Cách thêm thư viện vào pycharm:

Chuẩn bị file:



Code:

from .fxplc import FXPLC, RegisterDef, NotSupportedCommandError, NoResponseError, ResponseMalformedError



Code:

import binascii  
import enum  
import logging  
import struct  
from typing import Tuple, Union  
  
import serial  
  
logger = logging.getLogger("fxplc")  
  
  
class NotSupportedCommandError(Exception):  
 pass  
  
  
class NoResponseError(Exception):  
 pass  
  
  
class ResponseMalformedError(Exception):  
 pass  
  
  
STX = b"\x02" # Start of text  
ETX = b'\x03' # End of text  
EOT = b'\x04' # End of transmission  
ENQ = b'\x05' # Enquiry  
ACK = b'\x06' # Acknowledge  
LF = b'\x0A' # Line Feed  
CL = b'\x0C' # Clear  
CR = b'\x0D' # Carrier Return  
NAK = b'\x15' # Not Acknowledge  
  
registers\_map = {  
 "S": 0x0000,  
 "X": 0x0080,  
 "Y": 0x00a0,  
 "T": 0x00c0,  
 "M": 0x0100,  
 "D": 0x1000,  
}  
  
registers\_map\_counter = {  
 "T": 0x0800,  
}  
  
registers\_map\_bits = {  
 "S": 0x0000,  
 "X": 0x0400,  
 "Y": 0x0500,  
 "T": 0x0600,  
 "M": 0x0800,  
}  
  
  
class RegisterType(enum.Enum):  
 State = "S"  
 Input = "X"  
 Output = "Y"  
 Timer = "T"  
 Memory = "M"  
 Data = "D"  
  
  
class RegisterDef:  
 def \_\_init\_\_(self, type: RegisterType, num: int):  
 self.type = type  
 self.num = num  
  
 def \_\_str\_\_(self):  
 return f"{self.type.value}{self.num}"  
  
 def get\_bit\_image\_address(self) -> Tuple[int, int]:  
 top\_address = registers\_map[self.type.value]  
 return top\_address + self.num // 8, self.num % 8  
  
 @staticmethod  
 def parse(definition: str) -> 'RegisterDef':  
 return RegisterDef(type=RegisterType(definition[0]), num=int(definition[1:]))  
  
  
def calc\_checksum(payload):  
 return bytes(f"{sum(payload):02X}"[-2:].encode("ascii"))  
  
  
class FXPLC:  
 def \_\_init\_\_(self, port: str):  
 self.serial = serial.Serial(port=port, timeout=1,  
 baudrate=9600, bytesize=serial.SEVENBITS, parity=serial.PARITY\_EVEN, stopbits=serial.STOPBITS\_ONE)  
  
 def close(self):  
 self.serial.close()  
  
 def read\_bit(self, register: Union[RegisterDef, str]) -> bool:  
 if not isinstance(register, RegisterDef):  
 register = RegisterDef.parse(register)  
 addr, bit = register.get\_bit\_image\_address()  
  
 resp = self.read\_bytes(addr, 1)  
 return (resp[0] & (1 << bit)) != 0  
  
 def write\_bit(self, register: Union[RegisterDef, str], value: bool):  
 if not isinstance(register, RegisterDef):  
 register = RegisterDef.parse(register)  
 top\_address = registers\_map\_bits[register.type.value]  
 addr = top\_address + register.num  
  
 self.\_send\_command(7 if value else 8, struct.pack("<H", addr))  
  
 def read\_counter(self, register: Union[RegisterDef, str]) -> int:  
 if not isinstance(register, RegisterDef):  
 register = RegisterDef.parse(register)  
 addr = registers\_map\_counter[register.type.value] + register.num \* 2  
  
 resp = self.read\_bytes(addr, 2)  
  
 value = struct.unpack("<H", resp)[0]  
 return value  
  
 def read\_bytes(self, addr: int, count: int = 1) -> bytes:  
 req = struct.pack(">HB", addr, count)  
 resp = self.\_send\_command(0, req)  
 return resp  
  
 def write\_bytes(self, addr: int, values: bytes):  
 req = struct.pack(">HB", addr, len(values)) + values  
 self.\_send\_command(1, req)  
  
 def \_send\_command(self, cmd: int, data: bytes) -> bytes:  
 cmd\_hex = bytes([ord("0") + cmd])  
 payload\_hex = binascii.hexlify(data).upper()  
 logger.debug("TX [cmd | payload]: " + cmd\_hex.decode("ascii") + " | " + payload\_hex.decode("ascii"))  
 payload = cmd\_hex + payload\_hex  
  
 frame = STX + payload + ETX + calc\_checksum(payload + ETX)  
  
 self.serial.flushOutput()  
 self.serial.flushInput()  
 self.serial.write(frame)  
  
 return self.\_read\_response()  
  
 def \_read\_response(self):  
 def format\_code(\_code):  
 return f"RX [code]: {binascii.hexlify(\_code).decode('ascii')}"  
  
 def format\_code\_data(\_code, \_data):  
 return f"RX [code | payload]: {binascii.hexlify(\_code).decode('ascii')} | {\_data.decode('ascii')}"  
  
 code = self.serial.read(1)  
 if code == STX:  
 data = b""  
 while True:  
 d = self.serial.read(1)  
 if len(d) == 0:  
 logger.error(f"Invalid response - {format\_code\_data(code, data)}")  
 raise ResponseMalformedError()  
  
 if d == ETX:  
 break  
 data += d  
  
 logger.debug(format\_code\_data(code, data))  
  
 checksum = self.serial.read(2)  
 if len(checksum) != 2:  
 logger.error(f"Invalid response - {format\_code\_data(code, data)}")  
 raise ResponseMalformedError()  
  
 if calc\_checksum(data + ETX) != checksum:  
 logger.error(f"Wrong checksum - {format\_code\_data(code, data)}")  
 raise ResponseMalformedError()  
  
 return binascii.unhexlify(data)  
 elif code == NAK:  
 raise NotSupportedCommandError()  
 elif code == ACK:  
 logger.debug(f"{format\_code(code)} (ACK)")  
 else:  
 raise NoResponseError()  
  
  
\_\_all\_\_ = [  
 "NotSupportedCommandError",  
 "NoResponseError",  
 "ResponseMalformedError",  
 "RegisterType",  
 "RegisterDef",  
 "FXPLC",  
]



Code:

from setuptools import setup  
  
setup(  
 name='fxplc', # Tên thư viện  
 version='1.0.0',  
 description='A library for interfacing with FXPLC devices',  
 author='Your Name',  
 author\_email='your\_email@example.com',  
 packages=['fxplc\_lib'], # Tên package (thư mục)  
 install\_requires=['pyserial'], # Các thư viện phụ thuộc  
)

cấu trúc thư viện:

[tên project]-[fxplc\_lib]-[ \_\_init\_\_.py]

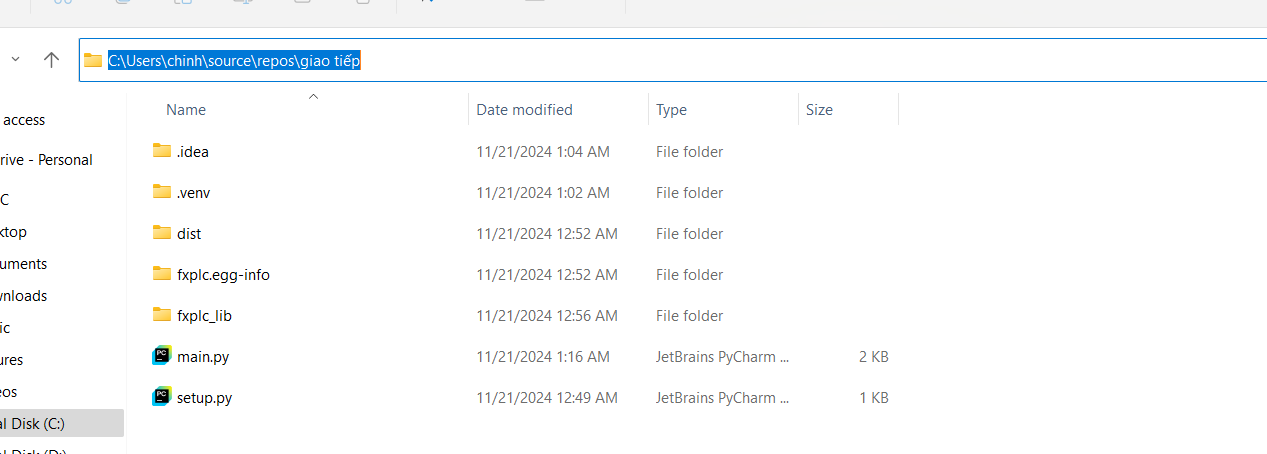
-[fxplc.py]

-[setup.py]

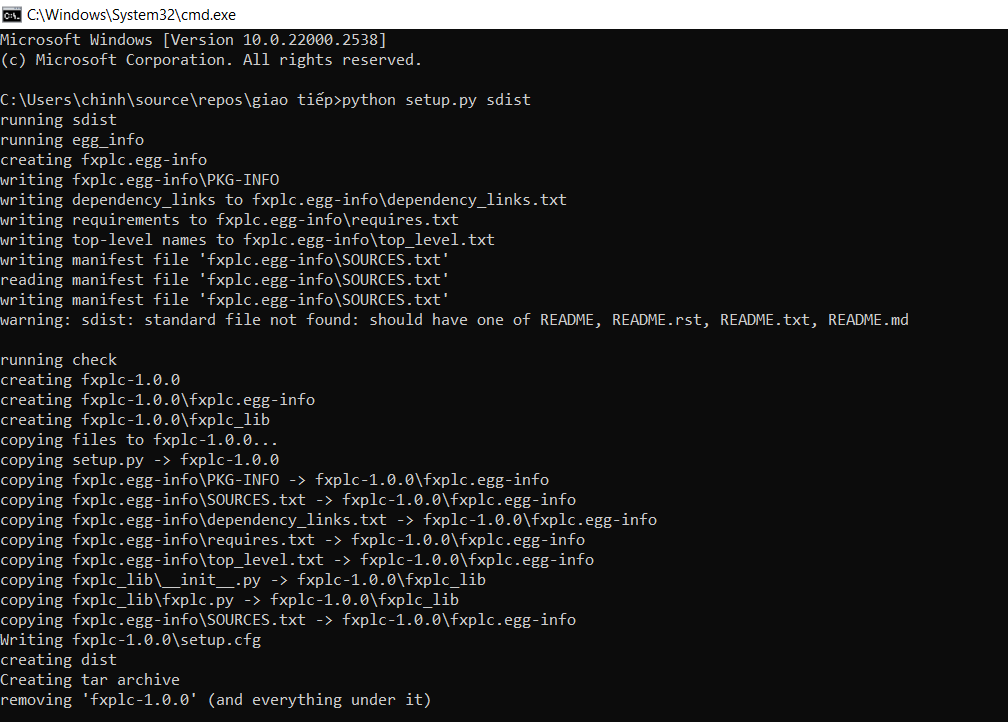
-[test]

Cách thực hiện:

* mở cmd nơi chứa các file thư viện nơi chứa file setup.py :

  
chạy dòng : python setup.py sdist

Để đóng gói thư viện , trường hợp xuất xuất hiện báo lỗi thiếu file .md thì kệ!



* **Cài đặt thư viện vào môi trường ảo (virtual environment)**:
* Mở terminal trong PyCharm.
* Chạy lệnh:

pip install /path/to/your/fxplc\_lib/dist/fxplc-1.0.0.tar.gz

và sử dụng bình thường!