ANALOG DEVICES
想像を超える可能性を
AHEAD OF WHAT'S POSSIBLE™

バーチャル・エレクトロニクス・ラボ：PythonとADLM2000によるオシロスコープの開発

著者：Christian Jason Garcia、ソフトウェア・システム・エンジニア
Arnie Mae Baes、ソフトウェア・システム・エンジニア

概要

本稿のテーマであるバーチャル・エレクトロニクス・ラボとは、ソフトウェアをベースとする一連の計測器のことです。エレクトロニクス用の各種実験装置をソフトウェア・アプリケーションとして実装し、実験室と同様の環境を構築することが目標です。この環境を利用することで、エレクトロニクスに関する様々な実験を行うことができます。必要な機材をすべてそろえた完全な実験室を用意するのには、多大な費用がかかります。また、その管理には大きな労力が必要になります。エレクトロニクス向けの実験室に求められるすべての機能がポケットに収まっている状況を想像してみてください。その可能性は無限大であることがわかるでしょう。

本稿では、アクティブ・ラーニング・モジュール「ADALM2000」を使用して、自分専用のバーチャルな実験用装置を開発する方法について説明します。ソフトウェアの開発に使用するプログラミング言語としては、シンプルかつオープンソースであることからPythonを選択することにします。PythonとADALM2000を組み合わせれば、様々な機能を提供するバーチャルな実験用装置を開発することができます。例えば、オシロスコープ、信号発生器、デジタル・マルチメータなどの機能を実現することが可能です。なかでも、本稿では、オシロスコープの機能に焦点を絞ることにします。オシロスコープは、実際の実験室で使用される最も基本的な計測器です。そのため、最初に開発するバーチャルな実験用装置としては最適なものだと言えるでしょう。

はじめに

計測器の業界は、急速かつ着実に仮想化の方向へと移行しています。すなわち、ソフトウェアをベースとし、PC上で稼働する計測器が提供されるようになっているのです。つまり、測定や制御の対象となるデバイスを接続するために使用する専用のハードウェアは、可能な限り最小限に抑えられるようになってきています。通常、ハードウェアはプラグイン式のボードに対応するようになっており、信号を直接デジタル化したり、スタンドアロンの計測器を制御したりするために使われます。バーチャルな計測器は、柔軟性、モジュール性、可搬性に優れています。

アナログ・デバイセズは、お客様が直面するあらゆる状況を想定して多種多様な電子モジュールを提供しています。ADALM2000もその1つの例です。これを利用すれば、技術者／開発者は、特定のニーズに応じて自分専用のバーチャル・エレクトロニクス・ラボを構築することができます。その開発作業の鍵を握るものがlibm2kというライブラリです。これを利用すれば、C++やC#、Pythonによって、ADALM2000を制御するためのソフトウェア・アプリケーションを容易に開発することができます。

オシロスコープとは何か？

オシロスコープは、エレクトロニクスの分野では欠かせない計測器です（図1）。一般的な回路、複雑な回路を対象として信号を解析する際に、多大な価値をもたらします。特に、最近のオシロスコープは、コンピュータとの接続機能も備えています。つまり、オシロスコープで取得した信号をデジタル・データとして保存し、後でPC上で解析を実施することができるということです。

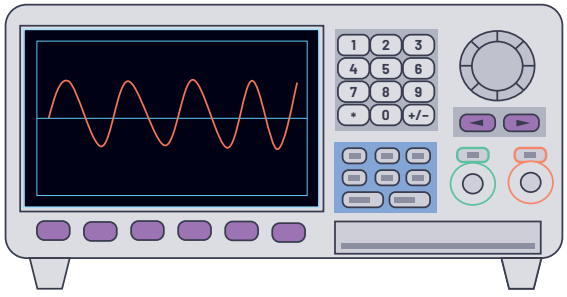

図1. オシロスコープの外観

オシロスコープは、アナログ／デジタルの信号波形を対象とし、時間軸で見た電圧の遷移や特性を表示するために使用されます。フロント・パネルに設けられた各種のボタンやつまみを使用し、アンプのトリガ、掃引時間、表示の設定を行うことで、より見やすい形で信号波形が表示されるよう調整することができます。

オシロスコープを使用すれば、特定の期間にわたる信号の挙動を把握することができます。このような機能は、一般的な回路の解析を行う上で不可欠です。また、各種回路の機能を検証する際にも役立ちます。つまり、エレクトロニクスを扱う実験室において、オシロスコープは不可欠な存在だと言えます。また、技術者が自身のニーズに応じてオシロスコープをカスタマイズできるようにすれば、特定の電子回路の解析能力をより高めることができます。

ADALM2000とは何か？

ADALM2000はアクティブ・ラーニング・モジュールというカテゴリの製品であり、多彩な機能を提供します。デジタル・オシロスコープ、ファンクション・ジェネレータ、ロジック・アナライザ、電圧計、スペクトラム・アナライザ、デジタル・バス・アナライザの機能に加え、2つのプログラマブル電源を備えています。ADALM2000は、ソフトウェア・パッケージ「Scopy」を使うことで容易に操作できます。学生や初級レベルの技術者であれば、この使い方が適しているでしょう。一方、ある程度のスキルを有するアプリケーション開発者であれば、libm2kライブラリを利用することで、独自のアプリケーション・インターフェースを開発することができます。また、ファームウェア開発者向けには、ADALM2000で直接実行できるカスタム・ソフトウェアやHDL（Hardware Description Language）コードを開発するためのオプションが提供されています。

開発に向けた事前の準備

まずは、開発作業を行うために事前に準備すべき事柄について説明します。

PythonとPyCharmのインストール

Pythonは、強力かつ習得が容易なオープンソースのプログラミング言語（プログラムの開発／実行環境）です。必要なソフトウェアは、Pythonの公式サイトからダウンロードすることができます。どのバージョンを使うべきかよくわからない場合には、Python 3.7を選択してください。

Pythonのプログラムは、統合開発環境（IDE：Integrated Development Environment）を使わなくても開発できます。ただ、ここではライブラリのダウンロードやデバッグの手間を省くために、IDEとしてPyCharmを使用することにします。PyCharmは、開発者にとって必須の様々なツールを提供してくれます。Pythonによるプログラム開発で最もよく使われているIDEだと言えるでしょう。JetBrainsの公式サイトからPyCharm（Community版）の最新バージョンをダウンロードしてください。

使用するライブラリ

Pythonのライブラリには、特定のアプリケーションで使用できるメソッドや関数が用意されています。本稿では、libm2k、matplotlib、NumPyを使用することにします。

libm2kの概要

Pythonを使用してADALM2000とのインターフェースを開発する場合には、libm2kライブラリをインストールします。libm2kは、Python、C#、MATLAB®、LabVIEW®とのバインディングが可能なC++のライブラリです。これをインストールすれば、以下に示す機能モジュールを利用できます。

- AnalogIn：オシロスコープまたは電圧計用のモジュールです。本稿では、主にこのモジュールの機能を利用します。
- AnalogOut：信号発生器用のモジュールです。
- Digital：ロジック・アナライザまたはパターン・ジェネレータ用のモジュールです。
- PowerSupply：定電圧ジェネレータ用のモジュールです。
- DMM：デジタル・マルチメータ用のモジュールです。

詳細については、libm2kのWikiページをご覧ください。

Libm2kのインストール

libm2kは、以下のステップに従うことでインストールすることができます。

- リリースのページにアクセスします。
  - 最新の実行可能バージョンをダウンロードします。例えば、Libm2k-0.4.0-Windows-Setup.exeを選択するといった具合です。
- ダウンロードしたファイルを実行します。「Setup」ウィンドウに「Select Additional Tasks」が表示されたら、「Install libm2k Python bindings」を必ず選択してください（図2）。
- インストール作業を終了します。以上の操作により、libm2kがPythonのデフォルトの環境にインストールされます。

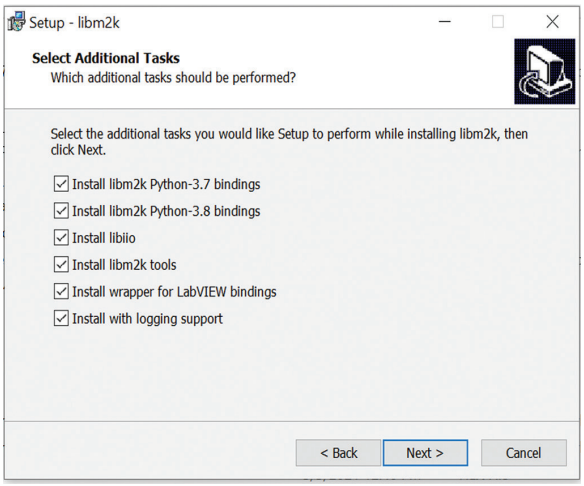

図2. libm2kのインストール用ウィンドウ

matplotlibの概要

本稿では、オシロスコープの画面を作成するためにmatplotlibライブラリを使用します。このライブラリは、Pythonで視覚化用の機能をカスタマイズしたい場合に便利なので、広く利用されています。詳細については、matplotlibのウェブサイトをご覧ください。

NumPyの概要

シンプルなオシロスコープであっても、各種の機能を実現するためには多くの数学的な計算が必要になります。NumPyライブラリは、複雑な計算を実現するためのシンプルな関数を提供してくれます。詳細については、NumPyのウェブサイトをご覧ください。

matplotlibとNumPyのインストール

matplotlibとNumPyをインストールする際には、PyCharm上で以下の操作を行います。

- 「File」→「Settings」→「Project Interpreter」を順に選択します。
- 「Settings」ウィンドウの右側にある「+」アイコンをクリックします。
- 「Available Packages」ウィンドウが表示されるので、検索ボックスを使って matplotlib と NumPy を検索します。
- インストールするバージョンを指定します（最新のバージョンを選択してください）。
- 「Install Package」ボタンをクリックします。

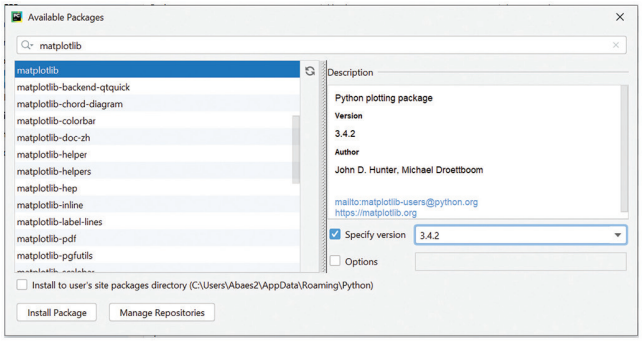

図3. ライブラリのパッケージのインストール。
PyCharm上で実行します。

ハードウェアのセットアップ

コーディングを始める前に、各種ハードウェアのセットアップを実施します。本稿では、以下に示すハードウェアを使用します。

- 信号源（可能であれば信号発生器を用意）
- ADALM2000
- プローブとクリップ

信号発生器を使用できる場合、図4、表1に示すように、プローブまたはクリップを使用してADALM2000をチャンネル1（Ch1）とチャンネル2（Ch2）に接続します。

他の信号源を使用する場合にも同様に構成します。最後に、USBポートを使ってADALM2000をPCに接続します。

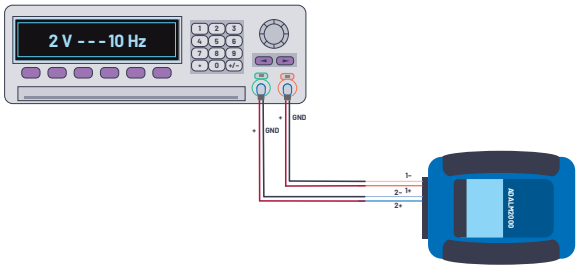

図4. 信号発生器と
ADALM2000のセットアップ

表1. 各端子の接続先

信号発生器	ADALM2000
Ch1のプラス側ワイヤ（+）	1+
Ch1のグラウンド	1–
Ch2のプラス側ワイヤ（+）	2+
Ch2のグラウンド	2–

シンプルなバーチャル・オシロスコープ

ここからは、プログラムの詳細についてコード・ブロックごとに解説していきます。また、コードによる処理の内容や、そのコードが必要な理由についての説明も加えます。その上で、基本的なコードを修正することで、個々のニーズに応じた最適な機能を追加することが可能であることを示します。

まず、バーチャル・オシロスコープの開発に使用する3つのライブラリ（libm2k、matplotlib、NumPy）をインポートします。

```
import libm2k
import matplotlib.pyplot as plt
import numpy as np
```

PCに接続された各ADALM2000は、一意的な識別子としてURI（Uniform Resource Identifier）を使用します。それにより、他のデバイスと区別されます。以下に示すコード・ブロックは、ADALM2000がPCに接続されていることを確認するためのものです。

```
# Detect ADALM2000 device connected to PC
uri = libm2k.getAllContexts()
if uri == ():
  print("No ADALM2000 found. Please replug ADALM2000 device.")
  exit(1)
else:
  print("Successfully connected to ADALM2000.")
```

PCに接続されたADALM2000が存在しない場合、上記のプログラムによって自動的に処理が終了します。

以下のコードにより、検出されたURIを使用して、ADALM2000に対する接続が実現されます。

```
# Connect to ADALM2000 with the detected uri
adalm2000_dev = libm2k.m2kOpen(uri[0])
```

複数のデバイスが接続されている場合、最初に検出されたADALM2000のURIにはuri[0]が対応することになります。

続いて、以下のコードによってA/Dコンバータ（ADC）とD/Aコンバータ（DAC）のキャリブレーションを実行します。

```
# Run the calibration for ADC and DAC
adalm2000_dev.calibrateADC()
adalm2000_dev.calibrateDAC()
```

キャリブレーションは、正確な測定値を確実に取得するために必要な重要なステップです。

続いて、サンプル・レートと期間を設定します。使用可能なサンプル・レートは、1kHz、10kHz、100kHz、1MHz、10MHz、100MHzのうちいずれかです。サンプル・レートとは、サンプリングを行う周期のことです。それによって、1秒間に取得されるサンプル（測定値を表すデジタル・データ）の数が決まります。一方、期間とは、サンプルの取得処理の対象となる時間のことです。例えば、サンプル・レートを1000、期間を3に設定すると、1秒あたり1000個のサンプルを3秒間にわたって取得することになります。その結果、計3000個のサンプルが取得されます（以下参照）。

```
# Set the sample rate and time duration
sample_rate = 1000  # Hz or sample/sec
duration = 3  # seconds (time duration of our data)
```

次に、チャンネル1の状態をイネーブルに設定します。また、同チャンネルをオシロスコープのアナログ入力として設定します（以下参照）。

```
# Enable and setup channel 1 as analog input for our oscilloscope
ocsi = adalm2000_dev.getAnalogIn()
ocsi.setSampleRate(sample_rate)
# Channel 1
ocsi.enableChannel(libm2k.ANALOG_IN_CHANNEL_1, True)
ocsi.setRange(libm2k.ANALOG_IN_CHANNEL_1, -10, 10) # range of voltage (from -10 to 10)
```

続いて、以下のコードを記述します。

```
# x-axis data
time_x = np.linspace(0, duration, (duration * sample_rate))
# y-axis data
data_y = ocsi.getSamples(duration * sample_rate)
```

このコードでは、等間隔のサンプルの数列を作成するために、NumPyのLinspace関数を使用しています。それにより、X軸（時間軸）のデータの数列が生成されます。同関数の第1引数と第2引数は、それぞれ数列の開始値と終了値を表しています。第3引数は、開始値と終了値の間に生成されるサンプルの数を表します。

この例では、開始値として0、終了値としては、上で期間として設定した値である3を指定しています。サンプルの数については、duration（期間）とsample_rate（サンプル・レート）の積を求める式を指定しています。それにより、必要なすべてのサンプルの数である3000が設定されることになります。また、この3000個のサンプルは、0から3までの間に均等に存在することになります。それらを、time_xに数列として格納します。

data_yには、ADALM2000を使用して収集したサンプルが格納されます。チャンネル1のサンプルはdata_y[0]、チャンネル2のサンプルはdata_y[1]に格納されることになります。信号の正確な周波数を表示するためには、time_xに格納したのと同じ数のサンプルを使用する必要があります。

次に、操作の対象となる図を作成します。それにはplt.subplots関数を使用します。同関数は、figureオブジェクト（figに格納）とaxesオブジェクト（axに格納）を返します。これらを使用してプロット全体をカスタマイズします。例えば、波形の視覚化に役立つグリッドを追加することができます。また、軸のラベルとY軸の上限値／下限値を追加することも可能です。ここでは、プロットに関する詳細を以下のように設定します。

```
# Create the figure that we will manipulate
fig, ax = plt.subplots()
plt.plot(time_x, data_y[0])
ax.grid()
ax.set_xlabel("Time (s)")
ax.set_ylim([-4, 4])
ax.set_ylabel("Voltage")
```

グラフの表示は、以下のコードによって実現します。

```
plt.show()
```

また、プログラムの最後の処理として、以下のコードによりコンテキストを破棄します。

```
libm2k.contextClose(adalm2000_dev)
```

このプログラムを実行すると、図5に示すような結果が表示されるはずです。

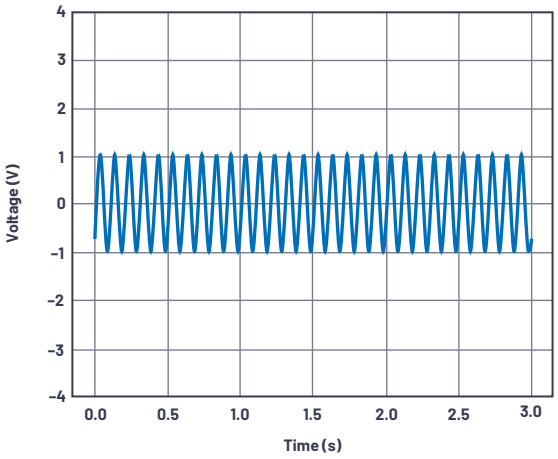

図5. プログラムの実行結果（その1）。1つのチャンネルから出力された正弦波が表示されています。これは1つの信号発生器から出力された10Hz、2V p-pの正弦波に相当します。

2チャンネルのバーチャル・オシロスコープ

続いては、上で作成したコードに対して更にコードを追加し、2チャンネルのバーチャル・オシロスコープを構築します。チャンネルをもう1つ追加するために、まずocsi.enableChannelとocsi.setRangeの行をコピーします。そして、第1引数をlibm2k.ANALOG_IN_CHANNEL_1からlibm2k.ANALOG_IN_CHANNEL_2に変更します（以下参照）。

```
# Enable and setup channel 1 and 2 as analog input for our oscilloscope
ocsi = adalm2000_dev.getAnalogIn()
ocsi.setSampleRate(sample_rate)
# Channel 1
ocsi.enableChannel(libm2k.ANALOG_IN_CHANNEL_1, True)
ocsi.setRange(libm2k.ANALOG_IN_CHANNEL_1, -10, 10) # range of voltage (from -10 to 10)
# Channel 2
ocsi.enableChannel(libm2k.ANALOG_IN_CHANNEL_2, True)
ocsi.setRange(libm2k.ANALOG_IN_CHANNEL_2, -10, 10) # range of voltage (from -10 to 10)
```

これにより、チャンネル2のプロットを追加した状態でグラフが生成されます。つまり、配列data_y[1]に格納されたデータがプロットされるということです。また、2つのプロットの色をカスタマイズし、それぞれを区別しやすくすることも可能です。以下のコードでは、チャンネル1にlightcoral（赤色）、チャンネル2にsteelblue（青色）を設定しています。

```
# Create the figure that we will manipulate
fig, ax = plt.subplots()
plt.plot(time_x, data_y[0], color='lightcoral') # channel 1 plot
plt.plot(time_x, data_y[1], color='steelblue') # channel 2 plot
ax.grid()
ax.set_xlabel("Time (s)")
ax.set_ylim([-4, 4])
ax.set_ylabel("Voltage")
```

実際にプログラムを実行すると、図6のような結果が表示されるはずです。

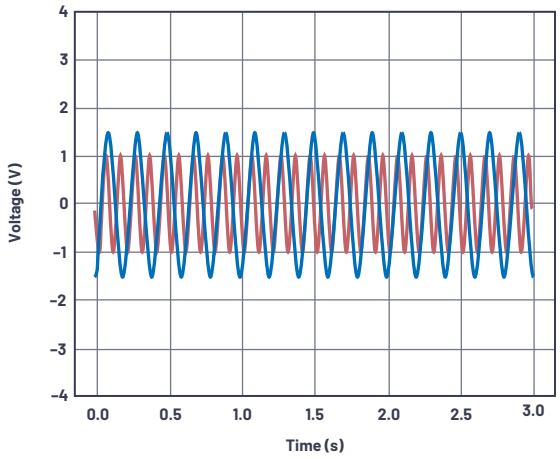

図6. プログラムの実行結果（その2）。2つのチャンネルから出力された正弦波が表示されています。それぞれ、チャンネル1の信号発生器から出力された10Hz、2V p-pの正弦波、チャンネル2の信号発生器から出力された5Hz、3V p-pの正弦波に相当します。

バーチャル・オシロスコープに対する機能の追加

続いては、追加の機能を実装することで、よりインタラクティブなバーチャル・オシロスコープを実現します。matplotlibには、様々なウィジェットが用意されています。ここでは上で作成したコードに対し、テキスト・ラベルとスライダのウィジェットを追加してみます。

まず、以下のようなコードにより、matplotlibのスライダをインポートします。

```
import libm2k
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.widgets import Slider
```

また、次のコードによって、時間とデータの配列をNumPyの配列に変換します。この配列は、後ほど示すコードで使用します。

```
# x-axis data
time_x = np.linspace(0, duration, (duration * sample_rate))
# y-axis data
data_y = ocsi.getSamples(duration * sample_rate)

# Convert to numpy arrays
data_y_np1 = np.array(data_y[0]) # data from ch1
data_y_np2 = np.array(data_y[1]) # data from ch2
time_x_np = np.array(time_x) # time data for x axis
```

取得したデータを使って計算を行い、それぞれの波形の特性を表す値を抽出してみましょう。以下に示すコードでは、取得した両チャンネルのデータを基にV_{pp}、V_{ave}、V_{rms}を算出しています。

```
# Compute for Vpp, Vave, and Vrms
v_pp_1 = abs(np.min(data_y_np1)) + abs(np.max(data_y_np1))
v_ave_1 = v_pp_1 / np.pi
v_rms_1 = v_pp_1 / (2 * np.sqrt(2))

v_pp_2 = abs(np.min(data_y_np2)) + abs(np.max(data_y_np2))
v_ave_2 = v_pp_2 / np.pi
v_rms_2 = v_pp_2 / (2 * np.sqrt(2))
```

このコードでは、V_{pp}を計算するために、data_yというNumPyの配列に格納されたデータの最大値と最小値の絶対値を加算しています。また、V_{ave}の計算は、V_{pp}の値をπで割るだけです。V_{rms}は、V_{pp}を$2\times\sqrt{2}$で割ることによって算出します。

以下のコードは、前のセクションで示したものと同様です。

```
# Create the figure and the waveforms that we will manipulate
fig, ax = plt.subplots()
wave1, = plt.plot(time_x_np, data_y_np1, color='lightcoral') # channel 1 plot
wave2, = plt.plot(time_x_np, data_y_np2, color='steelblue') # channel 2 plot
ax.grid()
ax.set_xlabel("Time (s)")
ax.set_ylim([-4, 4])
ax.set_ylabel("Voltage")
```

前のコードとの違いは、プロットの処理において、元の配列の代わりにNumPyの配列を使用していることです。また、波形のオブジェクトを作成している点も異なります。これらのオブジェクトは後ほど使用します。

続いて、V_{pp}、V_{ave}、V_{rms}の算出結果をグラフ中に表示する処理を追加します。それには、matplotlibのテキスト・ラベル・ウィジェットを使用します（以下参照）。

```
# Make a text label at the top of graph to show the computed Vpp, Vave and Vrms
label_ch1 = "Channel 1 : Vpp = {:.2f}  Vave = {:.2f}  Vrms = {:.2f}".format(v_pp_1, v_ave_1, v_rms_1)
label_ch2 = "\nChannel 2 : Vpp = {:.2f}  Vave = {:.2f}  Vrms = {:.2f}".format(v_pp_2, v_ave_2, v_rms_2)
fin_label = label_ch1 + label_ch2
plt.text(0.2, 3, fin_label, style='italic', bbox={'facecolor': 'paleturquoise', 'alpha': 0.5, 'pad': 5})
```

このコードでは、文字列のラベルとしてlabel_ch1、label_ch2を作成し、2つの文字列を連結することで最終的なラベルであるfin_labelを作成しています。テキスト・ラベルの作成は、plt.textによって行います。第1引数と第2引数である0.2と3は、テキストを配置する位置（X座標とY座標）を表します。第3引数は、表示する文字列です。第4引数と第5引数は、それぞれテキストとボックスのスタイルを表します。

次に、以下のコードによってオフセット・スライダを作成します。

```
# Adjust plot position so we can place the slider
plt.subplots_adjust(left=0.2)

# Make a vertically oriented slider to control the offset
ax_offset = plt.axes([0.05, 0.2, 0.0225, 0.63], facecolor='lemonchiffon')
offset_slider = Slider(ax=ax_offset, label="Offset", valmin=-2, valmax=2,
                       valinit=0, orientation="vertical")
```

このスライダの目的は、波形の基準レベルを調整することです。メインのプロットの左側を調整し、スライダを配置するためのスペースを確保します。plt.axesによって、スライダの大きさ、位置、フェイス・カラーを定義しています。また、Slider関数を使用して、特定のプロパティを備えたオフセット・スライダのオブジェクトを作成しています。

続いて、以下のコードを記述します。

```
# The function to be called anytime a slider's value changes
def update_offset(val):
    wave1.set_ydata(data_y_np1 + offset_slider.val)
    wave2.set_ydata(data_y_np2 + offset_slider.val)
    fig.canvas.draw_idle()

# Register the update_offset function with each slider
offset_slider.on_changed(update_offset)
```

このコードでは、update_offsetという関数を定義しています。また、同関数をoffset_sliderオブジェクトに登録しています。同関数は、スライダの値が変更されるたびに波形にオフセットを加えます。

作成したコードを実行すると、図7のような結果が表示されるはずです。

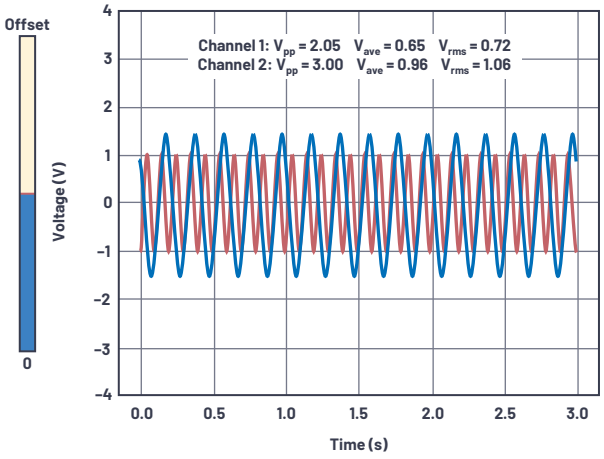

図7. プログラムの実行結果（その3）。2つのチャンネルの正弦波出力だけでなく、V_{pp}、V_{ave}、V_{rms}の算出結果、オフセット・スライダが表示されています。

ここで、スライダを使用してオフセットの値を変更してみてください。波形がリアルタイムに上下することがわかるはずです（図8）。

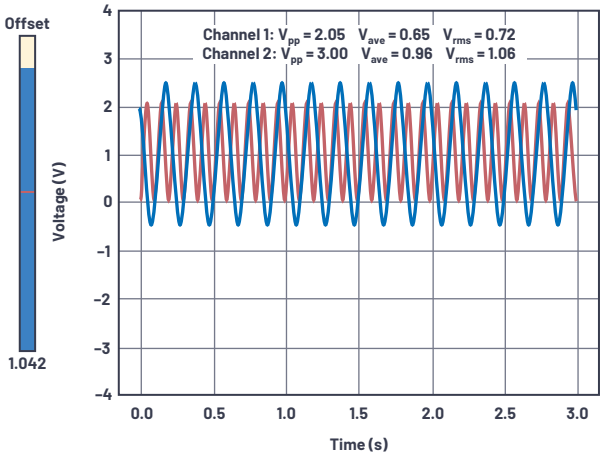

図8. オフセット・スライダ（左）を調整した結果。両正弦波のオフセットの値が変更され、波形が上側に移動しています。

まとめ

本稿では、まずバーチャル・エレクトロニクス・ラボを作成することによって得られる利便性について説明しました。その上で、ADALM2000とPythonを使用してバーチャル・オシロスコープを開発する方法を紹介しました。ソフトウェアに関する要件やハードウェアのセットアップ、具体的なコーディングの方法を理解していただけたはずです。

参考資料

「ADALM2000 Overview（ADALM2000の概要）」Analog Devices、2021年3月

Chandan Bhunia、Saikat Giri、Samrat Kar、Sudarshan Haldar「A Low-Cost PC-Based Virtual Oscilloscope（PCをベースとする低コストのバーチャル・オシロスコープ）」IEEE Transactions on Education、Vol. 47、No. 2、2004年5月

「Limb2k Examples: analog.py.（limb2kのサンプル：analog.py.）」Analog Devices

Amelia Tegen、Jeremy Wright「Oscilloscopes: The Digital Alternative: The Digital Scope's Capability in Measurement, Transient Capture, and Data Storage Is a Significant Improvement Over its Analog Counterpart（オシロスコープ：デジタルによる代替品：デジタル・オシロスコープの計測／過渡現象の捕捉／データ保存の機能はアナログ・オシロスコープをはるかに凌駕）」IEEE Potentials、Vol. 2、1983年

「What Is libm2k?（libm2kとは何か？）」Analog Devices Wiki、2021年4月

著者について

Arnie Mae Baes（arniemae.baes@analog.com）は、アナログ・デバイセズのソフトウェア・システム・エンジニアです。2019年にファームウェア・エンジニアとして入社しました。同年は、GUIとファームウェアの開発を中心に担当。2020年12月にコンスーマ・ソフトウェア・エンジニアリング・グループに加わりました。現在はファームウェアのテスト開発に注力しています。バタンガス州立大学で電子工学の学士号を取得しています。

Christian Jason Garcia（christian.garcia@analog.com）は、アナログ・デバイセズ（フィリピン　ゼネラル・トリアス）のファームウェア検証エンジニアです。2018年11月に入社しました。eモビリティ・グループで、SmartMeshネットワーク向けソフトウェアのテストとシステム検証を担当。聖トマス大学で電子／通信工学の学士号を取得しています。

EngineerZone®
オンライン・サポート・コミュニティ

アナログ・デバイセズのオンライン・サポート・コミュニティに参加すれば、各種の分野を専門とする技術者との連携を図ることができます。難易度の高い設計上の問題について問い合わせを行ったり、FAQを参照したり、ディスカッションに参加したりすることが可能です。

ADI EngineerZone™
SUPPORT COMMUNITY

Visit *ez.analog.com*

*英語版技術記事はこちらよりご覧いただけます。

ANALOG DEVICES
想像を超える可能性を
AHEAD OF WHAT'S POSSIBLE™

アナログ・デバイセズ株式会社

VISIT ANALOG.COM/JP

お住いの地域の本社、販売代理店などの情報は、analog.com/jp/contact をご覧ください。

オンラインサポートコミュニティEngineerZoneでは、アナログ・デバイセズのエキスパートへの質問、FAQの閲覧ができます。

TA23166-3/22（A）