Keysight EDU36311A와 **PYNQ AUP-ZU3**를 활용한 **PyVISA** 기반 테스트 자동화 **PoC** 튜토리얼 가이드

I. 서론 및 임베디드 테스트 자동화의 배경

A. 프로젝트 개요 및 목표 설정

본 보고서는 자일링스(Xilinx)의 Zynq UltraScale+ MPSoC 기반 AUP-ZU3 PYNQ 보드 (PYNQ 3.1.1 운영 환경)를 사용하여 키사이트(Keysight) EDU36311A 직류 전원 공급 장치를 원격 제어하고 측정하는 PyVISA 기반 테스트 자동화 개념 증명(Proof of Concept, PoC)을 수립하는 것을 목표로 한다. 이 PoC는 임베디드 리눅스 플랫폼에서 표준화된 계측기 제어 프로토콜(SCPI)을 효율적으로 활용하여 정밀한 실험 자동화 환경을 구축하는 방법을 상세히 설명한다 [User Query].

PYNQ 보드는 고성능 처리 시스템(PS)과 프로그래머블 로직(PL, FPGA)을 결합하여, 기존의 데스크톱 기반 자동화 환경(예: Windows/IVI)에서 벗어나, 데이터 수집 및 제어 로직을 하드웨어에 근접하게 배치할 수 있는 이점을 제공한다. 이 접근 방식은 특히 고속 제어 루프나 데이터 전송이 필요한 환경에서 높은 처리량과 낮은 지연 시간을 가능하게 한다. EDU36311A는 LAN(LXI) 및 USB 연결을 지원하는 3채널 직류 전원 공급 장치로, 테스트 벤치 자동화에 적합한 기능을 제공한다.[1]

B. 하드웨어 및 소프트웨어 필수 구성 요소 확인

성공적인 PoC 구현을 위해서는 다음 하드웨어 및 소프트웨어 구성 요소가 필요하다.

1. 제어 플랫폼**:** AUP-ZU3 PYNQ 보드. PYNQ 3.1.1 이미지가 마이크로 SD 카드에 로드되어

있어야 한다.[3]

2. 계측기**:** Keysight EDU36311A 삼중 출력 직류 전원 공급 장치 (90 W, LAN 및 USB 인터페이스 지원).[1]
3. 연결 수단**:** LAN 케이블 (LXI 통신용) 또는 USB 케이블 (USB-TMC 통신용).[4]
4. 소프트웨어 스택**:** Python 3.6 이상 환경에서 PyVISA와 순수 Python 기반 백엔드인 PyVISA-Py가 필요하다.[5] PYNQ 3.1.1 환경은 ARM 아키텍처 기반의 임베디드 리눅스이므로, 데스크톱 환경에서 주로 사용되는 벤더 제공의 32비트/64비트 IVI VISA 라이브러리(.dll,.so)를 사용할 수 없거나 지원이 불안정하다. 따라서 PyVISA-Py를 사용하는 것이 필수적이다.[5]

II. PYNQ AUP-ZU3 하드웨어 및 네트워크 준비

A. PYNQ 보드 초기 설정 및 부팅 과정

AUP-ZU3 PYNQ 보드를 초기화하는 과정은 자동화 환경을 구축하는 첫 단계이다.

1. 물리적 설정

PYNQ 이미지가 로드된 마이크로 SD 카드를 슬롯에 삽입해야 한다.[3] 부팅 스위치를 SD 위치로 설정하여 마이크로 SD 카드에서 운영 체제가 부팅되도록 지정한다.[3] 전원 공급을 위해 USB-C 케이블을 적절한 전원에 연결하고, 제어를 위해 또 다른 USB-C 케이블을 컴퓨터에 연결한다 (USB 가젯 모드 또는 이더넷 어댑터를 통한 연결).[4]

2. 부팅 시퀀스 확인

전원을 켜면 보드의 상태를 나타내는 LED 시퀀스를 통해 부팅 진행 상황을 확인할 수 있다. 먼저 전원 LED(ON LED)가 켜지는 것을 확인해야 하며, 이후 두 개의 PS LED(PS LED0, PS LED1)가 켜지고, PS LED1이 "심장 박동(heartbeat)" 패턴으로 깜박이기 시작한다.[4] 약 30초에서 60초 후에 DONE LED가 켜지는데, 이는 FPGA 비트스트림 다운로드가 완료되었음을 의미하는 FPGA

완료 신호이다.[4] 몇 초 후 PYNQ 운영 체제가 부팅을 완료하면 8개의 흰색 사용자 LED(LD0-LD7)가 깜박이고 4개의 RGB LED(RGB0-RGB3)가 교대로 색상을 깜박이며 보드가 준비되었음을 나타낸다.[4]

3. JupyterLab 접속

보드가 준비되면 호스트 컴퓨터에서 웹 브라우저(Chrome, Safari, Firefox 등 지원되는 브라우저 권장)를 열고 기본 IP 주소인 http://192.168.3.1/lab으로 접속한다. 기본 암호는 xilinx이다.[3] 이 JupyterLab 환경이 PyVISA 스크립트를 작성하고 실행할 자동화 환경이 된다.

B. LXI(LAN) 제어를 위한 네트워크 구성

PYNQ 보드에서 계측기 제어를 시작하기 전에, 네트워크 연결을 구성하는 것이 PyVISA 설치와 PoC 수행에 있어 가장 중요하다.

1. PYNQ의 기본 네트워크 모드와 의존성 설치 문제

PYNQ 보드는 기본적으로 DHCP 서버로 작동한다. 이는 PYNQ에 직접 연결된 호스트 컴퓨터에 IP 주소를 할당하는 가장 쉬운 방법이지만 [7], 이 모드에서는 PYNQ 보드가 외부 네트워크(인터넷)로부터 격리된다.[7] 이 네트워크 격리는 핵심 소프트웨어 스택인 PyVISA, PyVISA-Py 및 기타 시스템 라이브러리(libusb, pyusb)를 설치하는 데 치명적인 문제를 야기한다. 필수 패키지를 설치하려면 PYNQ 보드가 인터넷에 접근할 수 있어야 한다.

이 문제를 해결하는 방법은 두 가지이다:

1. **PYNQ DHCP** 클라이언트 모드 및 인터넷 공유**:** 호스트 컴퓨터가 PYNQ 보드에 인터넷 연결을 공유(브리징)하도록 설정해야 한다.[7] 이 방식은 복잡하지만, PYNQ에서 apt-get 및 pip 설치를 가능하게 한다.
2. 오프라인 설치**:** 필요한 모든 Python 및 시스템 패키지를 미리 다운로드하여 SD 카드에 저장한 후 오프라인으로 설치한다. (본 PoC에서는 편의상 인터넷 연결이 가능하다고 가정하고 진행한다.)

2. LXI 통신을 위한 전용 네트워크 구성

EDU36311A는 LAN(LXI Core) 연결을 지원하므로, 네트워크를 통한 원격 제어가 가능하다.[1] 안정적인 자동화를 위해 EDU36311A와 PYNQ 보드를 동일한 사설 네트워크에 연결하거나 직접 연결해야 한다.

- 정적 **IP** 설정 권장**:** DHCP 환경은 IP 주소 변경 가능성이 있어 자동화 시스템의 견고성을 해칠 수 있다. 따라서 EDU36311A에 고정 IP 주소를 할당하는 것이 강력히 권장된다.[8] 예를 들어, 전원 공급 장치에 192.168.1.100과 같은 정적 IP를 설정하고, PYNQ 보드 또한 192.168.1.101과 같이 동일한 서브넷에 속하는 정적 IP를 갖도록 구성해야 한다.[9]
- 네트워크 확인**:** PYNQ 보드의 Jupyter Terminal에서 ping <EDU36311A_IP> 명령어를 사용하여 전원 공급 장치에 네트워크 경로가 성공적으로 설정되었는지 확인한다.[10]

III. PYNQ 리눅스 환경에 PyVISA 제어 스택 설치

PYNQ 보드의 임베디드 리눅스 환경에 PyVISA를 설치하는 과정은 일반적인 데스크톱 리눅스 환경과 유사하지만, 특히 USB 제어 경로를 대비하여 필수적인 시스템 라이브러리 설치가 필요하다.

A. 핵심 PyVISA 구성 요소 설치

PYNQ는 ARM 아키텍처 기반이므로, PyVISA는 순수 Python으로 구현된 PyVISA-Py 백엔드를 명시적으로 또는 기본값으로 사용하게 된다.[5]

JupyterLab 내의 Terminal에서 다음 명령어를 실행하여 PyVISA 및 PyVISA-Py를 설치하거나 업데이트한다:

```
Bash
```

```
sudo pip3 install -U pyvisa pyvisa-py
```

pyvisa-py는 별도의 벤더 라이브러리 없이 VISA 표준의 제한된 하위 집합을 구현하므로

임베디드 환경에서 높은 이식성과 신뢰성을 제공한다.[5] TCPIP(LXI) 리소스의 경우, PyVISA-Py는 Python 표준 라이브러리의 socket 모듈에 의존하므로 추가 라이브러리가 필요하지 않다.[12]

B. USB-TMC 제어를 위한 리눅스 시스템 의존성 (선택 사항)

EDU36311A 제어에 LAN(LXI) 대신 USB를 사용하려면, PYNQ 리눅스 시스템에 추가적인 라이브러리가 반드시 설치되어야 한다. 이는 USB 계측기 제어(USB-TMC)를 위해서는 3계층의 소프트웨어 스택이 필요하기 때문이다: PyVISA-Py $\rightarrow$ PyUSB $\rightarrow$ 시스템 libusb.

PyVISA-Py 문서는 USB 리소스를 액세스하려면 PyUSB 설치가 필요함을 명시한다.[12] PyUSB는 다시 리눅스 커널과 상호 작용하기 위해 시스템 수준의 libusb 라이브러리에 의존한다.[13] 만약 이 기본 시스템 라이브러리가 누락되면, PyUSB 설치 또는 런타임에서 오류가 발생하게 된다.[14]

1. 시스템 libusb 라이브러리 설치

PYNQ 리눅스 환경은 데비안 기반이므로 apt-get 명령어를 사용하여 libusb 개발 파일을 설치한다.[15] (인터넷 연결이 필요하다.)

Bash

```
sudo apt-get update
sudo apt-get install libusb-1.0-0-dev
```

2. PyUSB Python 바인딩 설치

libusb 설치 후, PyVISA-Py가 USB 장치와 통신하는 데 사용하는 Python 바인딩인 pyusb를 설치한다.[13]

Bash

```
sudo pip3 install pyusb
```

IV. EDU36311A 연결 분석: LXI 대 USB-TMC

안정성과 설정 난이도를 고려할 때 LXI 연결이 PoC에 가장 권장되는 경로이지만, USB-TMC 연결 또한 전문적인 환경에서 요구될 수 있으므로 두 가지 방식 모두 상세히 다룬다.

A. LXI (LAN) 연결: 권장 경로

LXI는 표준 Python socket 모듈만으로 통신할 수 있어 PYNQ 환경에서 가장 적은 의존성을 요구하며, USB 장치 권한 문제를 완전히 회피할 수 있어 가장 안정적이다.[12]

LXI 리소스 문자열 구성

VISA 리소스 문자열은 인터페이스 유형, 호스트 주소(IP), 장치 이름 및 리소스 클래스를 지정해야 한다.[18]

- 구문: TCPIP[board]::host address::[LAN device name]::INSTR [18]
- 예시: EDU36311A에 고정 IP 192.168.1.100이 할당된 경우, 일반적인 Keysight LXI 장치의 기본 장치 이름인 inst0를 사용하여 다음과 같이 리소스 문자열을 구성한다.[19]
  TCPIP0::192.168.1.100::inst0::INSTR

B. USB-TMC 연결: 대체 경로 및 권한 관리

USB-TMC (Test and Measurement Class)를 통한 연결은 LXI를 사용할 수 없는 환경에서

유용하지만, 임베디드 리눅스에서 가장 까다로운 작업 중 하나인 장치 파일 권한 문제를 수반한다.

1. 계측기 **ID** 확인

USB 리소스 문자열을 구성하려면 EDU36311A의 Vendor ID (VID)와 Product ID (PID)가 필요하다.[20]

- Keysight EDU36311A VID: 0x2A8D [21]
- Keysight EDU36311A PID: 0x8F01 [21]
- **USB** 리소스 문자열 구문**:** USB[board]::manufacturer ID::model code::serial number.[18]

2. 리눅스 **USB** 권한 설정 **(udev** 규칙**)**

PYNQ 보드의 리눅스 배포판에서 기본 사용자(xilinx 또는 pynq)는 일반적으로 /dev/bus/usb 아래의 원시 USB 장치 파일에 대한 읽기/쓰기 권한이 없다. PyVISA-Py가 PyUSB를 통해 장치와 통신하려고 할 때, 이 권한 부족으로 인해 Permission denied 또는 No such device 오류가 발생하며 장치를 목록화하지 못한다.[12]

이 문제를 해결하려면, 해당 EDU36311A 장치에 대해서만 특정 사용자 그룹에 접근 권한을 부여하는 udev 규칙을 작성해야 한다.

udev 규칙 생성 및 적용 절차**:**

1. 규칙 파일 생성**:** Jupyter Terminal에서 다음 명령어를 사용하여 새 udev 규칙 파일을 생성한다.
   Bash
   sudo nano /etc/udev/rules.d/99-keysight-usb.rules
2. 규칙 내용 삽입**: VID 2a8d**와 **PID 8f01**에 해당하는 모든 **USB** 장치에 대해 **plugdev** 그룹**(PYNQ** 사용자가 일반적으로 속해 있는 그룹**)**에 읽기**/**쓰기 권한을 부여하는

규칙을 삽입한다.[22]

Keysight EDU36311A (VID 0x2A8D, PID 0x8F01)

SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="2a8d", ATTRS{idProduct}=="8f01", GROUP="plugdev", MODE="0660"

3. **udev** 규칙 다시 로드 및 트리거**:** 시스템이 새 규칙을 인식하도록 udev를 재로드하고 장치 변경을 트리거한다.
   Bash
   sudo udevadm control --reload-rules
   sudo udevadm trigger

4. 장치 재연결**:** EDU36311A를 PYNQ 보드에서 분리했다가 다시 연결하여 새 규칙이 장치에 적용되도록 한다.

C. VISA 리소스 식별자 표

성공적인 연결을 위해 필요한 EDU36311A의 VISA 리소스 식별자를 요약하면 다음과 같다.

Keysight EDU36311A VISA 리소스 식별자

인터페이스 유형	표준 **VISA** 구문	**EDU36311A** 필수 매개변수	예시 리소스 문자열
LAN (LXI)	TCPIP[board]::host address::INSTR	EDU36311A의 IP 주소 (예: 192.168.1.100)	TCPIP0::192.168.1.100::inst0::INSTR

USB-TMC	USB[board]::VID::PID::Serial::INSTR	VID: 0x2A8D, PID: 0x8F01, 고유 시리얼 번호	USB0::0x2A8D::0x8F01::CN60350010::0::INSTR

V. PyVISA 테스트 자동화 개념 증명: 채널 1 제어

이 섹션에서는 PYNQ Jupyter Notebook 환경에서 PyVISA를 사용하여 Keysight EDU36311A의 핵심 기능을 제어하고 측정하는 PoC 코드를 실행한다. PoC는 고전류 출력을 지원하는 채널 1 (6 V / 5 A)을 대상으로 한다.[2]

A. 초기화 및 리소스 연결

PyVISA를 가져오고 ResourceManager 객체를 생성한다. ARM 환경이므로 @py 옵션을 사용하여 PyVISA-Py 백엔드를 명시적으로 지정하는 것이 권장된다.[23]

Python

```
import pyvisa
import time

# 1. 리소스 문자열 설정 (LXI 예시 사용)
# 사용자의 환경에 맞게 IP 주소를 수정해야 합니다.
RESOURCE_STRING = 'TCPIP0::192.168.1.100::inst0::INSTR'

# 2. ResourceManager 초기화 (PyVISA-Py 백엔드 명시)
# 만약 백엔드를 찾지 못하는 OSError가 발생하면, 수동으로 경로를 지정할 수 있습니다.
try:
    rm = pyvisa.ResourceManager('@py')
except OSError as e:
    print(f"Error initializing ResourceManager: {e}")
    print("Ensure PyVISA and PyVISA-Py are installed correctly.")
    exit()
```

```
# 3. 사용 가능한 리소스 목록 확인 및 연결 검증
print("검색된 리소스 목록:", rm.list_resources()) # 장치가 성공적으로 감지되었는지 확인

# 4. EDU36311A에 연결
try:
    inst = rm.open_resource(RESOURCE_STRING)
    # 기기 응답 지연을 고려하여 타임아웃을 5초로 설정
    inst.timeout = 5000

except pyvisa.errors.VisaIOError as e:
    print(f"장치 연결 실패: {e}")
    print("네트워크 설정(IP, 포트) 또는 USB udev 규칙을 확인하십시오.")
    exit()
```

B. 계측기 확인 및 초기 설정

1. 장치 식별

가장 먼저 *IDN? (Identify) 쿼리 명령을 사용하여 계측기와 양방향 통신이 성공했는지 확인한다. 이 명령은 제조사, 모델명, 시리얼 번호 등을 반환한다.[6]

Python

```
identity = inst.query("*IDN?")
print(f"연결된 장치 정보: {identity.strip()}")
```

2. SCPI 채널 지정의 중요성

EDU36311A는 3채널 전원 공급 장치이므로, 출력과 관련된 모든 SCPI 명령(예: APPL, OUTP,

MEAS)은 반드시 채널 지정자(CH1, (@1), (@2), (@1:3) 등)를 포함해야 한다.[8] 이 채널 지정자가 누락되거나 잘못되면 명령이 의도치 않은 채널에 적용되거나 오류가 발생한다.

C. 출력 프로그래밍 및 활성화

채널 1에 원하는 전압과 최대 전류 제한을 설정하고 출력을 활성화한다.

Python

```
TARGET_VOLTAGE = 3.3  # V
CURRENT_LIMIT = 0.5   # A

# 1. 설정값 입력 (APPL CH1, <전압>, <전류>)
# 채널 1에 3.3V, 0.5A 제한을 설정
inst.write(f"APPL CH1, {TARGET_VOLTAGE}, {CURRENT_LIMIT}")
print(f"채널 1: 목표 전압 {TARGET_VOLTAGE} V, 전류 제한 {CURRENT_LIMIT} A 설정 완료.")

# 2. 출력 활성화 (OUTP ON,(@1))
inst.write("OUTP ON,(@1)") # 채널 1 출력 ON
print("채널 1 출력 활성화.")

# 3. 전압 안정화를 위해 잠시 대기 (필수)
# 전원 공급 장치가 설정된 값에 도달하고 안정화될 시간을 확보한다.
time.sleep(0.5)
```

D. 실시간 측정 및 데이터 검색

출력이 활성화된 후, MEAS:VOLT? 및 MEAS:CURR? 쿼리 명령을 사용하여 전원 공급 장치가 부하에 공급하는 실제 전압과 전류를 측정한다.[8]

Python

```
# 1. 실제 출력 전압 측정 (MEAS:VOLT? (@1))
# PyVISA의 query() 메서드는 명령을 보내고 응답을 읽는다. 응답은 문자열이므로 float으로 변환
measured_voltage_str = inst.query("MEAS:VOLT? (@1)")
measured_voltage = float(measured_voltage_str)

# 2. 실제 출력 전류 측정 (MEAS:CURR? (@1))
measured_current_str = inst.query("MEAS:CURR? (@1)")
measured_current = float(measured_current_str)

print("-" * 30)
print(f"측정 결과:")
print(f"설정 전압: {TARGET_VOLTAGE} V")
print(f"실제 출력 전압: {measured_voltage:.4f} V")
print(f"실제 출력 전류: {measured_current:.4f} A")
print("-" * 30)
```

E. 정리 및 세션 종료

테스트 자동화 스크립트의 마지막 단계는 항상 계측기를 안전한 상태로 되돌리고 리소스를 해제하는 것이다. 이는 안전상의 필수 조치이다.

Python

```
# 1. 출력 비활성화 (OUTP OFF,(@1))
inst.write("OUTP OFF,(@1)")
print("채널 1 출력 비활성화.")

# 2. VISA 연결 종료 및 리소스 해제
inst.close()
rm.close()
print("PyVISA 리소스 해제 완료.")
```

F. PoC를 위한 필수 SCPI 명령어 요약

EDU36311A 제어에 사용되는 핵심 SCPI 명령어를 정리하면 다음과 같다.

PoC를 위한 EDU36311A 핵심 SCPI 명령어 (채널 1)

기능	**SCPI** 명령어	**PyVISA** 메서드	설명
장치 확인	*IDN?	inst.query(...)	통신 확인 및 장치 정보 반환.
매개변수 설정	APPL CH1, 5.0, 1.0	inst.write(...)	채널 1의 전압(5.0V) 및 전류 제한(1.0A) 설정.
출력 활성화	OUTP ON, (@1)	inst.write(...)	채널 1의 전원 출력 활성화.
전압 측정	MEAS:VOLT? (@1)	inst.query(...)	채널 1의 실제 출력 전압 측정.
전류 측정	MEAS:CURR? (@1)	inst.query(...)	채널 1의 실제 출력 전류 측정.
출력 비활성화	OUTP OFF, (@1)	inst.write(...)	채널 1의 전원 출력 비활성화.

VI. 고급 자동화 기법 및 오류 처리

PYNQ 환경에서 PyVISA PoC를 산업 또는 연구 수준으로 확장하기 위해서는 반복적인 테스트 수행 능력과 안정적인 오류 처리 메커니즘이 필수적이다.

A. 반복 테스트 구현 (전압 스윕 예시)

자동화의 핵심은 전압이나 전류 설정을 체계적으로 변경하면서 부하의 응답을 측정하고 기록하는 스윕 테스트를 수행하는 것이다. PYNQ의 Python 환경은 이러한 반복 루프를 완벽하게 지원한다.

다음 코드는 1.0 V에서 5.0 V까지 1.0 V 단위로 전압을 증가시키며 부하 전류를 측정하는 예시이다.

Python

```
# 전압 스윕 테스트 설정
sweep_voltages = [1.0, 2.0, 3.0, 4.0, 5.0]
CURRENT_LIMIT = 2.0 # 안전을 위해 충분한 전류 제한 설정
results =

# 채널 1에 초기 전류 제한 설정
inst.write(f"APPL CH1, {sweep_voltages}, {CURRENT_LIMIT}")
inst.write("OUTP ON,(@1)")
time.sleep(1.0) # 출력 안정화 대기

for v_set in sweep_voltages:
    try:
        # 새로운 전압 설정
        inst.write(f"VOLT {v_set}, (@1)") # VOLT 명령은 APPL과 달리 전류 제한을 유지하면서 전압만 변경
[25]
        time.sleep(0.5) # 전압 조정 및 안정화 대기

        # 측정
        v_meas = float(inst.query("MEAS:VOLT? (@1)"))
        i_meas = float(inst.query("MEAS:CURR? (@1)"))

        results.append({'Set Voltage': v_set, 'Measured Voltage': v_meas, 'Measured Current': i_meas})
        print(f"V_set: {v_set:.1f} V, V_meas: {v_meas:.4f} V, I_meas: {i_meas:.4f} A")

    except pyvisa.errors.VisaIOError as e:
        print(f"통신 오류 발생: {e}")
        break

# 결과 출력 및 정리
```

```
inst.write("OUTP OFF,(@1)")
print("스윕 테스트 완료 및 출력 비활성화.")
# 데이터 로깅 로직 (예: CSV 파일로 저장) 추가 가능
```

측정 안정성의 보장은 자동화 테스트의 신뢰도에 직결된다. 전원 공급 장치가 APPL 명령을 처리하고 내부 출력단계를 조정하여 설정된 전압 또는 전류에 도달할 때까지는 일정 시간이 소요된다. 따라서 time.sleep()을 사용하여 측정 전에 충분한 지연 시간을 주는 것은 매우 중요하다.[23]

B. 오류 처리 및 타임아웃 관리

임베디드 리눅스 플랫폼에서 네트워크(LXI) 또는 USB-TMC 통신을 사용할 때, 느린 응답 시간이나 네트워크 지연으로 인해 PyVISA 타임아웃 오류 (VI_ERROR_TMO)가 발생할 수 있다.[23] 이는 특히 쿼리 작업에서 응답을 기다릴 때 흔히 발생한다.

1. 타임아웃 설정 조정

PyVISA는 리소스 객체에 대한 타임아웃 속성을 제공하며, 이를 밀리초 단위로 늘려서 해결할 수 있다. 기본값이 2000 ms인 경우, 다음과 같이 5000 ms(5초)로 늘릴 수 있다.

Python

```
inst.timeout = 5000 # 타임아웃을 5초로 설정
```

2. 쿼리 지연 **(Query Delay)** 활용

일부 계측기는 명령을 수신한 후 응답을 준비하는 데 시간이 걸린다. PyVISA는 쓰기 작업과 읽기 작업 사이에 자동으로 지연 시간을 추가하는 query_delay 속성을 제공하여 이러한 통신 지연을 처리할 수 있다.[23]

Python

```
inst.query_delay = 0.1 # 쓰기 후 100 ms 대기 후 읽기 시작
```

3. 안전한 리소스 해제 (try...finally 블록)

가장 중요한 안전 조치는 스크립트 실행 중 예기치 않은 오류가 발생하더라도 계측기 출력을 반드시 비활성화하도록 보장하는 것이다. 이는 try...finally 블록을 사용하여 구현해야 한다.

Python

```
try:
    # 모든 설정 및 측정 로직 (V.C, V.D, VI.A 섹션 내용)
    inst.write("APPL CH1, 5.0, 1.0")
    inst.write("OUTP ON,(@1)")
    #... 측정 루프...

finally:
    # 스크립트가 성공하거나 실패하더라도 이 블록은 항상 실행된다.
    inst.write("OUTP OFF,(@1)") # 안전을 위해 출력 비활성화
    inst.close()
    rm.close()
    print("리소스 최종 정리 완료.")
```

C. 다중 채널 프로그래밍

EDU36311A는 3개의 독립적인 출력을 제공하며 [2], PyVISA를 통해 여러 채널을 동시에 제어하거나 측정할 수 있다. 채널 리스트 구문 (@N,M) 또는 범위 구문 (@N:M)을 사용하여 이 기능을 활용한다.[8]

Python

```
# 채널 2와 채널 3에 동시에 명령 적용
# 채널 2: 20V/0.5A, 채널 3: 10V/0.2A 설정
inst.write("APPL CH2, 20.0, 0.5")
inst.write("APPL CH3, 10.0, 0.2")

# 채널 2와 3 동시에 출력 활성화
inst.write("OUTP ON,(@2,3)")

time.sleep(0.5)

# 채널 2와 3의 전압 측정
volt_ch2 = float(inst.query("MEAS:VOLT? (@2)"))
volt_ch3 = float(inst.query("MEAS:VOLT? (@3)"))

print(f"CH2 측정 전압: {volt_ch2} V, CH3 측정 전압: {volt_ch3} V")

# 다중 채널 출력 비활성화
inst.write("OUTP OFF,(@2,3)")
```

VII. 요약 및 결론

본 보고서는 AUP-ZU3 PYNQ 보드와 Keysight EDU36311A를 통합한 PyVISA 기반 테스트 자동화 PoC를 성공적으로 수립하기 위한 상세한 튜토리얼을 제공하였다. PYNQ의 임베디드 리눅스 환경은 PyVISA를 활용하여 고성능 계측기 자동화를 구현할 수 있는 강력한 플랫폼임을 입증하였다.

PoC의 성공은 하드웨어와 소프트웨어 스택의 신중한 사전 구성에 달려 있다. 특히 다음 세 가지 기술적 과제가 중요하게 다루어졌다.

1. **PyVISA** 백엔드 선택**:** ARM 아키텍처 환경에서는 범용 IVI-VISA 라이브러리 대신 순수 Python으로 구현된 PyVISA-Py를 사용하는 것이 유일하고 안정적인 경로임을 확인하였다.
2. 네트워크 연결 전략**:** PYNQ의 기본 네트워크 격리 모드(DHCP 서버)가 외부 패키지 설치를 방해한다는 점을 고려하여, 초기 환경 구축 시 인터넷 접근성을 확보하거나 LXI(LAN) 연결 시 정적 IP 설정을 통해 견고성을 확보하는 것이 중요함을 설명하였다. LXI는 USB-TMC에 비해 시스템 의존성이 낮아 PoC에 가장 권장되는 방식이다.

3. **USB** 권한 관리**:** 만약 USB-TMC를 사용해야 한다면, PYNQ의 비-루트 사용자가 USB 장치에 접근하기 위해 libusb, pyusb 라이브러리 스택을 설치하는 것 외에도, Keysight 장치의 VID/PID에 맞춘 리눅스 udev 규칙을 반드시 설정하여 장치 파일에 대한 접근 권한을 명시적으로 부여해야 함을 강조하였다.

EDU36311A 제어 측면에서는, 삼중 출력 전원 공급 장치의 특성상 모든 SCPI 명령에 채널 지정자(CH1 또는 (@N))를 포함해야 하며, APPL, OUTP ON/OFF, MEAS:VOLT?, MEAS:CURR?와 같은 표준 SCPI 명령어를 사용하여 완벽하게 프로그래밍 가능한 제어가 가능함을 입증하였다.[8]

결론적으로, PYNQ 보드는 단순한 임베디드 시스템을 넘어, 하이레벨 Python 제어 환경과 계측기 제어 프로토콜을 성공적으로 통합하는 강력하고 유연한 자동화 호스트 역할을 수행할 수 있다. 향후 이러한 PoC는 PYNQ의 PL(FPGA) 영역을 활용하여 초고속 데이터 수집 트리거 또는 실시간 제어 루프를 구현하는 방향으로 확장될 수 있다.

참고 자료

1. EDU36311A Smart Bench Essentials DC Power Supply - Keysight, 11월 5, 2025에 액세스, https://www.keysight.com/sg/en/product/EDU36311A/smart-bench-essentials-dc-power-supply-triple-outputs.html
2. Keysight EDU36311A DC Power Supply, Triple Output, 6V/5A, 2x 30V/1A, 90W, EDU Series, 11월 5, 2025에 액세스, https://www.testequity.com/product/31488-1-EDU36311A
3. Getting started with your PYNQ-ZU, 11월 5, 2025에 액세스, https://xilinx.github.io/PYNQ-ZU/getting_started.html
4. Getting started with your AUP-ZU3 | XUP PYNQ-ZU, 11월 5, 2025에 액세스, https://xilinx.github.io/AUP-ZU3/getting_started.html
5. Installation — PyVISA 1.15.1.dev34+gf58902ba8 documentation - Read the Docs, 11월 5, 2025에 액세스, https://pyvisa.readthedocs.io/en/latest/introduction/getting.html
6. PyVISA: Control your instruments with Python — PyVISA 1.15.1.dev34+gf58902ba8 documentation, 11월 5, 2025에 액세스, https://pyvisa.readthedocs.io/
7. Internet on the PYNQ board, 11월 5, 2025에 액세스, https://pynq.tue.nl/general/internet/
8. EDU36311A Triple Output Programmable DC Power Supply - User's Guide - Distrelec, 11월 5, 2025에 액세스, https://media.distrelec.com/Web/Downloads/_m/an/EDU36311A_mul_man.pdf
9. Assign a static IP address — Python productivity for Zynq (Pynq) - Read the Docs, 11월 5, 2025에 액세스, https://pynq.readthedocs.io/en/v2.7.0/appendix/assign_a_static_ip.html
10. Finding Your Linux IP Address - University of Rochester, 11월 5, 2025에 액세스, https://tech.rochester.edu/tutorials/finding-your-linux-ip-address/
11. Command to Check IP address in linux - GeeksforGeeks, 11월 5, 2025에 액세스, https://www.geeksforgeeks.org/linux-unix/command-to-check-ip-address-in-linux/

12. Installation — PyVISA-Py 0.7.3.dev21+gdb87594 documentation, 11월 5, 2025에 액세스, https://pyvisa.readthedocs.io/projects/pyvisa-py/en/latest/installation.html
13. pyusb/pyusb: Easy USB access for Python - GitHub, 11월 5, 2025에 액세스, https://github.com/pyusb/pyusb
14. python-ivi/python-usbtmc: Provides a USBTMC driver for controlling instruments over USB, 11월 5, 2025에 액세스, https://github.com/python-ivi/python-usbtmc
15. How to install libusb in Ubuntu - Stack Overflow, 11월 5, 2025에 액세스, https://stackoverflow.com/questions/4853389/how-to-install-libusb-in-ubuntu
16. How can I install libusb? - Ask Ubuntu, 11월 5, 2025에 액세스, https://askubuntu.com/questions/629619/how-can-i-install-libusb
17. How to Install PyUSB on Linux? - GeeksforGeeks, 11월 5, 2025에 액세스, https://www.geeksforgeeks.org/installation-guide/how-to-install-pyusb-on-linux/
18. VISA Resource Syntax and Examples - NI - National Instruments, 11월 5, 2025에 액세스, https://www.ni.com/docs/en-US/bundle/ni-visa/page/visa-resource-syntax-and-examples.html
19. EDU36311A Triple Output Programmable DC Power Supply User's Guide (English, French, and Spanish) - People, 11월 5, 2025에 액세스, https://people.ece.ubc.ca/~eng-services/files/manuals/Man_PowerSupply_KEY-EDU36311A.pdf
20. Resource classes — PyVISA 1.15.1.dev33+gb1bd8f250 documentation, 11월 5, 2025에 액세스, https://pyvisa.readthedocs.io/en/latest/api/resources.html
21. EDU36311A Triple Output Programmable DC Power Supply, 11월 5, 2025에 액세스, https://www.datatec.eu/media/b4/21/8c/1682505428/Keysight-EDU36311A-HB-1_DE.pdf?ts=1691652865
22. Can't connect to usb instrument · Issue #3 · pyvisa/pyvisa-py - GitHub, 11월 5, 2025에 액세스, https://github.com/pyvisa/pyvisa-py/issues/3
23. Communicating with your instrument — PyVISA 1.15.1.dev33+gb1bd8f250 documentation, 11월 5, 2025에 액세스, https://pyvisa.readthedocs.io/en/latest/introduction/communication.html
24. Python PyVISA USB connection issue - Stack Overflow, 11월 5, 2025에 액세스, https://stackoverflow.com/questions/36337335/python-pyvisa-usb-connection-issue
25. PyVisa-USB timeout error cannot be cleared without reboot? - Stack Overflow, 11월 5, 2025에 액세스, https://stackoverflow.com/questions/73759223/pyvisa-usb-timeout-error-cannot-be-cleared-without-reboot