Lab 2

Objectives

To install custom firmware on the controller unit, aquire time-sampled readings by sending command strings, and learn to visualize the data

Communication Channel with device

In [ ]:
```
# communication channel setup
import serial
import serial.tools.list_ports

VID = 61525
PID = 38912

ports = serial.tools.list_ports.comports()

device = None
for p in ports:
    if p.vid == VID and p.pid == PID:
        try:
            device = serial.Serial(p.device)
        except serial.SerialException:
            print('Reconnect the controller unit.')

if device is None:
    raise Exception('No suitable device detected.')
```

Setting VDCP

Command string

Format: 'dzAAAA\r'

- 'dz' instructs the controller unit to set the DC supply voltage.
- 'AAAA' represents the intended DC voltage value in **10 mV** increments, with zero-padding to fill up four-character spaces.
- '\r' is the carriage return character

Setting VDCP to 8V

In [ ]:
```
vdc = 8
cmd_setvdc = 'dz' + str(int(vdc*100)).zfill(4) + '\r'
print(cmd_setvdc)
```

```
dz0800
```

'int(vdc*100)' - converts the desired voltage balue to an integer by multiplying by 100.

'str().zfill()' - converts the above integer value into a string and adds '0' at the front until the string reaches the specified length indicated in the bracket.

Setting VDCP to 12V

```
In [ ]: vdc = 12
        cmd_setvdc = 'dz' + str(int(vdc*100)).zfill(4) + '\r'
```

Sending the command to the controller unit

```
In [ ]: device.write(bytes(cmd_setvdc, 'utf-8'))

Out[ ]: 7
```

Generating timed voltage singnals

Command string

Format: 'sBCCDDDEEEEEEEFFFFGGGG\r'

- 'sB' indicates the channel for generating the waveform with '1' or '2' replacing 'B' to represent W1 or W2, respectively.
- 'CC' denotes the waveform shape. Currently, three shapes are supported: sinusoidal ('00'), triangular ('10'), and sawtooth ('11').
- 'DDD' specifies the number of samples per oscillation cycle.
- 'EEEEEEE' represents the oscillation frequency in Hz.
- 'FFFF' is the signal amplitude in 10 mV.
- 'GGGG' is the DC offset in 10 mV.

Creating and measuring a sinusoidal waveform

2kHz freq, 0.5V Amplitude, 0V DC offset on W1

```
In [ ]: ns = 64              # samples per cycle; 64 generally enough
        freq = 2000          # frequency in Hz
        amp = 0.5            # amplitude
        offset = 0           # DC offset
        cmd_gensin = 's100' # W1 sinusoidal
        cmd_gensin += str(ns).zfill(3) + str(freq).zfill(7) + str(int(amp*100)).zfill(4)
        device.write(bytes(cmd_gensin, 'utf-8'))    #sending the command string to contr

Out[ ]: 23
```

Measurements

Vp = 0.5

Vrms(ideal) = 0.707Vp = 0.35335

Vrms(measured) = 0.318

Freq = 2kHz

Creating and measuring a triangular waveform

2kHz freq, 0.5V Amplitude, 0V DC offset on W1

In [ ]:
```
ns = 64             # samples per cycle; 64 generally enough
freq = 2000         # frequency in Hz
amp = 0.5           # amplitude
offset = 0          # DC offset
cmd_gensin = 's110' # W1 triangular
cmd_gensin += str(ns).zfill(3) + str(freq).zfill(7) + str(int(amp*100)).zfill(4)
device.write(bytes(cmd_gensin, 'utf-8'))    #sending the command string to contr
```

Out[ ]:
```
23
```

Measurements

Vp = 0.5

Vrms(ideal - sinusoidal factor) = 0.707Vp = 0.3535V Vrms(ideal - triangular factor) = 0.577Vp = 0.2885V

Vrms(measured) = 0.258V

Freq = 2kHz

Discussion of the change of meaurements

The root mean square (RMS) value of a waveform, commonly abbreviated as Vrms, is a measure of its effective voltage. It's basically the equivalent DC voltage that would cause the same amount of power dissipation in a resistive load as the original waveform. The RMS values of various waveforms are calculated using the distribution of their energy. The relationship between the RMS value and the peak amplitude is not constant for all waveforms; it is determined by the waveform's shape and properties.

The RMS value of sinusoidal and triangular waves varies primarily due to their differing waveform shapes and energy distribution over time.

1. Sinusoidal Wave:

   - For a sinusoidal wave, the RMS value can be calculated using the formula: Vrms/sqrt(2), where Vp is the peak amplitude of the sinusoidal wave.
   - Sinusoidal waves have a smooth, continuous variation from peak to peak, and the RMS value is about 0.707 times the peak value.
   - 0.707 is calculated mathematically as the RMS value of a sine wave by integrating the square of the waveform across a complete cycle and then taking the square root of the result.

2. Triangular Wave:

   - For a triangular wave, the RMS value can be calculated based on its peak amplitude and the shape of the waveform.
   - For a triangle waveform, the RMS value is roughly 0.577 times the peak amplitude.
   - 0.577 is calculated using the mathematics of a triangle waveform and its energy distribution. It entails computing the area under the triangle waveform and executing the relevant mathematical calculations to determine the RMS value.
   - The exact formula for calculating the RMS value of a triangular wave is a bit more involved than for a sinusoidal wave due to its shape.

The relationship between RMS value and peak amplitude is determined by the waveform's shape and energy distribution across time. Triangular waveforms spend more time around their peak values than sinusoidal waveforms, resulting in a higher RMS value than would be expected if you just scaled it by 0.707. **However, the exact factor of 0.577 is unique to the triangle waveform and its mathematical characteristics.**

In conclusion, the 0.577 factor for triangular waveforms and the 0.707 factor for sinusoidal waveforms are obtained from their respective mathematical calculations of RMS values and are fundamental features of these waveforms.

Reading timed voltage signals

Command String

Format: 'm1HHHHHHIJJJKKKLMMMNNN\r'

- 'm1', instructs the controller unit to read timed voltage signals from CH1 and CH2.

- 'HHHHHH' represents the sampling frequency given in Hz. For our current implementation of the controller unit, the maximum sampling rate that can reliably read both channels is 210kHz.

- 'IJJJKKK' and 'LMMMNNN' are the coupling mode ('I' and 'L'), gain ('JJJ' and 'MMM'), and DC offset ('KKK' and 'NNN') parameters for CH1 and CH2, respectively. These values are

crucial for configuring the ADC to accurately measure the signal within the expected voltage range. Calibration is necessary to determine the values of these parameters for different voltage ranges. In particular, the DC bias values can vary between units and need to be adjusted individually.

Creating the sinusoidal signal

In [ ]:
```
ns = 64                 # samples per cycle; 64 generally enough
freq = 2000             # frequency in Hz
amp = 0.5               # amplitude
offset = 0              # DC offset
```

```
cmd_gensin = 's100' # W1 sinusoidal
cmd_gensin += str(ns).zfill(3) + str(freq).zfill(7) + str(int(amp*100)).zfill(4)
device.write(bytes(cmd_gensin, 'utf-8'))    #sending the command string to contr
```

```
Out[ ]: 23
```

Reading and Output of sampled data

```
In [ ]: fs = 200000         # sampling frequency
        c1 = 0              # CH1 DC coupling mode; 0 for signals within around ±1 V
        gain1 = 138         # CH1 gain; set the range of voltage that can be read
        dco1 = 130          # CH1 offset; set the zero level
        c2 = 0              # CH2 DC coupling mode
        gain2 = 138         # CH2 gain
        dco2 = 130          # CH2 offset
        cmd_readosc = 'm1' + str(fs).zfill(6) + str(c1) + str(gain1).zfill(3) + str(dco1
```

Creating the byte array and read data into it

```
In [ ]: bytedata = bytearray(4000)     # create a byte array of size 4000
        device.reset_input_buffer()  # clear data in input buffer
        device.write(bytes(cmd_readosc, 'utf-8'))
        device.readline()            # skip line containing the command string
        device.readinto(bytedata)
```

```
Out[ ]: 4000
```

Converting byte array into 2-dimensional NumPy array for processing

```
In [ ]: import numpy as np

        data = np.frombuffer(bytedata, dtype='uint16').reshape((2, 1000))
```

Visualising the timed voltage signals

Matching the length of time array and length of sampled data

```
In [ ]: t = np.arange(1000)/fs
```

Visualising the data using Plotly

Plot: 't' vs 'data[x, :]

x = 0 : Channel 1

x = 1 : Channel 2

Channel 1

```
In [ ]: import plotly

        fig = plotly.graph_objs.Figure()
```

```
fig.add_trace(plotly.graph_objs.Scatter(x=t*1e3, y=data[0, :]))
fig.update_layout(xaxis_title='Time (ms)', yaxis_title='Voltage (a.u.)')
```

Channel 2

In [ ]:
```
import plotly

fig = plotly.graph_objs.Figure()
fig.add_trace(plotly.graph_objs.Scatter(x=t*1e3, y=data[1, :]))
fig.update_layout(xaxis_title='Time (ms)', yaxis_title='Voltage (a.u.)')
```

Yes, both channel produce idential signals as the Micro-controller unit shares the same clock signal for both channel 1 and channel 2. Therefore it is expected that both channels produces idential waveforms

Close Connection

In [ ]:
```
device.close()
```

Open-ended Question

Given that the maximum sampling rate of the ADC that can reliably read both channels is 210 kHZ, what is the highest signal frequency it can capture? What happens if the ADC tries to pick up a signal that is higher than that?

Due to the requirement known as the Nyquist sampling frequency, the sampling frequency of the ADC should be at least twice of the maximum frequency of the analog signal.

Given that the maximum sampling rate of the ADC is 210kHz, the highest signal frequency it can capture accurately is 210kHz / 2 = 105kHz.

If the ADC captures a signal higher than its Nyquist Frequency (in this case, 105kHz), *Aliasing can occur.*

Aliasing occurs when the signal frequency surpasses half the sampling frequency, resulting in distortions and inconsistencies in the digital representation of the signal.

Aliasing happens when higher-frequency components of a signal fold back into the frequency range below the Nyquist frequency, resulting in misleading signals or distortion. These spurious signals might disrupt the proper representation of the underlying signal, making it difficult to analyse or comprehend.

To avoid aliasing, the ADC's sampling frequency must be at least twice the highest frequency component of the signal being sampled. This ensures that the Nyquist-Shannon sampling theorem is met and that the signal is represented correctly in the digital domain. If the signal frequency exceeds the Nyquist frequency, proper antialiasing filters or downsampling techniques may be required to reduce aliasing before sampling.

What is the time span of a time series data that has 1,000 data points sampled at 200 kHz sampling frequency?

Time Span = Number of Data Points / Sampling Frequency

Data points (N) = 1000 Sampling Frequency (*f*) = 200kHz

Therefore time span = 1000/200k = 0.005 seconds

or time span (frequency) = 200Hz