Pulsed NMR of Ferroin and Ferriin

Jordan J. Gosselin, and Cassandra S. Niman
Modern Physics Lab 173, Bio&Quantum, University of California, San Diego
June 13, 2008

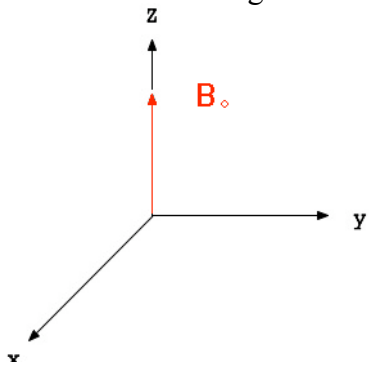
Introduction:

Pulsed NMR is used to characterize spin-spin and spin-lattice interactions. Due to our interest in the ferroin catalyzed B-Z reaction we attempted to characterize ferroin and ferriin using pulsed NMR with hopes to run the pulsed NMR on a B-Z reaction and observe changes from ferroin to ferriin in the reaction. This reaction could not be imaged using NMRI as reported by Ying Gao et.al.

Characterizing T2 for ferriin gave expected results, while finding T2* for proved difficult due to the low signal to noise observed. The FID of ferroin was observed as expected, and T2* was characterized. Attempting to measure T2 for ferroin lead to interesting observations of phase shifting in the spin-echo, which lead to an inability to properly characterize T2 and question the accuracy of the results found for ferriin T2.

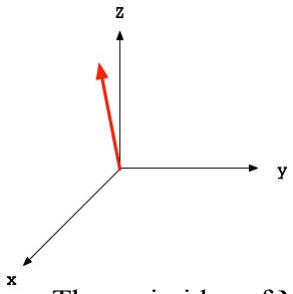
Background:

Pulsed NMR is a spectroscopy technique that takes advantage of the intrinsic particle property of spin. Specifically, it focuses on the spins of the protons in the nucleus. The methodology involves placing a sample within a set of orthogonal coils, which comprise the x, y, and z-axis. We have defined the x-direction as the transmission coil, the y-direction as the reception coil, and the z-direction is our magnet coil. A DC current is fed through the magnetic coils to create a magnetic field, Bo, in the z-direction.



The spin state of each proton in a nucleus has a magnetic moment given by $\underline{\mu} = \gamma * \underline{S}$ (underlined variables represent vectors; S is the spin vector, γ is the gyromagnetic ratio). The energy of the state is determined by the inner product of the magnetic moment with the magnetic field ($E = -\underline{\mu} \cdot \underline{B}$). Because Bo is only in the z-direction only the z component of μ is important. This component is given by $\mu_z = \hbar m$, where $m = \pm \frac{1}{2}$ for protons.

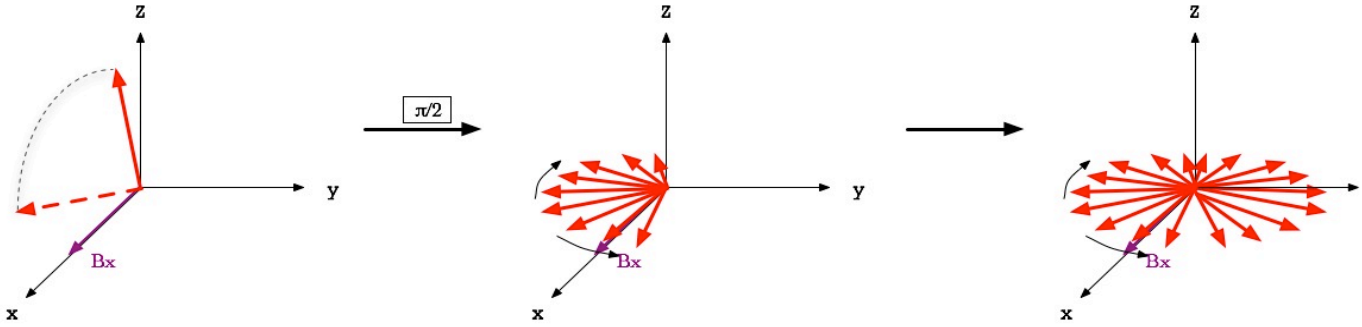
The above is a quantum mechanical description of the system; we are only concerned with the net effects of these magnetic moments, so we will treat NMR classically with the exception of the preceding explanation of magnetic moments due to spin. The individual magnetic moments due to spin can be summed into a net magnetic moment, which is what is seen with the receptor coils. The ratio of the populations of the spin up and spin down states (S+ and S- respectively) can be estimated by the Maxwell-Boltzmann distribution. The equation is (Number S+)/(Number S-) = $\exp(-\Delta E_{\pm}/kT)$, where k is Boltzmann's constant (Pochapsky 28). The net magnetic moment is directly proportional to this difference in the population of the spin states.



This net magnetic moment precesses about the z-axis with an angular frequency given by $\omega = \gamma * Bo$. For the purposes of our experiment we tuned Bo to give an ω of $\sim 2*\pi*8$ MHz. This was done because we were using a function generator that generated ~8 Mhz sine waves.

The main idea of NMR is to rotate this net magnetic moment into the x-y plane; here we can measure it's amplitude and determine the time it takes to relax to it's initial state. This is most easily visualized in a reference frame rotating at the same frequency as ω. In this reference frame $\omega' = \gamma * Bo'$. Since our reference frame rotates at ω with respect to the laboratory frame $\omega' = 0$, therefore $Bo' = 0$. The net magnetic moment is stationary in this frame. We apply an 8 MHz pulse in the x-direction in the laboratory frame. In the rotating frame this appears to be a stationary magnetic moment in the x' direction due to the decomposition of the 8 MHz sine wave into two vectors rotating with opposite ω (an interested reader should refer to the first chapter of Bruch's book; all others must accept this as true). This moment provides a torque on the magnetic moment for the duration of the pulse width. This will rotate the magnetic moment around the x' axis (rotating frame) an angle θ given by the equation $\theta = \gamma * Bx$ (where Bx is the peak amplitude of the oscillating magnetic field applied in the laboratory frame). The equation for θ can be easily derived from the equation for ω.

When the magnetic moment is rotated into the x-y plane the net magnetic moment will decrease due to the phase spreading in the x-y plane caused by inhomogenity of the magnetic field throughout the sample.



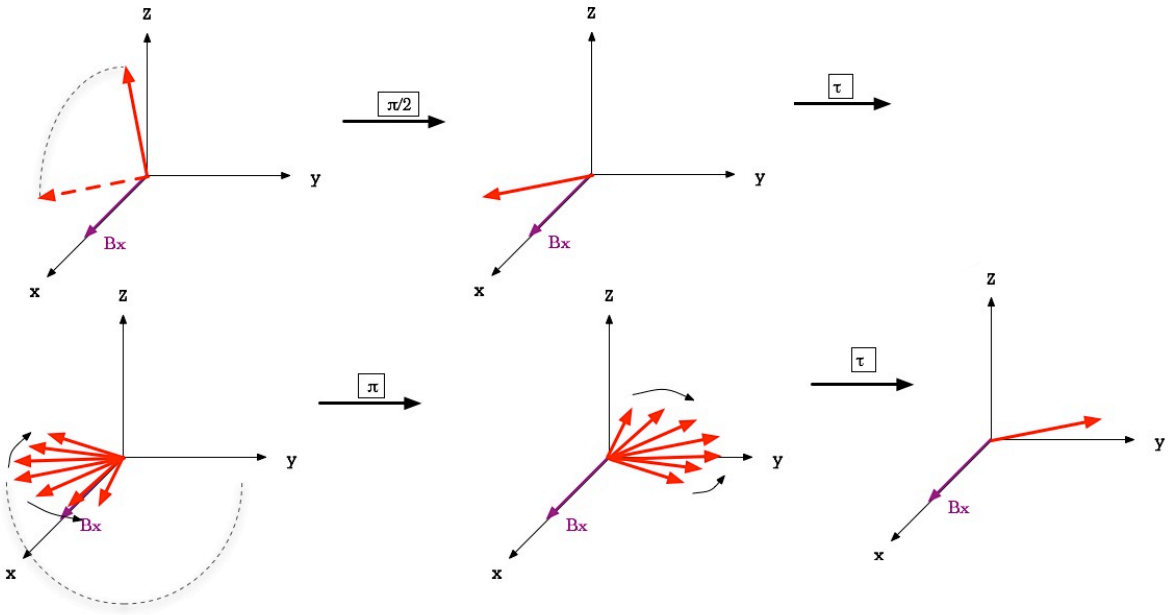
The above is the phase spreading in the rotating reference frame. This relative phase spreading can be utilized to characterize a substance in pulsed NMR.

The main idea behind pulsed NMR is to use pulses with an angle of π or $\pi/2$ in sequence to measure various characteristic relaxation times.

Methodology:

For our project we used a pulse generating/data acquisition MatLab code developed by Erik Flister combined with a simple averaging routine designed by Jordan Gosselin. This allowed us to perform multiple trials and average the results extremely rapidly. The mode of the dataset sizes was 64 trials; the data taking/averaging process for these trials took approximately thirty seconds to one minute.

We measured T2 and T2* for ferriin and ferroin respectively. We applied a pulse with an angle of $\pi/2$ followed by a pulse with an angle of π in an attempt to recover a spin echo and measure T2. This is a common technique, which is mentioned in all the books we referenced (bibliographical information given following the report).



Another method we used was the Carl-Purcell method. This method uses multiple π pulses following the initial $\pi/2$ pulse. We attempted to use this method unsuccessfully. Our problems are discussed in the "Experimental Results" section below.

Experimental Results:

Our measurements are of ferroin and ferriin. We tried to characterize T2 for these samples. By changing the time the pulses were high and maximizing the amplitude of the FID and Spin Echo we found the time length of a $\pi/2$ and π pulse for each sample. Ferroin and ferriin have a $\pi/2$ pulse size of 25μs and 10μs respectively. For each sample the length of the pulses stayed constant for the remainder of our measurements. For each setting of τ, the time between the pulses, 64 or 128 raw data samples were acquired and averaged. This produces a smooth data set to start with.

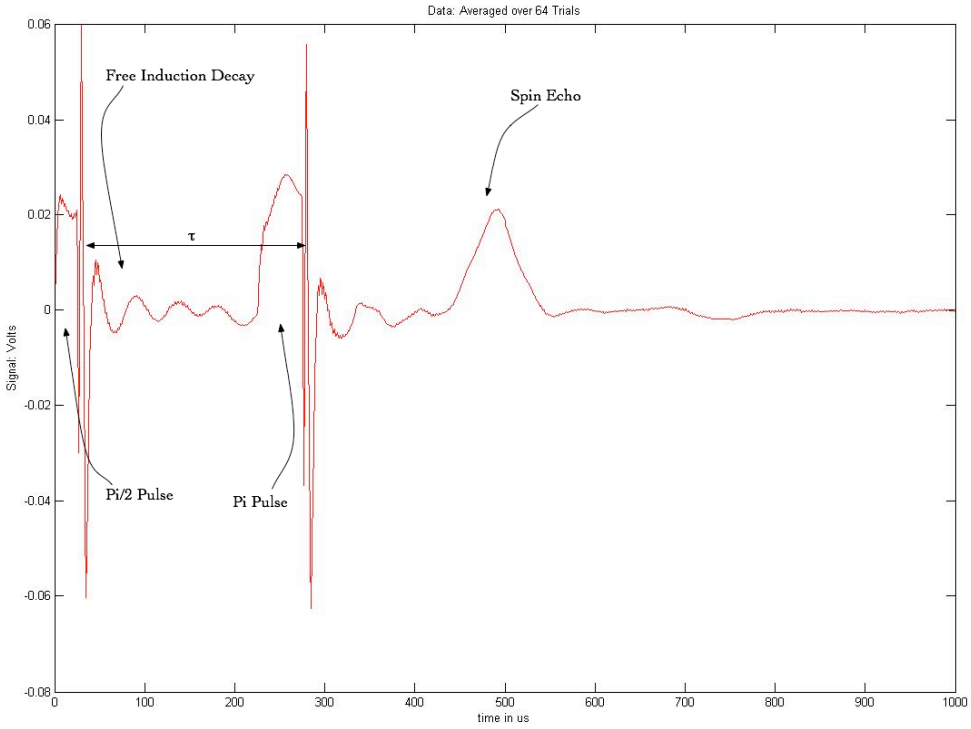


Fig1: 64 averaged data sets taken of ferroin. There is a clear FID and spin-echo as expected. The strange shape of the second pulse is likely due to an overlapping of the pulse and the tail of the FID.

Ferriin Data & Analysis:

A π/2 pulse for ferriin is about 10μs. The data for the FID following the π/2 pulse is shown below. Because the signal is so small most of it gets lost in the noise, there is no clear exponential decay of the peaks, and this data could not be used to characterize T2*

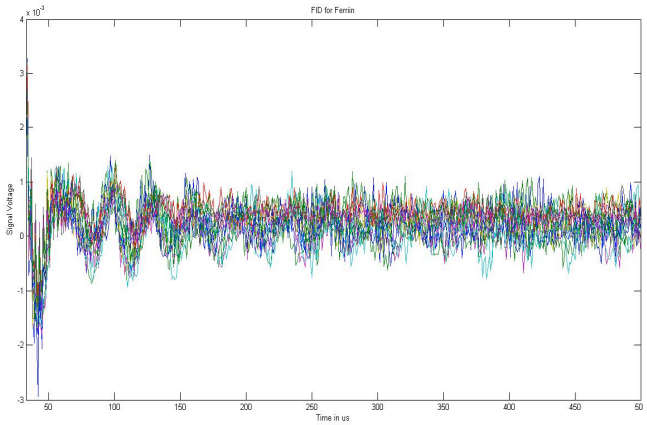


Fig2: FID for Ferriin sample.

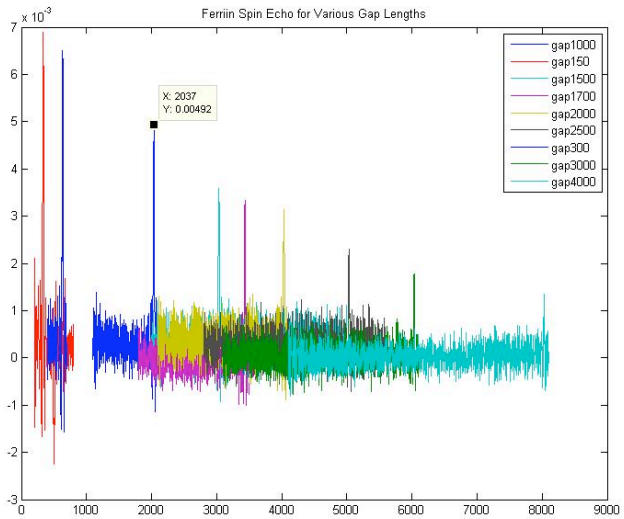


Fig3: Spin-echoes for Ferriin

However the spin-echo data for this data set looked as expected. The amplitude of the spin-echo decays exponentially as τ increased. There is a DC-offset for each averaged data set above. To quickly get rid of this, 30 points past all the echoes, noise about the GND signal, were averaged and subtracted from the data. The peaks fit to an exponential decay resulting in the equation:

V(t) = 0.0062 exp(-t/5000) ; here T2 = 5000µs.

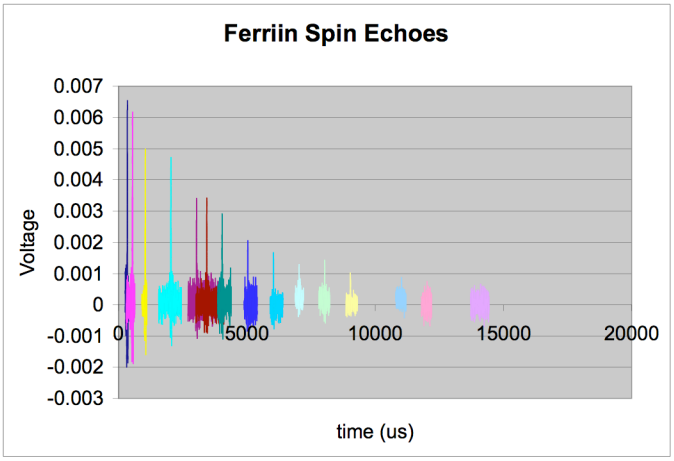


Fig6: Ferriin spin-echoes after DC-offset adjustments

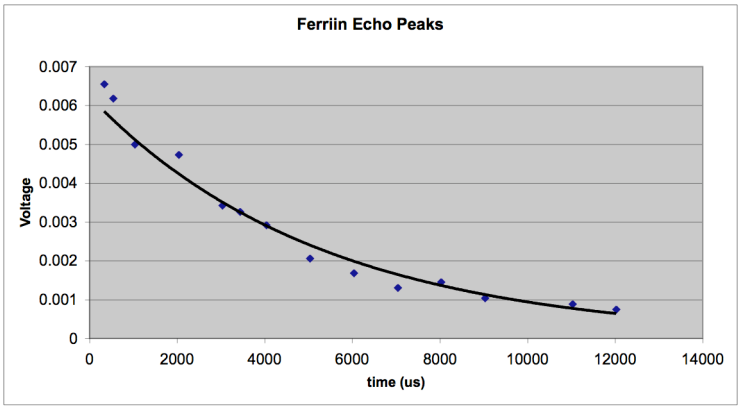


Fig7: Peaks of Ferriin spin-echoes. Fit to the curve: V(t) = 0.0062 exp(-t/5000)

Ferroin Data & Analysis:

As previously stated the π/2 pulse for ferroin was measured around 25µs. The data used to characterize at the FID observed is a set of more than 1150 raw data sets averaged. This was used to find T2*.

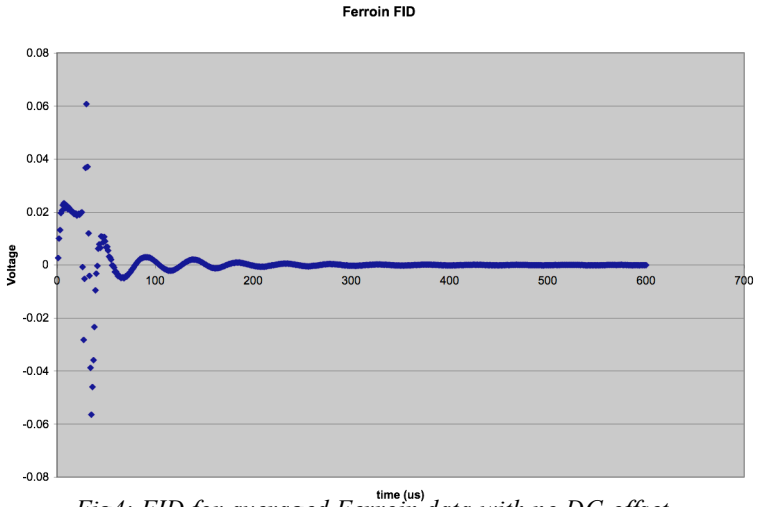


Fig4: FID for averaged Ferroin data with no DC-offset.

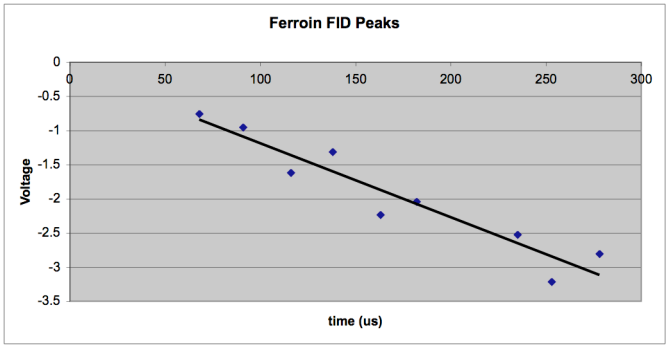


Fig5: Taking the natural log of the peaks and subtracting any remaining DC offset the peaks fit to:V(t) = -t/92.59 - 0.1024

The FID for Ferroin is very clear. We removed the DC offset in the same way explained above. The natural log of the peaks was fit to a line. The T2* factor here is 92.59µs.

To try to get a more accurate fit we used fitting software to fit a sinusoidal exponential decay to the FID, first using a constant T2* of 92.56μs and ten letting T2* vary.

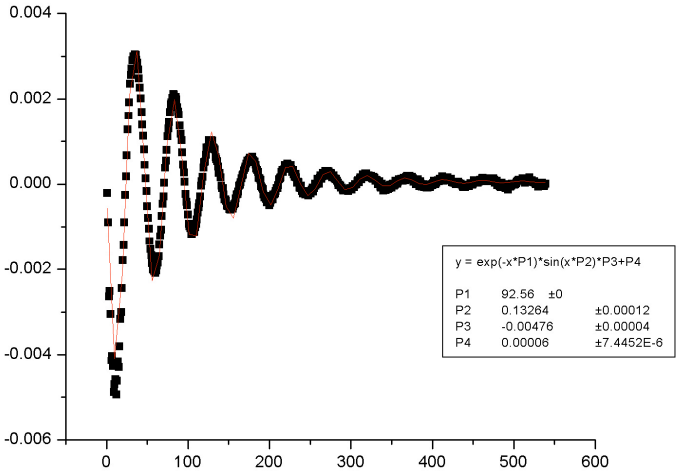


Fig6: Sinusoidal exponential decay fit of FID using T2 = 92.56μs*

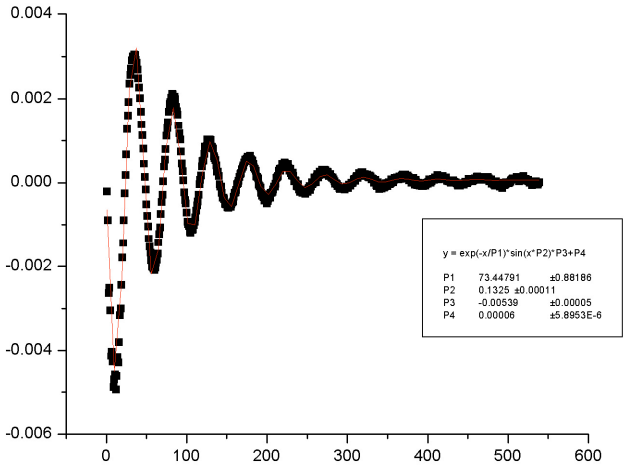


Fig7: Curve fit allowing T2 to vary giving T2* = 73.44μs*

The data for the spin echo, which would give us, a T2 value is shown below.

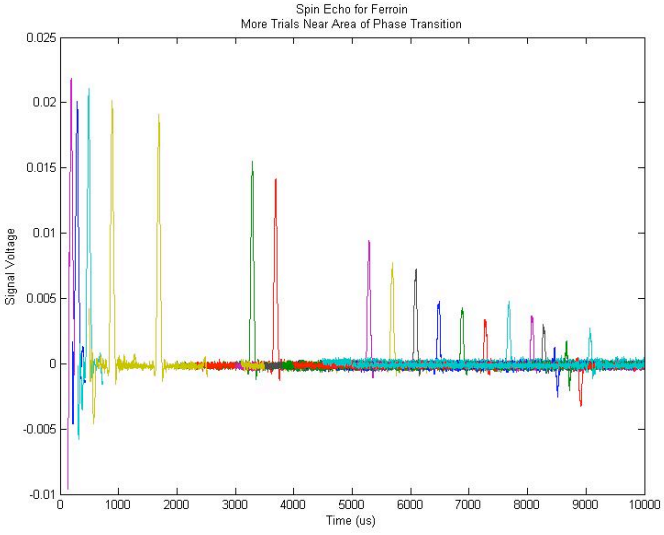


Fig8: Ferroin spin-echoes, 20 data sets for different τ.

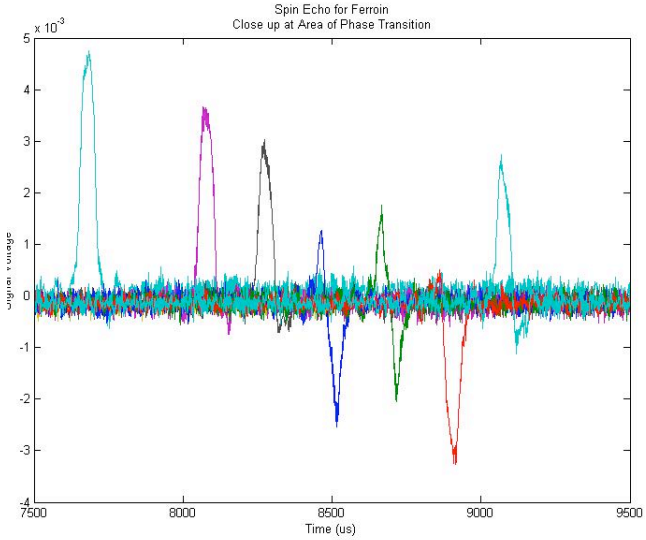
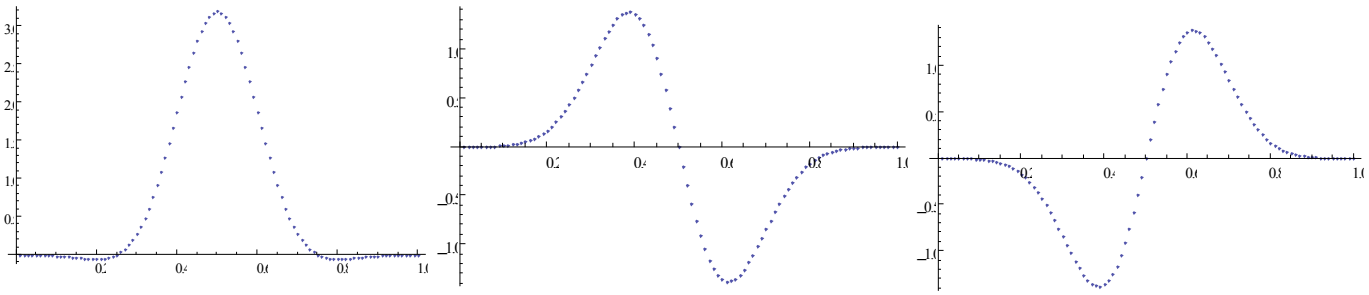


Fig9: Close up of phase-shifted spin-echoes.

In fig8 you can see the phase shifts of the spin echoes. The signal is read from a coil in the y-direction, but part of the spin-echo amplitude is in the x-direction for many of the data taken. Without knowing the actual phase change or the full amplitude there is no way to find a T2 value.

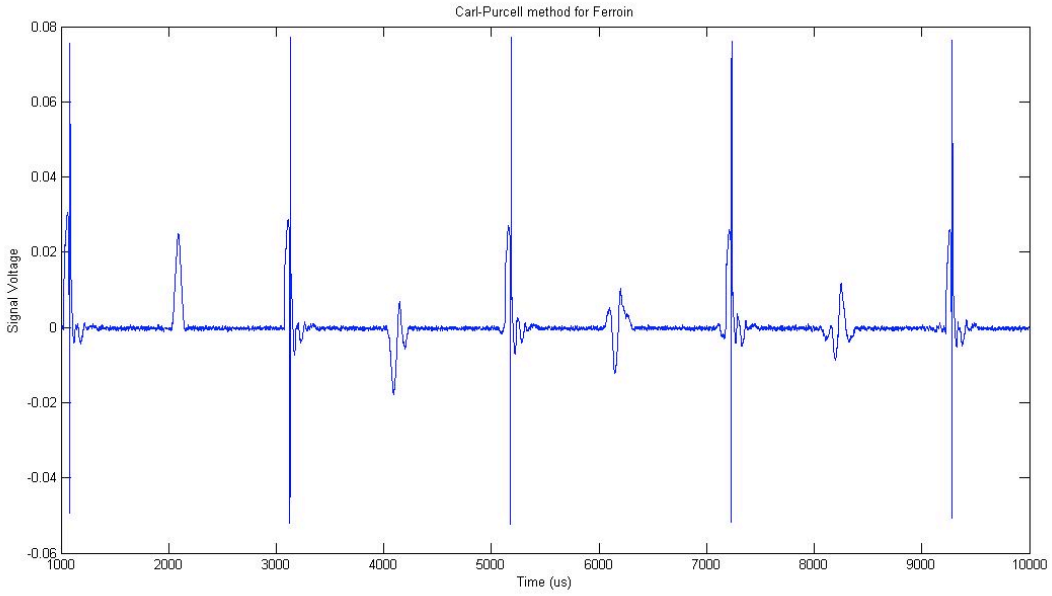
This phase shift is most likely due to the median angular velocity of the relative phases not matching perfectly with the applied field of 8 MHz. To confirm this effect resembles a phase shift we modeled the spin echo as a cosine function with its amplitude modulated by a Gaussian function centered at a half period of the cosine.



Above you can see this function with the cosine function phase shifted by pi. This appears similar to the ideal spin echo. Next to that is the cosine function phase shifted by +pi/2 and – pi/2. You can see that these resemble our non-ideal spin echoes.

Due to these irregular phase shifts, proper spin echo amplitude was hard to measure. It was for this reason that we resorted to measuring T2* off of the FID. The phase of the FID was very consistent from trial to trial, and its amplitude was large; this allowed for easy measurement.

One method commonly used for measuring T2 is the Carl-Purcell method. However, we were unsuccessful in using this method successive spin echoes would not be in phase. This made it difficult to measure the decrease in amplitude.



In the picture above the first four spin echoes are shown. It can be seen that each spin echo is out of phase with the previous spin echoes.

A modification to this technique was suggested by Meiboom and Gill (Brey 9) to overcome this phase problem by shiftime the phase of the 180-degree pulse by 90-degrees (transmitting across the x and y axis alternately). However, the lab equipment is currently set for reception in only one direction and transmission in only one other, orthogonal direction so we were unable to attempt this technique. Suggestions of how future lab groups might attempt to get around this problem are in the next section.

We do not consider our results very accurate. Although we do not see it in the ferriin data, the phase shift effect give major cause for concern with our ferriin T2 results. Also T2 and T2* are theoretically closely related, but in our data for ferriin and ferroin they seem to be different by orders of magnitude. There is much to be desired of the data, and because of this included is another section to address possible improvements for the next lab group.

Recommendations for Future Experiments:

Firstly we suggest that a T1 measurement is made, this measurement is known to be easier to make, and might give more insight into the spin-lattice interactions in the sample. As for T2 measurements the phase shift in the spin-echo needs to be addressed. We think this inconsistent phase is due to an unstable magnetic field. If the magnetic field is not constant in the z-direction the rotation of the magnetic moment may be out of phase with the 8Mhz pulse supplied in the x-direction. We changed the magnetic field to try to get it in phase, but were unable to keep a steady current flow into the magnetic could. To test this a hall probe could be used to see how stable the magnetic field is. A way to measure the full spin echo even with an unstable magnetic field is to not only measure from the y-direction but also the x-direction. This means using the coil currently used for transmitting, in the x-direction, to also receive. With this information the entire amplitude of the spin-echo in the x-y plane could be calculated. Finally we would like to recommend that a measurement of a ferroin catalyzed B-Z is done once ferriin and ferroin are correctly characterized. This can be done with the current data acquisition system. It would be interesting to see if the oscillating reaction could be detected. Tests have been done to test if the ferroin catalyzed B-Z could be imaged using NMRI and it was found to be difficult to image the reaction, but using the pulsed NMR there may be interesting results.

Works Referenced:

Magnetic Resonance Imaging of Ruthenium-, Cerium-, and ferroin-Catalyzed Belousov-Zhabotinky Reactions
Gao, Y., Cross, A.R., and Armstrong, R.L.
J.Phys. Chem, 100, 2, 10159 – 1016, 1996, 10.1021/jp9531691

Bruch, Martha. **Nmr Spectroscopy Techniques.** New York: M. Dekker, 1996.

Cowan, B. **Nuclear Magnetic Resonance and Relaxation.** Cambridge: Cambridge University Press, 1997.

Pochapsky, , Thoma et.al. **Nmr for Physical and Biological Scientists.** Washington: Taylor & Francis, 2007.

Brey, Wallace. **Pulse Methods in 1d and 2d Liquid-Phase Nmr.** Boston: Academic Press, 1988.
***All these books are available in the Science and Engineering library at UCSD*

Special Thanks

Erik Flister – Excellent MatLab coding, really the only thing that got the experiment of the ground. Freaking awesome.
Earl Dolnick – Resident pulsed NMR enthusiast ad best B-field tuning, spin-echo finder we know.
Professor Kleinfeld – Got the equipment we needed, and all the ferriin from horse spleen we never dreamed of.
Jordan Gosselin – Showed the data who's boss, and made many graphs with pretty colors.
Cassandra Niman – Created vector diagrams for describing NMR theory, painstakingly, by hand. Also discovered some totally sweet transitions for our presentation; they kept everyone awake, except Prof. Kleinfeld.

This is a copy of the data acquisition and Pulse code developed by Erik Flister, with a short data averaging sequence developed by Jordan Gosselin. It has been included with the report for the sake of preserving Erik's work on David Kleinfeld's class website forever, or for the duration that Dr. Kleinfeld maintains his website, whichever is shorter.

```
function pulseUIerror
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameters to set

hpath='C:\Program Files\National Instruments\NI-DAQ\DAQmx ANSI C Dev\include\NIDAQmx.h';
libpath='C:\WINNT\system32\nicaiu.dll'; %WINNT

recordingPath='C:\Documents and Settings\Undergrad Labs\Desktop\recordings\DeleteisOKFerroinOnePi';

pulseDev='Dev1';
counterNames={'/ctr0','/ctr2'}; %pulseA pulseB (can't use ctr1, cuz numPiPulses>1 causes us to use coupled
counters)
recordingDev='Dev2';
recordingChans=[0 1]; %data pulses

numTrials=16; %trials per recording
trialGap=.1; %in seconds

dPulse=10; %default pulseA microseconds
dGap=100; %default gap microseconds
mPulse=10000; %max pulseA microseconds
mGap=mPulse; %max gap microseconds

pulseFactor=2; % scale factor for pulseB duration (in terms of pulseA duration)
gapFactor = 2;
numPiPulses=10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clc
close all
format long g
daqreset

disp('initializing nidaqmx...')

% if length(recordingChans)~=length(counterNames)+1
%     error('must supply one more recordingChan than counterNames')
% end

left = 100;
bottom = 300;
width = 700;

margin = 25;
bWidth = 60;
bHeight = 30;
sWidth = width-2*margin;
sHeight = 20;
aHeight = 100;
```

```
tWidth = sWidth;
tHeight = 15;
tMargin = 5;

height = 6*margin+aHeight+2*tHeight+2*sHeight+bHeight;

hMainFigure   =   figure(...
   'MenuBar','none', ...
   'Toolbar','none', ...
   'Name', mfilename, ...
   'NumberTitle','off', ...
   'Position',[left bottom width height], ...
   'Resize', 'off',...
   'Units','pixels',...
   'CloseRequestFcn',@closeCallback,...
   'Visible','off');
hPlotAxes     =   axes(...
   'Parent', hMainFigure, ...
   'Units', 'pixels', ...
   'Position',[margin 2*margin width-2*margin aHeight]);
hUpdateButton  =   uicontrol(...
   'Style','togglebutton', ...
   'Parent', hMainFigure, ...
   'Units','pixels',...
   'Position',[margin 5*margin+aHeight+2*sHeight+2*tHeight bWidth bHeight],...
   'String','Start',...
   'Value',false,...
   'Callback', @hUpdateButtonCallback);
hPulseSlider = uicontrol(...
   'Parent', hMainFigure, ...
   'Style','slider',...
   'Units','pixels',...
   'Position',[margin aHeight+4*margin+sHeight+tHeight sWidth sHeight],...
   'Max',mPulse,'Min',1,'Value',dPulse,...
   'SliderStep',[1 10]/(mPulse-1),...
   'Callback', @hSliderCallback);
hGapSlider = uicontrol(...
   'Parent', hMainFigure, ...
   'Style','slider',...
   'Units','pixels',...
   'Position',[margin aHeight+3*margin sWidth sHeight],...
   'Max',mGap,'Min',1,'Value',dGap,...
   'SliderStep',[1 10]/(mGap-1),...
   'Callback', @hSliderCallback);
hPulseText = uicontrol(...
   'Parent', hMainFigure,...
   'Style','text',...
   'Units','pixels',...
   'Position',[margin aHeight+4*margin+2*sHeight+tHeight+tMargin tWidth tHeight-tMargin],...
   'String','',...
   'HorizontalAlignment','left');
hGapText = uicontrol(...
   'Parent', hMainFigure,...
   'Style','text',...
   'Units','pixels',...
   'Position',[margin aHeight+3*margin+sHeight+tMargin tWidth tHeight-tMargin],...
```

```
    'String','',...
    'HorizontalAlignment','left');

running=false;
pulses=false;
nidaqmx='nidaqmx';
tasks={};
update();
initNidaq();
set(hMainFigure,'Visible','on');

    function closeCallback(src,event)
        if running || pulses
            errordlg('can''t close while running -- wait for stop or hit stop.','error','modal');
        else
            cleanup();
        end
    end

    function hUpdateButtonCallback(hObject, eventdata)
        running=~running;
        update();
    end

    function hSliderCallback(hObject, eventdata)
        set(hObject,'Value',round(get(hObject,'Value')));
        update();
    end

    function update
        set(hUpdateButton,'Value',running);
        if running
            set(hUpdateButton,'String','Stop');
        else
            set(hUpdateButton,'String','Start');
        end

        set(hPulseText,'String',sprintf('Pulse: %g us',get(hPulseSlider,'Value')));
        set(hGapText,'String',sprintf('Gap: %g us',get(hGapSlider,'Value')));

        p=zeros(ceil(1.25*(mGap+(1+pulseFactor)*mPulse)),2);
        p((1:get(hPulseSlider,'Value'))+1,1)=1;
        p((get(hPulseSlider,'Value')+get(hGapSlider,'Value'))+(1:pulseFactor*get(hPulseSlider,'Value'))+1,2)=1;
        plot(hPlotAxes,0:length(p)-1,p);
        set(hPlotAxes,'YTick',[]);
        xlabel(hPlotAxes,'microseconds');
        ylim(hPlotAxes,[-1 2]);
        xlim(hPlotAxes,[0 size(p,1)-1]);

        if running && ~pulses
            startPulses();
        end
    end

    function ai=startRecording
        warning('off','MATLAB:MKDIR:DirectoryExists');
```

```
        mkdir(recordingPath);
        warning('on','MATLAB:MKDIR:DirectoryExists');

        ai = analoginput('nidaq',recordingDev);
        set(ai,'SamplesPerTrigger',inf);
        addchannel(ai,recordingChans);

        sr=propinfo(ai,'SampleRate');
        sr=setverify(ai,'SampleRate',max(sr.ConstraintValue));
        sr=setverify(ai,'SampleRate',1000000);
        disp(sprintf('sampling at %d hz',sr));
        %set(ai,'Coupling','DC');
        disp(sprintf('coupling is %s',ai.Channel(1).Coupling))
        %ai.Channel.Coupling = 'DC';
        disp(sprintf('input type is %s',get(ai,'InputType')))
            %pseudodifferential input: channels
            %all referred to a common ground but this ground is not connected to the computer
            %ground.
        %set(ai,'InputType','SingleEnded');
        d=daqhwinfo(ai);
        %d.InputRanges
        maxGain=d.InputRanges(find(diff(d.InputRanges')==min(diff(d.InputRanges'))),:);
        ir=setverify(ai.Channel(1),'InputRange',maxGain); %sets gain
        disp(['InputRange: ' num2str(ir)]);

        ir=setverify(ai.Channel(2),'InputRange',[-10 10]); %sets gain

        set(ai,'LogFileName',fullfile(recordingPath,sprintf(['recording.usPulse.%d.usGap.%d.' datestr(now,30)
'.daq'],get(hPulseSlider,'Value'),get(hGapSlider,'Value'))));
        set(ai,'LoggingMode','Disk'); %Disk&Memory

        start(ai)

        while ~strcmp(get(ai,'Logging'),'On')
            %block til ai started, otherwise miss first few pulses
        end
    end

    function startPulses
        if ~pulses && running

            pulses=true;

            ai=startRecording();

            usPulse = get(hPulseSlider,'Value');
            usBtwPulses = get(hGapSlider,'Value');

            taskname='pulses';

            %constants from NIDAQmx.h
            cDAQmx_Val_Hz  =  10373;
            cDAQmx_Val_Low = 10214;
            cDAQmx_Val_ContSamps = 10123;
            cDAQmx_Val_FiniteSamps = 10178;
            cDAQmx_Val_Seconds = 10364;
```

```
cDAQmx_Val_Falling = 10171;
cDAQmx_Val_WaitInfinitely = -1.0;

if isempty(tasks)
   try
      for i=1:length(counterNames)
         task=libpointer('uint32Ptr',0);
         [errCode gTaskname tasks{i}]=calllib(nidaqmx,'DAQmxCreateTask',[taskname num2str(i)],task);
         %task is supposed to come back as a uint32Ptr, but it's just a uint32
         checkErr(errCode,'DAQmxCreateTask');
      end

      % docs claim that for cDAQmx_Val_Low, pulse will start in hi phase
      errCode=calllib(nidaqmx,'DAQmxCreateCOPulseChanTime',tasks{1},[pulseDev
counterNames{1}],'',cDAQmx_Val_Seconds,cDAQmx_Val_Low,0,usPulse/(2*1000000),usPulse/1000000);
      checkErr(errCode,'DAQmxCreateCOPulseChanTime');

      errCode=calllib(nidaqmx,'DAQmxCreateCOPulseChanTime',tasks{2},[pulseDev
counterNames{2}],'',cDAQmx_Val_Seconds,cDAQmx_Val_Low,usBtwPulses/1000000,gapFactor*usBtwPulses/10
00000,pulseFactor*usPulse/1000000);
      checkErr(errCode,'DAQmxCreateCOPulseChanTime');

      if numPiPulses>1

errCode=calllib(nidaqmx,'DAQmxCfgImplicitTiming',tasks{2},cDAQmx_Val_FiniteSamps,numPiPulses);
         checkErr(errCode,'DAQmxCfgImplicitTiming');
      end

      [errCode]=calllib(nidaqmx,'DAQmxCfgDigEdgeStartTrig',tasks{2},['/' pulseDev
'/Ctr0InternalOutput'],cDAQmx_Val_Falling);
      checkErr(errCode,'DAQmxCfgDigEdgeStartTrig');

      rR=libpointer('doublePtr',0); %assume same for all channels...
      [errCode garbage rR]=calllib(nidaqmx,'DAQmxGetCOCtrTimebaseRate',tasks{1},[pulseDev
counterNames{1}],rR);
      checkErr(errCode,'DAQmxGetCOCtrTimebaseRate');
      disp(sprintf('counter master timebase: %g',rR))
      disp(sprintf('generating %d double pulse trials',numTrials));

      for t=1:numTrials
         for i=length(tasks):-1:1
            [errCode]=calllib(nidaqmx,'DAQmxStartTask',tasks{i});
            checkErr(errCode,'DAQmxStartTask');
         end

         [errCode]=calllib(nidaqmx,'DAQmxWaitUntilTaskDone',tasks{end},cDAQmx_Val_WaitInfinitely);
         checkErr(errCode,'DAQmxWaitUntilTaskDone');

         for i=1:length(tasks)
            [errCode]=calllib(nidaqmx,'DAQmxStopTask',tasks{i});
            checkErr(errCode,'DAQmxStopTask');
         end

         if ~running
            break
         end
```

```
            if mod(t,100)==0
               disp(sprintf('done w/trial %d of %d',t,numTrials))
            end

            tic;
            while toc<trialGap %pause is unreliable, can return too soon!
            end
         end

         disp('pulses stopped')

         for i=1:length(tasks)
            errCode=calllib(nidaqmx,'DAQmxClearTask',tasks{i});
            checkErr(errCode,'DAQmxClearTask');
         end
         tasks={};
      catch
         x=lasterror;
         x.message
         x.stack.line
         cleanup();
      end
   else
      error('startPulses called with nonempty tasks')
   end

   stop(ai);

   while strcmp(get(ai,'Logging'),'On')
      %block til ai stopped, otherwise might read unfinished file
   end

   if true
      [data,time,abstime,events,info] = daqread(get(ai,'LogFileName'),'DataFormat','Native');
      %Jordan added this
      %And he really hopes it will work
%-------------------------------------------------------------------------
      %This is now set up to average straight out of pulseui

clear datasplit
clear average
index = 0;
%changes hPipulse
hPipulse = get(hPulseSlider,'Value');
%changes gap
Gap = get(hGapSlider,'Value');


flag= 0;

v = 1;
kend = length(data);
k = v;
```

```
while(v <= kend)
for k = v:kend
   if (data(k,2)>1.0 && flag == 0)
      for n = 1:4
         if data(k+n,2) > 4.0
            flag = 1;
            index = index+1;
            break
          end

      end
   end

   if flag == 1
      if ((k+ceil((3*hPipulse+Gap+(0.05*(1e6)))-10))<=kend)

      datasplit(:,index) = data(k:k+ceil((3*hPipulse+Gap+(0.05*(1e6)))-10),1);
      end
      flag = 0;
      v = k + ceil((3*hPipulse+Gap+(0.05*(1e6)))-10);
      break
   end


end
if ((kend - k)<=ceil((3*hPipulse+Gap+(0.05*(1e6)))-10))

   break

end

end
clear abstime data ans events flag info k kend n printk time v
average = mean(datasplit(:,:),2);
matsave = sprintf('C:\\Documents and Settings\\Undergrad
Labs\\Desktop\\recordings\\usPulse.%d.usGap.%d.%s.mat',hPipulse,Gap,datestr(now,30));
save(matsave,'average','datasplit');
display(['Average saved under name: ' matsave])
display('Number of data points taken:')
display(index)
clear average index datasplit hPipulse Gap

%--------------------------------------------------------------------------

            %figure
            %plot(time,data)
            %xlabel('seconds'), ylabel('volts')
         end

         delete(ai);

         %analyzePulses(data(:,[1:length(counterNames)]),time);

         running=false;
         pulses=false;
         update();
```

```
    else
        pulses
        running
        error('startPulses called with pulses==true or running==false')
    end
end

function analyzePulses(data,time)
    figure
    bins=100;
    d=diff(data>1);
    [rups cups]=find(d>0);
    [rdowns cdowns]=find(d<0);
    for j=1:size(d,2)
        p{j,1}=time(rups(cups==j))*1000000;
        p{j,2}=time(rdowns(cdowns==j))*1000000;

        if length(p{j,1})~=length(p{j,2})
            disp(sprintf('chan: %d, num ups: %d, num downs: %d',j,length(p{j,1}),length(p{j,2})))
            error('pulse edge problem')
        end

        subplot(size(d,2)+2,2,2*j-1)
        hist(p{j,2}-p{j,1},bins);
        xlabel('microseconds')
        ylabel('count')
        title(sprintf('pulse %d: %d total',j,length(p{j,1})))

        subplot(size(d,2)+2,2,j*2)
        plot(p{j,2}-p{j,1})
        ylabel('microseconds pulse width')
        xlabel('pulse number')
    end

    if length(p{1,1})~=length(p{2,1})
        disp(sprintf('chan 1: %d pulses, chan 2: %d pulses',length(p{1,1}),length(p{2,1})))
        error('didn''t see same number of pulses in each channel')
    end

    subplot(size(d,2)+2,2,2*(size(d,2)+1)-1)
    hist(p{2,1}-p{1,2},bins);
    title('gap lengths')
    xlabel('microseconds')
    ylabel('count')

    subplot(size(d,2)+2,2,2*(size(d,2)+1))
    plot(p{2,1}-p{1,2})
    ylabel('microseconds gap length')
    xlabel('gap number')

    p{1,1}=p{1,1}(2:end)/1000;
    p{2,2}=p{2,2}(1:end-1)/1000;
    subplot(size(d,2)+2,2,2*(size(d,2)+2)-1)
    hist(p{1,1}-p{2,2},bins);
    title('intertrial intervals')
    xlabel('milliseconds')
```

```
    ylabel('count')

    subplot(size(d,2)+2,2,2*(size(d,2)+2))
    plot(p{1,1}-p{2,2})
    ylabel('milliseconds intertrial interval')
    xlabel('intertrial interval number')
end

function initNidaq
    try
        [notfound warnings]=loadlibrary(libpath, hpath, 'alias', nidaqmx);
    catch
        x=lasterror;
        x.message
        x.stack.line
        cleanup();
    end
end

function checkErr(e,fcn)
    knownWarnings=[200010];
    %don't know why we're getting this:
    %warning 200010: Measurements: Requested value is not a supported value for this property.
    %Property: DAQmx_AI_Max
    %You Have Requested:  42.420000
    %(valid range is -42 to 42)
    %seems to be coming from matlab's data acq ai object, not our task

    lib=nidaqmx;
    if e~=0 && ~ismember(e,knownWarnings)
        sprintf('error in %s\n',fcn)
        if e>0
            sprintf('got warning %d',e)
        elseif e<0
            sprintf('got error %d',e)
        end

        try
            errBuff=libpointer();
            [errBuffSize errBuff]=calllib(lib,'DAQmxGetExtendedErrorInfo',errBuff,0);
            errBuff=repmat(' ',1,errBuffSize);
            [newerr errBuff]=calllib(lib,'DAQmxGetExtendedErrorInfo',errBuff,errBuffSize);
            if newerr~=0
                'why is newerr ~=0?'
            end
            errBuff
        catch
            x=lasterror;
            x.message
            x.stack.line
        end

        error([lib ' error'])
    end
end
```

```
    function cleanup()
        lib=nidaqmx;
        if ~isempty(tasks)
            try
                for i=1:length(tasks)
                    errCode=calllib(lib,'DAQmxClearTask',tasks{i});
                    checkErr(errCode,'DAQmxClearTask');
                end
            catch
                x=lasterror;
                x.message
                x.stack.line
            end
            tasks={};
        end
        if libisloaded(lib)
            unloadlibrary(lib)
        end
        if strcmp(get(hMainFigure,'BeingDeleted'),'off')
            delete(hMainFigure)
            close all
        end
    end
end
```