ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робота
Перший (бакалаврський) рівень

Програмно-апаратна система виявлення патологій на ЕКГ

Автор:
Михайло Шиленко
студ. гр. КІУКІ-21-3

Керівник:
Ольга Єрошенко
ст. викл. каф. ЕОМ

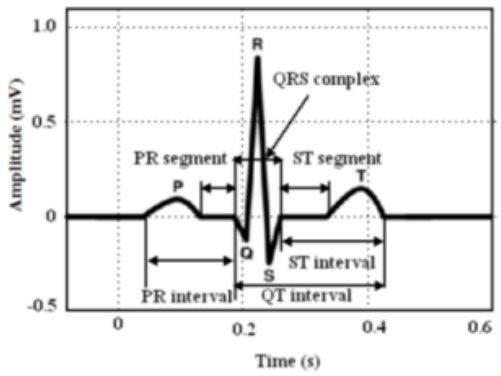
Мета і задачі роботи

Мета: Реалізація програмно-апаратної системи зчитування, фільтрації та аналізу електрокардіографічного сигналу.

Задачі:

- Визначення характеристик сигналу ЕКГ;
- Розробка аналогового вхідного модуля;
- Синтез віконного цифрового фільтру;
- Застосування алгоритму виявлення зубців на ЕКГ.

ЕКГ. Визначення



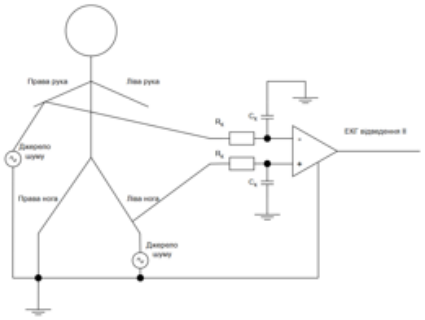
Електрокардіограма (ЕКГ) — це графічне відображення електричної активності серця, зафіксоване за допомогою електродів на поверхні тіла

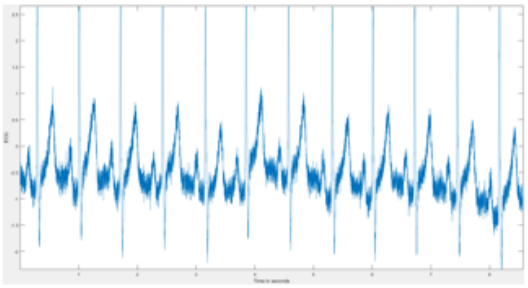
Характеристики сигналу ЕКГ

- Розмах сигналу ЕКГ у нормі дорівнює 1мВ.
- Використовуваний коефіцієнт підсилення аналогового вхідного модуля повинен дорівнювати 1000 (60 дБ).
- ЕКГ передає корисну інформацію в полосі частот 0,05–100 Гц.
- Рекомендована для ЕКГ частота дискретизації 500 Гц.

Типи завад на ЕКГ

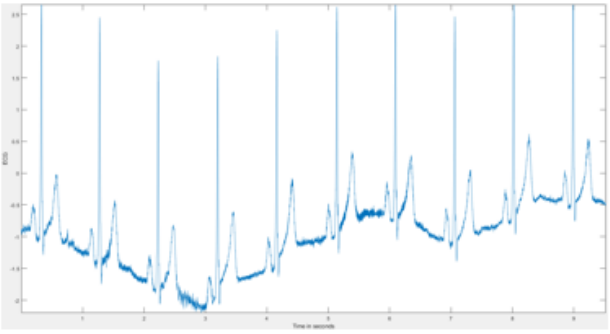
- Високочастотні завади, спричинені електромагнітними наведеннями від мережі енергопостачання (50/60 Гц).



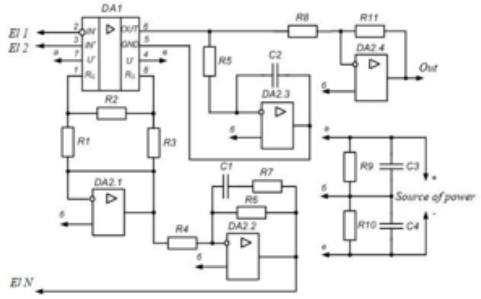


Типи завад на ЕКГ

- Низькочастотні завади (дрейф ізолінії), викликані рухом пацієнта (0,05–0,3 Гц)



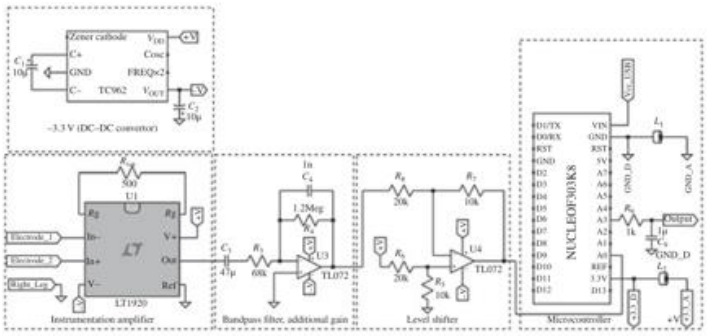
Існуючі рішення (вхідні модулі)



$$G = 1 + \frac{49,4k\Omega}{R_G - 1}$$

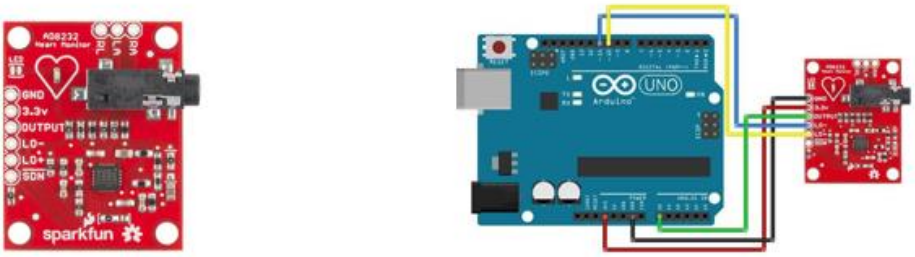
Принципова схема підсилювача біопотенціалів на основі AD620

Існуючі рішення (вхідні модулі)



Принципова схема реєстрації ЕКГ на основі мікроконтролера

Існуючі рішення (вхідні модулі)



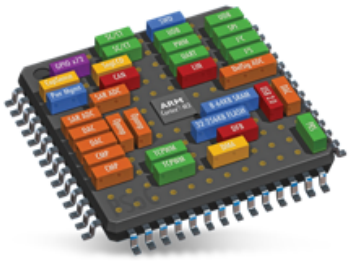
Модуль на базі інтегральної мікросхеми AD8232

Існуючі рішення (програмний модуль)

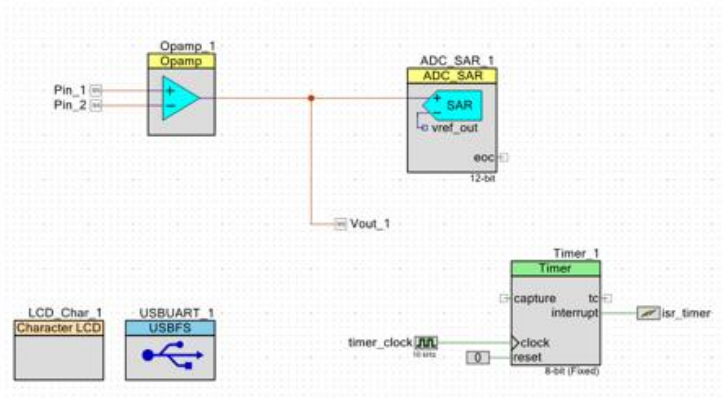
Інтерфейс програми ECG Analyzer

Розробка аналогового вхідного модуля

- У якості основи для аналогового модуля обрано мікроконтролер сімейства PSoC 5LP — завдяки його здатності до гнучкого налаштування внутрішніх аналогових і цифрових блоків.

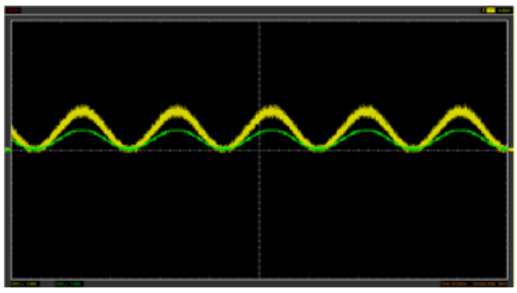


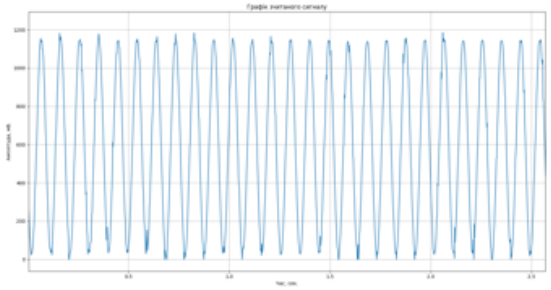
Розробка аналогового вхідного модуля



Схематичне зображення розробленого аналогового вхідного модуля

Верифікація аналогового вхідного модуля





Синтез віконного фільтру

- Основною ідеєю синтезу віконного фільтру є ідеальний ФНЧ із вже відомою частотою зрізу та імпульсною характеристикою, що задається виразом

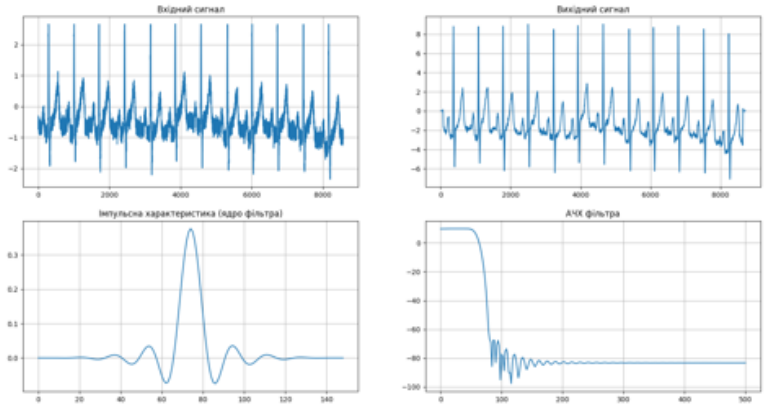
$$h[i] = \frac{\sin(2\pi f_c i)}{\pi i}$$

Синтез віконного фільтру

- На практиці даний алгоритм не може бути реалізований, оскільки коливання функції sin(x)/x не затухають до нульового значення на всьому проміжку дійсних чисел.
- Замість цього виконується усічення функції до M+1 відліків, що дозволяє реалізувати фільтр зі кінченою імпульсною характеристикою.
- Однак такий крок призводить до появи бічних пелюсток в імпульсній характеристиці фільтра, що проявляється у вигляді додаткових спотворень. Для придушення цих спотворень і застосовують віконні функції.

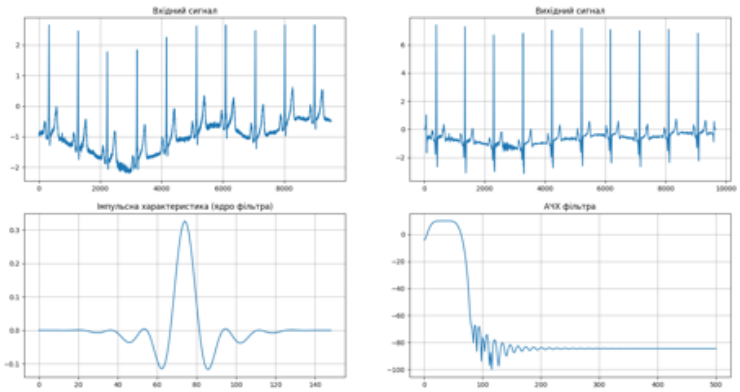
$$w[i] = 0,42 - 0,5\cos(2\pi i / M) + 0,08\cos(4\pi i / M)$$

Верифікація синтезованого фільтру



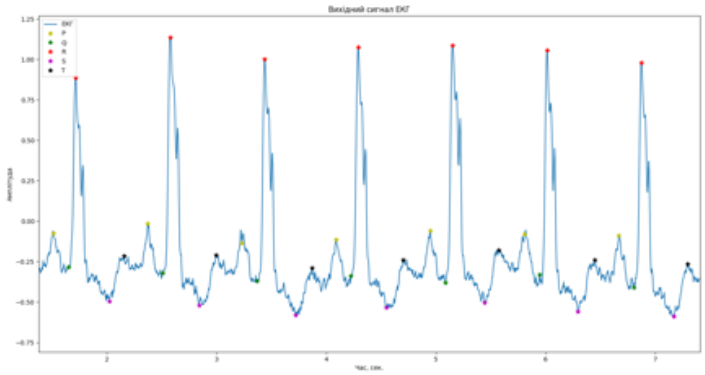
Результат придушення високочастотної завади

Верифікація синтезованого фільтру



Результат придушення низькочастотної завади

Автоматизований пошук зубців на ЕКГ



Результат роботи алгоритму пошуку зубців на ЕКГ

Висновки

- В даній роботі були представлені методи побудови системи програмно-апаратної системи виявлення патологій на ЕКГ.
- Основну увагу приділено процесу збору та обробки інформації, реалізації цифрового смугового фільтра засобами мови програмування Python.
- Розроблена система демонструє повний цикл обробки ЕКГ-сигналу – від зчитування сигналів за допомогою аналогового вхідного модуля до програмного аналізу.

ДОДАТОК Б

Початковий код програми визначення зубців на ЕКГ

Лістинг Б.1 – Початковий код програми визначення зубців на ЕКГ

```
import numpy as np
import matplotlib.pyplot as plt

# Read digitized ECG
# Reading the file - assuming 3 columns format
data = np.loadtxt(r".\signals\chf01-heartfailure.txt")
f = data.T  # Transpose to match MATLAB format [3,inf]
t = np.array(f[0, :])   # First row
y = np.array(f[1, :])   #Second row

Td = t[1] - t[0]  # Sampling period

# Set masks for finding peaks
RRmask = 215
RQmask = 15
RSmask = 79
RPmask = 38
RTmask = 102

plt.figure()
plt.plot(t, y)  # Plot original ECG
plt.title('Вихідний сигнал ЕКГ')
plt.xlabel('Час, сек.')
plt.ylabel('Амплітуда')

# Procedure for finding R peak

nr = 0.21
nq = 0.03
ns = -0.18
np_ = 0.04
nt = -0.14

dy = np.diff(y)  # Find derivative
finR = round(RRmask + 0.1 * RRmask)
nR = [0]  # Initialize list
aR = [y[0]]  # Initialize list

# Find first R peak
for i in range(1, finR):
    if i < len(y) and i - 1 < len(dy):
        if (y[i] > aR[0]) and (dy[i - 1] <= nr):
            nR[0] = i  # Sample number of first peak
            aR[0] = y[i]  # Amplitude of first peak
```

```
# Find remaining R peaks
start = round(nR[0] + 0.8 * RRmask)
fin = round(nR[0] + 1.15 * RRmask)
N = len(y)
n = 1  # Python 0-based, so starting from 1 for second element

# Initialize arrays for storing results
RR = []
RRR = []
QQ = []
SS = []
PP = []
TT = []
nQ = []
aQ = []
nS = []
aS = []
nP = []
aP = []
nT = []
aT = []

while fin < N:
    nR.append(start)
    aR.append(y[start])

    for i in range(start, min(fin + 1, N)):
        if i - 1 < len(dy):
            if (y[i] > aR[n]) and (dy[i - 1] <= nr):
                nR[n] = i  # Array of R peak sample numbers
                aR[n] = y[i]  # Array of R peak amplitudes

    # Calculate working RR interval
    work = nR[n] - nR[n - 1]
    RR.append(work)
    RRR.append([nR[n - 1], aR[n - 1]])

    # Check duration of working RR interval
    if (work < round(0.95 * RRmask)) and (work > round(1.05 *
RRmask)):
        RRmask = work

    start = round(nR[n] + 0.8 * RRmask)
    fin = round(nR[n] + 1.15 * RRmask)

    # Determine Q peak
    st = round(nR[n - 1] - 1.15 * RQmask)
    finish = round(nR[n - 1] - 0.75 * RQmask)

    st = max(0, st)  # Ensure we don't go below 0
    finish = min(N - 1, finish)  # Ensure we don't exceed array
bounds
```

```
    nQ.append(st)
    aQ.append(y[st])

    for i in range(st, finish + 1):
        if i < N and i - 1 < len(dy):
            if (y[i] < aQ[n - 1]) and (abs(dy[i - 1]) >= nq):
                nQ[n - 1] = i  # Array of Q peak sample numbers
                aQ[n - 1] = y[i]  # Array of Q peak amplitudes

    QQ.append([nQ[n - 1], aQ[n - 1]])

    # Determine S peak
    st = round(nR[n - 1])
    finish = round(nR[n - 1] + 1.5 * RSmask)

    st = max(0, st)
    finish = min(N - 1, finish)

    nS.append(st)
    aS.append(y[st])

    for i in range(st, finish + 1):
        if i < N and i - 1 < len(dy):
            if (y[i] < aS[n - 1]) and (abs(dy[i - 1]) >= ns):
                nS[n - 1] = i  # Array of S peak sample numbers
                aS[n - 1] = y[i]  # Array of S peak amplitudes

    SS.append([nS[n - 1], aS[n - 1]])

    # Determine P peak
    st = round(nR[n - 1] - 1.35 * RPmask)  # Changed 1.35 to
1.01 (comment from original)
    finish = round(nR[n - 1] - 0.65 * RPmask)

    st = max(0, st)
    finish = min(N - 1, finish)

    nP.append(st)
    aP.append(y[st])

    for i in range(st, finish + 1):
        if i < N and i - 1 < len(dy):
            if (y[i] > aP[n - 1]) and ((dy[i - 1] >= np_) and
(dy[i - 1] < nq)):
                nP[n - 1] = i  # Array of P peak sample numbers
                aP[n - 1] = y[i]  # Array of P peak amplitudes

    PP.append([nP[n - 1], aP[n - 1]])

    # Determine T peak
    st = round(nR[n - 1] + 0.75 * RTmask)
    finish = round(nR[n - 1] + 1.2 * RTmask)
```

```
    st = max(0, st)
    finish = min(N - 1, finish)

    nT.append(st)
    aT.append(y[st])

    for i in range(st, finish + 1):
        if i < N and i - 1 < len(dy):
            if (y[i] > aT[n - 1]) and ((dy[i - 1] >= nt) and
(dy[i - 1] < nr)):
                nT[n - 1] = i  # Array of T peak sample numbers
                aT[n - 1] = y[i]  # Array of T peak amplitudes

    TT.append([nT[n - 1], aT[n - 1]])

    n = n + 1

# Convert lists to numpy arrays for easier indexing
nP = np.array(nP)
aP = np.array(aP)
nQ = np.array(nQ)
aQ = np.array(aQ)
nR = np.array(nR)
aR = np.array(aR)
nS = np.array(nS)
aS = np.array(aS)
nT = np.array(nT)
aT = np.array(aT)

# Plot the detected peaks
plt.plot(t[nP], aP, '*y', markersize=8)  # P peaks
plt.plot(t[nQ], aQ, '*g', markersize=8)  # Q peaks
plt.plot(t[nR], aR, '*r', markersize=8)  # R peaks
plt.plot(t[nS], aS, '*m', markersize=8)  # S peaks
plt.plot(t[nT], aT, '*k', markersize=8)  # T peaks

plt.legend(['ЕКГ', 'P', 'Q', 'R', 'S', 'T'])
plt.show()
```