22장. 오실로스코프 PC로 제어하기(Python)

1. 학습개요

목표

1. Python은 PC에 설치가 필요한 개발 환경임을 이해하고, Jupyter Notebook 기반 실습 환경을 구성한다.
2. PyVISA를 이용해 RIGOL 오실로스코프와 PC를 연결하고 VISA 리소스 검색 – 장비 연결 – SCPI 전송 흐름을 익힌다.
3. 수신한 바이너리(TMC) 데이터를 헤더 처리 – 배열 변환 – 전압/시간 축 복원 – 파형 시각화까지 완주할 수 있다.

※ 교육을 진행하기 위해서는 Python과 pyVISA가 별도로 설치되어 있어야 합니다. Python 개발 환경 중 해당 교재에서는 Jupyter Notebook 기반의 실습 환경으로 구성되었음을 안내합니다.

2. 이론

PC 제어는 사람이 화면에서 하던 측정/저장/분석을 반복 가능하게 자동화하는 방법입니다. 오실로스코프는 SCPI(Standard Command for Programmable Instruments) 명령으로 제어할 수 있고, Python에서는 보통 VISA 드라이버 + PyVISA 조합으로 장비와 통신합니다.

2.1 Python 개발 환경

그림 22-1. Python

- Python은 PC에 별도로 설치해야 하는 프로그래밍 언어입니다.
- 실습에서는 Jupyter Notebook(브라우저 기반 개발 환경)을 사용하면, 코드를 셀 단위로 실행하며 단계별 검증이 가능합니다.

2.2 PC – 오실로스코프 통신 구조 (VISA + SCPI)

- VISA: USB/LAN/GPIB 같은 물리 연결을 추상화해 장비를 찾아 연결하는 표준 인터페이스
- PyVISA: Python에서 VISA를 쓰기 위한 라이브러리
- SCPI: 오실로스코프가 이해하는 제어 명령어(예: `:RUN`, `:STOP`, `:WAV:DATA?`)
- 실제 Python을 사용해서 코드를 짤 때, rm.list_resources()에서 장비가 보이지 않으면, 케이블/드라이버/인터페이스 등부터 점검해야 합니다.

2.3 파형 데이터 수진 포맷(TMC)과 헤더 처리에 대한 이해

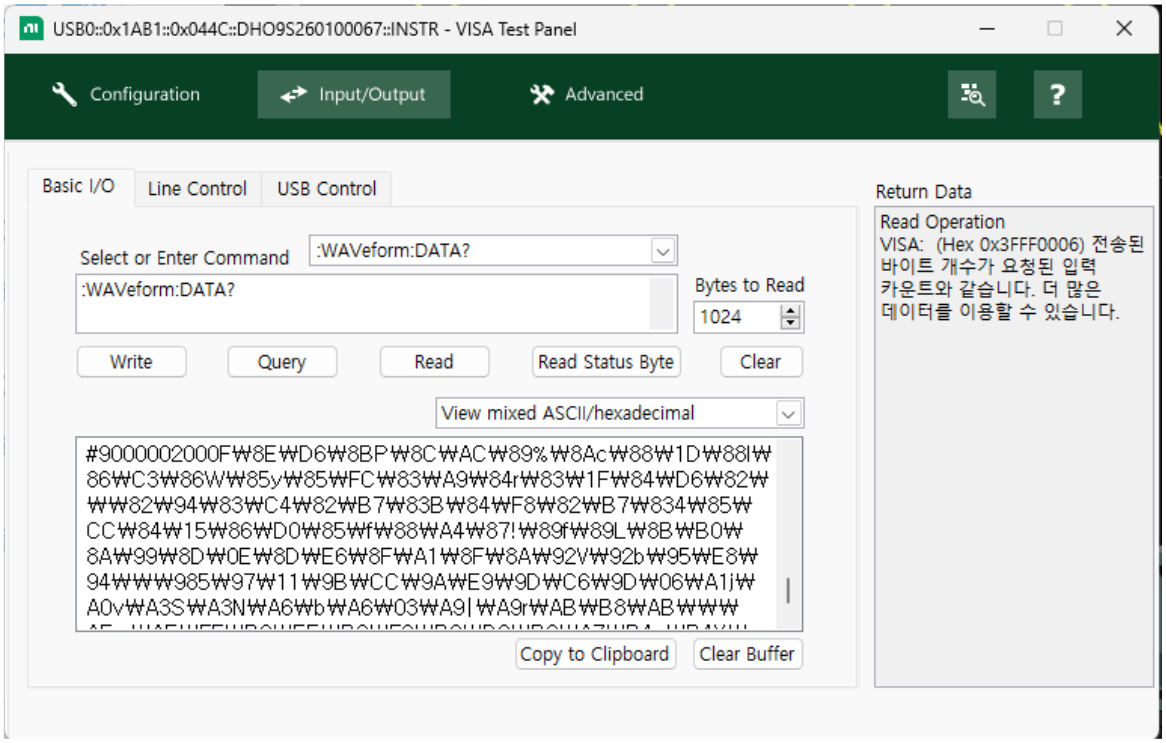

그림 22-2. :WAV:DATA? 의 결과 데이터 신호 예시

오실로스코프에서 `:WAV:DATA?` 명령으로 파형 데이터를 요청하면, VISA 읽기 함수는 일반적으로 TMC 데이터 블록 헤더 + 실제 파형 데이터 + 터미네이터 형태로 데이터를 반환합니다.

따라서 수신된 데이터를 그대로 그래프로 그릴 수는 없고, 먼저 헤더 구조를 해석하여 실제 파형 데이터 구간만 분리하는 과정이 필요합니다.

TMC 헤더는 다음과 같은 형식으로 구성됩니다.

- `#NXXXXXX`
- `#` : TMC 데이터 블록의 시작을 알리는 식별자
- `N` : 뒤따르는 데이터 길이 문자열의 자릿수
- `XXXXXX` : 실제 파형 데이터 길이를 ASCII 문자열로 표현한 값

예를 들어 다음과 같은 데이터가 수신되었다고 가정해 보겠습니다.

`#9000001000XXXX…`

이 경우의 의미는 다음과 같습니다.

`#` : 이 데이터가 TMC 블록 형식임을 의미합니다.

`9` : 뒤이어 나오는 데이터 길이 정보가 9자리 ASCII 문자열로 표현되어 있다는 뜻입니다.

`000001000` : 실제 파형 데이터 길이가 1000바이트임을 의미합니다.

즉, 이 구조에서는 첫 번째 바이트 #, 두 번째 바이트 9, 그 다음 9바이트 000001000까지가 헤더이며, 그 뒤부터가 실제 파형 데이터입니다. 따라서 전체 수신 데이터에서 실제 파형을 추출하려면 다음 순서로 처리해야 합니다.

① 첫 바이트가 #인지 확인
- #가 아니면 예상한 TMC 형식이 아닐 수 있으므로 예외 처리가 필요합니다.

② 두 번째 바이트를 읽어 길이 정보의 자릿수 N을 파악
- 예를 들어 9이면, 그 다음 9바이트가 실제 데이터 길이를 나타냅니다.

③ N자리 ASCII 문자열을 정수로 변환하여 실제 파형 데이터 길이를 계산
- 예: 000001000 → 1000

④ 헤더 길이를 계산
- 전체 헤더 길이 = # 1바이트 + N 1바이트 + 길이 문자열 N바이트
- 예: 1 + 1 + 9 = 11바이트

⑤ 헤더 이후부터 실제 데이터 길이만큼 잘라서 파형 데이터만 추출
- 이 과정을 통해 헤더는 무시하고 순수 파형 데이터만 남길 수 있습니다.

⑥ 끝에 포함될 수 있는 터미네이터를 제거
- VISA 통신에서는 수신 데이터 맨 끝에 줄바꿈 문자나 종료 문자(₩n 등)가 붙을 수 있으므로, 실제 데이터 길이만큼만 정확히 잘라 사용하는 것이 중요합니다.

⑦ 추출된 파형 데이터를 바이트 배열로 변환
- 이후 np.frombuffer() 등을 사용하면 NumPy 배열로 쉽게 변환할 수 있으며, 전압 변환식에 적용하거나 그래프로 표시할 수 있습니다.

- read_raw()로 받은 전체 데이터 길이와 실제 파형 데이터 길이(data_len)는 서로 다를 수 있습니다.
- 이는 수신 데이터에 TMC 헤더와 터미네이터가 포함되어 있기 때문입니다.
- 헤더를 제거하지 않고 바로 np.frombuffer()로 변환하면, 배열의 앞부분에 헤더 값이 데이터로 포함되어 파형이 깨져 보일 수 있습니다.
- 따라서 터미네이터를 별도로 제거하기보다는 TMC 헤더에서 확인한 실제 데이터 길이(data_len)만큼 정확히 슬라이싱하여 사용하는 방식이 가장 안전합니다.
- 또한 장비 설정에 따라 파형 데이터 포맷이 달라질 수 있기 때문에, :WAV:FORM BYTE로 받아오는 형태를 BYTE로 설정해 놓고 받아오는 것이 좋습니다.
- 만약 WORD 형식을 사용하는 경우에는 데이터 타입이 uint16 등으로 달라질 수 있으며, ASCII 형식에서는 데이터가 문자열 형태로 전달되기 때문에 파싱 방식이 달라집니다. 따라서 Python으로 파형 데이터를 처리하기 전에 :WAV:FORM? 명령어를 통해 현재 파형 데이터 포맷을 확인하는 것이 좋습니다.

2.4 Raw 코드값을 “실제 전압/시간축”으로 복원하기

```
#전압 변환 파라미터 조회
yinc = float(scope.query(':WAV:YINC?')) #Y축 증분값 (전압 단위)
yref = float(scope.query(':WAV:YREF?')) #Y축 기준 레벨
yor = float(scope.query(':WAV:YOR?')) #Y축 오프셋
#전압값으로 변환: (rawdata - yor - yref) x yinc
voltages = (data-yor-yref)*yinc
print(f"변환된 전압값 샘플: {voltages[:10]}")
변환된 전압값 샘플: [1.348293 1.655499 1.928571 2.150442 2.321112 2.457648 2.577117 2.662452
 2.73072  2.781921]
#시간축 계산을 위한 파라미터 조회
xinc = float(scope.query(':WAV:XINC?')) #X축 증분값 (시간 단위)
xor = float(scope.query(':WAV:XOR?')) #X축 시작 지치
xref = float(scope.query(':WAV:XREF?')) #X축 기준값
#시간축 값으로 변환
times = np.arange(len(voltages))*xinc+xor
```

그림 22-3. Raw 코드값의 전압/시간 값을 복원하는 코드

오실로스코프는 화면에 보이는 전압/시간 스케일 정보를 SCPI로 제공합니다.

- 전압 변환 계수
    - :WAV:YINC? (전압 1 LSB당 증분)
    - :WAV:YREF? (기준 레벨)
    - :WAV:YOR? (오프셋)
- 전압 변환 예시(교육용 공식):
    - V = (data - YOR - YREF) × YINC
- 시간축 계수
    - :WAV:XINC? (시간 증분)
    - :WAV:XOR? (시간 시작 오프셋)
    - :WAV:XREF? (시간 기준)
- 시간축 생성 예시:
    - t = np.arange(N) * XINC + XOR

주의: 모델/펌웨어에 따라 계수 정의가 약간 다를 수 있으니, 결과가 이상하면 YREF/YOR를 포함/제외한 변환식을 비교해 확인합니다.

3. 기본 실습

준비물: PC(Windows 권장), Python (Jupyter Notebook, PyVISA, NumPy, Matplotlib 준비), 오실로스코프, USB 혹은 LAN 케이블

3.1 실습 1 – 개발 환경 구성 (Python + 라이브러리 설치)

① Python(및 Jupyter Notebook)을 설치합니다.

② 터미널(명령 프롬프트)에서 아래 라이브러리를 설치합니다.

```
[1]: pip install pyvisa matplotlib numpy
```

③ 설치 후에는 드라이버/경로 인식 문제를 줄이기 위해, 개발환경을 재시작합니다. (pyVISA는 PC에 VISA 백엔드(예: NI-VISA Driver 등)가 필요할 수 있습니다. 장비가 검색되지 않으면 VISA 드라이버 설치 여부를 먼저 확인하세요.

3.2 실습 2 – 장비 검색 및 연결 (리소스 확인)

① Jupyter Notebook을 열고 새 파일을 만든 뒤 (예: DHO_Waveform_DEMO)로 저장합니다.

② 아래 코드를 이용해 연결된 장비 목록을 확인합니다.

```
[1]: #pyVISA numpy, matplotlib 모듈 불러기기

[2]: import pyvisa

[3]: import numpy as np

[*]: import matplotlib.pyplot as plt

     Matplotlib is building the font cache; this may take a moment.

[*]: import time

[*]: #Jupyter 내에서 그래프를 셀 안에 출력

[*]: %matplotlib inline

[*]: # VISA 리소스 매니저 생성 (연결된 장비 검색 가능)

[*]: rm = pyvisa.ResourceManager()

[*]: print(rm.list_resources()) #연결된 장비 목록 출력
```

③ print에서 나온 오실로스코프 리소스 문자열을 확인하고, 아래와 같이 open_resource()로 연결합니다.

```
[10]: print(rm.list_resources()) #연결된 장비 목록 출력
      ('USB0::0x1AB1::0x044C::DHO9S260100067::INSTR', 'TCPIP0::169.254.39.10::inst0::INSTR', 'ASRL10::INSTR')

[ ]: #오실로스코프에 연결 (VISA 리소스 참고하여 입력)

[ ]: scope = rm.open_resource('USB0::0x1AB1::0x044C::DHO9S260100067::INSTR')
```

3.3 실습 3 – SCPI로 파형 데이터 요청 및 수신 (RAW)

아래 흐름으로 진행합니다.

- 파형 소스: CHAN1
- 읽기 모드: MAX (전체 화면 범위 데이터)
- 포맷: BYTE
- 안정적 캡처를 위해 :RUN 후 잠깐 대기하고 :STOP으로 정지한 뒤 읽기

3.4 실습 4 – TMC 헤더 제거, NumPy 배열 변환, 전압/시간축 복원, 그래프 출력

아래 Python 코드를 한 번에 실행 가능한 기준 예제로 작성하여 진행합니다.

```
</> Python
import pyvisa
import numpy as np
import matplotlib.pyplot as plt
import time

%matplotlib inline

rm = pyvisa.ResourceManager()
print(rm.list_resources())

# 장비 리소스 문자열은 PC/연결 방식에 따라 달라질 수 있습니다.
scope = rm.open_resource('USB0::0x1AB1::0x044D::DHO8A253200440::INSTR')
scope.timeout = 10000   # ms

# 파형 설정 및 정지 캡처
scope.write(':WAV:SOUR CHAN1')
scope.write(':WAV:MODE MAX')
scope.write(':WAV:FORM BYTE')
scope.write(':RUN')
time.sleep(0.2)
scope.write(':STOP')

# 데이터 수신
scope.write(':WAV:DATA?')
raw_data = scope.read_raw()
print(f"수신된 전체 데이터 길이: {len(raw_data)}")

# TMC 헤더 처리
if raw_data[0:1] != b'#':
    raise ValueError("Invalid TMC header")

header_len_digits = int(raw_data[1:2])   # N
data_len = int(raw_data[2:2 + header_len_digits])   # 데이터 길이
```

```
data_start = 2 + header_len_digits
waveform_bytes = raw_data[data_start : data_start + data_len]

# 바이트 → 배열
data = np.frombuffer(waveform_bytes, dtype=np.uint8)
print(f"받은 파형 포인트 수: {len(data)}")

# 전압 변환 계수
yinc = float(scope.query(':WAV:YINC?'))
yref = float(scope.query(':WAV:YREF?'))
yor  = float(scope.query(':WAV:YOR?'))

voltages = (data - yor - yref) * yinc
print(f"변환된 전압 값 샘플: {voltages[:10]}")

# 시간축 계수
xinc = float(scope.query(':WAV:XINC?'))
xorg = float(scope.query(':WAV:XOR?'))
xref = float(scope.query(':WAV:XREF?'))  # 필요 시 참고용

times = np.arange(len(voltages)) * xinc + xorg

# 그래프 출력
plt.figure(figsize=(12, 5))
plt.plot(times, voltages, label='CH1 Waveform')
plt.title("RIGOL DHO - CH1 Waveform (Python + PyVISA)")
plt.xlabel("Time (s)")
plt.ylabel("Voltage (V)")
plt.grid(True)
plt.tight_layout()
plt.show()

# 연결 종료
scope.close()
```

4. 결과 확인

- `rm.list_resource()` 결과를 캡쳐하여 제출합니다.
- `수신된 전체 데이터 길이`, `받은 파형 포인트 수` 출력 로그를 제출합니다.
- 변환된 전압 샘플 (예: `voltages[:10]`) 출력 로그를 제출합니다.
- Matplotlib로 출력된 파형 그래프 캡처, 오실로스코프 화면 캡처한 후 두가지 화면 비교하여 제출합니다.

5. 확인 문제

1. Python으로 오실로스코프를 제어할 때, VISA와 SCPI는 각각 어떤 역할을 하나요?

   _____

2. `:WAV:DATA?` 명령어로 받은 데이터에서 TMC 헤더를 제거해야 하는 이유는 무엇인가요?

   _____

3. Raw 바이트 데이터를 실제 전압으로 바꾸기 위해 사용하는 파라미터 3가지를 쓰고, 전압 변환식을 적어보세요.

   _____

6. 핵심요약

- Python은 별도 설치가 필요한 프로그래밍 언어이며, Jupyter Notebook을 쓰면 단계별 검증이 쉽다.
- PyVISA로 장비를 검색/연결하고, SCPI 명령으로 `RUN/STOP`, `WAV:DATA?` 등을 제어한다
- `:WAV:DATA?` 응답은 TMC 바이너리 블록인 경우가 많아 헤더 파싱 후 순수 데이터만 추출해야 한다.
- `YINC/YREF/YOR`, `XINC/XOR,XREF` 값을 읽어 전압/시간축을 복원하면 오실로스코프 화면과 같은 파형을 PC에서 재현할 수 있다.

7. 참고 및 부록

용어집:

- SCPI: 계측기 제어용 표준 명령어 체계
- VISA: 계측기 통신 표준 인터페이스
- TMC Header: 데이터 길이 정보를 포함한 전송 헤더
- PyVISA: Python에서 VISA를 사용하기 위한 라이브러리