자동화 계측 제어를 위한 실용 가이드: **PyVISA**를 이용한 **Keysight** 파워 서플라이 및 오실로스코프 제어

1장**:** 테스트 환경 준비

자동화된 테스트 시스템을 구축하는 첫 단계는 소프트웨어와 하드웨어 환경을 정확하게 구성하고 검증하는 것입니다. 이 과정은 계측기 자동화에서 가장 빈번하게 문제가 발생하는 지점이므로, 견고한 기반을 다지는 것이 매우 중요합니다. 본 장에서는 VISA 프레임워크의 개념부터 시작하여 필요한 소프트웨어 설치, 그리고 계측기 연결 및 주소 확인에 이르는 전 과정을 상세히 다룹니다.

1.1: VISA 프레임워크와 **PyVISA**의 역할

계측기 자동화를 이해하기 위해서는 먼저 통신 아키텍처(VISA), 계측기 언어(SCPI), 그리고 프로그래밍 인터페이스(PyVISA)라는 세 가지 계층화된 모델을 명확히 구분해야 합니다. 이 개념적 분리는 문제 발생 시 원인을 체계적으로 진단하는 데 강력한 사고의 틀을 제공합니다.

- **VISA (Virtual Instrument Software Architecture):** VISA는 테스트 및 측정 장비를 위한 표준화된 통신 계층입니다.[1] 이는 USB, LAN (TCPIP), GPIB와 같은 물리적 인터페이스의 복잡성을 추상화하여, 어떤 연결 방식을 사용하든 일관된 방법으로 계측기와 통신할 수 있도록 합니다. VISA는 메시지를 어떻게 전달할지(전송 프로토콜)에 대한 규칙을 정의하지만, 메시지의 내용(명령어) 자체에는 관여하지 않습니다. 이는 마치 국제 우편 시스템과 같아서, 편지의 내용과 상관없이 정해진 주소로 안정적으로 배달하는 역할을 수행합니다.
- **SCPI (Standard Commands for Programmable Instruments):** SCPI는 대부분의 최신 계측기에서 사용하는 표준 명령어 언어입니다.[3] 이 언어는 계측기의 기능을 제어하고 측정 결과를 요청하기 위한 어휘와 문법을 정의합니다. 예를 들어, $*IDN?$은 계측기의 식별 정보를 요청하는 쿼리이며, $:SOURce:VOLTage 5.0$은 전압을 5.0V로 설정하는 명령어입니다. SCPI는 VISA라는 우편 시스템을 통해 전달되는 '편지의

내용'에 해당합니다.

- **PyVISA:** PyVISA는 VISA 라이브러리를 파이썬에서 쉽게 사용할 수 있도록 감싼(wrapping) 고수준 파이썬 패키지입니다.[1] 이는 객체 지향적인 인터페이스를 제공하여 개발자가 저수준의 통신 프로토콜 대신 테스트 로직 자체에 집중할 수 있게 해줍니다. 즉, PyVISA는 파이썬 스크립트가 VISA라는 '우편 서비스'에 SCPI로 작성된 '편지'를 보내고 답장을 받을 수 있도록 해주는 편리한 창구 역할을 합니다.

이러한 계층적 구조를 이해하면, 자동화 스크립트가 작동하지 않을 때 문제의 원인이 연결 자체에 있는지(VISA 백엔드 문제), 명령어 문법에 있는지(SCPI 구문 오류), 아니면 스크립트 로직에 있는지(PyVISA 사용법 문제)를 체계적으로 파악할 수 있습니다.

1.2: 소프트웨어 설치 및 구성

자동화 환경을 구축하기 위해 다음 세 가지 핵심 소프트웨어를 설치해야 합니다.

- 파이썬**(Python)** 설치**:** 최신 버전의 파이썬(3.6 이상 권장)을 공식 웹사이트에서 다운로드하여 설치합니다.[6] 설치 과정에서 'Add Python to PATH' 또는 'PATH에 Python 추가' 옵션을 반드시 선택해야 합니다. 이 옵션을 활성화하면 명령 프롬프트나 터미널에서 pip과 같은 파이썬 패키지 관리자를 쉽게 사용할 수 있습니다.
- **PyVISA** 설치**:** 파이썬 설치가 완료되면, 명령 프롬프트(cmd)나 PowerShell을 관리자 권한으로 실행하고 다음 명령어를 입력하여 PyVISA 패키지를 설치합니다.
  ```
  Bash
  pip install pyvisa
  ```
  이 명령어는 파이썬 패키지 인덱스(PyPI)에서 최신 버전의 PyVISA를 다운로드하여 자동으로 설치합니다.[6]
- **VISA** 백엔드**(Backend)** 설치**:** PyVISA는 VISA 라이브러리에 대한 '프론트엔드' 역할을 하므로, 시스템에 실제 통신을 담당할 '백엔드' 라이브러리가 설치되어 있어야 합니다.[8] Keysight 계측기를 제어하는 경우,
  Keysight IO Libraries Suite를 설치하는 것이 가장 효율적이고 안정적인 방법입니다. 이 스위트에는 Keysight VISA 라이브러리뿐만 아니라, 이후 과정에서 필수적으로 사용될 Keysight Connection Expert 유틸리티가 포함되어 있습니다.[10] Keysight 공식 웹사이트에서 무료로 다운로드할 수 있습니다.

여기서 매우 중요한 기술적 제약 사항이 있습니다. 설치된 파이썬 인터프리터의 비트(bitness)와 VISA 라이브러리의 비트가 반드시 일치해야 합니다. 예를 들어, 64비트 파이썬은 32비트 VISA 라이브러리(DLL)를 로드할 수 없으며, 그 반대도 마찬가지입니다.[8] 비트 불일치는 스크립트 실행 시 혼란스러운 오류를 발생시키는 주된 원인이므로, 64비트 파이썬을 사용한다면 반드시 64비트 버전의 Keysight IO Libraries Suite를 설치해야 합니다. 이 점을

사전에 확인함으로써 수많은 잠재적 문제 해결 시간을 절약할 수 있습니다.

1.3: Keysight Connection Expert를 이용한 계측기 연결 및 주소 식별

소프트웨어 설치가 완료되면, 이제 물리적으로 계측기를 PC에 연결하고 통신 주소를 확인할 차례입니다. Keysight Connection Expert는 이 과정을 매우 간단하게 만들어주는 강력한 도구입니다.

1. **Connection Expert** 실행**:** Windows 시작 메뉴나 작업 표시줄의 IO 아이콘을 통해 Keysight Connection Expert를 실행합니다.[10] 프로그램이 실행되면 연결된 인터페이스(USB, LAN 등)와 자동으로 탐색된 계측기 목록이 표시됩니다.[13] 대부분의 최신 USB 및 로컬 네트워크상의 LAN 계측기는 별도의 설정 없이 자동으로 인식됩니다.[15]
2. **VISA** 리소스 문자열**(**주소**)** 찾기**:** 목록에서 제어하려는 계측기를 선택하면, 오른쪽 패널에 해당 계측기의 상세 정보가 나타납니다. 여기서 가장 중요한 정보는 'VISA Address' 또는 'VISA 리소스 문자열'입니다. 이 문자열은 PyVISA 스크립트에서 계측기를 고유하게 식별하는 데 사용되는 주소입니다.
   - **USB** 연결 예시**:** $USB0::0x2A8D::0x1301::MY57000123::INSTR$
   - LAN (TCPIP) 연결 예시: $TCPIP0::192.168.1.10::inst0::INSTR$
     이 주소는 인터페이스 유형, 제조업체 ID, 모델 ID, 시리얼 번호 등의 정보를 포함하는 표준화된 형식입니다.12 이 주소를 복사하여 스크립트에 사용할 준비를 합니다.
3. 통신 검증**:** 파이썬 코드를 작성하기 전에 Connection Expert 내에서 통신이 정상적으로 이루어지는지 확인하는 것이 좋습니다. 계측기를 선택한 상태에서 'Send Commands To This Instrument' 또는 'Interactive IO' 탭으로 이동합니다. 입력 창에 표준 SCPI 쿼리인 $*IDN?$을 입력하고 'Send & Read' 버튼을 클릭합니다. 잠시 후 응답 창에 계측기의 제조업체, 모델명, 시리얼 번호, 펌웨어 버전 등이 포함된 식별 문자열이 표시되면, 하드웨어 연결부터 VISA 라이브러리까지 모든 통신 스택이 정상적으로 작동하고 있음을 의미합니다.[12] 이 단계를 통해 실제 코딩 단계에서 발생할 수 있는 연결 관련 문제를 미리 배제할 수 있습니다.

2장**: PyVISA** 프로그래밍의 기초

테스트 환경 준비가 완료되었으니, 이제 실제 파이썬 코드를 작성하여 계측기와 통신하는 방법을 배울 차례입니다. 본 장에서는 PyVISA를 사용하여 통신을 설정하고, SCPI 명령어를 보내고 응답을 받으며, 어떠한 상황에서도 안정적으로 작동하는 견고한 자동화 스크립트를 작성하기 위한 핵심 기법들을 다룹니다.

2.1: 통신 설정

PyVISA를 이용한 계측기 통신은 세 가지 기본적인 단계로 이루어집니다.

1. **PyVISA** 라이브러리 임포트**:** 스크립트의 가장 처음에 import pyvisa 구문을 사용하여 PyVISA 라이브러리를 가져옵니다.
2. 리소스 매니저**(Resource Manager)** 생성**:** rm = pyvisa.ResourceManager() 코드를 실행하여 리소스 매니저 객체를 생성합니다. 이 객체는 시스템에 설치된 VISA 백엔드 라이브러리를 찾아 로드하고, 사용 가능한 계측기 리소스를 관리하는 역할을 합니다.[5]
3. 계측기 리소스 열기**:** instrument = rm.open_resource('VISA_ADDRESS_HERE') 코드를 사용하여 특정 계측기와의 통신 세션을 엽니다. 여기서 'VISA_ADDRESS_HERE' 부분에는 앞서 Keysight Connection Expert에서 확인한 계측기의 VISA 리소스 문자열을 정확하게 입력해야 합니다. 이 함수가 성공적으로 실행되면, instrument라는 객체를 통해 해당 계측기를 제어할 수 있게 됩니다.

다음은 이 세 단계를 보여주는 완전한 코드 예시입니다.

Python

```
import pyvisa

# 1.3절에서 확인한 실제 계측기의 VISA 주소로 변경해야 합니다.
VISA_ADDRESS = 'USB0::0x2A8D::0x1301::MY57000123::INSTR'

# 리소스 매니저 생성
rm = pyvisa.ResourceManager()

# 지정된 주소의 계측기와 통신 세션 열기
instrument = rm.open_resource(VISA_ADDRESS)

# 계측기 식별 정보 요청 및 출력
print(instrument.query('*IDN?'))

# 통신 세션 닫기
instrument.close()
```

2.2: 계측기의 언어: SCPI와 PyVISA 메서드

계측기와의 실질적인 상호작용은 SCPI 명령어를 PyVISA가 제공하는 메서드를 통해 주고받음으로써 이루어집니다.

- **SCPI 명령어 구조:** SCPI 명령어는 일반적으로 콜론(:)으로 구분되는 계층적인 키워드로 구성됩니다. 예를 들어, SOURce:VOLTage는 소스(출력) 하위 시스템의 전압 관련 기능을 의미합니다. 명령어 끝에 물음표(?)가 붙으면 해당 설정값을 요청하는 '쿼리(query)'가 되고, 물음표가 없으면 값을 설정하는 '명령(command)'이 됩니다.[3]
- **PyVISA의 핵심 통신 메서드:** PyVISA는 SCPI 통신을 위해 세 가지 주요 메서드를 제공합니다.
  - instrument.write('COMMAND'): 계측기로부터 응답을 기대하지 않는 명령을 보낼 때 사용합니다. 예를 들어, 계측기를 초기화하는 instrument.write('*RST')와 같이 사용됩니다.
  - response = instrument.read(): 계측기의 출력 버퍼에 있는 데이터를 읽어올 때 사용합니다. 일반적으로 write 메서드로 데이터 요청 명령을 보낸 후에 사용됩니다.
  - response = instrument.query('QUERY?'): SCPI 쿼리를 보내고 그 응답을 즉시 읽어오는 가장 편리하고 일반적으로 사용되는 메서드입니다. 이 메서드는 내부적으로 write와 read를 순차적으로 실행하는 것과 동일합니다.[5] 예를 들어, response = instrument.query('*IDN?')는 식별 정보 쿼리를 보내고 응답 문자열을 response 변수에 저장합니다.

2.3: 견고한 자동화 스크립트 작성법

단순히 명령어를 주고받는 것을 넘어, 실제 테스트 환경에서 안정적으로 동작하는 스크립트를 작성하기 위해서는 예외 처리와 동기화 메커니즘을 반드시 고려해야 합니다.

- **타임아웃(Timeout) 관리:** 스크립트가 계측기의 응답을 무한정 기다리다 멈추는 현상을 방지하기 위해 통신 타임아웃을 설정하는 것이 중요합니다. 타임아웃은 밀리초(ms) 단위로 설정하며, 지정된 시간 내에 응답이 없으면 PyVISA는 예외(Exception)를 발생시킵니다. 일반적으로 5초(5000ms) 정도가 적절합니다.

  Python
  ```
  instrument.timeout = 5000  # 타임아웃을 5000ms (5초)로 설정
  ```

  18

- **리소스 관리와 try...finally:** 계측기와의 통신 세션은 사용이 끝난 후 반드시 instrument.close()를 호출하여 명시적으로 닫아주어야 합니다. 만약 스크립트 실행 중 오류가 발생하여 프로그램이 비정상적으로 종료되면, close()가 호출되지 않아 리소스가 계속 점유되는 문제가 발생할 수 있습니다. 파이썬의 try...finally 구문은 이러한 문제를

해결하는 표준적인 방법입니다. finally 블록 안의 코드는 try 블록에서 오류가 발생하든 안 하든 항상 실행됨을 보장하므로, 리소스 해제 코드를 여기에 위치시키는 것이 안전합니다.[18]

```
Python
instrument = None  # 변수 초기화
try:
    instrument = rm.open_resource(VISA_ADDRESS)
    #... 계측기 제어 코드...
finally:
    if instrument:
        instrument.close()
```

- 오류 처리 및 동기화**:** 신뢰성 있는 자동화 스크립트는 단순히 명령을 보내는 것에서 그치지 않고, 해당 명령이 올바르게 실행되었는지 확인하고 계측기의 동작 완료를 기다리는 메커니즘을 포함해야 합니다. 이를 위해 두 가지 중요한 SCPI 명령어를 활용할 수 있습니다.
  - 사후 오류 검사 **(SYSTem:ERRor?):** 이 쿼리는 계측기의 오류 큐(queue)에 저장된 가장 오래된 오류를 반환합니다. 오류가 없으면 "+0, No error"와 같은 응답을 반환합니다. 스크립트의 중요한 단계마다 이 쿼리를 호출하여 이전에 실행된 명령어들에서 문제가 발생하지 않았는지 확인하는 것이 좋습니다. 오류가 발견되면, 오류가 더 이상 없을 때까지 반복적으로 쿼리하여 모든 오류 메시지를 확인할 수 있습니다. 이는 '반응적(reactive)' 오류 처리 방식입니다.[18]
  - 사전 동작 완료 대기 **(*OPC?):** 'Operation Complete' 쿼리는 이전에 전송된 모든 명령의 실행이 완료될 때까지 계측기가 응답을 보류하도록 합니다. 모든 작업이 끝나면 '1'을 반환합니다. 전압 램프 설정이나 파형 수집과 같이 실행에 시간이 걸리는 명령 직후에 이 쿼리를 사용하면, 스크립트가 다음 단계로 넘어가기 전에 계측기가 준비될 때까지 안전하게 기다리게 할 수 있습니다. 이는 '능동적(proactive)' 동기화 방식입니다.[7]

다음은 SYSTem:ERRor?를 확인하는 재사용 가능한 파이썬 함수 예시입니다.

```
Python
def check_instrument_errors(instrument, command_context=""):
    """계측기의 오류 큐를 확인하고 오류가 있으면 출력합니다."""
    while True:
        error_string = instrument.query(':SYSTem:ERRor?')
        if 'No error' in error_string or '+0' in error_string:
            break
        else:
```

```
        print(f"계측기 오류 발생 ({command_context}): {error_string.strip()}")
```

이 함수를 스크립트의 주요 지점, 예를 들어 설정 명령 그룹을 보낸 후에 호출하면 스크립트의 안정성을 크게 향상시킬 수 있습니다.

3장: Keysight 파워 서플라이 자동화

이제 2장에서 배운 PyVISA 프로그래밍 기초를 실제 계측기에 적용해볼 차례입니다. 본 장에서는 Keysight의 E36300 시리즈와 같은 일반적인 프로그래머블 DC 파워 서플라이를 자동화하는 방법을 다룹니다. 채널 선택, 전압 및 전류 설정, 출력 활성화, 그리고 실제 출력값 측정에 이르는 핵심적인 제어 과정을 실용적인 스크립트 예제와 함께 단계별로 학습합니다.

3.1: 핵심 파워 서플라이 SCPI 명령어

프로그래머블 파워 서플라이를 제어하는 데 사용되는 SCPI 명령어는 대부분 직관적이며 몇 가지 핵심적인 명령어로 대부분의 작업을 수행할 수 있습니다. 아래 표는 파워 서플라이 자동화에 필수적인 SCPI 명령어들을 요약한 것입니다.[20]

표 1: 일반적인 Keysight 파워 서플라이 SCPI 명령어

명령어	설명	예시
INSTrument:SELect <CH>	제어할 출력 채널을 선택합니다.	INST:SEL CH1
VOLTage <val>	선택된 채널의 출력 전압을 설정합니다.	VOLT 3.3
CURRent <val>	선택된 채널의 전류 제한 값을 설정합니다.	CURR 0.1
OUTPut:STATe <ON|OFF>	출력 릴레이를 켜거나(ON) 끕니다(OFF).	OUTP ON

MEASure:VOLTage? [<CH>]	지정된 채널의 실제 출력 전압을 측정하여 반환합니다.	MEAS:VOLT? CH1
MEASure:CURRent? [<CH>]	지정된 채널의 실제 출력 전류를 측정하여 반환합니다.	MEAS:CURR? CH1
*RST	계측기를 공장 기본 설정으로 초기화합니다.	*RST
*IDN?	계측기의 식별 정보를 요청합니다.	*IDN?

참고: 대괄호 `` 안의 키워드는 생략 가능합니다. 예를 들어, *SOURce:VOLTage*는 *VOLT*로 축약하여 사용할 수 있습니다.

3.2: 실용 스크립트**:** 출력 설정 및 활성화

다음은 파워 서플라이의 특정 채널에 원하는 전압과 전류를 설정하고 출력을 활성화하는 표준적인 파워업(power-up) 시퀀스를 수행하는 완전한 파이썬 스크립트입니다. 이 스크립트에는 2장에서 다룬 견고한 프로그래밍 기법(리소스 관리, 오류 확인)이 모두 포함되어 있습니다.

Python

```
import pyvisa
import time

# 파워 서플라이의 VISA 주소 (Connection Expert에서 확인한 값으로 변경)
PSU_VISA_ADDRESS = 'USB0::0x2A8D::0x0F01::MY58001234::INSTR'

def check_instrument_errors(instrument, command_context=""):
    """계측기의 오류 큐를 확인하고 오류가 있으면 출력합니다."""
    while True:
```

```
        error_string = instrument.query(':SYSTem:ERRor?')
        if 'No error' in error_string or '+0' in error_string:
            break
        else:
            print(f"계측기 오류 발생 ({command_context}): {error_string.strip()}")

def setup_power_supply():
    """파워 서플라이를 설정하고 출력을 활성화하는 함수."""
    rm = pyvisa.ResourceManager()
    psu = None
    try:
        # 리소스 열기 및 타임아웃 설정
        psu = rm.open_resource(PSU_VISA_ADDRESS)
        psu.timeout = 5000
        psu.write_termination = '\n'
        psu.read_termination = '\n'

        # 계측기 초기화 및 식별 정보 확인
        psu.write('*RST')
        print(f"연결된 장비: {psu.query('*IDN?').strip()}")
        check_instrument_errors(psu, "초기화 후")

        # 채널 1 선택
        psu.write('INST:SEL CH1')

        # 전압 3.3V, 전류 제한 0.1A 설정
        # APPLY 명령은 전압과 전류를 한 번에 설정하는 편리한 방법입니다.
        psu.write('APPL 3.3, 0.1')

        # 설정 후 오류 확인
        check_instrument_errors(psu, "APPL 명령 후")

        # 출력 활성화
        psu.write('OUTP ON')
        print("채널 1의 출력을 3.3V, 0.1A로 설정하고 활성화했습니다.")

        # 잠시 대기
        time.sleep(1)

        # 최종 오류 확인
        check_instrument_errors(psu, "출력 활성화 후")

    except pyvisa.errors.VisaIOError as e:
```

```
        print(f"VISA I/O 오류: {e}")
    finally:
        if psu:
            # 안전을 위해 출력을 비활성화하고 연결을 닫습니다.
            psu.write('OUTP OFF')
            psu.close()
            print("파워 서플라이 출력을 비활성화하고 연결을 종료했습니다.")

if __name__ == "__main__":
    setup_power_supply()
```

이 스크립트는 단순히 명령을 나열하는 것이 아니라, 각 단계마다 오류를 확인하고 try...finally 구조를 통해 예외가 발생하더라도 안전하게 리소스를 해제하는 안정적인 자동화 코드의 좋은 예시입니다.

3.3: 실용 스크립트**:** 실제 출력 측정

출력을 설정하는 것만큼이나 중요한 것은 실제 부하에 공급되는 전압과 전류를 정확하게 측정하는 것입니다. 다음 스크립트는 이전 예제에 측정 단계를 추가한 것입니다. MEASure 쿼리를 사용하여 실제 출력값을 읽어오고, 이를 프로그램에서 활용할 수 있도록 숫자 데이터(float)로 변환하는 과정을 보여줍니다.

Python

```
#... (이전 스크립트의 check_instrument_errors 함수 및 주소 정의는 동일)...

def setup_and_measure_psu():
    """파워 서플라이를 설정, 활성화하고 실제 출력값을 측정하는 함수."""
    rm = pyvisa.ResourceManager()
    psu = None
    try:
        psu = rm.open_resource(PSU_VISA_ADDRESS)
        psu.timeout = 5000
        psu.write_termination = '\n'
        psu.read_termination = '\n'

        print(f"연결된 장비: {psu.query('*IDN?').strip()}")
```

```
        # 채널 1에 5.0V, 0.5A 설정
        psu.write('APPL CH1, 5.0, 0.5')

        # 출력 활성화
        psu.write('OUTP:STAT ON, (@1)')
        print("채널 1의 출력을 5.0V, 0.5A로 설정하고 활성화했습니다.")

        # 출력이 안정화될 시간을 줍니다.
        time.sleep(2)

        # 채널 1의 실제 출력 전압 측정
        measured_voltage_str = psu.query('MEAS:VOLT? CH1')
        # 계측기 응답(문자열)을 float 형태로 변환
        measured_voltage = float(measured_voltage_str)

        # 채널 1의 실제 출력 전류 측정
        measured_current_str = psu.query('MEAS:CURR? CH1')
        measured_current = float(measured_current_str)

        print(f"측정된 전압: {measured_voltage:.4f} V")
        print(f"측정된 전류: {measured_current:.4f} A")

        check_instrument_errors(psu, "측정 후")

    except pyvisa.errors.VisaIOError as e:
        print(f"VISA I/O 오류: {e}")
    except ValueError:
        print("계측기로부터 받은 응답을 숫자로 변환하는 데 실패했습니다.")
    finally:
        if psu:
            psu.write('OUTP OFF')
            psu.close()
            print("파워 서플라이 출력을 비활성화하고 연결을 종료했습니다.")

if __name__ == "__main__":
    setup_and_measure_psu()
```

이 예제는 계측기로부터 받은 응답이 줄 바꿈 문자(\n) 등을 포함한 문자열 형태라는 점을 명확히 보여줍니다. 따라서 float() 함수를 사용하여 이를 파이썬에서 계산이나 비교에 사용할 수 있는 숫자 데이터 타입으로 변환하는 과정이 필수적입니다. 이 과정은 모든 종류의 계측기에서 측정값을 다룰 때 공통적으로 적용되는 중요한 단계입니다.

4장: Keysight 오실로스코프 자동화

파워 서플라이 제어에 이어, 이번에는 더 복잡하고 다양한 기능을 가진 오실로스코프를 자동화하는 방법을 다룹니다. 오실로스코프 자동화의 핵심은 수직(Vertical), 수평(Horizontal), 트리거(Trigger) 설정을 프로그래밍 방식으로 제어하여 원하는 파형을 안정적으로 포착하는 것입니다. 본 장에서는 기본적인 설정부터 시작하여, 오실로스코프의 내장 측정 기능을 활용하는 효율적인 방법까지 학습합니다.

4.1: 필수 오실로스코프 SCPI 설정 명령어

오실로스코프의 SCPI 명령어는 기능별로 체계적인 하위 시스템 구조를 가집니다. 예를 들어, 채널 관련 설정은 :CHANnel, 시간 축 설정은 :TIMebase, 트리거 설정은 :TRIGger, 데이터 수집 관련 설정은 :ACQuire 하위 시스템에 속해 있습니다.[17] 자동화를 시작하기 위해 알아야 할 가장 기본적인 명령어들은 다음과 같습니다.

표 2: 기본 오실로스코프 설정 SCPI 명령어

명령어	설명	예시
*RST	오실로스코프를 공장 기본 설정으로 초기화합니다.	*RST
:CHANnel<n>:SCALe <val>	채널 <n>의 수직 스케일(Volts/div)을 설정합니다.	:CHAN1:SCAL 500E-3 (500 mV/div)
:CHANnel<n>:OFFSet <val>	채널 <n>의 수직 오프셋(위치)을 설정합니다.	:CHAN1:OFFS -1.5 (-1.5 V)
:TIMebase:SCALe <val>	수평 스케일(Seconds/div)을 설정합니다.	:TIM:SCAL 1E-6 (1 µs/div)
:TIMebase:POSition <val>	수평 위치(지연)를	:TIM:POS 0

	설정합니다.	
:TRIGger:EDGE:SOURce <src>	에지 트리거의 소스를 지정합니다 (예: CHAN1).	:TRIG:EDGE:SOUR CHAN1
:TRIGger:EDGE:LEVel <val>	에지 트리거의 전압 레벨을 설정합니다.	:TRIG:EDGE:LEV 1.65 (1.65 V)
:RUN, :STOP, :SINGle	파형 수집 상태를 제어합니다 (연속, 정지, 단일).	:RUN

4.2: 실용 스크립트**:** 기본 파형 관측 설정

다음은 3.3V 진폭의 1kHz 구형파를 안정적으로 관측하기 위해 오실로스코프를 설정하는 완전한 파이썬 스크립트입니다. 이 스크립트는 여러 설정 명령어를 순차적으로 전송하여 특정 측정 환경을 구성하는 방법을 보여줍니다.

Python

```
import pyvisa
import time

# 오실로스코프의 VISA 주소 (Connection Expert에서 확인한 값으로 변경)
SCOPE_VISA_ADDRESS = 'USB0::0x2A8D::0x1797::MY59001234::INSTR'

def check_instrument_errors(instrument, command_context=""):
    #... (이전 장의 함수와 동일)...
    while True:
        error_string = instrument.query(':SYSTem:ERRor?')
        if 'No error' in error_string or '+0' in error_string:
            break
        else:
            print(f"계측기 오류 발생 ({command_context}): {error_string.strip()}")
```

```
def setup_oscilloscope_basic():
    """오실로스코프의 기본 설정을 수행하는 함수."""
    rm = pyvisa.ResourceManager()
    scope = None
    try:
        scope = rm.open_resource(SCOPE_VISA_ADDRESS)
        scope.timeout = 10000
        scope.write_termination = '\n'
        scope.read_termination = '\n'

        # 계측기 초기화 및 식별 정보 확인
        scope.write('*RST')
        # *RST는 완료까지 시간이 걸릴 수 있으므로 *OPC?로 대기
        scope.query('*OPC?')
        print(f"연결된 장비: {scope.query('*IDN?').strip()}")

        # --- 설정 시작 ---
        print("오실로스코프 설정 시작...")

        # 타임베이스 설정: 100us/div
        scope.write(':TIMebase:SCALe 100E-6')

        # 채널 1 설정
        scope.write(':CHANnel1:DISPlay ON')   # 채널 1 켜기
        scope.write(':CHANnel1:SCALe 1.0')    # 수직 스케일 1V/div
        scope.write(':CHANnel1:OFFSet 1.65')  # 수직 오프셋 1.65V (0-3.3V 신호의 중앙)

        # 트리거 설정
        scope.write(':TRIGger:MODE EDGE')          # 에지 트리거 모드
        scope.write(':TRIGger:EDGE:SOURce CHAN1')  # 트리거 소스를 채널 1로
        scope.write(':TRIGger:EDGE:SLOPe POS')     # 상승 에지에서 트리거
        scope.write(':TRIGger:EDGE:LEVel 1.65')    # 트리거 레벨 1.65V

        # 수집 시작
        scope.write(':RUN')

        print("설정 완료. 1kHz, 3.3V 구형파 관측 준비 완료.")
        check_instrument_errors(scope, "최종 설정 후")

        # 설정이 적용된 것을 확인하기 위해 잠시 대기
        time.sleep(2)

    except pyvisa.errors.VisaIOError as e:
```

```
        print(f"VISA I/O 오류: {e}")
    finally:
        if scope:
            scope.close()
            print("오실로스코프 연결을 종료했습니다.")

if __name__ == "__main__":
    setup_oscilloscope_basic()
```

이 스크립트는 계측기 초기화(*RST)와 같이 시간이 소요될 수 있는 작업 후에 *OPC? 쿼리를 사용하여 동기화하는 좋은 예를 보여줍니다. 이를 통해 이전 명령이 완전히 완료된 후에 다음 명령이 실행되도록 보장할 수 있습니다.

4.3: 자동화된 측정 수행

많은 경우, 파형의 특정 파라미터(예: 전압, 주파수, 상승 시간 등)만이 필요합니다. 이럴 때 전체 파형 데이터를 PC로 전송하여 분석하는 것은 비효율적입니다. 대신, 오실로스코프의 강력한 내장 측정 하드웨어를 활용하는 것이 훨씬 빠르고 효율적입니다. :MEASure 하위 시스템의 쿼리들을 사용하면 이러한 자동화된 측정을 쉽게 수행할 수 있습니다.[17]

다음 스크립트는 채널 1에 입력된 신호의 Vpp(피크-투-피크 전압), 주파수, 상승 시간을 자동으로 측정하고 그 결과를 가져오는 방법을 보여줍니다.

Python

```
#... (이전 스크립트의 check_instrument_errors 함수 및 주소 정의는 동일)...

def perform_auto_measurements():
    """오실로스코프의 자동 측정 기능을 사용하는 함수."""
    rm = pyvisa.ResourceManager()
    scope = None
    try:
        scope = rm.open_resource(SCOPE_VISA_ADDRESS)
        scope.timeout = 10000
        scope.write_termination = '\n'
        scope.read_termination = '\n'
```

```
        print(f"연결된 장비: {scope.query('*IDN?').strip()}")

        # 자동 설정을 통해 신호를 화면에 맞춤
        scope.write(':AUTOscale')
        scope.query('*OPC?') # Autoscale 완료 대기
        print("자동 설정(Autoscale) 완료.")
        time.sleep(2) # 측정이 안정화될 시간

        # --- 자동 측정 수행 ---
        # Vpp 측정 (소스 직접 지정)
        vpp_str = scope.query(':MEASure:VPP? CHAN1')
        vpp = float(vpp_str)
        print(f"측정된 Vpp: {vpp:.4f} V")

        # 주파수 측정
        freq_str = scope.query(':MEASure:FREQuency? CHAN1')
        freq = float(freq_str)
        print(f"측정된 주파수: {freq:.2f} Hz")

        # 상승 시간 측정
        # 일부 신호에서는 측정이 불가능할 수 있음 (예: DC 신호)
        try:
            risetime_str = scope.query(':MEASure:RISEtime? CHAN1')
            risetime = float(risetime_str)
            print(f"측정된 상승 시간: {risetime * 1e9:.2f} ns")
        except pyvisa.errors.VisaIOError:
            # 계측기가 값을 측정할 수 없을 때 (예: 9.9E+37 반환) 오류가 발생할 수 있음
            print("상승 시간을 측정할 수 없습니다 (에지가 감지되지 않음).")

        check_instrument_errors(scope, "자동 측정 후")

    except pyvisa.errors.VisaIOError as e:
        print(f"VISA I/O 오류: {e}")
    finally:
        if scope:
            scope.close()
            print("오실로스코프 연결을 종료했습니다.")

if __name__ == "__main__":
    perform_auto_measurements()
```

이 방법은 데이터 전송 오버헤드를 최소화하고 테스트 시간을 크게 단축시키는 매우 강력한 자동화 기법입니다. 측정 소스는 :MEASure:VPP? CHAN1과 같이 쿼리 명령어에 직접 명시할 수

있어 코드를 더 명확하게 만들 수 있습니다.[17]

5장: 고급 응용: 파형 데이터 수집 및 분석

오실로스코프의 내장 측정 기능만으로는 부족하고, PC에서 직접 파형 데이터를 분석하거나 저장해야 하는 경우가 있습니다. 본 장에서는 오실로스코프로부터 원시(raw) 파형 데이터를 PC로 전송받아 실제 전압과 시간 값으로 변환하고, 이를 그래프로 시각화하는 가장 복잡하면서도 강력한 자동화 기술을 상세히 다룹니다.

5.1: 파형 데이터 전송의 이론

파형 데이터는 크게 두 가지 형식, 즉 ASCII와 바이너리(Binary)로 전송될 수 있습니다.

- **ASCII 형식:** 사람이 읽을 수 있는 텍스트 형태로 데이터를 전송합니다. 디버깅이 쉽다는 장점이 있지만, 데이터 양이 매우 크고 전송 속도가 현저히 느려 대량의 데이터를 다루기에는 부적합합니다.[17]
- 바이너리 형식: 데이터를 ADC(Analog-to-Digital Converter)의 원시 정수 값 그대로 전송합니다. 데이터 크기가 작아 전송 속도가 매우 빠르지만, PC에서 이 데이터를 의미 있는 값(전압, 시간)으로 변환하기 위한 추가적인 처리 과정이 필요합니다. 효율적인 자동화를 위해서는 바이너리 형식을 사용하는 것이 일반적입니다.[17]

바이너리 데이터 전송을 위한 핵심 SCPI 명령어는 다음과 같습니다.

- :WAVeform:SOURce <src>: 데이터를 가져올 소스(예: CHAN1)를 지정합니다.
- :WAVeform:FORMat <format>: 데이터 형식을 지정합니다. BYTE(8비트 정수) 또는 WORD(16비트 정수)를 주로 사용합니다.
- :WAVeform:POINts <count>: 전송받을 데이터 포인트의 수를 설정합니다.
- :WAVeform:DATA?: 설정된 형식에 따라 실제 파형 데이터를 요청하는 쿼리입니다.[17]

5.2: 파형 데이터의 로제타 스톤, 프리앰블(Preamble)

:WAVeform:DATA? 쿼리를 통해 받은 바이너리 데이터는 단순한 정수 값의 나열일 뿐, 그 자체로는 아무런 의미가 없습니다. 이 원시 ADC 값들을 실제 전압과 시간 단위로 변환하기 위해서는 스케일링 및 오프셋 정보가 반드시 필요합니다. 이 모든 핵심 정보는 파형

프리앰블**(waveform preamble)** 에 담겨 있으며, :WAVeform:PREamble? 쿼리를 통해 얻을 수 있습니다. 프리앰블은 원시 데이터를 해석하는 방법을 알려주는 '로제타 스톤'과 같은 역할을 합니다.

:WAVeform:PREamble? 쿼리는 콤마로 구분된 10개의 값으로 구성된 문자열을 반환합니다. 각 값의 의미는 다음과 같습니다.[17]

인덱스	이름	설명
0	Format	데이터 형식 (0: BYTE, 1: WORD, 4: ASCII)
1	Type	수집 유형 (0: Normal, 1: Peak Detect, 2: Average)
2	Points	전송된 데이터 포인트의 수
3	Count	항상 1
4	XIncrement	X축(시간) 증분값. 데이터 포인트 간의 시간 간격(초).
5	XOrigin	X축(시간) 원점. 첫 번째 데이터 포인트의 시간(초).
6	XReference	X축 원점의 기준이 되는 데이터 포인트 인덱스 (보통 0).
7	YIncrement	Y축(전압) 증분값. ADC 1 count 당 전압 값.
8	YOrigin	Y축(전압) 원점. 화면 중앙에 해당하는 전압 값(오프셋).
9	YReference	Y축 원점의 기준이 되는 ADC count 값.

이 프리앰블 값들을 사용하여 원시 ADC 데이터( )를 실제 전압( )과 시간( )으로 변환하는

공식은 다음과 같습니다.[17]

전압 변환 (Y축):

시간 변환 (X축):

(여기서 i는 0부터 시작하는 데이터 포인트의 인덱스입니다.)
이 두 공식을 이해하고 적용하는 것이 파형 데이터 수집의 가장 핵심적인 부분입니다.

5.3: 파형 수집 및 플로팅을 위한 완전한 파이썬 스크립트

이제 이론을 실제 코드로 구현해 보겠습니다. 다음은 오실로스코프에서 단일 파형을 수집하고, 프리앰블 정보를 이용해 변환한 후, matplotlib 라이브러리를 사용하여 그래프로 그리는 완전한 스크립트입니다. 이 스크립트를 실행하기 위해서는 numpy와 matplotlib 라이브러리가 추가로 설치되어 있어야 합니다 (pip install numpy matplotlib).

Python

```
import pyvisa
import numpy as np
import matplotlib.pyplot as plt
import time

# 오실로스코프의 VISA 주소
SCOPE_VISA_ADDRESS = 'USB0::0x2A8D::0x1797::MY59001234::INSTR'

def acquire_and_plot_waveform():
    """오실로스코프에서 파형을 수집하고 그래프로 그리는 함수."""
    rm = pyvisa.ResourceManager()
    scope = None
    try:
        scope = rm.open_resource(SCOPE_VISA_ADDRESS)
        scope.timeout = 15000  # 데이터 전송을 위해 타임아웃을 길게 설정
        scope.write_termination = '\n'
        scope.read_termination = '\n'
```

```
    print(f"연결된 장비: {scope.query('*IDN?').strip()}")

    # --- 오실로스코프 설정 ---
    scope.write('*RST; *CLS') # 초기화 및 상태 레지스터 클리어
    scope.write(':TIMebase:SCALe 100E-6')
    scope.write(':CHANnel1:SCALe 1.0')
    scope.write(':TRIGger:EDGE:SOURce CHAN1')
    scope.write(':TRIGger:EDGE:LEVel 1.65')

    # --- 파형 수집 준비 ---
    scope.write(':WAVeform:SOURce CHAN1')
    scope.write(':WAVeform:FORMat WORD')  # 16비트 바이너리 형식
    scope.write(':WAVeform:BYTeorder LSBFirst') # 바이트 순서 설정 (PC 표준)

    # 단일 수집 실행 및 완료 대기
    scope.write(':SINGle')
    scope.query('*OPC?')
    print("단일 파형 수집 완료.")

    # --- 프리앰블 정보 요청 및 파싱 ---
    preamble_str = scope.query(':WAVeform:PREamble?')
    preamble = preamble_str.strip().split(',')

    # 프리앰블 값들을 숫자 형태로 변환
    x_increment = float(preamble)
    x_origin = float(preamble)
    y_increment = float(preamble)
    y_origin = float(preamble)
    y_reference = int(float(preamble))
    num_points = int(float(preamble))

    print("프리앰블 정보 획득 완료.")

    # --- 원시 파형 데이터 요청 ---
    # query_binary_values는 헤더와 데이터를 자동으로 파싱해주는 강력한 기능
    raw_data = scope.query_binary_values(
        ':WAVeform:DATA?',
        datatype='h',        # 'h'는 short integer (16-bit)를 의미
        is_big_endian=False   # LSBFirst이므로 False
    )
    print(f"{len(raw_data)}개의 데이터 포인트 수신 완료.")

    # --- 데이터 변환 ---
```

```
        # NumPy를 사용하여 전체 배열에 대해 한 번에 계산 (매우 효율적)
        print("데이터 변환 중...")
        voltage_values = (np.array(raw_data) - y_reference) * y_increment + y_origin

        # 시간 축 데이터 생성
        time_values = np.arange(0, num_points * x_increment, x_increment) + x_origin

        # --- 그래프 플로팅 ---
        print("그래프 생성 중...")
        plt.figure(figsize=(10, 6))
        plt.plot(time_values * 1e6, voltage_values) # 시간 축을 us 단위로 표시
        plt.title('Acquired Waveform')
        plt.xlabel('Time (us)')
        plt.ylabel('Voltage (V)')
        plt.grid(True)
        plt.show()

    except pyvisa.errors.VisaIOError as e:
        print(f"VISA I/O 오류: {e}")
    finally:
        if scope:
            scope.close()
            print("오실로스코프 연결을 종료했습니다.")

if __name__ == "__main__":
    acquire_and_plot_waveform()
```

이 스크립트는 PyVISA의 query_binary_values 메서드를 활용하여 복잡한 바이너리 데이터 파싱을 간소화하는 방법을 보여줍니다. 또한, numpy를 사용하여 대규모 데이터 배열에 대한 수학적 변환을 매우 빠르고 효율적으로 수행합니다. 이 예제는 파형 분석 자동화의 완결된 형태를 보여주는 핵심적인 결과물입니다.

6장: 통합 테스트 시나리오: 샘플 자동화 테스트 루틴

지금까지 배운 모든 내용을 종합하여, 파워 서플라이와 오실로스코프를 함께 제어하는 현실적인 자동화 테스트 시나리오를 구현해 봅니다. 이 마지막 장은 개별 계측기 제어 기술을 하나의 통합된 테스트 시퀀스로 엮어내는 과정을 보여줌으로써, 실무에 바로 적용할 수 있는 능력을 완성하는 것을 목표로 합니다.

6.1: 테스트 케이스 정의

- 테스트 목표**:** 가상의 테스트 대상 장치(DUT, Device Under Test)의 '전원 인가 후 출력 안정화 시간(turn-on time)'을 특성화합니다.
- 테스트 절차**:**
  1. 파워 서플라이를 사용하여 DUT에 5V의 전원 스텝을 인가하여 전원을 켭니다.
  2. 오실로스코프는 DUT의 출력 단자에 연결되어 있으며, 파워 서플라이가 켜지는 순간을 트리거하여 DUT의 출력 신호 파형을 포착합니다.
  3. 오실로스코프의 자동 측정 기능을 사용하여 포착된 파형의 상승 시간(rise time)을 측정합니다.
  4. 테스트 결과(설정 전압, 측정된 상승 시간)를 기록하고 출력합니다.

이 시나리오는 전원 제어와 신호 측정이 연동되는 가장 일반적인 자동화 테스트 유형 중 하나입니다.

6.2: 통합 파이썬 스크립트

다음은 위에서 정의한 테스트 케이스를 수행하는 단일 통합 스크립트입니다. 이 스크립트는 각 계측기를 초기화하고, 테스트 시퀀스를 실행하며, 사용된 모든 리소스를 안전하게 해제하는 잘 구조화된 함수들로 구성되어 있습니다.

Python

```
import pyvisa
import time
import csv
from datetime import datetime

# --- 계측기 VISA 주소 ---
PSU_VISA_ADDRESS = 'USB0::0x2A8D::0x0F01::MY58001234::INSTR'
SCOPE_VISA_ADDRESS = 'USB0::0x2A8D::0x1797::MY59001234::INSTR'

def check_instrument_errors(instrument, command_context=""):
    #... (이전 장의 함수와 동일)...
```

```
    pass

def run_dut_turn_on_test(psu, scope, voltage=5.0):
    """DUT 턴온 테스트 시퀀스를 실행하고 상승 시간을 반환합니다."""
    print(f"\n--- {voltage}V 턴온 테스트 시작 ---")

    # --- 오실로스코프 설정 ---
    scope.write('*RST; *CLS')
    scope.write(':CHANnel1:SCALe 1.0')      # 1V/div
    scope.write(':CHANnel1:OFFSet 2.5')     # 0-5V 신호의 중앙
    scope.write(':TIMebase:SCALe 100E-6')   # 100us/div
    # 트리거를 Normal 모드로 설정하여, 트리거 이벤트가 있을 때만 수집
    scope.write(':TRIGger:SWEep NORMal')
    scope.write(':TRIGger:EDGE:SOURce CHAN1')
    scope.write(':TRIGger:EDGE:SLOPe POS')
    scope.write(':TRIGger:EDGE:LEVel 2.5')  # 5V의 50% 지점

    # 단일 수집 대기 상태로 만듦
    scope.write(':SINGle')
    time.sleep(0.5) # 오실로스코프가 대기 상태로 전환될 시간

    # --- 파워 서플라이 설정 및 출력 ---
    psu.write('*RST; *CLS')
    # 채널 1에 지정된 전압, 1A 전류 제한 설정
    psu.write(f'APPL CH1, {voltage}, 1.0')

    # 전원 인가 (이 순간 오실로스코프가 트리거됨)
    psu.write('OUTP:STAT ON, (@1)')
    print(f"DUT에 {voltage}V 전원 인가.")

    # 트리거가 발생하고 파형 수집이 완료될 때까지 충분히 대기
    time.sleep(3)

    # --- 결과 측정 ---
    risetime = float('nan') # Not a Number로 초기화
    try:
        risetime_str = scope.query(':MEASure:RISEtime? CHAN1')
        risetime = float(risetime_str)
        print(f"측정된 상승 시간: {risetime * 1e6:.2f} us")
    except pyvisa.errors.VisaIOError:
        print("상승 시간을 측정할 수 없습니다.")

    # --- 정리 ---
```

```
    psu.write('OUTP:STAT OFF, (@1)')
    print("DUT 전원 차단.")

    return risetime

def main():
    """메인 실행 함수: 계측기 연결, 테스트 실행, 리소스 해제."""
    rm = pyvisa.ResourceManager()
    psu = None
    scope = None

    results =

    try:
        # 계측기 리소스 열기
        psu = rm.open_resource(PSU_VISA_ADDRESS)
        scope = rm.open_resource(SCOPE_VISA_ADDRESS)

        # 타임아웃 등 기본 설정
        for inst in [psu, scope]:
            inst.timeout = 10000
            inst.write_termination = '\n'
            inst.read_termination = '\n'
            print(f"연결 성공: {inst.query('*IDN?').strip()}")

        # 테스트 실행
        test_voltage = 5.0
        rise_time_result = run_dut_turn_on_test(psu, scope, test_voltage)
        results.append({
            'timestamp': datetime.now().isoformat(),
            'voltage_setting_V': test_voltage,
            'rise_time_s': rise_time_result
        })

        # 결과 파일로 저장 (CSV)
        with open('test_results.csv', 'w', newline='') as f:
            writer = csv.DictWriter(f, fieldnames=results.keys())
            writer.writeheader()
            writer.writerows(results)
        print("\n테스트 결과를 'test_results.csv' 파일에 저장했습니다.")

    except pyvisa.errors.VisaIOError as e:
        print(f"치명적인 VISA 오류 발생: {e}")
```

```
    except Exception as e:
        print(f"알 수 없는 오류 발생: {e}")
    finally:
        # 모든 리소스 안전하게 해제
        if psu:
            psu.write('OUTP OFF')
            psu.close()
        if scope:
            scope.close()
        print("\n모든 계측기 연결을 종료했습니다.")

if __name__ == "__main__":
    main()
```

이 스크립트는 두 계측기 간의 동작 순서를 정밀하게 제어하는 방법을 보여줍니다. 오실로스코프를 먼저 '단일 수집 대기' 상태로 만든 다음, 파워 서플라이의 출력을 켜서 트리거 이벤트를 발생시키는 것이 핵심입니다. 테스트가 끝난 후에는 결과를 간단한 CSV 파일로 저장하여 데이터 로깅의 기본 형태를 구현합니다.

6.3: 결론 및 다음 단계

이 가이드를 통해 독자들은 자동화 테스트 환경을 구축하는 기초부터 시작하여, PyVISA를 이용해 Keysight 파워 서플라이와 오실로스코프를 제어하는 핵심 기술들을 습득했습니다. 다루어진 주요 기술은 다음과 같습니다.

- VISA, SCPI, PyVISA의 개념적 이해 및 소프트웨어 환경 설정.
- Keysight Connection Expert를 이용한 VISA 주소 확인 및 통신 검증.
- try...finally, 오류 확인, 동기화를 포함한 견고한 스크립트 작성법.
- 파워 서플라이의 전압/전류 설정 및 출력 제어.
- 오실로스코프의 기본 설정, 자동 측정 기능 활용, 그리고 원시 파형 데이터 수집 및 분석.

이 가이드에서 제공된 지식과 예제 코드는 다양한 자동화 테스트 시나리오를 개발하기 위한 견고한 출발점입니다. 여기서 더 나아가 다음과 같은 방향으로 기술을 확장해 볼 수 있습니다.

- 테스트 반복 및 파라미터 스윕**(Sweep):** 전압을 3.3V, 5.0V, 12.0V 등으로 변경하며 테스트를 반복 실행하고, 각 조건에서의 결과를 기록하는 루프를 구현합니다.
- 고급 데이터 분석**:** 수집한 전체 파형 데이터를 numpy나 scipy와 같은 라이브러리를 사용하여 FFT(주파수 분석), 필터링 등 복잡한 신호 처리 작업을 수행합니다.
- 객체 지향 프로그래밍**(OOP)** 적용**:** 각 계측기를 클래스(Class)로 추상화하여 코드를 더 모듈화하고 재사용성을 높입니다. 예를 들어, KeysightPSU 클래스를 만들고 set_voltage, measure_current와 같은 메서드를 구현할 수 있습니다.

- **GUI(Graphical User Interface)** 통합**:** PyQt나 Tkinter와 같은 라이브러리를 사용하여 테스트를 제어하고 결과를 시각적으로 표시하는 사용자 인터페이스를 개발합니다.

자동화 계측 제어는 테스트의 효율성, 반복성, 정확성을 극대화하는 현대 엔지니어링의 필수 기술입니다. 이 가이드가 그 여정을 시작하는 데 훌륭한 동반자가 되기를 바랍니다.

부록**: SCPI** 명령어 참조 테이블

표 **1:** 일반적인 **Keysight** 파워 서플라이 **SCPI** 명령어

명령어	설명	예시
INSTrument:SELect <CH>	제어할 출력 채널을 선택합니다.	INST:SEL CH1
VOLTage <val>	선택된 채널의 출력 전압을 설정합니다.	VOLT 3.3
CURRent <val>	선택된 채널의 전류 제한 값을 설정합니다.	CURR 0.1
OUTPut:STATe <ON|OFF>	출력 릴레이를 켜거나(ON) 끕니다(OFF).	OUTP ON
MEASure:VOLTage? [<CH>]	지정된 채널의 실제 출력 전압을 측정하여 반환합니다.	MEAS:VOLT? CH1
MEASure:CURRent? [<CH>]	지정된 채널의 실제 출력 전류를 측정하여 반환합니다.	MEAS:CURR? CH1
*RST	계측기를 공장 기본 설정으로 초기화합니다.	*RST
*IDN?	계측기의 식별 정보를	*IDN?

	요청합니다.	

표 **2:** 기본 오실로스코프 설정 **SCPI** 명령어

명령어	설명	예시
*RST	오실로스코프를 공장 기본 설정으로 초기화합니다.	*RST
:CHANnel<n>:SCALe <val>	채널 <n>의 수직 스케일(Volts/div)을 설정합니다.	:CHAN1:SCAL 500E-3 (500 mV/div)
:CHANnel<n>:OFFSet <val>	채널 <n>의 수직 오프셋(위치)을 설정합니다.	:CHAN1:OFFS -1.5 (-1.5 V)
:TIMebase:SCALe <val>	수평 스케일(Seconds/div)을 설정합니다.	:TIM:SCAL 1E-6 (1 µs/div)
:TIMebase:POSition <val>	수평 위치(지연)를 설정합니다.	:TIM:POS 0
:TRIGger:EDGE:SOURce <src>	에지 트리거의 소스를 지정합니다 (예: CHAN1).	:TRIG:EDGE:SOUR CHAN1
:TRIGger:EDGE:LEVel <val>	에지 트리거의 전압 레벨을 설정합니다.	:TRIG:EDGE:LEV 1.65 (1.65 V)
:RUN, :STOP, :SINGle	파형 수집 상태를 제어합니다 (연속, 정지, 단일).	:RUN

참고 자료

1. PyVISA: Control your instruments with Python — PyVISA 1.15.1.dev27+g908d86744 documentation, 9월 30, 2025에 액세스, https://pyvisa.readthedocs.io/
2. Keysight Technologies - System Developer Guide - Using USB in the Test and

Measurement Environment Application Note, 9월 30, 2025에 액세스, https://keysight.zinfi.net/concierge/OEMs/keysight/wwwcontent/Attachments/pdf/6773_5989-1417EN.pdf
3. System Power Supply Programming - Keysight, 9월 30, 2025에 액세스, https://www.keysight.com/us/en/assets/7018-06572/white-papers/5992-3841.pdf
4. 664xA 665xA 667xA 668xA, 669xA DC Power Supplies ... - Keysight, 9월 30, 2025에 액세스, https://www.keysight.com/us/en/assets/7018-09132/technical-overviews/5964-8269.pdf
5. Tutorial — PyVISA 1.5 documentation - Read the Docs, 9월 30, 2025에 액세스, https://pyvisa.readthedocs.io/en/1.5-docs/tutorial.html
6. PyVISA · PyPI, 9월 30, 2025에 액세스, https://pypi.org/project/PyVISA/
7. Getting Started with Oscilloscope Automation and Python - Tektronix, 9월 30, 2025에 액세스, https://www.tek.com/en/documents/technical-brief/getting-started-with-oscilloscope-automation-and-python
8. Installation — PyVISA 1.15.0 documentation - Read the Docs, 9월 30, 2025에 액세스, https://pyvisa.readthedocs.io/en/stable/introduction/getting.html
9. Installation — PyVISA 1.15.1.dev27+g908d86744 documentation - Read the Docs, 9월 30, 2025에 액세스, https://pyvisa.readthedocs.io/en/latest/introduction/getting.html
10. Keysight IO Libraries Suite 2019 (for Windows), 9월 30, 2025에 액세스, https://www.keysight.com/us/en/assets/9018-18746/installation-guides/9018-18746.htm
11. Installation — PyVISA 1.8 documentation - Read the Docs, 9월 30, 2025에 액세스, https://pyvisa.readthedocs.io/en/1.8/getting.html
12. Keysight E5810B LAN/GPIB/USB Gateway - TestEquity, 9월 30, 2025에 액세스, https://assets.testequity.com/te1/Documents/pdf/keysight/E5810B-Quick-Start.pdf
13. Connection Expert - Keysight, 9월 30, 2025에 액세스, https://helpfiles.keysight.com/IO_Libraries_Suite/English/IOLS_Linux/Web_Connection_Expert/Index.htm
14. Connection Expert - Keysight, 9월 30, 2025에 액세스, https://helpfiles.keysight.com/IO_Libraries_Suite/English/IOLS_Windows/Connection_Expert_New/Index.htm
15. Connecting Your Instruments - Keysight, 9월 30, 2025에 액세스, https://helpfiles.keysight.com/BenchVueSoftware_HDML5HelpFiles/PlatformApp/English/Content/Connectivity%20Guide/Connecting%20Your%20Instruments.htm
16. Addressing a Session - Keysight, 9월 30, 2025에 액세스, https://helpfiles.keysight.com/IO_Libraries_Suite/English/IOLS_Linux/VISA/Content/UsersGuide/chapter3/Addressing%20a%20Session.htm
17. Keysight InfiniiVision 3000 X-Series Oscilloscopes ... - Batronix, 9월 30, 2025에 액세스, https://www.batronix.com/files/Keysight/Oszilloskope/3000XT/3000XT-Program

ming.pdf
18. Python Example - Keysight, 9월 30, 2025에 액세스, https://helpfiles.keysight.com/csg/FFProgrammingHelpWebHelp/Examples/Python_Example.htm
19. Acquiring data from oscilloscope using pyvisa - Stack Overflow, 9월 30, 2025에 액세스, https://stackoverflow.com/questions/28012009/acquiring-data-from-oscilloscope-using-pyvisa
20. E36300 Series Programming Guide | Keysight, 9월 30, 2025에 액세스, https://www.keysight.com/us/en/assets/9018-04577/programming-guides/9018-04577.pdf
21. E36300 Series Programmable DC Power Supplies - Batronix, 9월 30, 2025에 액세스, https://www.batronix.com/files/Keysight/Netzgeraete/E36300-Programming.pdf
22. Reading and Writing values — PyVISA 1.8 documentation, 9월 30, 2025에 액세스, https://pyvisa.readthedocs.io/en/1.8/rvalues.html