLz100);
step(CL);
xlim([0 0.02])
axes('position',[0.45 0.175 0.4 0.3])
box on
% index1 = 0.001<FPGAStep100(:,1) & FPGAStep100(:,1)< 0.002;
% index2 = 0.001<normalStep100(:,1) & normalStep100(:,1)< 0.002;
hold on
plot(FPGAStep100(:,1),FPGAStep100(:,2),'b','LineWidth',1.5)
```

```
plot(normalStep100(:,1),normalStep100(:,2),'r','LineWidth',1.5)
step(CLz100);
step(CL);
xlim([0.001,0.003])

figure(2)
hold on
plot(FPGAStep1000(:,1),FPGAStep1000(:,2),'b','LineWidth',1.5);
plot(normalStep1000(:,1),normalStep1000(:,2),'r','LineWidth',1.5);
step(CLz1000);
step(CL);
xlim([0 0.02])
axes('position',[0.45 0.175 0.4 0.3])
box on
hold on
plot(FPGAStep1000(:,1),FPGAStep1000(:,2),'b','LineWidth',1.5)
plot(normalStep1000(:,1),normalStep1000(:,2),'r','LineWidth',1.5)
step(CLz1000);
step(CL);
xlim([0.001,0.003])
ylim([0.4 0.8])

figure(3)
hold on
plot(FPGAStep10000(:,1),FPGAStep10000(:,2));
plot(normalStep10000(:,1),normalStep10000(:,2));
step(CLz10000);
step(CL);
axis([0 0.05 -5 5]);

figure()
hold on
plot(FPGAjitter100);
plot(normalJitter100(3:end));
xlim([0 1000]);
legend('FPGA','Default');
fprintf("RMS FPGA jitter 100Hz :  %0.3f \n",rms(FPGAjitter10000(2:300)));
fprintf("RMS Normal jitter 100Hz : %0.3f \n",rms(normalJitter10000(3:end)));
```

```
figure()
hold on
plot(FPGAjitter1000(2:end));
plot(normalJitter1000(3:end));
xlim([0 1000]);
legend('FPGA','Default');
fprintf("RMS FPGA jitter 1kHz :  %0.3f \n",rms(FPGAjitter1000(2:300)));
fprintf("RMS Normal jitter 1kHz : %0.3f \n",rms(normalJitter1000(3:303)));

figure()
hold on
plot(FPGAjitter10000(2:end));
plot(normalJitter10000(3:end));
xlim([0 300]);
legend('FPGA','Default');
fprintf("RMS FPGA jitter 10kHz :  %0.3f \n",rms(FPGAjitter100(1:300)));
fprintf("RMS Normal jitter 10kHz : %0.3f \n",rms(normalJitter100(3:303)));

figure()
hold on
bode(7*G);
bode(7*Gz10000)
```

A.2 LabVIEW Code