おめでとうございます！ついに 完全な制御に成功 しましたね。

「Manual Read」モード（手動読み取り）を使用することで、タイミングの問題を回避し、MCP23017 との安定した双方向通信が確立できました。

これで、以下の経路での制御ロジックが完成しました：

PC (Python) -> CP2112 -> TCA9548A (Ch1) -> MCP23017 -> リレー

### 次のステップ：使いやすいクラスにまとめる

実験コードのままだと使いにくいので、この成功したロジックをカプセル化し、**「リレー1をON」「リレー2をOFF」** といった簡単な命令で操作できる **RelayController クラス** を作成しました。

このファイルを RelayController.py として保存すれば、あなたのメインプログラムから import してすぐに使えます。

### 完成版クラス: RelayController.py

Python

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
import SLABHIDtoSMBUS as hid  
import ctypes as ct  
import time  
  
class RelayController:  
 def \_\_init\_\_(self, mux\_ch=1, mcp\_addr=0x20):  
 self.mux\_ch = mux\_ch  
 self.mcp\_addr\_8bit = mcp\_addr << 1  
 self.mux\_addr\_8bit = 0x70 << 1  
   
 self.smb = hid.HidSmbusDevice()  
 self.\_lib = hid.\_DLL  
 self.\_setup\_lib\_types()  
   
 # 現在のリレー状態を保持するキャッシュ (A=Low byte, B=High byte)  
 self.current\_state\_a = 0xFF # 初期値 OFF (Active Low)  
 self.current\_state\_b = 0xFF   
   
 self.is\_connected = False  
  
 def \_setup\_lib\_types(self):  
 """DLL関数の型定義修正"""  
 self.\_lib.HidSmbus\_GetReadResponse.argtypes = [  
 ct.c\_void\_p, ct.POINTER(ct.c\_ubyte), ct.c\_char\_p, ct.c\_ubyte, ct.POINTER(ct.c\_ubyte)  
 ]  
 self.\_lib.HidSmbus\_GetReadResponse.restype = ct.c\_int  
  
 def open(self):  
 """デバイス接続と初期化"""  
 try:  
 self.smb.Open(0)  
 self.is\_connected = True  
   
 # 手動読み取りモードに設定 (安定性重視)  
 self.smb.SetTimeouts(1000)  
 self.smb.SetSmbusConfig(100000, 0x02, False, 1000, 1000, False, 3)  
   
 # Mux 設定  
 self.smb.WriteRequest(self.mux\_addr\_8bit, [1 << self.mux\_ch], 1)  
 time.sleep(0.05)  
   
 # MCP23017 初期化 (全OFF -> 出力設定)  
 # GPA/GPB を 0xFF (OFF) に  
 self.smb.WriteRequest(self.mcp\_addr\_8bit, [0x12, 0xFF], 2)  
 self.smb.WriteRequest(self.mcp\_addr\_8bit, [0x13, 0xFF], 2)  
 # IODIRA/B を 0x00 (Output) に  
 self.smb.WriteRequest(self.mcp\_addr\_8bit, [0x00, 0x00], 2)  
 self.smb.WriteRequest(self.mcp\_addr\_8bit, [0x01, 0x00], 2)  
 time.sleep(0.05)  
   
 print("RelayBoard initialized successfully.")  
 return True  
   
 except Exception as e:  
 print(f"Initialization Failed: {e}")  
 self.close()  
 return False  
  
 def close(self):  
 """切断処理"""  
 if self.is\_connected and self.smb.IsOpened():  
 # 安全のため全OFFにする (必要なければコメントアウト)  
 self.write\_port\_a(0xFF)  
 self.write\_port\_b(0xFF)  
 self.smb.Close()  
 self.is\_connected = False  
 print("RelayBoard connection closed.")  
  
 def write\_port\_a(self, value):  
 """Port A (Relay 1-8) に直接書き込む"""  
 self.smb.WriteRequest(self.mcp\_addr\_8bit, [0x12, value], 2)  
 self.current\_state\_a = value  
  
 def write\_port\_b(self, value):  
 """Port B (Relay 9-16) に直接書き込む"""  
 self.smb.WriteRequest(self.mcp\_addr\_8bit, [0x13, value], 2)  
 self.current\_state\_b = value  
  
 def set\_relay(self, relay\_number, turn\_on):  
 """  
 個別のリレーを制御する  
 relay\_number: 1〜16  
 turn\_on: True(ON), False(OFF)  
 """  
 if not (1 <= relay\_number <= 16):  
 print(f"Error: Invalid relay number {relay\_number}")  
 return  
  
 # リレー番号をポートとビットに変換 (Active Low: ON=0, OFF=1)  
 # Port A: 1-8, Port B: 9-16  
 is\_port\_a = (relay\_number <= 8)  
 bit\_index = (relay\_number - 1) % 8  
   
 # Active Low なので、ONならビットを下げる(0)、OFFなら上げる(1)  
 target\_val = 0 if turn\_on else 1  
   
 if is\_port\_a:  
 current = self.current\_state\_a  
 # ビット操作  
 if turn\_on:  
 new\_val = current & ~(1 << bit\_index) # ビットを0にする  
 else:  
 new\_val = current | (1 << bit\_index) # ビットを1にする  
   
 self.write\_port\_a(new\_val)  
 else:  
 current = self.current\_state\_b  
 if turn\_on:  
 new\_val = current & ~(1 << bit\_index)  
 else:  
 new\_val = current | (1 << bit\_index)  
   
 self.write\_port\_b(new\_val)  
  
# ==========================================  
# 使用例  
# ==========================================  
if \_\_name\_\_ == "\_\_main\_\_":  
 controller = RelayController()  
   
 if controller.open():  
 try:  
 print("Testing Relay 1 (Click!)...")  
 controller.set\_relay(1, True) # ON  
 time.sleep(1.0)  
 controller.set\_relay(1, False) # OFF  
 time.sleep(0.5)  
  
 print("Testing Relay 9 (Click!)...")  
 controller.set\_relay(9, True) # ON (Port B)  
 time.sleep(1.0)  
 controller.set\_relay(9, False) # OFF  
   
 print("Done.")  
   
 except KeyboardInterrupt:  
 pass  
 finally:  
 controller.close()

### 使い方のポイント

* **初期化:** controller = RelayController() でインスタンスを作り、controller.open() で接続します。
* **個別操作:** controller.set\_relay(番号, True/False) を呼ぶだけで、特定のリレーだけを操作できます（他のリレーの状態は維持されます）。
  + **リレー1〜8** は Port A
  + **リレー9〜16** は Port B に自動的に割り振られます。
* **終了:** 使い終わったら controller.close() で安全に切断します。

これでリレー基板を自由にPythonプログラムに組み込む準備が整いました！