**醫學信號處理系統架構規劃 (從Delphi到Python轉換)**

基於您的需求，我將為您規劃一個完整的醫學信號處理系統轉換與開發方案，特別針對心電圖(ECG)分析、HRV和多重尺度熵分析。

**系統架構規劃**

**1. 分層式架構建議**

text

[硬體層]

↑

[串口通訊層] (comport.pas 轉換)

↑

[信號採集控制層] (wl\_controlmc.pas/dfm 轉換)

↑

[信號預處理層] (濾波、去噪)

↑

[特徵檢測層] (R-Peak、RR間期)

↑

[分析層] (HRV、MSE)

↑

[UI展示層] (Qt Designer)

**2. 模組轉換優先順序**

1. **基礎通訊模組** (comport.pas → Python)
2. **控制界面模組** (wl\_controlmc → PyQt)
3. **核心算法模組** (新增ECG分析)

**一、基礎模組轉換策略**

**1. 串口通訊模組 (comport.pas → Python)**

**Python替代方案:**

python

import serial

from serial.tools import list\_ports

class ComPort:

def \_\_init\_\_(self):

self.ser = serial.Serial()

self.baudrate = 115200

self.timeout = 1

def list\_ports(self):

"""替代Delphi中的列舉串口功能"""

return [port.device for port in list\_ports.comports()]

def open(self, port\_name):

"""開啟串口"""

self.ser.port = port\_name

self.ser.baudrate = self.baudrate

self.ser.timeout = self.timeout

try:

self.ser.open()

return True

except serial.SerialException:

return False

def read\_data(self, length=1024):

"""讀取數據"""

return self.ser.read(length)

def write\_data(self, data):

"""寫入數據"""

self.ser.write(data)

def close(self):

"""關閉串口"""

self.ser.close()

**2. 控制界面模組 (wl\_controlmc)**

**轉換建議步驟:**

1. 使用Qt Designer重建DFM界面
2. 保留原有業務邏輯結構
3. 轉換VCL事件到Qt信號槽

**關鍵結構轉換示例:**

python

from PyQt5.QtCore import QObject, pyqtSignal

class WLControlMC(QObject):

data\_received = pyqtSignal(bytes) *# 替代Delphi中的事件*

def \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.com\_port = ComPort()

self.sample\_rate = 1000 *# 默認採樣率*

self.ecg\_buffer = []

def start\_acquisition(self):

"""開始採集信號"""

if self.com\_port.open('COM3'):

*# 啟動定時器讀取數據*

self.timer.start(100) *# 100ms讀取一次*

def process\_raw\_data(self, raw\_data):

"""處理原始數據"""

*# 添加您的數據解析邏輯*

self.ecg\_buffer.extend(self.\_parse\_ecg\_data(raw\_data))

self.data\_received.bytes.emit(self.ecg\_buffer[-1000:]) *# 發送最近1000點*

**二、ECG分析模組設計**

**1. 模組結構設計**

text

ecg\_processor/

├── \_\_init\_\_.py

├── preprocessor.py # 信號預處理

├── qrs\_detector.py # R峰檢測

├── hrv\_analyzer.py # HRV分析

└── mse\_analyzer.py # 多重尺度熵

**2. R-Peak檢測基礎實現**

python

import numpy as np

from scipy.signal import find\_peaks

class QRSDetector:

def \_\_init\_\_(self, sample\_rate=1000):

self.sample\_rate = sample\_rate

self.window\_size = int(0.12 \* sample\_rate) *# 120ms窗口*

def detect\_r\_peaks(self, ecg\_signal):

"""三階段R峰檢測算法"""

*# 1. 帶通濾波 (0.5-40Hz)*

filtered = self.\_bandpass\_filter(ecg\_signal)

*# 2. 微分平方增強*

diff\_squared = np.diff(filtered) \*\* 2

*# 3. 移動平均積分*

integrated = np.convolve(diff\_squared, np.ones(self.window\_size)/self.window\_size, mode='same')

*# 4. 自適應閾值檢測*

peaks, \_ = find\_peaks(integrated, distance=self.sample\_rate\*0.6) *# 最小RR間隔600ms*

return peaks

def \_bandpass\_filter(self, signal):

"""簡化版帶通濾波器"""

*# 實際實現應使用更專業的濾波器如Butterworth*

from scipy.signal import butter, filtfilt

nyq = 0.5 \* self.sample\_rate

low = 0.5 / nyq

high = 40.0 / nyq

b, a = butter(4, [low, high], btype='band')

return filtfilt(b, a, signal)

**3. HRV分析模組框架**

python

class HRVAnalyzer:

def \_\_init\_\_(self):

self.rr\_intervals = []

def add\_r\_peaks(self, r\_peaks):

"""更新RR間期數據"""

if len(r\_peaks) >= 2:

new\_intervals = np.diff(r\_peaks) *# 單位:採樣點*

self.rr\_intervals.extend(new\_intervals)

def time\_domain\_analysis(self):

"""時域分析"""

if not self.rr\_intervals:

return {}

rr\_ms = np.array(self.rr\_intervals) \* (1000.0 / self.sample\_rate)

return {

'mean\_rr': np.mean(rr\_ms),

'sdnn': np.std(rr\_ms), *# 總體標準差*

'rmssd': np.sqrt(np.mean(np.diff(rr\_ms) \*\* 2)), *# 相鄰RR間期差值的均方根*

'nn50': sum(np.abs(np.diff(rr\_ms)) > 50), *# 相鄰RR間期差異>50ms的次數*

}

def frequency\_domain\_analysis(self):

"""頻域分析"""

*# 實現Lomb-Scargle周期圖等算法*

pass

**三、開發路線圖建議**

1. **第一階段 (基礎轉換)**
   * 完成comport.pas到Python的轉換
   * 使用Qt Designer重建主界面
   * 實現基本數據採集流程
2. **第二階段 (核心算法)**
   * 實現R峰檢測基礎算法
   * 開發實時波形顯示
   * 建立RR間期緩衝區
3. **第三階段 (高級分析)**
   * 實現時域HRV分析
   * 開發頻域HRV分析
   * 整合多重尺度熵算法
4. **第四階段 (優化與驗證)**
   * 算法性能優化
   * 臨床數據驗證
   * 用戶界面改進

**四、關鍵技術建議**

1. **實時處理架構:**

python

class RealTimeProcessor:

def \_\_init\_\_(self):

self.buffer\_size = 5000 *# 5秒緩衝*

self.ecg\_buffer = np.zeros(self.buffer\_size)

self.index = 0

def add\_data(self, new\_data):

"""環形緩衝區實現"""

n = len(new\_data)

if self.index + n <= self.buffer\_size:

self.ecg\_buffer[self.index:self.index+n] = new\_data

else:

part1 = self.buffer\_size - self.index

self.ecg\_buffer[self.index:] = new\_data[:part1]

self.ecg\_buffer[:n-part1] = new\_data[part1:]

self.index = (self.index + n) % self.buffer\_size

1. **性能關鍵點:**
   * 使用numpy進行向量化運算
   * 對長時間記錄採用分段處理
   * 考慮使用Numba加速關鍵算法
2. **推薦Python庫:**

text

- PyQt5/PySide6 (界面)

- pyqtgraph (高性能繪圖)

- numpy/scipy (數值計算)

- heartpy (HRV分析參考)

- neurokit2 (生物信號處理)

需要我針對哪個具體模組提供更詳細的實現示例？或是您希望先專注於某個特定部分的轉換？