Chắc chắn rồi! Dưới đây là một chương trình Python hoàn chỉnh để truyền dữ liệu qua cổng COM theo đúng cấu trúc khung truyền bạn yêu cầu. Chương trình sử dụng thư viện numpy để tạo dữ liệu sóng hiệu quả và threading để có thể vừa gửi dữ liệu liên tục, vừa lắng nghe lệnh "exit" để dừng chương trình.

Giới thiệu

Chương trình sẽ thực hiện các công việc sau:

1. **Tạo dữ liệu:** Tạo trước 1024 mẫu cho 5 dạng sóng (3 sóng sin, 2 sóng tam giác) theo đúng biên độ và độ lệch pha.
2. **Đóng gói khung truyền:** Với mỗi bước lặp, lấy 5 giá trị dữ liệu và đóng gói chúng thành một khung 22 byte theo định dạng: Start Byte + Data 1-5 + End Byte.
3. **Gửi dữ liệu:** Gửi khung truyền qua cổng COM đã chọn.
4. **Lắng nghe lệnh dừng:** Sử dụng một luồng (thread) riêng biệt để lắng nghe dữ liệu đến. Nếu nhận được chuỗi "exit", chương trình sẽ dừng lại một cách an toàn.
5. **Lặp lại:** Quá trình gửi dữ liệu được lặp lại liên tục qua 1024 mẫu cho đến khi có lệnh dừng.

---

1. Chuẩn bị

Trước tiên, bạn cần cài đặt hai thư viện cần thiết là pyserial (để giao tiếp cổng COM) và numpy (để xử lý mảng và tạo sóng).

Mở terminal hoặc command prompt và chạy lệnh sau:

Bash

```
pip install pyserial numpy
```

---

2. Mã nguồn chương trình

Sao chép và dán mã nguồn dưới đây vào một tệp tin, ví dụ com_transmitter.py.

Quan trọng: Hãy thay đổi giá trị của biến COM_PORT thành tên cổng COM ảo hoặc thật trên máy của bạn (ví dụ: 'COM3' trên Windows hoặc '/dev/ttyUSB0' trên Linux).

Python

```
import serial
import time
import numpy as np
import struct
import threading

# -- CẤU HÌNH --
# Thay đổi thành cổng COM của bạn (ví dụ: 'COM3' trên Windows, '/dev/ttyUSB0' trên Linux)
COM_PORT = 'COM_PORT_CUA_BAN'
BAUD_RATE = 115200
SEND_INTERVAL_S = 0.01 # Thời gian nghỉ giữa mỗi lần gửi (giây), ví dụ 0.01s = 100Hz

# -- CÁC THAM SỐ SÓNG --
STEPS = 1024
# Dữ liệu 1, 2, 3: Sóng sine
AMP_SINE = 500000
# Dữ liệu 4, 5: Sóng tam giác
AMP_TRIANGLE = 100000

# Biến toàn cục để báo hiệu cho luồng gửi dừng lại
stop_event = threading.Event()

def generate_wave_data():
    """
    Tạo ra 5 bộ dữ liệu sóng và trả về dưới dạng tuple các mảng numpy.
    """
    print("[INFO] Đang tạo dữ liệu sóng...")
    # Tạo một trục thời gian chuẩn hóa từ 0 đến 2*pi
    t = np.linspace(0, 2 * np.pi, STEPS, endpoint=False)

    # Dữ liệu 1: Sóng sine, pha 0 độ
    data1 = (AMP_SINE * np.sin(t)).astype(np.int32)
```

```
    # Dữ liệu 2: Sóng sine, lệch pha 120 độ (2*pi/3)
    data2 = (AMP_SINE * np.sin(t + 2 * np.pi / 3)).astype(np.int32)

    # Dữ liệu 3: Sóng sine, lệch pha 240 độ (4*pi/3)
    data3 = (AMP_SINE * np.sin(t + 4 * np.pi / 3)).astype(np.int32)

    # Dữ liệu 4: Sóng tam giác
    # Tạo sóng răng cưa từ -A đến +A rồi lấy giá trị tuyệt đối, sau đó điều chỉnh
    sawtooth = np.linspace(-AMP_TRIANGLE, AMP_TRIANGLE, STEPS // 2)
    data4 = np.concatenate([sawtooth, sawtooth[::-1]]).astype(np.int32)

    # Dữ liệu 5: Sóng tam giác, lệch pha 90 độ (dịch chuyển 1/4 chu kỳ)
    shift_amount = STEPS // 4
    data5 = np.roll(data4, shift_amount).astype(np.int32)

    print("[INFO] Đã tạo dữ liệu sóng thành công.")
    return data1, data2, data3, data4, data5

def create_frame(d1, d2, d3, d4, d5):
    """
    Đóng gói 5 giá trị số nguyên 32-bit thành một khung truyền 22 byte.
    <i: little-endian signed 32-bit integer
    """
    start_byte = b'\x03'
    end_byte = b'\xFC'

    # Đóng gói 5 số nguyên 32-bit có dấu theo định dạng little-endian
    payload = struct.pack('<iiiii', d1, d2, d3, d4, d5)

    return start_byte + payload + end_byte

def serial_listener(ser):
    """
    Chạy trong một luồng riêng để lắng nghe dữ liệu từ cổng COM.
    Nếu nhận được 'exit', sẽ kích hoạt stop_event.
    """
    while not stop_event.is_set():
        try:
            if ser.in_waiting > 0:
                # Đọc dữ liệu đến, chuyển thành chuỗi và loại bỏ khoảng trắng
                line = ser.readline().decode('utf-8', errors='ignore').strip()
                if line.lower() == 'exit':
                    print("\n[INFO] Nhận được lệnh 'exit'. Đang dừng chương trình...")
```

```
                stop_event.set() # Báo cho luồng chính dừng lại
                break
        except serial.SerialException:
            print("\n[LỖI] Cổng serial đã bị ngắt. Dừng luồng lắng nghe.")
            stop_event.set()
            break
        except Exception as e:
            # Bỏ qua các lỗi nhỏ khác để chương trình không bị crash
            pass
        time.sleep(0.1) # Giảm tải CPU

def main():
    """
    Hàm chính để chạy chương trình.
    """
    # 1. Tạo dữ liệu sóng
    d1, d2, d3, d4, d5 = generate_wave_data()

    # 2. Mở cổng COM và xử lý ngoại lệ
    ser = None
    try:
        ser = serial.Serial(COM_PORT, BAUD_RATE, timeout=1)
        print(f"[INFO] Đã mở cổng {COM_PORT} với baud rate {BAUD_RATE}.")
        # Xóa bộ đệm đầu vào và đầu ra để bắt đầu sạch
        ser.reset_input_buffer()
        ser.reset_output_buffer()

        # 3. Khởi tạo và chạy luồng lắng nghe
        listener_thread = threading.Thread(target=serial_listener, args=(ser,))
        listener_thread.daemon = True  # Luồng sẽ tự động kết thúc khi chương trình chính thoát
        listener_thread.start()
        print("[INFO] Đã khởi động luồng lắng nghe lệnh 'exit'.")
        print("[INFO] Bắt đầu truyền dữ liệu...")

        # 4. Vòng lặp truyền dữ liệu chính
        step_index = 0
        while not stop_event.is_set():
            # Lấy dữ liệu tại bước hiện tại
            val1, val2, val3, val4, val5 = (
                d1[step_index],
                d2[step_index],
                d3[step_index],
                d4[step_index],
```

```
            d5[step_index]
        )

        # Tạo khung truyền
        frame_to_send = create_frame(val1, val2, val3, val4, val5)

        # Gửi khung truyền
        ser.write(frame_to_send)

        # In trạng thái ra màn hình (ghi đè trên cùng một dòng)
        print(f"-> Bước {step_index+1}/{STEPS} | Frame: {frame_to_send.hex(' ')}      \r", end="")

        # Chuyển sang bước tiếp theo, quay vòng khi hết chu kỳ
        step_index = (step_index + 1) % STEPS

        # Đợi một khoảng thời gian ngắn
        time.sleep(SEND_INTERVAL_S)

    except serial.SerialException as e:
        print(f"\n[LỖI] Không thể mở hoặc ghi vào cổng {COM_PORT}: {e}")
        print("[GỢI Ý] Hãy kiểm tra lại tên cổng COM, đảm bảo nó không bị chiếm bởi chương trình khác và đã được cắm vào.")
    except Exception as e:
        print(f"\n[LỖI] Đã xảy ra lỗi không mong muốn: {e}")
    finally:
        # 5. Dọn dẹp và đóng cổng
        if listener_thread and listener_thread.is_alive():
            listener_thread.join(timeout=1) # Chờ luồng con kết thúc

        if ser and ser.is_open:
            ser.close()
            print(f"\n[INFO] Đã đóng cổng {COM_PORT}.")
        print("[INFO] Chương trình đã kết thúc.")

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

---

3. Cách sử dụng

1. **Kết nối phần cứng:** Kết nối thiết bị của bạn vào cổng COM của máy tính. Nếu bạn chỉ muốn thử nghiệm, bạn có thể sử dụng một cặp cổng COM ảo (virtual COM port pair) và một chương trình terminal (như PuTTY, Tera Term, hoặc CoolTerm) để xem dữ liệu.
2. **Chỉnh sửa cấu hình:** Mở tệp com_transmitter.py và thay đổi COM_PORT = 'COM_PORT_CUA_BAN' thành cổng COM chính xác của bạn.
3. **Chạy chương trình:** Mở terminal/command prompt, điều hướng đến thư mục chứa tệp và chạy lệnh:
   Bash
   python com_transmitter.py

4. **Quan sát:** Chương trình sẽ bắt đầu gửi dữ liệu. Trên màn hình, bạn sẽ thấy trạng thái gửi được cập nhật liên tục.
5. **Dừng chương trình:** Để dừng chương trình, bạn cần gửi chuỗi exit (kèm theo ký tự xuống dòng) đến cổng COM mà chương trình đang sử dụng. Nếu bạn đang dùng một chương trình terminal để theo dõi, chỉ cần gõ exit và nhấn Enter. Chương trình Python sẽ nhận được lệnh, in ra thông báo và tự động kết thúc.

Giải thích chi tiết về mã nguồn

- **struct.pack('<iiiii', ...)**: Đây là phần cốt lõi của việc đóng gói.
  - '<': Báo hiệu rằng dữ liệu được sắp xếp theo kiểu **little-endian** (byte có trọng số thấp nhất được đặt ở địa chỉ nhỏ nhất), đúng như yêu cầu của bạn (bit 7-0 đi trước).
  - 'i': Đại diện cho một kiểu dữ liệu **integer 32-bit có dấu** (signed 32-bit integer).
  - 'iiiii': Lặp lại 5 lần để đóng gói 5 giá trị số nguyên.
- **threading.Event()**: Đây là một cơ chế an toàn để giao tiếp giữa các luồng. Luồng chính kiểm tra stop_event.is_set(). Luồng lắng nghe gọi stop_event.set() khi nhận được lệnh "exit". Cách này tốt hơn nhiều so với việc dùng một biến cờ (flag) toàn cục vì nó tránh được các vấn đề về đồng bộ hóa (race conditions).
- **numpy**: Thư viện này cực kỳ mạnh mẽ để tạo và tính toán trên các mảng lớn. Việc tạo trước toàn bộ 1024 điểm dữ liệu sẽ nhanh hơn nhiều so với việc tính toán giá trị sin/cos trong mỗi vòng lặp.
- **Khối try...except...finally**: Đảm bảo rằng dù chương trình có gặp lỗi (ví dụ: không mở được cổng COM) hay kết thúc bình thường, cổng COM sẽ luôn được đóng (ser.close()). Đây là một thói quen lập trình tốt để tránh để lại tài nguyên bị treo.