In [ ]:
```
# For use with VScope v1.4r1
# Law Choi Look      ecclaw@ntu.edu.sg           07 Mar 2022 v1.3
# Wonkeun Chang      wonkeun.chang@ntu.edu.sg    22 Jul 2023 v2.7
```

In [ ]:
```
import serial
import serial.tools.list_ports

import threading
import time

import plotly
from ipywidgets import interactive_output,fixed,Button,ToggleButtons,SelectionSl
from IPython.display import display

import numpy as np

VID=61525
PID=38912
BAUDRATE=115200
BUFFERSIZE=1000
```

In [ ]:
```
# VScopeBoard class

class VScopeBoard:

    def __init__(self,vid=VID,pid=PID,baudrate=BAUDRATE):
        ports=serial.tools.list_ports.comports()
        for p in ports:
            if p.vid==vid and p.pid==pid:
                self.device=serial.Serial(p.device,baudrate=baudrate)
        if not hasattr(self,'device'):
            raise Exception('No controller unit detected')
        self.device.close()

        ########################################################################
        # Enter your calbiration data below
        # W1/W2 DC bias
        self.w1bias=0.141 # W1 DC bias reading in V; set it to 0 before calibrat
        self.w2bias=0.101 # W2 DC bias reading in V; set it to 0 before calibrat
        ########################################################################
        # CH1/CH2: gain and offset look-up table
        # 'dc' calibrate at each vscale
        # 'dc'=-bias/'amp'+'dc'
        self.p1={0.02 :{'gain':224,'amp':0.0950,'dc':2,'dco':130,'dco_c2':171,'a
                 0.05 :{'gain':208,'amp':0.1500,'dc':1.91,'dco':130,'dco_c2':168
                 0.1  :{'gain':160,'amp':0.3900,'dc':1.83,'dco':130,'dco_c2':159
                 0.2  :{'gain':138,'amp':0.5500,'dc':1.81,'dco':130,'dco_c2':157
                 0.5  :{'gain':220,'amp':2.1700,'dc':2.06,'dco':130,'dco_c2':130
                 1    :{'gain':212,'amp':2.8500,'dc':2.0,'dco':130,'dco_c2':130,
                 2    :{'gain':160,'amp':7.9000,'dc':1.85,'dco':130,'dco_c2':130
                 5    :{'gain':106,'amp':19.100,'dc':1.81,'dco':130,'dco_c2':130
        self.p2={0.02 :{'gain':224,'amp':0.0950,'dc':2.03,'dco':130,'dco_c2':171
                 0.05 :{'gain':208,'amp':0.1500,'dc':1.93,'dco':130,'dco_c2':167
                 0.1  :{'gain':160,'amp':0.3900,'dc':1.84,'dco':130,'dco_c2':159
                 0.2  :{'gain':138,'amp':0.5500,'dc':1.82,'dco':130,'dco_c2':158
                 0.5  :{'gain':220,'amp':2.1700,'dc':2.08,'dco':130,'dco_c2':130
                 1    :{'gain':212,'amp':2.8500,'dc':2.02,'dco':130,'dco_c2':130
                 2    :{'gain':160,'amp':7.9000,'dc':1.86,'dco':130,'dco_c2':130
```

```
            5    :{'gain':106,'amp':18.800,'dc':1.81,'dco':130,'dco_c2':130
    ########################################################################

# Set DC power supply voltage VDCP and VDCN
def set_vdc(self,voltage):
    cmd='dz'+str(int(voltage*100)).zfill(4)+'\r'
    self.device.open()
    self.device.reset_output_buffer()
    self.device.write(bytes(cmd,'utf-8'))
    self.device.close()

# Generate waveforms on W1 and W2
def generate_wave(self,channel,shape,amp,freq,offset):
    if channel==1:
        offset-=self.w1bias
        cmd='s1'
    else:
        offset-=self.w2bias
        cmd='s2'
    ns=64
    if shape=='Triangular':
        cmd+=str(10).zfill(2)
    elif shape=='Sawtooth':
        cmd+=str(11).zfill(2)
    else:
        cmd+=str(0).zfill(2)
    cmd+=str(ns).zfill(3)+str(freq).zfill(7)+str(int(amp*100)).zfill(4)+str(
    self.device.open()
    self.device.reset_output_buffer()
    self.device.write(bytes(cmd,'utf-8'))
    self.device.close()

# Capture oscilloscope traces on CH1 and CH2
def capture_oscscope(self,tbase,vscale1,coupling1,vscale2,coupling2):
    if tbase>1100:
        fs=50000
    elif tbase>510:
        fs=100000
    else:
        fs=200000 # maximum sampling rate=210 kHz
    if coupling1=='DC':
        c1=0 if vscale1<0.4 else 1
    else:
        c1=2 if coupling1=='AC' else 3
    if coupling2=='DC':
        c2=0 if vscale2<0.4 else 1
    else:
        c2=2 if coupling2=='AC' else 3
    if c1==2:
        dco1=self.p1[vscale1]['dco_c2']
        aco1=self.p1[vscale1]['aco_c2']
    else:
        dco1=self.p1[vscale1]['dco']
        aco1=0
    if c2==2:
        dco2=self.p2[vscale1]['dco_c2']
        aco2=self.p2[vscale1]['aco_c2']
    else:
        dco2=self.p2[vscale1]['dco']
        aco2=0
```

```
        cmd='m1'+str(fs).zfill(6)+str(c1)+str(self.p1[vscale1]['gain']).zfill(3)
        bytedata=bytearray(BUFFERSIZE*4)
        self.device.open()
        self.device.reset_output_buffer()
        self.device.reset_input_buffer()
        self.device.write(bytes(cmd,'utf-8'))
        self.device.readline()
        self.device.readinto(bytedata)
        self.device.close()
        data=np.frombuffer(bytedata,dtype='uint16').reshape((2,BUFFERSIZE))
        raw1=aco1+self.p1[vscale1]['amp']*(self.p1[vscale1]['dc']-1.5*data[0,:]/
        raw2=aco2+self.p2[vscale2]['amp']*(self.p2[vscale2]['dc']-1.5*data[1,:]/
        # Whittaer-Shannon interpolation for signal reconstruction on a finer gr
        n=2**9
        t=np.arange(n)*10*tbase*1e-6/n
        ch1=np.sum(np.multiply(raw1,np.transpose(np.sinc(t*fs-np.reshape(np.aran
        ch2=np.sum(np.multiply(raw2,np.transpose(np.sinc(t*fs-np.reshape(np.aran
        return t,ch1,ch2,raw1,raw2

    # Measure DC voltage on PC0 and PC1
    def measure_volt(self):
        cmd='m2\r'
        bytedata=bytearray(4)
        self.device.open()
        self.device.reset_output_buffer()
        self.device.reset_input_buffer()
        self.device.write(bytes(cmd,'utf-8'))
        self.device.readline()
        self.device.readinto(bytedata)
        self.device.close()
        data=np.frombuffer(bytedata,dtype='uint16')
        v1=3.3*data[0]/4095
        v2=3.3*data[1]/4095
        return v1,v2
```

In [ ]:

```
# RepeatTimer class

class RepeatTimer:

    def __init__(self,interval,function,*args,**kwargs):
        self.interval=interval
        self.function=function
        self.args=args
        self.kwargs=kwargs
        self.is_running=False

    def _run(self):
        self.is_running=False
        try:
            self.function(*self.args,**self.kwargs)
        except:
            pass
        self.start()

    def start(self):
        if not self.is_running:
            self._timer=threading.Timer(self.interval,self._run)
            self._timer.start()
            self.is_running=True
```

```
    def stop(self):
        try:
            self._timer.cancel()
            self.is_running=False
        except:
            pass
```

In [ ]:
```
vscope=VScopeBoard()
```

In [ ]:
```
# # Calibrate CH1 and CH2 DC offset voltages
# # Work only in DC coupling mode
# # 1. Connect CH1 and CH2 pins to GND
# # 2. Execute all cells above
# # 3. Uncomment this cell and execute
# # 4. Note down the new 'dc' values and incorporate in the look-up table in cap
# # 5. Comment back this cell once completed

# n_calibrate=10
# vdiv_list=[0.02,0.05,0.1,0.2,0.5,1,2,5]
# ch1_dc=list()
# ch2_dc=list()
# for vdiv in vdiv_list:
#     _,_,_,y1,y2=vscope.capture_oscscope(1000,vdiv,'DC',vdiv,'DC')
#     for i in range(n_calibrate-1):
#         time.sleep(0.5)
#         _,_,_,ch1,ch2=vscope.capture_oscscope(1000,vdiv,'DC',vdiv,'DC')
#         y1+=ch1
#         y2+=ch2
#     y1/=n_calibrate
#     y2/=n_calibrate
#     ch1_dc.append(vscope.p1[vdiv]['dc']-np.mean(y1)/vscope.p1[vdiv]['amp'])
#     ch2_dc.append(vscope.p2[vdiv]['dc']-np.mean(y2)/vscope.p2[vdiv]['amp'])
#     print('Completed calibrating '+str(vdiv).rjust(4)+' V/Div')
# for i, vdiv in enumerate(vdiv_list):
#     print('\'dc\' CH1 '+str(vdiv).rjust(4)+' V/Div: '+str(round(ch1_dc[i],2)))
# for i, vdiv in enumerate(vdiv_list):
#     print('\'dc\' CH2 '+str(vdiv).rjust(4)+' V/Div: '+str(round(ch2_dc[i],2)))
```

In [ ]:
```
# DC voltage supply interface

vdc=FloatSlider(min=5.5,max=13.5,step=0.1,value=5.5,continuous_update=False,read
vdc_ui=HBox([Label(value='VDCP/VDCN (±V)',layout=Layout(width='105px')),vdc],lay

ps_html='<h1 style=\'text-align:center;font-size:16px;background-color:lightcora
ps_title=HTML(value=ps_html,layout=Layout(width='820px'))
ps_ui=HBox([vdc_ui],layout=Layout(justify_content='space-around',width='820px',h

interactive_output(vscope.set_vdc,{'voltage':vdc})
```

Out[ ]: Output()

In [ ]:
```
# Waveform generator interface

shape1=ToggleButtons(options=['Sinusoidal','Triangular','Sawtooth'],value='Sinus
amp1=FloatSlider(min=0,max=5,step=0.01,value=0,readout_format='.2f',continuous_u
freq1=IntSlider(min=100,max=50000,step=100,value=1000,continuous_update=False)
offset1=FloatSlider(min=-6,max=6,step=0.01,value=0,readout_format='.2f',continuc

shape2=ToggleButtons(options=['Sinusoidal','Triangular','Sawtooth'],value='Sinus
```

```
amp2=FloatSlider(min=0,max=5,step=0.01,value=0,readout_format='.2f',continuous_u
freq2=IntSlider(min=100,max=50000,step=100,value=1000,continuous_update=False)
offset2=FloatSlider(min=-6,max=6,step=0.01,value=0,readout_format='.2f',continuc

shape1_ui=HBox([Label(value='Shape',layout=Layout(width='105px',display='flex',j
amp1_ui=HBox([Label(value='Amplitude (V)',layout=Layout(width='105px',display='f
freq1_ui=HBox([Label(value='Frequency (Hz)',layout=Layout(width='105px',display=
offset1_ui=HBox([Label(value='Offset (V)',layout=Layout(width='105px',display='f

shape2_ui=HBox([Label(value='Shape',layout=Layout(width='105px',display='flex',j
amp2_ui=HBox([Label(value='Amplitude (V)',layout=Layout(width='105px',display='f
freq2_ui=HBox([Label(value='Frequency (Hz)',layout=Layout(width='105px',display=
offset2_ui=HBox([Label(value='Offset (V)',layout=Layout(width='105px',display='f

wg1_ui=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;m
wg2_ui=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;m

wg_ui=HBox([wg1_ui,wg2_ui],layout=Layout(justify_content='space-around',width='8

wg_html='<h1 style=\'text-align:center;font-size:16px;background-color:lightblue
wg_title=HTML(value=wg_html,layout=Layout(width='820px'))

interactive_output(vscope.generate_wave,{'channel':fixed(1),'shape':shape1,'amp'
interactive_output(vscope.generate_wave,{'channel':fixed(2),'shape':shape2,'amp'
```

Out[ ]: Output()

In [ ]:

```
# Oscilloscope interface

fig=plotly.graph_objs.FigureWidget()
fig.update_layout(width=380,height=410,margin=dict(l=2,r=2,t=2,b=32),paper_bgcol
fig.update_layout(xaxis=dict(showticklabels=False,showgrid=True,gridwidth=1,grid
fig.update_layout(yaxis1=dict(tickmode='array',tickvals=[0],ticks='inside',tickw
fig.update_layout(yaxis2=dict(tickmode='array',tickvals=[0],ticks='inside',tickw
fig.update_layout(yaxis3=dict(range=[-5,5],dtick=1,showticklabels=False,showgrid
fig.add_trace(plotly.graph_objs.Scatter(x=[],y=[],name='CH1',yaxis='y1'))
fig.add_trace(plotly.graph_objs.Scatter(x=[],y=[],name='CH2',yaxis='y2'))
fig.add_trace(plotly.graph_objs.Scatter(x=[],y=[],yaxis='y3'))
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color='white',
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color='white',
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color=plotly.c
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color=plotly.c
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color='black',

coupling1=ToggleButtons(options=['DC','AC','GND'],value='DC',style={'button_widt
vscale1=SelectionSlider(options=[0.02,0.05,0.1,0.2,0.5,1,2,5],value=0.2,continuc
vpos1=FloatSlider(min=-5,max=5,step=0.01,value=0,readout_format='.2f',continuous

coupling2=ToggleButtons(options=['DC','AC','GND'],value='DC',style={'button_widt
vscale2=SelectionSlider(options=[0.02,0.05,0.1,0.2,0.5,1,2,5],value=0.2,continuc
vpos2=FloatSlider(min=-5,max=5,step=0.01,value=0,readout_format='.2f',continuous

tbase=SelectionSlider(options=[20,50,100,200,500,1000,2000],value=500,continuous

capture=Button(description='Capture',button_style='primary')

coupling1_ui=HBox([Label(value='Coupling',layout=Layout(width='105px',display='f
vscale1_ui=HBox([Label(value='Volts/Div',layout=Layout(width='105px',display='fl
vpos1_ui=HBox([Label(value='Y-Pos.',layout=Layout(width='105px',display='flex',j
```

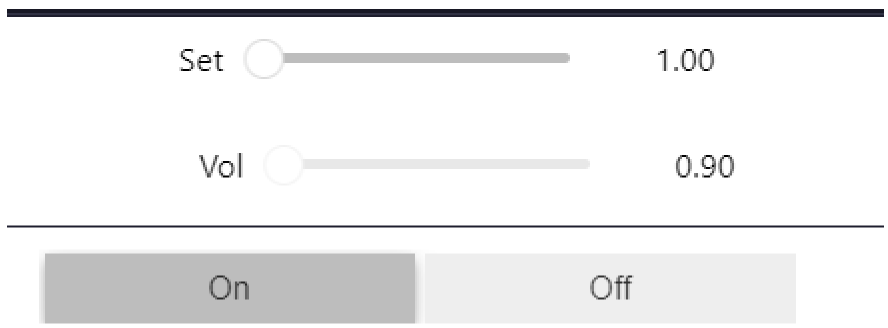
```
coupling2_ui=HBox([Label(value='Coupling',layout=Layout(width='105px',display='f
vscale2_ui=HBox([Label(value='Volts/Div',layout=Layout(width='105px',display='fl
vpos2_ui=HBox([Label(value='Y-Pos.',layout=Layout(width='105px',display='flex',j

tbase_ui=HBox([Label(value='Time/Div (µS)',layout=Layout(width='105px',display='

osc1=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;mar
osc2=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;mar
osc3=VBox([tbase_ui,capture],layout=Layout(align_items='center'))
osc_ui=HBox([VBox([fig],layout=Layout(justify_content='space-around')),VBox([osc

osc_html='<h1 style=\'text-align:center;font-size:16px;background-color:lightgre
osc_title=HTML(value=osc_html,layout=Layout(width='820px'))

def calchar(y,dt,tbase):
    n_avg=5
    y=np.convolve(y,np.ones(n_avg),mode='valid')/n_avg # moving average
    rms=np.sqrt(np.mean(y**2))
    ptp=np.ptp(y)
    if tbase<510:
        y=np.pad(y,(0,int(0.01*len(y)/(10*tbase*1e-6))-len(y)),'constant') # ens
    freq=np.abs(np.fft.fftfreq(len(y))[np.argmax(np.abs(np.fft.fft(y-np.mean(y))
    return rms,freq,ptp

def txtchar(ch,rms,freq,ptp):
    text='CH1    ' if ch==1 else 'CH2    '
    text=text+'RMS: {0:6.2f} mV    '.format(rms*1e3) if rms<1 else text+'RMS: {0
    text=text+'Freq: {0:6.3f} kHz    '.format(freq*1e-3) if freq>1e3 else text+'
    text=text+'Vpp: {0:6.2f} mV'.format(ptp*1e3) if ptp<1 else text+'Vpp: {0:6.2
    return text

def capture_oscscope(empty=None):
    if coupling1.value=='AC' and vscale1.value>0.4:
        print('Only DC coupling is available for V/Div > 0.5. Selecting DC coupl
        coupling1.value='DC'
    if coupling2.value=='AC' and vscale2.value>0.4:
        print('Only DC coupling is available for V/Div > 0.5. Selecting DC coupl
        coupling2.value='DC'
    t,ch1,ch2,_,_=vscope.capture_oscscope(tbase.value,vscale1.value,coupling1.va
    fig.data[0]['x']=fig.data[1]['x']=t
    fig.data[0]['y']=ch1
    fig.data[1]['y']=ch2
    fig.update_layout(xaxis=dict(range=[0,10*tbase.value*1e-6],dtick=tbase.value
    fig.update_layout(yaxis1=dict(range=[vscale1.value*(-vpos1.value-5),vscale1.
    fig.update_layout(yaxis2=dict(range=[vscale2.value*(-vpos2.value-5),vscale2.
    rms1,freq1,ptp1=calchar(ch1,t[1],tbase.value)
    rms2,freq2,ptp2=calchar(ch2,t[1],tbase.value)
    fig.update_layout(annotations=[dict(text=txtchar(1,rms1,freq1,ptp1)),dict(te

capture_oscscope()

capture.on_click(capture_oscscope)
tbase.observe(capture_oscscope,'value')
vscale1.observe(capture_oscscope,'value')
vpos1.observe(capture_oscscope,'value')
coupling1.observe(capture_oscscope,'value')
vscale2.observe(capture_oscscope,'value')
vpos2.observe(capture_oscscope,'value')
coupling2.observe(capture_oscscope,'value')
```

In [ ]:
```python
set = FloatSlider(description='Set', continuous_update=False, value=10)
vol = FloatSlider(description='Vol', disabled=True)
switch = ToggleButtons(options=['On', 'Off'], value='Off')
```

In [ ]:
```python
va_min = -2.7    # vC must be between -0.15 V (loudest) and 0.5 V (softest)
va_max = 9.0
max_volume = 3.3 # maximum voltage that can be read by PC0 and PC1
```

In [ ]:
```python
delta = 0.01 # va voltage increment/decrement size in each iteration

u = 0.0 # initial va
vscope.generate_wave(2, None, 0, 1000, u) # set initial va at 0
def control(): # this function gets called repeatedly once the power is on
    global u # ensure variable u created outside can be used in the function
    v_volume, _ = vscope.measure_volt() # voltage readings at PC0/PC1; discard P
    r0 = set.value # user set volume level taken from the set slider
    z = v_volume/max_volume*100 # measured volume as a fraction of maximum volum
    if z < r0:
        u-=delta
    elif z > r0:
        u+=delta
    u = np.clip(u, va_min, va_max) # ensure u stays within the allowed range of
    vscope.generate_wave(2, None, 0, 1000, u) # set the W2 offset to the new u
    vol.value = z # adjust the vol slider to the detected volume level

auto = RepeatTimer(0.01, control) # call control() function every 10 ms

def power_onoff(empty=None): # switch on/off audio amplifier
    if switch.value == 'On':
        vscope.set_vdc(13.5) # switch on the amplifier
        time.sleep(0.5) # pause for 0.5 s to avoid port access congestion
        auto.start() # start the timer
    else:
        auto.stop() # stop the timer
        time.sleep(0.5) # pause for 0.5 s to avoid port access congestion
        vscope.set_vdc(5.5) # switch off the amplifier

switch.observe(power_onoff, 'value') # call power_onoff() function
display(set, vol, switch) # display the user interface elements
```

```
FloatSlider(value=10.0, continuous_update=False, description='Set')
FloatSlider(value=0.0, description='Vol', disabled=True)
ToggleButtons(index=1, options=('On', 'Off'), value='Off')
```



In [ ]:
```python
# display(ps_title,ps_ui)
# display(wg_title,wg_ui)
# display(osc_title,osc_ui)
```

Adjusting the Delta to a smaller value would result in a smaller increment in the volume adjustments. Resulting in a smoother increase or decrease to the volume change. Higher Delta would make the volume to have sudden spikes and might result in a distorting or 'choppy' sound. This happens as the Delta is directly affecting the Va which acts as the volume control.

Open Ended Question

From the RepeatTimer class, the accuracy of the time interval between calls to the control function is accurate and reliable enough. When changes to 1s instead of 10ms, the volume can be seen changing at an accurate rate of 1s. This shows that every 10ms, control() will be reliably be called. Additional observation was seen when the timing was changed to 100us. A faster change to the volume can be observed.

However, the RepeatTimer class has a bit of a delay where whenever the class is called, a check if function given is able to run is done before setting the interval for the function. Even though this might not cause might of an issue for this case, if a function with more complexity or even with a time delay would cause an delay at the start of the class and might not be reliable anymore.