Міністерство освіти і науки України
ДВНЗ «Донецький національний технічний університет»

Кафедра комп'ютерної інженерії

# Пояснювальна записка

до курсової роботи
з дисципліни «Методи та засоби розробки системних програм»

на тему

«Організація виконання періодичних завдань»

Виконав студент
групи КІ-16
напряму підготовки «Комп'ютерна інженерія»
*Копилов К.О.*

Керівник
ст. викл. кафедри КІ *Шевченко О.Г.*

Оцінка

Національна шкала _____

Кількість балів: _____

Член комісії

_____ _____
(підпис) (прізвище та ініціали)

Покровськ, 2018

ЗМІСТ

# ВСТУП

Системне програмування (чи програмування систем) - підрозділ програмування, що полягає в роботі над системним програмним забезпеченням.

Визначення "системне" підкреслює той факт, що результати цього виду програмування істотно міняють властивості і можливості обчислювальної системи. В той же час безперечним залишається той факт, що певною мірою цей результат має місце при застосуванні будь-яких програм, що виконуються в обчислювальній системі. Тому між програмуванням "системним" і "несистемним" (прикладним програмуванням) немає чіткої межі. Одна з основних відмінних рис системного програмування в порівнянні з прикладним полягає в тому, що результатом останнього являється випуск програм для взаємодії з користувачем (наприклад, текстовий процесор). Тоді як результатом системного програмування є випуск програм для взаємодії з апаратним забезпеченням (наприклад, дефрагментація жорсткого диска), що має на увазі сильну залежність таких програм від апаратної частини. Зокрема можна виділити наступне:

- програміст повинен зважати специфіку апаратної частини і інші, часто унікальні властивості системи в якій функціонує програма, використати ці властивості, наприклад, застосовуючи спеціально оптимізований для цієї архітектури алгоритм;
- часто використовується низькорівнева мова програмування або такий діалект мови програмування, яка дозволяє функціонування в оточенні з обмеженим набором системних ресурсів
- працює максимально ефективно і має мінімальне запізнювання за часом завершення [1].

# 1. ЗАСОБИ ОС WINDOWS ДЛЯ ОРГАНІЗАЦІЇ ПЕРІОДИЧНИХ ДІЙ

Операційна система Windows дозволяє для кожного додатка створити декілька віртуальних таймерів. Усі ці таймери працюють по перериваннях одного фізичного таймера.

Оскільки робота Windows грунтована на передачі повідомлень, логічно було б припустити, що і робота віртуального таймера також грунтована на передачі повідомлень. І справді, додаток може замовити для будь-якого свого вікна декілька таймерів, які періодично посилатимуть у функцію вікна повідомлення з кодом WM_TIMER.

Є і інший спосіб, також грунтований на передачі повідомлень. При використанні цього способу повідомлення WM_TIMER посилаються не функції вікна, а спеціальної функції, описаної з ключовим словом _export. Ця функція нагадує функцію вікна і, так само як і функція вікна, викликається не з додатка, а з Windows. Функції, які викликаються з Windows, мають спеціальний пролог і епілог і називаються функціями зворотного виклику (callback function). Функція вікна і функція, спеціально призначена для обробки повідомлень таймера, є прикладами функцій зворотного виклику. На жаль, точність віртуального таймера залишає бажати кращого. Повідомлення таймера проходять через чергу додатка, до того ж інший додаток може блокувати на деякий час роботу нашого додатка. Тому повідомлення від таймера приходять в загальному випадку нерегулярно. Крім того, незважаючи на можливість вказівки інтервалів часу в мілісекундах, реальна дискретність таймера визначається періодом переривань, що посилаються таймером. Цей період (тобто тривалість одного такту таймера) можна упізнати за допомогою функції GetTimerResolution:

```
DWORD WINAPI GetTimerResolution(void);
```

Нерегулярність приходу повідомлень таймера не викликає особливих проблем, якщо не йдеться про роботу в режимі реального часу. Системи реального часу, грунтовані на Windows, повинні використати для пристроїв введення/виведення, критичних до швидкості реакції системи спеціальні драйвери. Строго кажучи, операційна система Windows не призначена для роботи в режимі реального часу. Windows орієнтована на роботу з людиною, коли невеликі затримки подій в часі не мають ніякого значення.

**Створення і знищення таймера**

Для створення віртуального таймера додаток повинен використати функцію SetTimer:

```
UINT WINAPI SetTimer(HWND hwnd, UINT idTimer,UINT uTimeout, TIMERPROC tmprc);
```

Перший параметр функції (hwnd) повинен містити ідентифікатор вікна, функція якого отримуватиме повідомлення від таймера, або NULL. У останньому випадку із створюваним таймером не зв'язується ніяке вікно і повідомлення від таймера приходитимуть в спеціально створену для цього функцію. Другий параметр (idTimer) визначає ідентифікатор таймера (він не має дорівнювати нулю). Ідентифікатор використовується тільки у тому випадку, якщо перший параметр функції SetTimer містить ідентифікатор вікна. Оскільки для одного вікна можна створити декілька таймерів, для того, щоб розрізняти повідомлення, що приходять від різних таймерів, додаток при створенні повинен забезпечити кожен таймер власним ідентифікатором. Якщо перший параметр вказаний як NULL, другий параметр функції ігнорується, оскільки для таймера задана спеціальна функція, одержуюча повідомлення тільки від цього таймера.

Третій параметр (uTimeout) визначає період отримання повідомлень від таймера в мілісекундах. Врахуйте, що фізичний таймер тікає приблизно 18,21

разів в секунду (точне значення складає 1000/54,925). Тому, навіть якщо ви вкажете, що таймер повинен тікати кожну мілісекунду, повідомлення приходитимуть з інтервалом не менше 55 мілісекунд. Останній параметр (tmprc) визначає адресу функції, яка отримуватиме повідомлення WM_TIMER (ми називатимемо цю функцію функцією таймера). Цей параметр необхідно обов'язково вказати, якщо перший параметр функції SetTimer рівний NULL. Повертане функцією SetTimer значення є ідентифікатором створеного таймера (якщо в якості першого параметра функції було вказано значення NULL). У будь-якому разі функція SetTimer повертає нульове значення, якщо вона не змогла створити таймер. У Windows версії 3.0 максимальна кількість створених в усій системі таймерів була 16. Для Windows версії 3.1 це обмеження зняте. Проте, якщо додаток більше не потребує послуг таймера, воно повинне знищити таймер, викликавши функцію KillTimer:

```
BOOL WINAPI KillTimer (HWND hwnd, UINT idTimer);
```

Перший параметр функції (hwnd) визначає ідентифікатор вікна, вказаний при створенні таймера функцією SetTimer. Другий параметр (idTimer) - ідентифікатор знищуваного таймера. Це має бути або той ідентифікатор, який ви вказали при створенні таймера (якщо таймер створювався для вікна), або значення, отримане при створенні таймера від функції SetTimer (для таймера, що має власну функцію обробки повідомлень). Функція KillTimer повертає значення TRUE при успішному знищенні таймера або FALSE, якщо вона не змогла знайти таймер з вказаним ідентифікатором [2].

# 2. СТРУКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Курсова робота була виконана з використанням вільного фреймворка Qt. Для визначення виробника і моделі центрального процесора були використані асемблерні вставки. Проводилося дослідження двох таймерів:

1. QTimer (C++),
2. Timer (QML).

Програма розділена на два рівня:

1. рівень QML,
2. рівень C++.

Логіка програми була написана на C++, графічний інтерфейс користувача на QML. Обидві частини програми активно між собою спілкуються за допомогою інструментарію сигналів і слотів. При деякій події генерується сигнал, і по цьому сигналу відбувається виконання слота. Слот - це звичайний метод, який може бути виконаний і без сигналу до його виконання. Крім того, між рівнями відбувається активний обмін об'єктами, інформацію можна передати як з C++ в QML, так і навпаки.

# 3. ОПИС ПРОГРАМНИХ МОДУЛІВ

Qt - кросплатформний фреймворк для розробки ПЗ на мові програмування C++, дозволяє запускати написане з його допомогою програмне забезпечення у більшості сучасних операційних систем шляхом простої компіляції програми для кожної системи без зміни початкового коду.

Включає усі основні класи, які можуть знадобитися при розробці прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Є повністю об'єктно-орієнтованим, розширюваним і таким, що підтримує техніку компонентного програмування.

Відмітна особливість - використання метаоб’єктного компілятора - попередньої системи обробки початкового коду. Розширення можливостей забезпечується системою плагінів, які можливо розміщувати безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності виджетів, пов'язаною з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна [3]. Варто відмітити, що уся кросплатформність цього проекту зводиться нанівець через використання компілятора Visual Studio, який підтримує асемблерні вставки синтаксису MASM32.

QML - декларативна мова програмування, яка заснована на JavaScript, призначена для дизайну додатків, які роблять основний упор на призначений для користувача інтерфейс. Є частиною Qt Quick, середовища розробки призначеного для користувача інтерфейсу, поширюваної разом з Qt.

QML-документ є деревом елементів. QML елемент, так само як і елемент Qt, є сукупністю блоків: графічних (таких, як rectangle, image) і поведінкових (таких, як state, transition, animation). Ці елементи можуть бути

об'єднані, щоб побудувати комплексні компоненти, починаючи від простих кнопок і повзунків і закінчуючи повноцінними додатками, працюючими з інтернетом.

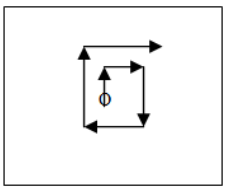
QML елементи можуть бути доповнені стандартними JavaScript вставками шляхом вбудовування .js файлів. Також вони можуть бути розширені C++ компонентами через Qt framework [4].

# 4. МЕТОДИКА РОБОТИ

## Розробка періодичної функції. QML

У моєму варіанті було задано побудувати періодичну функцію, показану на рисунку 2.1, функція повинна мати наступні властивості:

- працювати нескінченно, поки користувач не натисне поєднання клавіш або не натисне на кнопку закриття форми,
- після отримання команди завершення роботи вивести інформацію про підтримку технології PGE, виробника і моделі центрального процесора,
- при наступному натисненні кнопки закриття - закрити додаток.



Заповнення – послідовно по спіралі, що розкручується від центру у напряму годинникової стрілки.
Знищення – послідовно в зворотному порядку.
Формат виводу – 1 символ.

Рисунок 4.1 – Формат виведення періодичної функції.

Як можна було бачити на малюнку, елементи "спіралі" мають вертикальне і горизонтальне положення. Для малювання елементів у вертикальному положенні використовувався елемент QML Column, для малювання в горизонтальному - Row. Звідси витікає, що уся періодична функція є набором рядків і стовпців, розташованих по різних координатах. В якості дочірніх елементів для елементів Row і Column були елементи Text і Repeater. Розглянемо принцип малювання:

1. створити елемент Row / Column,
2. вказати координати, використати елемент Repeater, який відмалює елементи рядка або стовпця в заданому напрямі,
3. створити потрібну кількість елементів,

4. створити новий рядок або стовпець з координатами, прилеглими до попереднього елементу,
5. почати все спочатку.

Розглянемо декілька елементів періодичної функції:

```
1. Column {
    anchors.centerIn: parent
    id: colId
    Repeater {
        id: repeat1
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

2. Row {
    x: 329; y: 182
    id: rowId
    Repeater {
        id: repeat2
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}
```

Як можна бачити, першим елементом нашої спіралі є стовпець, використовується елемент Column, а другий елемент рядок - Row. Параметр anchors вказує, до якої частини форми має бути прикріплений зміст Column, в

нашому випадку старт періодичної функції йде з центру форми, у властивості id вказується унікальна назва елементу для того, щоб можна було отримати доступ до його властивостей і модифікувати їх при потребі з будь-якої частини програми. Як було сказано раніше, елемент Repeater потрібний для малювання елементів в заданому напрямі. Властивість model вказує, скільки елементів потрібно відмалювати, за умовчанням встановлено значення 0, оскільки відмалювання пов'язане з тиком таймера, model інкрементується потрібне кількість разів при кожному тикі таймера. Далі йде елемент Text, в якому вказується текст, який виводитиметься, шрифт, розмір і колір. Як можна бачити, явного тексту у властивості text немає, там вказана змінна, яка береться з рівня C++, в якості змінної передається наявність підтримки технології PGE, а оскільки формат виведення один символ - виводиться або Y, або N. Властивості color теж встановлена змінна, її значення теж погоджене з технічним завданням і дорівнює воно "lightgreen". Тобто періодична функція виводиться в яскраво-зеленому кольорі. Властивості елементу Row такі ж як і Column, тільки явно вказані координати, де закінчилося відмалювання Column.

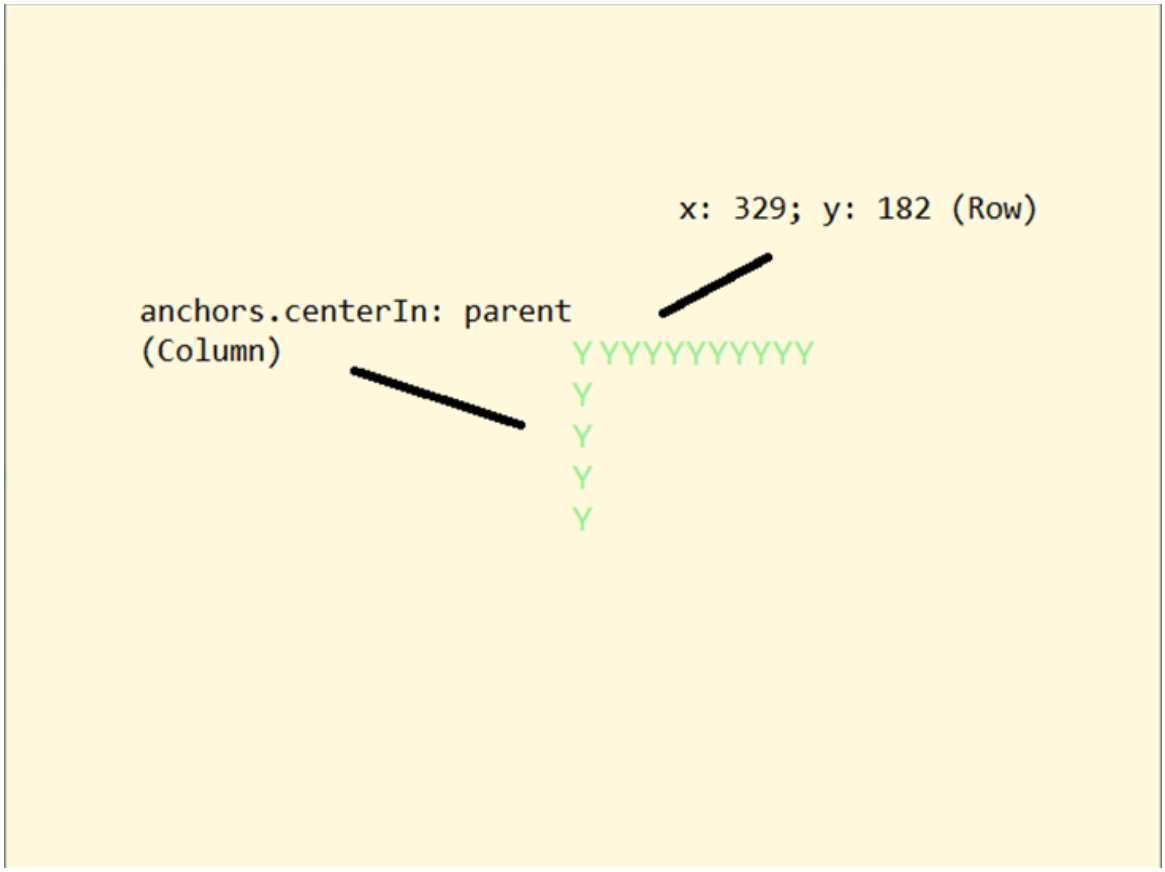



Рисунок 4.3 – Компонування складових частин періодичної функції.

Далі треба відлічити потрібну кількість елементів періодичної функції для створення спіралі. На малюнку 4.3 можна побачити 5 елементів по вертикалі і 11 по горизонталі, наступні частини спіралі формуються за таким же принципом. Щоб сформувати спіраль далі, треба створити новий елемент стовпця, ввести його координати під елементом рядка, відлічити потрібну кількість елементів і продовжити далі. Для управління усіма частинами періодичної функції була створена окрема функція в рівні QML, завданням якої є циклічно заповнювати і очищати форму елементами. Функція викликається при кожному спрацьовуванні таймера, в ній ведеться облік одної змінної, яка потрібна для роботи періодичної функції. Розглянемо декілька елементів періодичної функції:

```
1.      if(times >= 0 && times <= 5)
  ++repeat1.model
2.      if(times > 5 && times <= 15)
```

```
++repeat2.model
```

Суть роботи полягає в створенні елементів шляхом перевірки значення допоміжної змінної, яка інкрементується кожного разу при виклику функції, як було показано вище, по вертикалі відмалювало 5 символів, що відповідає першому випадку, по горизонталі 11, але у функції 10, тому що один елемент по горизонталі залишився від елементу по вертикалі. Для заповнення усієї форми (640x480) треба виконати 338 ітерацій. Після заповнення форми відбувається її очищення, по одному елементу з кінця. Після 678 ітерації обнуляється властивість model кожних Repeater'ів, обнуляється допоміжна змінна, і функція розпочинає свою роботу з початку.

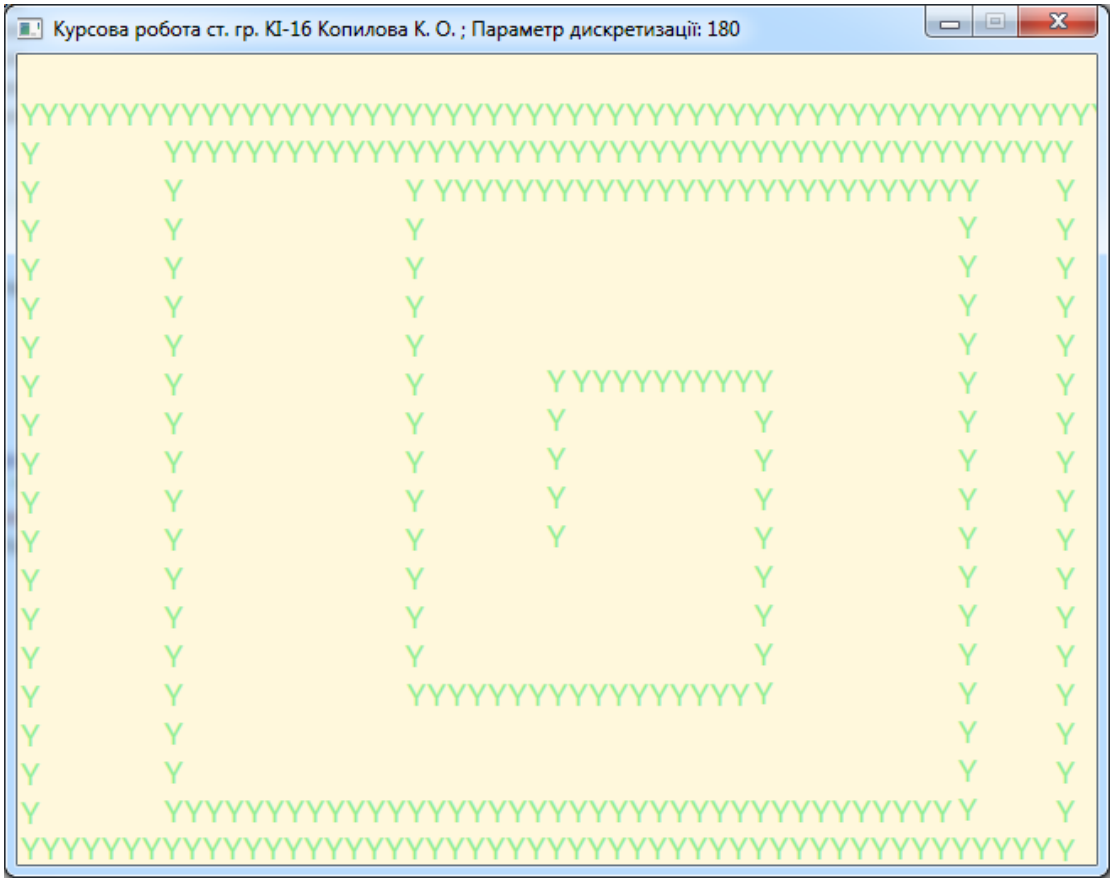



Рисунок 4.4 – Повністю заповнена форма елементами періодичної функції, якщо PGE підтримується.

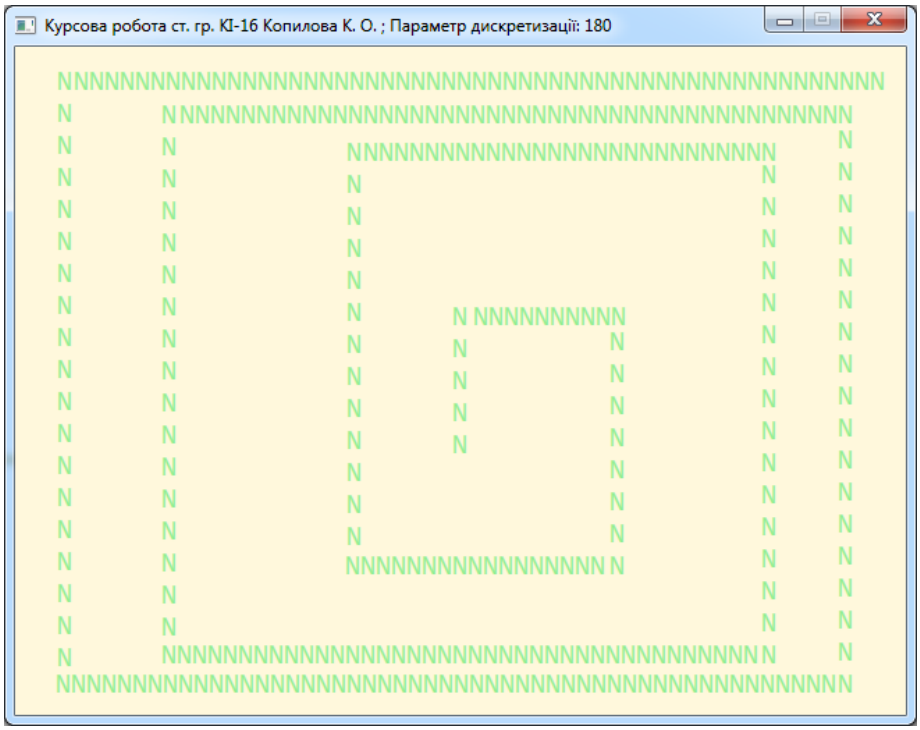



Рисунок 4.4.1 - Повністю заповнена форма елементами періодичної функції, якщо не PGE підтримується.

Після запуску додатка, користувача зустрічає форма, показана на рисунку 4.5.

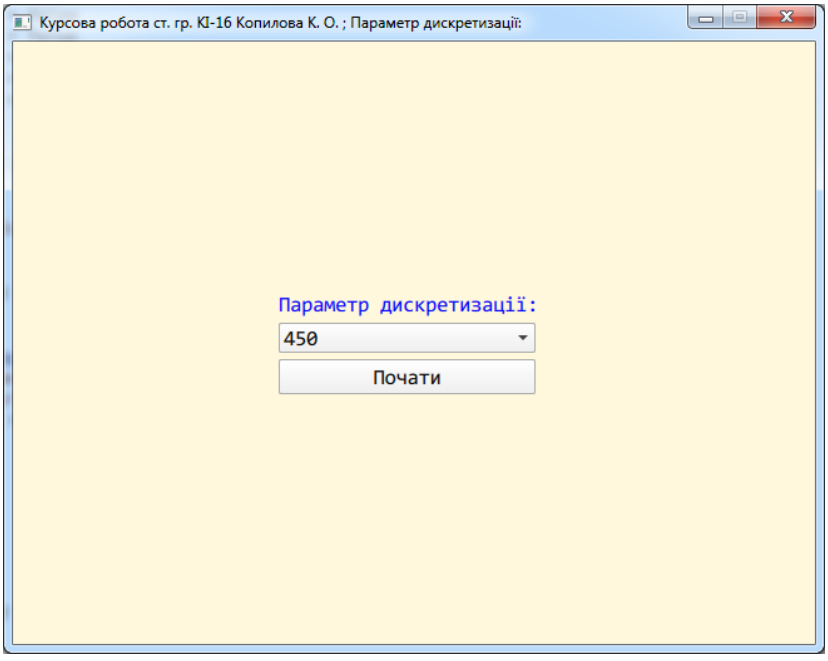



Рисунок 4.5 – Початкова форма додатка.

Тут ми можемо бачити запрошення вибрати потрібний параметр дискретизації (у ТЗ вказані 450, 1080, 180) і почати роботу, натиснувши на відповідну кнопку. Після натиснення кнопки "Почати", додаток почне працювати до тих пір, поки не буде натиснута комбінація клавіш Ctrl+C або не буде натиснута кнопка закриття форми. Після натиснення на кнопку закриття форми або виконання поєднання клавіш, нас зустріне форма, показана на малюнку 4.6.

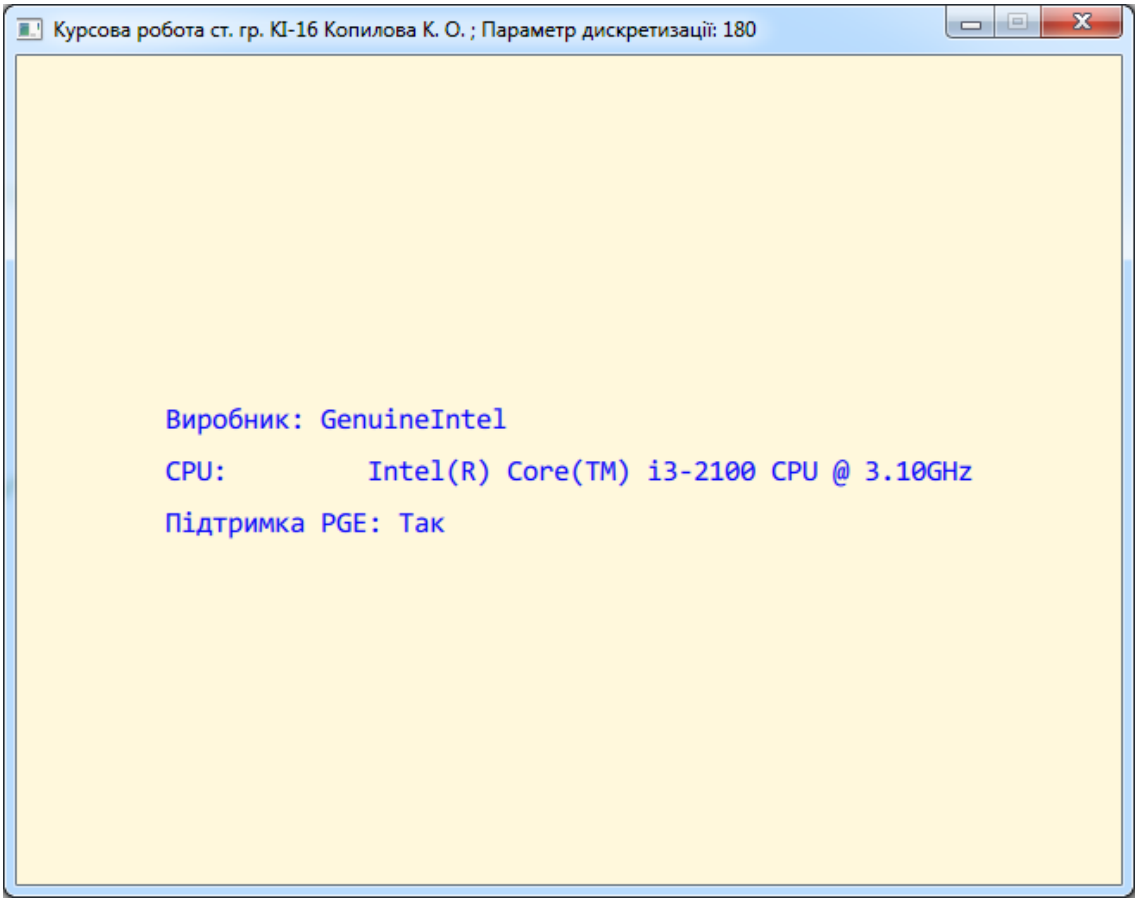



Рисунок 4.6 – Кінцева форма.

Для створення поєднання клавіш в QML використовується елемент Shortcut. Розглянемо його у рамках використання цього додатка.

```
Shortcut {
    id: endShortcut // ідентифікатор
```

```
            sequence: "Ctrl+C" // поєднання клавіш
            onActivated: endFunction() //що викликати при спрацьовуванні поєднання, викликається кінцева функція, яка відображає інформацію
            enabled: false // спочатку неактивний, активується при хоч би одному проході періодичної функції
        }
```

Для управління кнопкою закриття форми, використовується елемент onClosing. Була створена змінна типу int, значення якої за умовчанням дорівнює нулю. При натисненні на кнопку закриття вікна, відбувається виклик кінцевої функції, її значення інкрементується, і як тільки воно стає рівним двом або більше - форма закривається. Крім того, треба перевірити, що стався хоч би один прохід періодичної функції, якщо не стався, то по натисненню на кнопку закриття відбувається закриття.

```
        onClosing: {
            close.accepted = false
            if(times == 0)
                close.accepted = true
            endFunction()
            if(canClose >= 2)
                close.accepted = true
    }
```

**Розробка періодичної функції. Зв'язок QML і C++**

Як було написано раніше, робилося тестування двох таймерів: QML і C++, тому було створено 2 додатка, в першому використовується таймер C++ - "QTimer", в другому таймер QML - "Timer". Для зв'язування періодичної функції і таймера використовувався інструментарій сигналів і слотів, схема роботи якого показана на малюнку 4.7 [5].

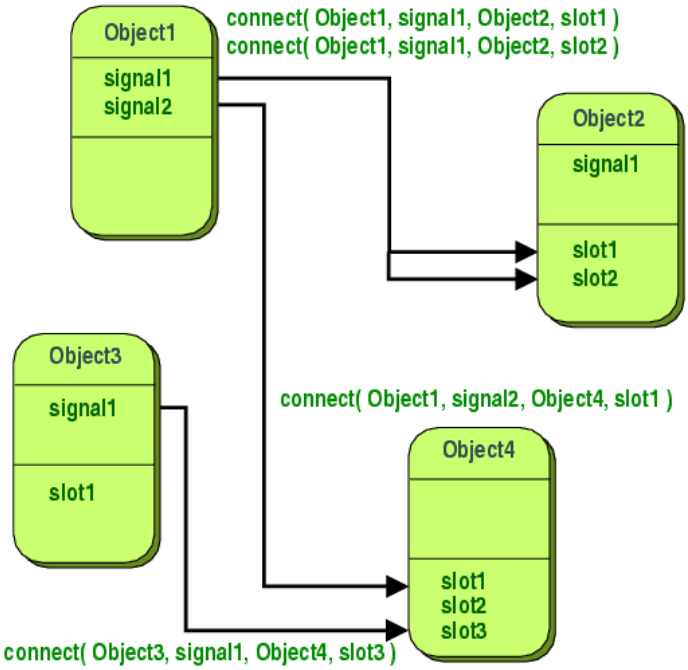



Рисунок 4.5 – Схема роботи сигналів і слотів.

На рисунку можна бачити, що об'єкти генерують сигнали, і до них прив'язані слоти. Наприклад, користувач натискає кнопку, при натисненні генерується сигнал onClicked, і виконується функція someFunc(), зв'язування об'єктів відбувається за допомогою методу connect() об'єкту QObject.

```
QObject::connect(&object1, SIGNAL(onClicked()), &object2, SLOT(someFunc()));
```

Для використання функціонала сигналів і слотів, клас має бути успадкований від класу QObject, також потрібна обов'язкова наявність макросу Q_OBJECT. Слоти оголошуються в заголовному файлі класу, в розділі з вказівкою модифікатора доступу і словом slots, наприклад:

```
private slots:

void someSlot();
```

Сигнали теж вказуються в заголовному файлі класу, проте їх модифікатор доступу може бути тільки public. Наприклад:

```
signals:

someSignal();
```

Створимо клас, в якому користуватимемося таймером. Я назвав його GetCPUData. Клас окрім використання таймера він дозволяє отримати рядки назви процесора, його виробника і підтримки технології PGE. У заголовному файлі в секції private оголосимо об'єкт класу QTimer *timer; у конструкторі ініціюємо об'єкт timer = new QTimer(this). За умовчанням, класи, успадковані від QObject, приймають як параметр конструктора його батьківський об'єкт, це треба для звільнення динамічно виділеної пам'яті і відвертання витоків пам'яті. В якості аргументу конструктору класу QTimer, був переданий аргумент this, тобто, при знищенні об'єкту класу GetCPUData, знищаться усі його дочірні об'єкти, і пам'ять буде звільнена. Періодична функція і розрахунок погрішності в рівні C++ визначені у фукнції void periodicFunction(). Зв'яжемо періодичну функцію і таймер за допомогою методу connect:

```
connect(timer, SIGNAL(timeout()), this, SLOT(periodicFunction()));
```

Тут: timer - об'єкт класу QTimer, timeout() - сигнал від таймера, який сигналізує про закінчення інтервалу, this - вказівник на об'єкт класу, в якому створений timer, periodicFunction() - наша періодична функція. Установка таймера на потрібну частоту дискретизації відбувається у функції getInterval(unsigned int interval), функція отримує значення інтервалу з QML рівня програми. На малюнку 4.5 була показана початкова форма програми, в якій розміщено запрошення вибрати параметр дискретизації. В якості елементу для вибору був використаний QML ComboBox, у властивості model якого були встановлені доступні параметри дискретизації. По натисненню на кнопку "Почати" відбувається генерація сигналу, слотом для якого виступає функція startFuction() з QML рівня, в якій відбувається розбір значень з ComboBox за допомогою оператора множинного вибору switch(), вибране

значення вирушає в C++ рівень. Для виклику слотів C++ в QML треба прокинути необхідний об'єкт класу. У моєму проекті движок QML був створений у функції main:

```
  QGuiApplication app(argc, argv);
        QQmlApplicationEngine                                    engine;
engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
        if (engine.rootObjects().isEmpty())
    return -1;
```

В об'єкта engine класу QQmlApplicationEngine є метод rootContext() setContextProperty(const QString &name, QObject *value), перший аргумент якого служить для визначення ідентифікатора, по якому можна звернутися в QML рівні, другий для передачі об'єкту.

Створимо об’єкт класу GetCPUData в функції main:

```
  GetCPUData *cpuData = new GetCPUData();
```

І зробимо його доступним в QML:

```
  engine.rootContext()->setContextProperty("cpuData", cpuData);
```

Такими маніпуляціями в QML стали доступні сигнали та слоти із C++. Окрім передачі об’єкта, передаються також рядки інформації про процесор і підтримку технології PGE:

```
      QString cpu = cpuData->getCPUName();
      QString manufacturer = cpuData->getCPUManufacturer();
      QString pge;
      (cpuData->isPGESupported()) ? pge = "Y" : pge = "N";
      engine.rootContext()->setContextProperty("pgeSupport", pge);
      engine.rootContext()->setContextProperty("processorModel", cpu);
      engine.rootContext()->setContextProperty("processorManufacturer",
manufacturer);
```

Розглянемо установку таймера в C++ на необхідне значення з QML:

```
function startFunction() {

    switch(cbox.currentIndex) {
    case 0:
        window.title += "450"
        cpuData.getInterval(450)
        break;
    case 1:
        window.title += "1080"
        cpuData.getInterval(1080)
        break;
    case 2:
        window.title += "180"
        cpuData.getInterval(180)
        break;
    }
    startSettings.visible = false
    }
```

Тут відбувається розбір елементів ComboBox, і викликається слот getInterval з відповідним значенням, а в заголовок вікна поміщається параметр дискретизації. Після вибору частоти, елемент вибору ховається і починається виконання періодичної функції.

У методі getInverval(unsigned int interval) відбувається налаштування на вибраний параметр дискретизації і запускається таймер.

```
void GetCPUData::getInterval(unsigned int interval)
{
    switch(interval) {
    case 450:
        timer->setInterval(450);
        break;
    case 1080:
        timer->setInterval(1080);
        break;
    case 180:
        timer->setInterval(180);
```

```
            break;
        }
        timer->start();
    }
```

Для зупинки періодичної функції використовується вже раніше розглянутий механізм поєднання клавіш і маніпуляцій з клавішею "закрити" в заголовку вікна. У QML викликається функція endFunction(), яка показує сумарну інформацію і викликає C++ слот stopPeriondicFunction(), в якому відбувається зупинка таймера:

```
void GetCPUData::stopPeriondicFunction()
{
    timer->stop();
}
```

**Розрахунок погрішності QTimer (C++)**

Для розрахунку погрішності скористаємося функцією DWORD WINAPI GetTickCount (void) [6]. Функція повертає кількість мілісекунд з моменту запуску системи, крім того з її допомогою можна заміряти інтервал таймера. Скористаємося змінною, в яку запишемо число, яке поверне GetTickCount(). Викличемо періодичну функцію з QML, з поточного значення GetTickCount() віднімемо попереднє, отримаємо інтервал, порівняємо його із заданим і отримаємо погрішність. Далі треба записати фактичний час виконання в текстовий файл. Для запису в текстовий файл скористаємося класами QFile і QTextStream. Створимо об'єкти цих класів в секції private нашого класу:

```
    QFile file;
    QTextStream out;
```

Ініціалізуємо їх в конструкторі:

```
        file.setFileName("./out.txt"); // шлях, куди буде збережений файл
file.open(QIODevice::WriteOnly | QIODevice::Text); // властивості відкриття
файлу
        out.setDevice(&file); // посилання на об'єкт класу file
```

Розглянемо виклик періодичної функції і розрахунок фактичного часу виконання.

```
void GetCPUData::periodicFunction()
{
    t1 = GetTickCount();
    emit catchQmlEvent();
    if(t2 == 0) t2 = t1;
    qDebug() << t1 - t2 << endl
    out << t1 - t2 << endl;
    out.flush();
    t2 = t1;
}
```

У змінну t1 записується кількість тиків, ключове слово emit генерує сигнал, слотом для якого є періодична функція в шарі QML. При першому проході функції друге значення ще не отримане, тому на виведення піде реальна кількість мілісекунд із старту системи, перевірка дозволяє вивести 0, замість великого числа, яке може збивати з пантелику. Для зручності відладки було використано виведення в налагоджувальну консоль за допомогою класу QDebug. Далі, в об'єкт out класу QTextStream за допомогою перевантаженного побітового зсуву вліво передається різниця мілісекунд між першим і подальшим виконанням функції, фактичний час виконання записується у файл після виклику методу flush. Щоб зберегти попереднє значення змінної t1, використовується змінна t2. Для з'єднання сигналу з C++ і слота в QML використовується елемент Connections:

```
        Connections {
            target: cpuData // об'єкт класу
            onCatchQmlEvent: period() // що виконувати при генерації сигналу
```

```
}
```

Як можна було помітити, в C++ сигнал для виклику періодичної функції називається catchQmlEvent(), але в QML він називається onCatchQmlEvent, так виходить через особливості QML, який сам додає префікс "on" до назви сигналу, і перетворює наступну букву назви сигналу до верхнього регістра.

Отриманий алгоритм роботи :

1. по сигналу від QTimer викликається C++ слот,
2. заміряється час виконання,
3. викликається періодична функція з QML,
4. розраховується фактичний час виконання,
5. фактичний час виконання записується у файл.

**Розрахунок погрішності Timer (QML)**

Здебільшого логіка роботи програми залишилася незмінною, в цьому застосуванні тепер використовується сигнал не від QTimer (C++), а від Timer (QML). Розглянемо створення QML- таймера.

```
Timer {
      id: timer
      onTriggered: cpuData.periodicFunction() // виклик періодичної функції з C++
      repeat: true // встановлення таймера на повтор
}
```

Тепер вибір інтервалу зроблений в QML, для перебору значень використовується оператор switch(), усі дії з налаштування таймера відбуваються у функції startFunction().

```
function startFunction() {

        switch(cbox.currentIndex) {
        case 0:
            timer.interval = 450 // установка інтервалу
            window.title += "450"
            break;
        case 1:
            timer.interval = 1080
            window.title += "1080"
            break;
        case 2:
            timer.interval = 180
            window.title += "180"
            break;
        }
        startSettings.visible = false
        timer.running = true // запуск таймера
  }
```

Алгоритм роботи для QML-частини :

1. згенерувати сигнал,
2. викликати періодичну функцію з C++, яка викличе періодичну функцію з QML,
3. отримати реальну дискретизацію,
4. вивести її у файл.

Логіка роботи обох додатків однакова, за винятком використання різних таймерів.

# 5. ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ

## Дослідження додатка на базі QTimer (C++)

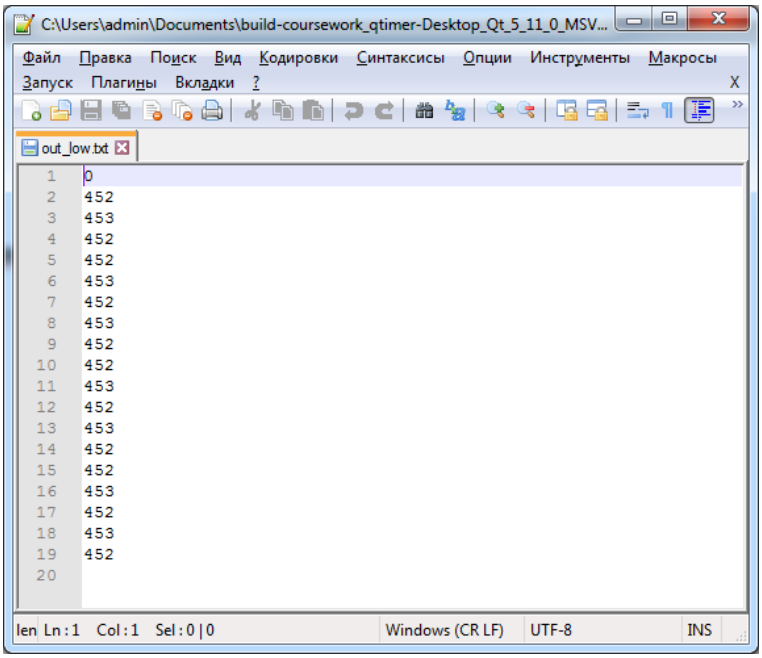



Рисунок 5.1 – Фактичний час виконання при частоті дискретизації 450 мс. і завантаженням ЦП до 30%.

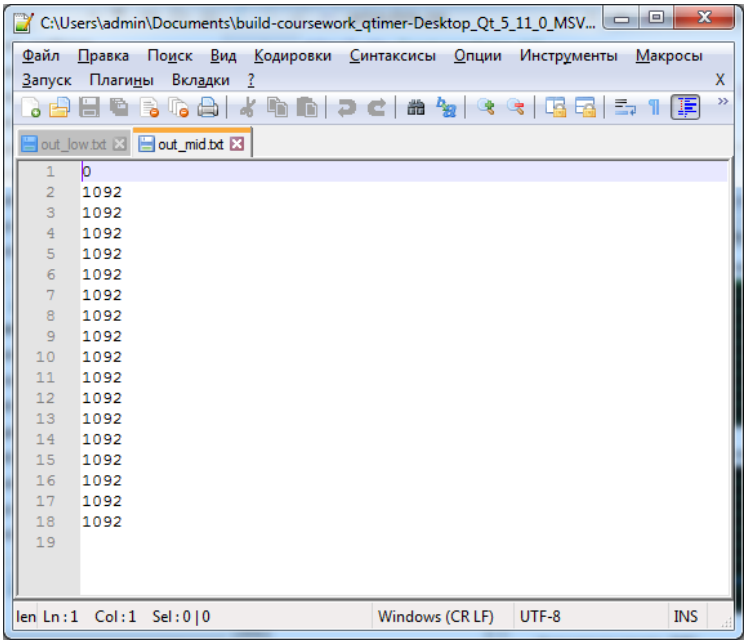



Рисунок 5.2 – Фактичний час виконання при частоті дискретизації 1080 мс. і завантаженням ЦП до 70%.

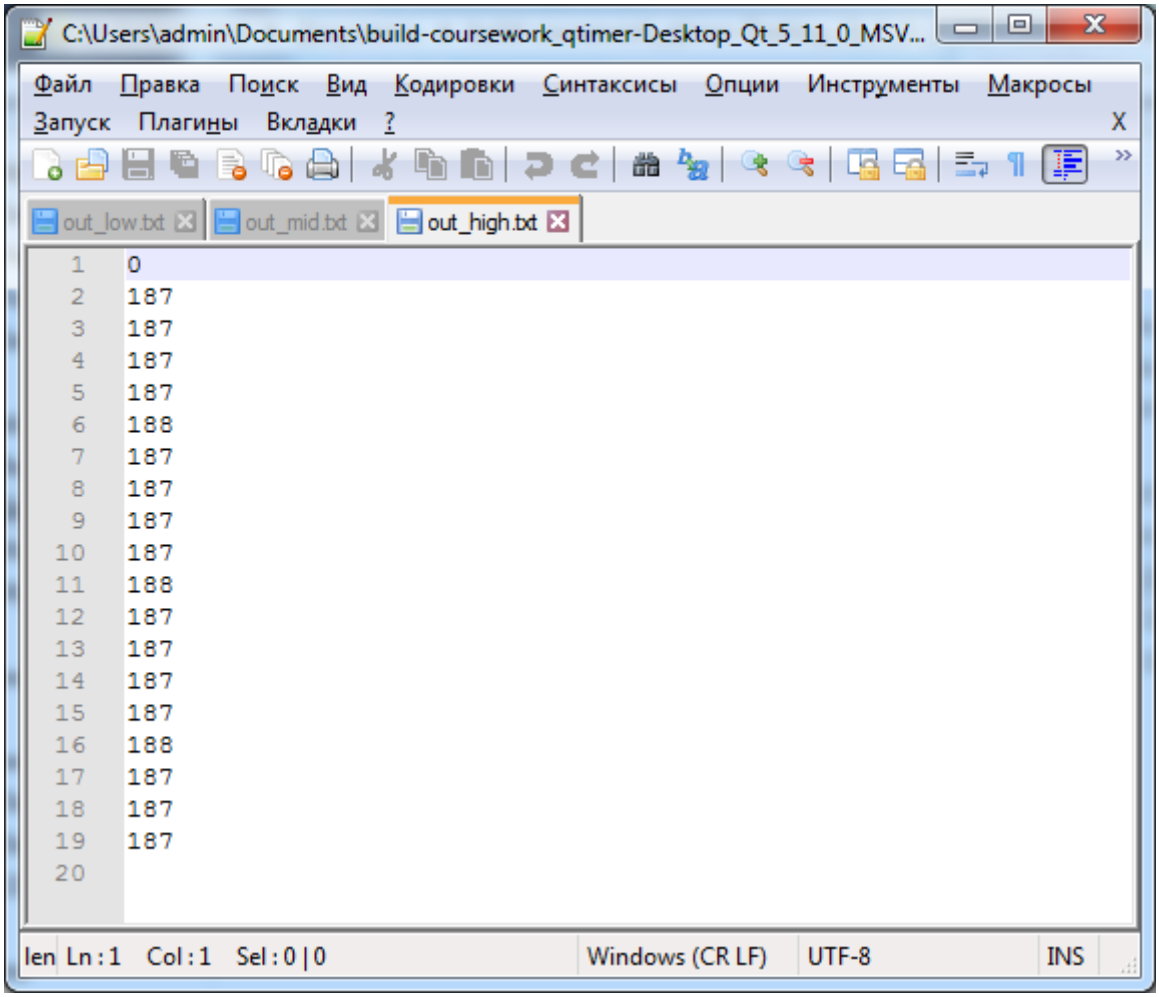



Рисунок 5.3 - Фактичний час виконання при частоті дискретизації 180 мс. і завантаженням ЦП до 100%.

На основі наявних даних побудуємо графіки, в облік візьмемо 10 виконань.

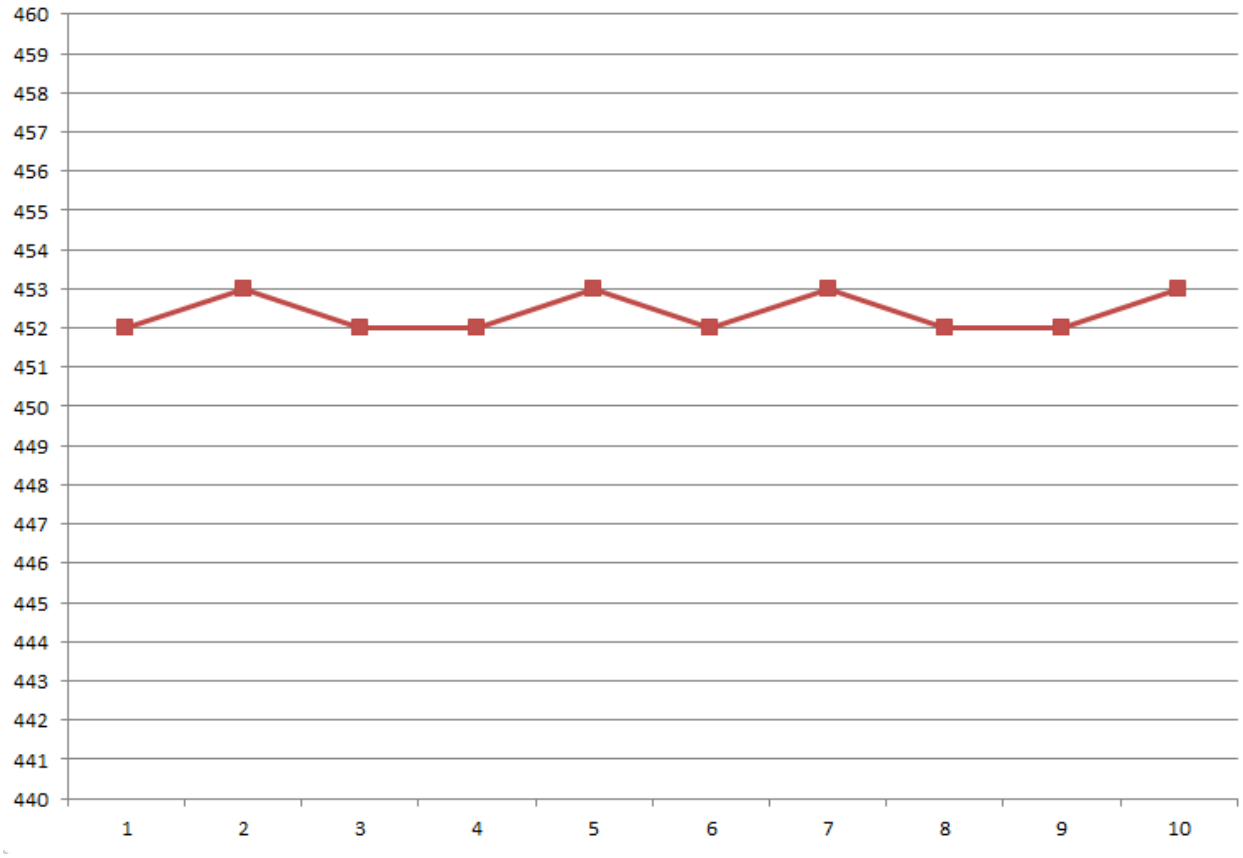



Рисунок 5.4 – Графік фактичного часу виконання при завантаженні ЦП до 30% і вибраній частоті дискретизації 450 мс.

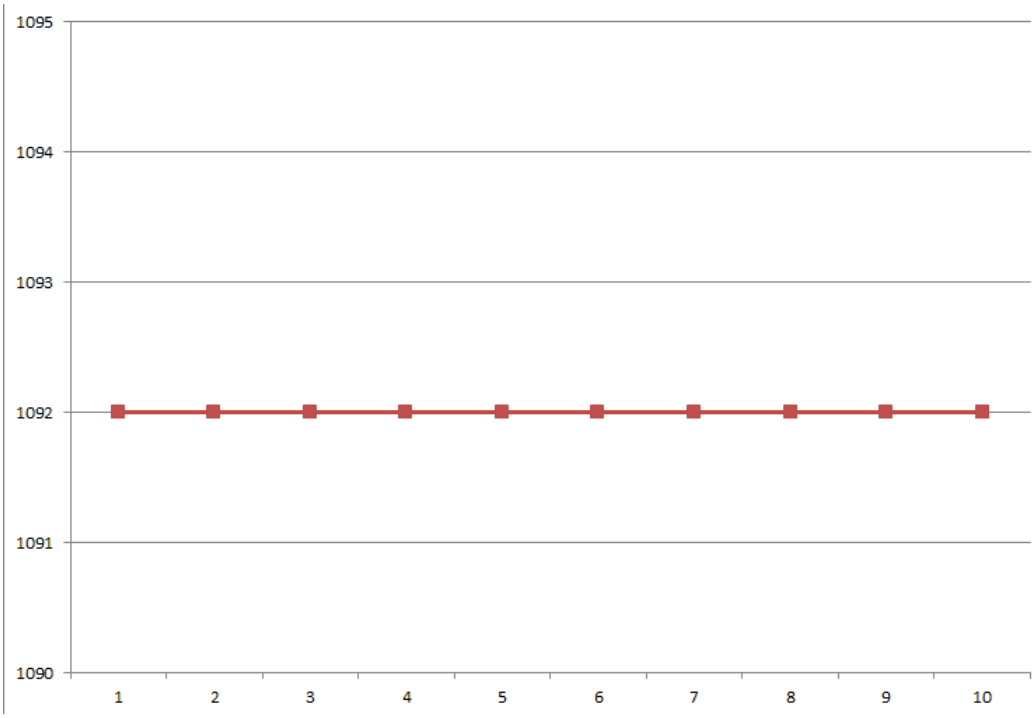



Рисунок 5.5 - Графік фактичного часу виконання при завантаженні ЦП до 70% і вибраній частоті дискретизації 1080 мс.

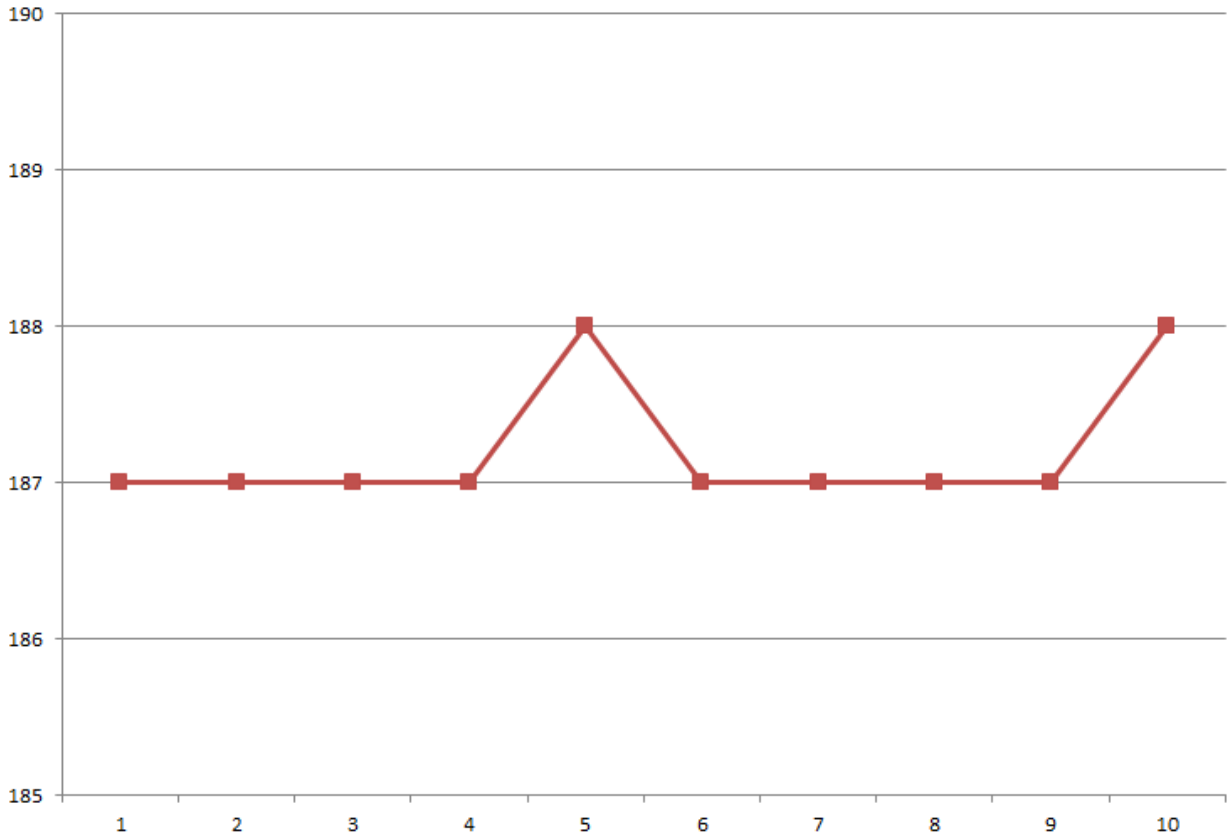



Рисунок 5.6 - Графік фактичного часу виконання при завантаженні ЦП до 100% і вибраній частоті дискретизації 1080 мс.

**Дослідження додатка на базі Timer (QML)**

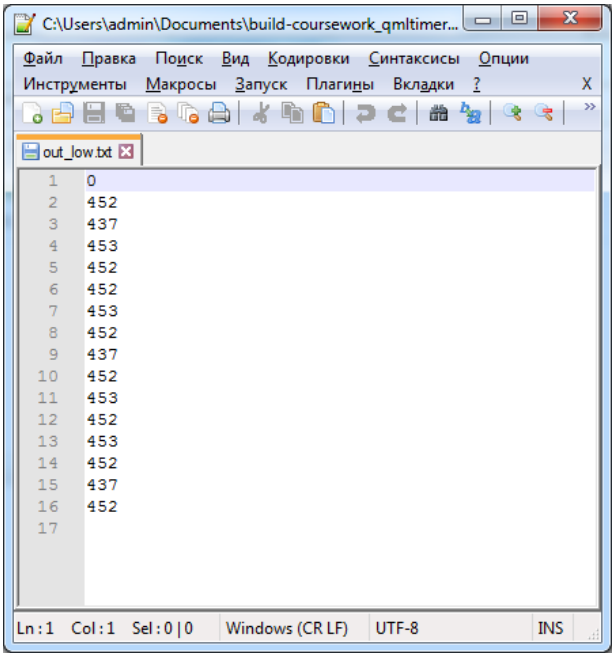



Рисунок 5.7 - Фактичний час виконання при частоті дискретизації 450 мс. і завантаженням ЦП до 30%.

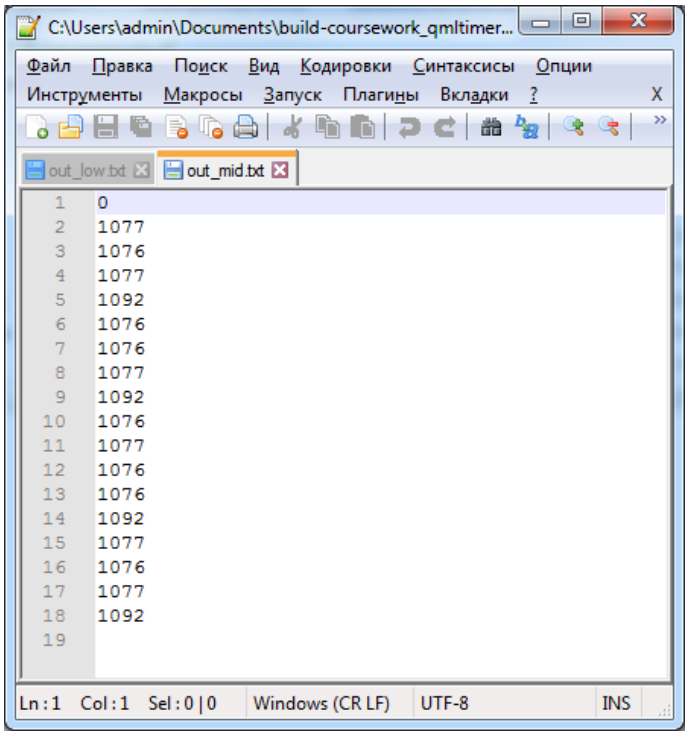



Рисунок 5.8 - Фактичний час виконання при частоті дискретизації 1080 мс. і завантаженням ЦП до 70%.

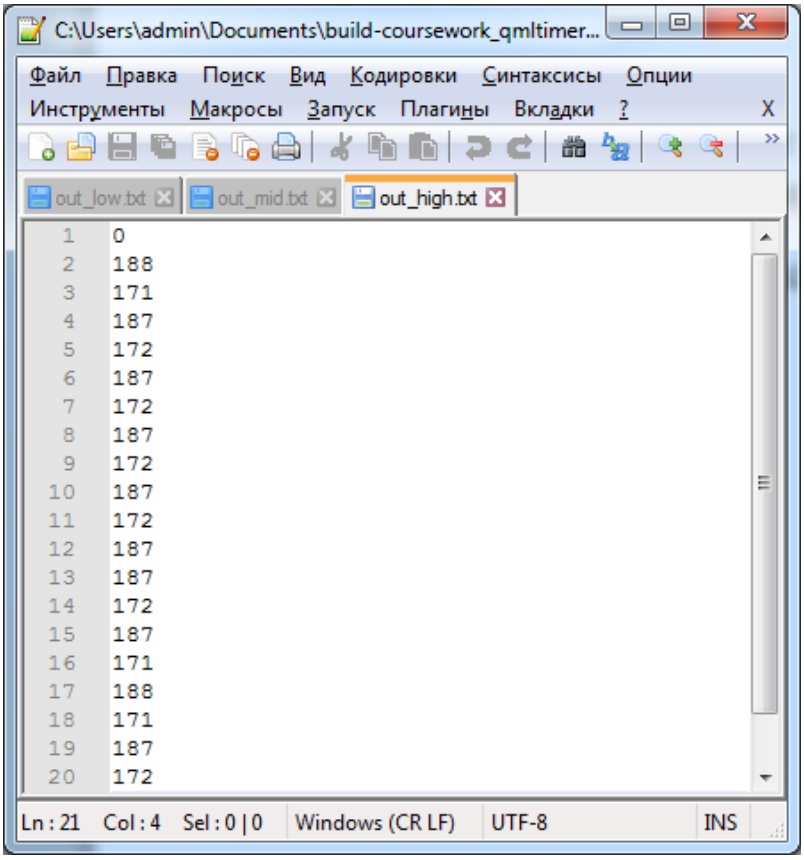



Рисунок 5.9 - Фактичний час виконання при частоті дискретизації 180 мс. і завантаженням ЦП до 100%.

На основі наявних даних побудуємо графіки, в облік візьмемо все ті ж 10 виконань.

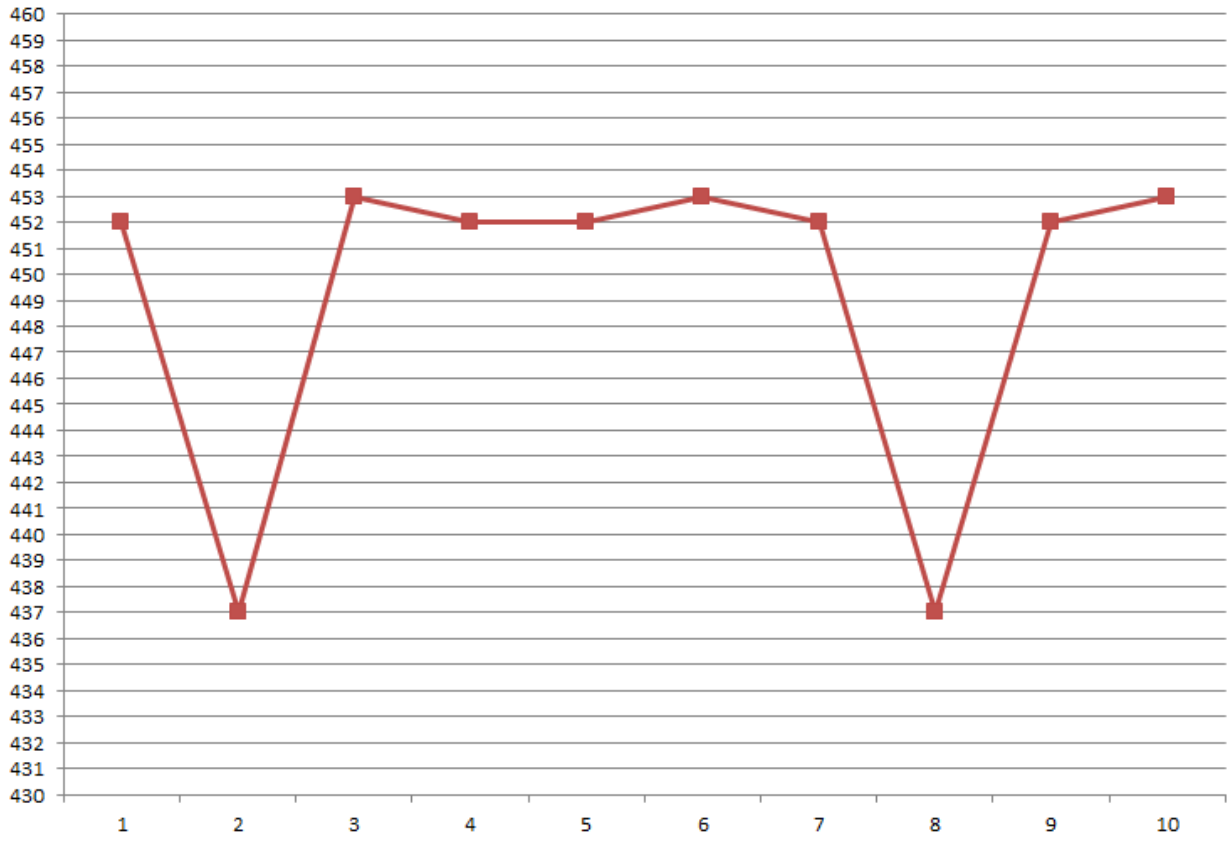



Рисунок 5.10 - Графік фактичного часу виконання при завантаженні ЦП до 30% і вибраній частоті дискретизації 450 мс.

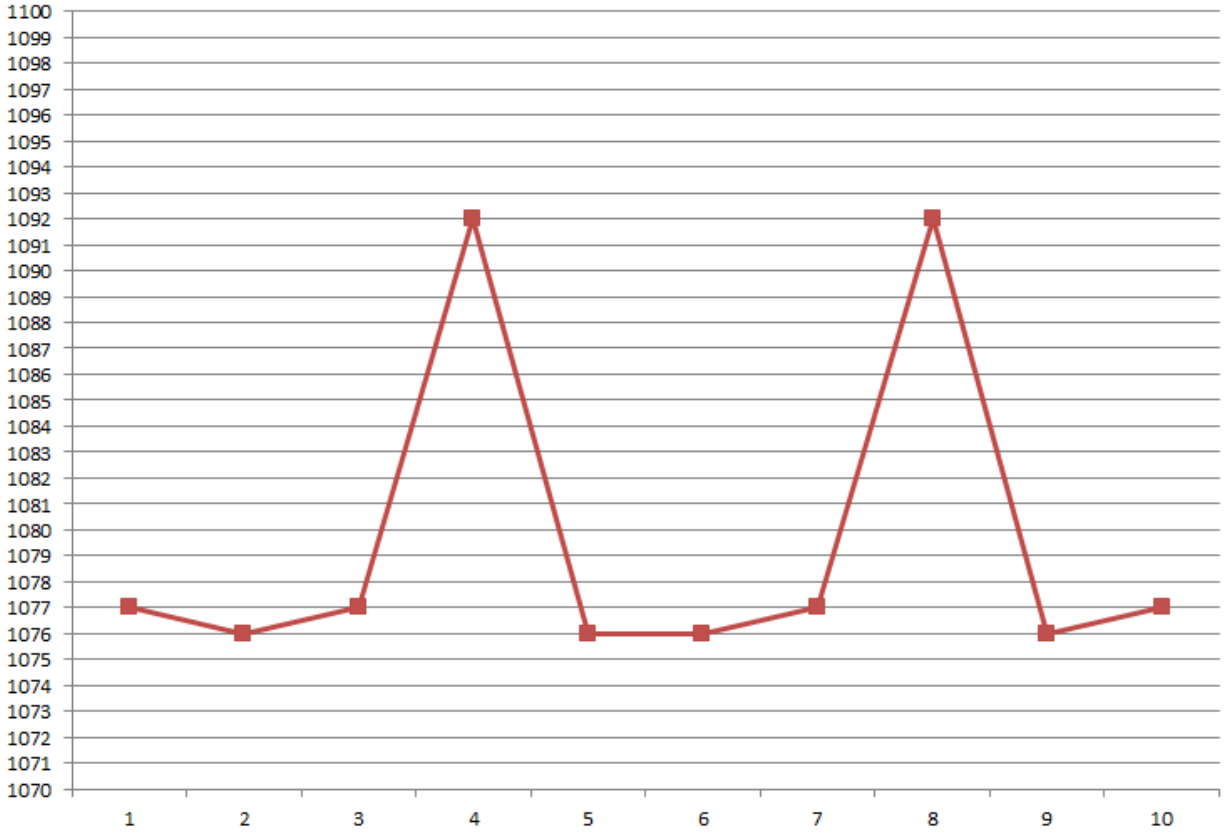



Рисунок 5.11 - Графік фактичного часу виконання при завантаженні ЦП до 70% і вибраній частоті дискретизації 1080 мс.

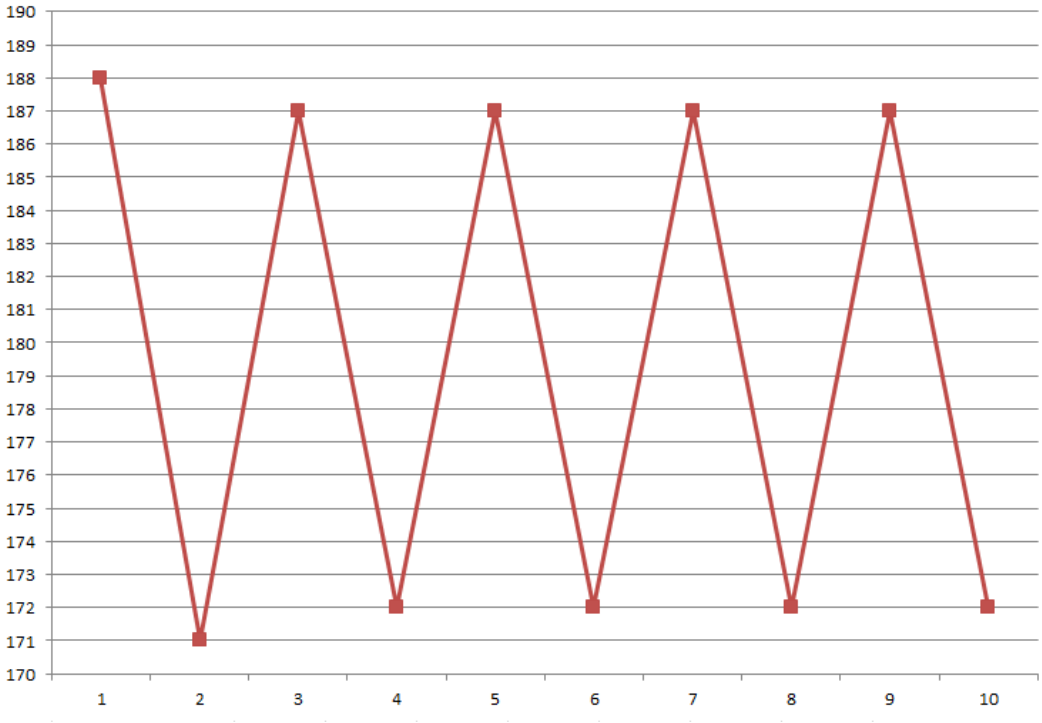



Рисунок 5.12 - Графік фактичного часу виконання при завантаженні ЦП до 100% і вибраній частоті дискретизації 180 мс.

Середня похибка для QTimer (C++):

- 450 мс., до 30% навантаження ЦП – 2,4 мс;
- 1080 мс., до 70% навантаження ЦП – 12 мс;
- 180 мс., до 100% навантаження ЦП – 7.2 мс.

Середня похибка для Timer (QML):

- 450 мс., до 30% навантаження ЦП – 0.7 мс;
- 1080 мс., до 70% навантаження ЦП – 0.4 мс;
- 180 мс., до 100% навантаження ЦП – 0.5 мс.

# ВИСНОВОК

Виконавши цю курсову роботу, я навчився досліджувати погрішність таймерів, отримав досвід в розробці графічного інтерфейсу за допомогою мови QML. Отримав досвід стикування двох мов: QML і C++, мова асемблера і C++, навчився будувати графіки в MS Excel. Навчився визначати підтримку тих, або інших технологій процесора, дізнаватися його назву та виробника (за допомогою асемблерної мнемоніки CPUID).

Що було виконано:

- визначення назви виробника процесору, на якому функціонує ПЗ,
- визначення розширених можливостей процесору,
- вибір та використання засобів встановлення часового інтервалу,
- організація функціонування періодичної частини додатку,
- обчислення та запис у файл фактичного часу періодизації,
- забезпечення коректного завершення виконання з відображенням результуючої інформації,
- реалізація алгоритму наповнення та очищення екранної форми.

Що можна було б ще виконати:

- зробити додаток кросплатформним,
- використати інший метод малювання елементів,
- переписати додаток так, щоб при запуску був вибір який таймер використати.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Системне програмування [Електронний ресурс] – режим доступу: https://dic.academic.ru/dic.nsf/ruwiki/29431
2. Використання таймера [Електронний ресурс] – режим доступу: http://www.codenet.ru/progr/visualc/vc/10.php
3. Qt [Електронний ресурс] – режим доступу: https://ru.wikipedia.org/wiki/Qt
4. QML [Електронний ресурс] – режим доступу: https://ru.wikipedia.org/wiki/QML
5. Сигнали та слоти [Електронний ресурс] – режим доступу: https://doc.qt.io/archives/3.3/signalsandslots.html
6. Функція GetTickCount() [Електронний ресурс] – режим доступу: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms724408(v=vs.85).aspx

# ДОДАТОК А – ЛІСТИНГ РЕАЛІЗАЦІЇ ДОДАТКА З QTimer (C++)

## getcpudata.h

```
#ifndef GETCPUDATA_H
#define GETCPUDATA_H

#include <QObject>
#include <QTimer>
#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <Windows.h>

class GetCPUData : public QObject
{
    Q_OBJECT
public:
    explicit GetCPUData(QObject *parent = nullptr);
    QString getCPUManufacturer();
    QString getCPUName();
    bool isPGESupported();
private:
    QTimer *timer;
    QFile file;
    QTextStream out;
    unsigned int t1 = 0;
    unsigned int t2 = 0;
private slots:
    void periodicFunction();
public slots:
    void getInterval(unsigned int interval);
    void stopPeriondicFunction();
signals:
    void catchQmlEvent();
};

#endif // GETCPUDATA_H
```

## getcpudata.cpp

```
#include "getcpudata.h"

GetCPUData::GetCPUData(QObject *parent) : QObject(parent)
{
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(periodicFunction()));
    file.setFileName("./out.txt");
    file.open(QIODevice::WriteOnly | QIODevice::Text);
    out.setDevice(&file);
}

QString GetCPUData::getCPUName()
{
    char processorManufacturer[12];
    __asm
```

```
    {
        lea edi, processorManufacturer
        xor eax, eax
        cpuid
        mov[edi], ebx
        mov[edi + 4], edx
        mov[edi + 8], ecx
        mov processorManufacturer[12], 0
    }
    QString cpuManufacturer(processorManufacturer);
    return cpuManufacturer;
}

QString GetCPUData::getCPUManufacturer()
{
    char processorInformation[49];
    __asm
    {
    xor edi, edi
    lea edi, processorInformation
    mov eax, 80000002h
    cpuid
    mov[edi], eax
    mov[edi + 4], ebx
    mov[edi + 8], ecx
    mov[edi + 12], edx
    mov eax, 80000003h
    cpuid
    mov[edi + 16], eax
    mov[edi + 20], ebx
    mov[edi + 24], ecx
    mov[edi + 28], edx
    mov eax, 80000004h
    cpuid
    mov[edi + 32], eax
    mov[edi + 36], ebx
    mov[edi + 40], ecx
    mov[edi + 44], edx
    mov processorInformation[48], 0
    }
    QString cpuModel(processorInformation);
    return cpuModel;
}

bool GetCPUData::isPGESupported()
{
    bool pge;
    __asm
    {
        mov eax, 1
        cpuid
        bt edx, 13
        jc M1
        jnc M2
        M1:
            mov pge, 1
            jmp M3
        M2:
            mov pge, 0
        M3:
    }
```

```
    return pge;
}

void GetCPUData::periodicFunction()
{
    t1 = GetTickCount();
    emit catchQmlEvent();
    if(t2 == 0) t2 = t1;
    qDebug() << t1 - t2;
    out << t1 - t2 << endl;
    out.flush();
    t2 = t1;
}

void GetCPUData::getInterval(unsigned int interval)
{
    switch(interval) {
    case 450:
        timer->setInterval(450);
        break;
    case 1080:
        timer->setInterval(1080);
        break;
    case 180:
        timer->setInterval(180);
        break;
    }
    timer->start();
}

void GetCPUData::stopPeriondicFunction()
{
    timer->stop();
}
```

## main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include <QQuickStyle>
#include "getcpudata.h"


int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    GetCPUData *cpuData = new GetCPUData();
    QString cpu = cpuData->getCPUName();
    QString manufacturer = cpuData->getCPUManufacturer();
    QString pge;
    (cpuData->isPGESupported()) ? pge = "Y" : pge = "N";
    engine.rootContext()->setContextProperty("pgeSupport", pge);
    engine.rootContext()->setContextProperty("processorModel", cpu);
    engine.rootContext()->setContextProperty("processorManufacturer",
manufacturer);
    engine.rootContext()->setContextProperty("cpuData", cpuData);
```

```
    QQuickStyle::setStyle("Fusion");

    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

## main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.3
import QtQuick.Controls 2.2

ApplicationWindow {
    id: window
    visible: true
    title: qsTr("Курсова робота ст. гр. КІ-16 Копилова К. О. ; Параметр
дискретизації: ")
    color: "#FFF8DC"
    property string functionColor: "lightgreen"
    property int times: 0
    property int canClose: 0
    maximumHeight: 480
    maximumWidth: 640
    minimumHeight: 480
    minimumWidth: 640

    onClosing: {
        close.accepted = false
        if(times == 0)
            close.accepted = true
        endFunction()
        if(canClose == 2)
            close.accepted = true
    }

    Column {
        anchors.centerIn: parent
        spacing: 5
        id: startSettings
        Text {
            id: text
            text: "Параметр дискретизації: "
            font.family: "Consolas"
            font.pointSize: 13
            color: "blue"
        }

        ComboBox {
            id: cbox
            model: [450, 1080, 180]
            width: 205
            font.family: "Consolas"
            font.pointSize: 13
        }

        Button {
            text: "Почати"
            onPressed: startFunction()
```

```
        width: cbox.width
        font.family: cbox.font
        font.pointSize: 13
    }
}

Shortcut {
    id: endShortcut
    sequence: "Ctrl+C"
    onActivated: endFunction()
    enabled: false
}

Connections {
    target: cpuData
    onCatchQmlEvent: period()
}

function startFunction() {

    switch(cbox.currentIndex) {
    case 0:
        window.title += "450"
        cpuData.getInterval(450)
        break;
    case 1:
        window.title += "1080"
        cpuData.getInterval(1080)
        break;
    case 2:
        window.title += "180"
        cpuData.getInterval(180)
        break;
    }
    startSettings.visible = false
}

Column {
    id: summaryInformation
    visible: false
    anchors.centerIn: parent
    spacing: 10

    Text {
        id: cpu
        text: "Виробник: " + processorModel
        font.family: "Consolas"
        font.pointSize: 13
        color: "blue"
    }

    Text {
        id: manufacturer
        text: "CPU: " + processorManufacturer
        font.family: "Consolas"
        font.pointSize: 13
        color: "blue"
    }

    Text {
        id: pgeSup
```

```
            text: (pgeSupport) === "Y" ? "Підтримка PGE: Так" : "Підтримка
PGE: Hi"
            font.family: "Consolas"
            font.pointSize: 13
            color: manufacturer.color
        }
    }

    function endFunction() {
        summaryInformation.visible = true
        cpuData.stopPeriondicFunction()
        ++canClose
        colId.visible = false
        colId2.visible = false
        rowId.visible = false
        rowId1.visible = false
        colId3.visible = false
        rowId2.visible = false
        colId4.visible = false
        rowId3.visible = false
        colId5.visible = false
        rowId4.visible = false
        colId6.visible = false
        rowId5.visible = false
        colId7.visible = false
        rowId6.visible = false
    }


////////////////////////////////////////////////////////////////////////////

    Column {
        anchors.centerIn: parent
        id: colId
        Repeater {
            id: repeat1
            model: 0
            Text {
                text: pgeSupport
                font.family: "Consolas"
                font.pointSize: 15
                color: functionColor
            }
        }
    }

    Row {
        x: 329; y: 182
        id: rowId
        Repeater {
            id: repeat2
            model: 0
            Text {
                text: pgeSupport
                font.family: "Consolas"
                font.pointSize: 15
                color: functionColor
            }
        }
    }
```

```
Column {
    id: colId2
    x: 427; y: 200
    Repeater {
        id: repeat3
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
    x: 424; y: 361
    id: rowId1
    width: 0
    layoutDirection: Qt.RightToLeft
    Repeater {
        id: repeat4
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId3
    x: 238; y: 363
    height: 0
    rotation: 180
    Repeater {
        id: repeat5
        model: 0
        Text {
            rotation: 180
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
    x: 250; y: 64
    id: rowId2
    Repeater {
        id: repeat6
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
```

```
        }
    }
}

Column {
    id: colId4
    x: 536; y: 80
    Repeater {
        id: repeat7
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
    x: 534; y: 425
    id: rowId3
    width: 0
    layoutDirection: Qt.RightToLeft
    Repeater {
        id: repeat8
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId5
    x: 105; y: 428
    height: 0
    rotation: 180
    Repeater {
        id: repeat9
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
            rotation: 180
        }
    }
}

Row {
    x: 118; y: 37
    id: rowId4
    Repeater {
        id: repeat10
        model: 0
        Text {
```

```
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId6
    x: 591; y: 55
    Repeater {
        id: repeat11
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
    x: 590; y: 446
    id: rowId5
    width: 0
    layoutDirection: Qt.RightToLeft
    Repeater {
        id: repeat12
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId7
    x: 30; y: 450
    height: 0
    rotation: 180
    Repeater {
        id: repeat13
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
            rotation: 180
        }
    }
}

Row {
    x: 42; y: 13
    id: rowId6
```

```
    Repeater {
        id: repeat14
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

function period() {
    ++times
    endShortcut.enabled = true
    if(times >= 0 && times <= 5)
        ++repeat1.model
    if(times > 5 && times <= 15)
        ++repeat2.model
    if(times > 15 && times <= 23)
        ++repeat3.model
    if(times > 23 && times <= 40)
        ++repeat4.model
    if(times > 40 && times <= 53)
        ++repeat5.model
    if(times > 53 && times <= 80)
        ++repeat6.model++
    if(times > 80 && times <= 96)
        ++repeat7.model
    if(times > 96 && times <= 135)
        ++repeat8.model
    if(times > 135 && times <= 152)
        ++repeat9.model
    if(times > 152 && times <= 196)
        ++repeat10.model
    if(times > 196 && times <= 214)
        ++repeat11.model
    if(times > 214 && times <= 265)
        ++repeat12.model
    if(times > 265 && times <= 284)
        ++repeat13.model
    if(times > 284 && times <= 338)
        ++repeat14.model
    if(times > 338 && times <= 392)
        --repeat14.model
    if(times > 392 && times <= 413)
        --repeat13.model
    if(times > 413 && times <= 464)
        --repeat12.model
    if(times > 464 && times <= 482)
        --repeat11.model
    if(times > 482 && times <= 526)
        --repeat10.model
    if(times > 526 && times <= 543)
        --repeat9.model
    if(times > 543 && times <= 582)
        --repeat8.model
    if(times > 582 && times <= 598)
        --repeat7.model
    if(times > 598 && times <= 625)
        --repeat6.model
```

```
            if(times > 625 && times <= 638)
                --repeat5.model
            if(times > 638 && times <= 655)
                --repeat4.model
            if(times > 655 && times <= 663)
                --repeat3.model
            if(times > 663 && times <= 673)
                --repeat2.model
            if(times >= 673 && times <= 678) {
                --repeat1.model
                repeat1.model = 0
                repeat2.model = 0
                repeat3.model = 0
                repeat4.model = 0
                repeat5.model = 0
                repeat6.model = 0
                repeat7.model = 0
                repeat8.model = 0
                repeat9.model = 0
                repeat10.model = 0
                repeat11.model = 0
                repeat12.model = 0
                repeat13.model = 0
                repeat14.model = 0
                times = 0
            }
            if(pgeSupport === "Y") {
                colId2.x = 437
                colId2.y = 206
                rowId1.x = 435
                rowId1.y = 368
                colId3.x = 230
                colId3.y = 368
                rowId2.x = 247
                rowId2.y = 69
                colId4.x = 558
                colId4.y = 91
                rowId3.x = 555
                rowId3.y = 437
                colId5.x = 87
                colId5.y = 437
                rowId4.x = 99
                rowId4.y = 46
                colId6.x = 616
                colId6.y = 69
                rowId5.x = 614
                rowId5.y = 459
                colId7.x = 2
                colId7.y = 461
                rowId6.x = 13
                rowId6.y = 24
            }
        }
    }
```

# ДОДАТОК Б – ЛІСТИНГ РЕАЛІЗАЦІЇ ДОДАТКА 3 Timer (QML)

## getcpudata.h

```
#ifndef GETCPUDATA_H
#define GETCPUDATA_H

#include <QObject>
#include <QTimer>
#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <Windows.h>

class GetCPUData : public QObject
{
    Q_OBJECT
public:
    explicit GetCPUData(QObject *parent = nullptr);
    QString getCPUManufacturer();
    QString getCPUName();
    bool isPGESupported();
private:
    QFile file;
    QTextStream out;
    unsigned int t1 = 0;
    unsigned int t2 = 0;
public slots:
    void periodicFunction();
signals:
    void catchQmlEvent();
};

#endif // GETCPUDATA_H
```

## getcpudata.cpp

```
#include "getcpudata.h"

GetCPUData::GetCPUData(QObject *parent) : QObject(parent)
{
    file.setFileName("./out.txt");
    file.open(QIODevice::WriteOnly | QIODevice::Text);
    out.setDevice(&file);
}

QString GetCPUData::getCPUManufacturer()
{
    char processorManufacturer[12];
    __asm
    {
        lea edi, processorManufacturer
        xor eax, eax
        cpuid
        mov[edi], ebx
        mov[edi + 4], edx
        mov[edi + 8], ecx
        mov processorManufacturer[12], 0
```

```
    }
    QString cpuManufacturer(processorManufacturer);
    return cpuManufacturer;
}

QString GetCPUData::getCPUName()
{
    char processorInformation[49];
    __asm
    {
    xor edi, edi
    lea edi, processorInformation
    mov eax, 80000002h
    cpuid
    mov[edi], eax
    mov[edi + 4], ebx
    mov[edi + 8], ecx
    mov[edi + 12], edx
    mov eax, 80000003h
    cpuid
    mov[edi + 16], eax
    mov[edi + 20], ebx
    mov[edi + 24], ecx
    mov[edi + 28], edx
    mov eax, 80000004h
    cpuid
    mov[edi + 32], eax
    mov[edi + 36], ebx
    mov[edi + 40], ecx
    mov[edi + 44], edx
    mov processorInformation[48], 0
    }
    QString cpuModel(processorInformation);
    return cpuModel;
}

bool GetCPUData::isPGESupported()
{
    bool pge;
    __asm
    {
        mov eax, 1
        cpuid
        bt edx, 13
        jc M1
        jnc M2
        M1:
            mov pge, 1
            jmp M3
        M2:
            mov pge, 0
        M3:
    }
    return pge;
}

void GetCPUData::periodicFunction()
{
    t1 = GetTickCount();
    emit catchQmlEvent();
    if(t2 == 0) t2 = t1;
```

```
    qDebug() << t1 - t2;
    out << t1 - t2 << endl;
    out.flush();
    t2 = t1;
}
```

main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "getcpudata.h"
#include <QQuickStyle>


int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    GetCPUData *cpuData = new GetCPUData();
    QString cpu = cpuData->getCPUName();
    QString manufacturer = cpuData->getCPUManufacturer();
    QString pge;
    QQuickStyle::setStyle("Fusion");
    (cpuData->isPGESupported()) ? pge = "Y" : pge = "N";
    engine.rootContext()->setContextProperty("pgeSupport", pge);
    engine.rootContext()->setContextProperty("processorModel", cpu);
    engine.rootContext()->setContextProperty("processorManufacturer",
manufacturer);
    engine.rootContext()->setContextProperty("cpuData", cpuData);
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.11
import QtQuick.Controls 2.2

ApplicationWindow {
    id: window
    visible: true
    title: qsTr("Курсова робота ст. гр. КІ-16 Копилова К. О. ; Параметр
дискретизації: ")
    color: "#FFF8DC"
    property string functionColor: "lightgreen"
    property int times: 0
    property int canClose: 0
    maximumHeight: 480
    maximumWidth: 640
    minimumHeight: 480
    minimumWidth: 640

    onClosing: {
        close.accepted = false
```

```
    if(times == 0)
        close.accepted = true
    endFunction()
    if(canClose >= 2)
        close.accepted = true
}

Timer {
    id: timer
    onTriggered: cpuData.periodicFunction()
    repeat: true
}

Column {
    anchors.centerIn: parent
    spacing: 5
    id: startSettings
    Text {
        id: text
        text: "Параметр дискретизації: "
        font.family: "Consolas"
        font.pointSize: 13
        color: "blue"
    }

    ComboBox {
        id: cbox
        model: [450, 1080, 180]
        width: 205
        font.family: "Consolas"
        font.pointSize: 13
    }

    Button {
        text: "Почати"
        onPressed: startFunction()
        width: cbox.width
        font.family: cbox.font
        font.pointSize: 13
    }
}

Shortcut {
    id: endShortcut
    sequence: "Ctrl+C"
    onActivated: endFunction()
    enabled: false
}

Connections {
    target: cpuData
    onCatchQmlEvent: period()
}

function startFunction() {

    switch(cbox.currentIndex) {
    case 0:
        timer.interval = 450
        window.title += "450"
        break;
```

```
            case 1:
                timer.interval = 1080
                window.title += "1080"
                break;
            case 2:
                timer.interval = 180
                window.title += "180"
                break;
            }
            startSettings.visible = false
            timer.running = true
        }

        Column {
            id: summaryInformation
            visible: false
            anchors.centerIn: parent
            spacing: 10

            Text {
                id: cpu
                text: "Виробник: " + processorModel
                font.family: "Consolas"
                font.pointSize: 13
                color: "blue"
            }

            Text {
                id: manufacturer
                text: "CPU: " + processorManufacturer
                font.family: "Consolas"
                font.pointSize: 13
                color: "blue"
            }

            Text {
                id: pgeSup
                text: (pgeSupport) === "Y" ? "Підтримка PGE: Так" : "Підтримка
PGE: Ні"
                font.family: "Consolas"
                font.pointSize: 13
                color: manufacturer.color
            }
        }

        function endFunction() {
            summaryInformation.visible = true
            ++canClose
            timer.running = false
            colId.visible = false
            colId2.visible = false
            rowId.visible = false
            rowId1.visible = false
            colId3.visible = false
            rowId2.visible = false
            colId4.visible = false
            rowId3.visible = false
            colId5.visible = false
            rowId4.visible = false
            colId6.visible = false
            rowId5.visible = false
```

```
        colId7.visible = false
        rowId6.visible = false
    }


////////////////////////////////////////////////////////////////////////////
///

    Column {
        anchors.centerIn: parent
        id: colId
        Repeater {
            id: repeat1
            model: 0
            Text {
                text: pgeSupport
                font.family: "Consolas"
                font.pointSize: 15
                color: functionColor
            }
        }
    }

    Row {
        x: 329; y: 182
        id: rowId
        Repeater {
            id: repeat2
            model: 0
            Text {
                text: pgeSupport
                font.family: "Consolas"
                font.pointSize: 15
                color: functionColor
            }
        }
    }

    Column {
        id: colId2
        x: 427; y: 200
        Repeater {
            id: repeat3
            model: 0
            Text {
                text: pgeSupport
                font.family: "Consolas"
                font.pointSize: 15
                color: functionColor
            }
        }
    }

    Row {
        x: 424; y: 361
        id: rowId1
        width: 0
        layoutDirection: Qt.RightToLeft
        Repeater {
            id: repeat4
            model: 0
```

```
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId3
    x: 238; y: 363
    height: 0
    rotation: 180
    Repeater {
        id: repeat5
        model: 0
        Text {
            rotation: 180
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
    x: 250; y: 64
    id: rowId2
    Repeater {
        id: repeat6
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId4
    x: 536; y: 80
    Repeater {
        id: repeat7
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
    x: 534; y: 425
    id: rowId3
    width: 0
```

```
    layoutDirection: Qt.RightToLeft
    Repeater {
        id: repeat8
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Column {
    id: colId5
    x: 105; y: 428
    height: 0
    rotation: 180
    Repeater {
        id: repeat9
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
            rotation: 180
        }
    }
}

Row {
    x: 118; y: 37
    id: rowId4
    Repeater {
        id: repeat10
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}
Column {
    id: colId6
    x: 591; y: 55
    Repeater {
        id: repeat11
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}

Row {
```

```
    x: 590; y: 446
    id: rowId5
    width: 0
    layoutDirection: Qt.RightToLeft
    Repeater {
        id: repeat12
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}
Column {
    id: colId7
    x: 30; y: 450
    height: 0
    rotation: 180
    Repeater {
        id: repeat13
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
            rotation: 180
        }
    }
}
Row {
    x: 42; y: 13
    id: rowId6
    Repeater {
        id: repeat14
        model: 0
        Text {
            text: pgeSupport
            font.family: "Consolas"
            font.pointSize: 15
            color: functionColor
        }
    }
}
function period() {
    ++times
    ++counter
    endShortcut.enabled = true
    if(times >= 0 && times <= 5)
        ++repeat1.model
    if(times > 5 && times <= 15)
        ++repeat2.model
    if(times > 15 && times <= 23)
        ++repeat3.model
    if(times > 23 && times <= 40)
        ++repeat4.model
    if(times > 40 && times <= 53)
        ++repeat5.model
    if(times > 53 && times <= 80)
```

```
    ++repeat6.model++
if(times > 80 && times <= 96)
    ++repeat7.model
if(times > 96 && times <= 135)
    ++repeat8.model
if(times > 135 && times <= 152)
    ++repeat9.model
if(times > 152 && times <= 196)
    ++repeat10.model
if(times > 196 && times <= 214)
    ++repeat11.model
if(times > 214 && times <= 265)
    ++repeat12.model
if(times > 265 && times <= 284)
    ++repeat13.model
if(times > 284 && times <= 338)
    ++repeat14.model
if(times > 338 && times <= 392)
    --repeat14.model
if(times > 392 && times <= 413)
    --repeat13.model
if(times > 413 && times <= 464)
    --repeat12.model
if(times > 464 && times <= 482)
    --repeat11.model
if(times > 482 && times <= 526)
    --repeat10.model
if(times > 526 && times <= 543)
    --repeat9.model
if(times > 543 && times <= 582)
    --repeat8.model
if(times > 582 && times <= 598)
    --repeat7.model
if(times > 598 && times <= 625)
    --repeat6.model
if(times > 625 && times <= 638)
    --repeat5.model
if(times > 638 && times <= 655)
    --repeat4.model
if(times > 655 && times <= 663)
    --repeat3.model
if(times > 663 && times <= 673)
    --repeat2.model
if(times >= 673 && times <= 678) {
    --repeat1.model
    repeat1.model = 0
    repeat2.model = 0
    repeat3.model = 0
    repeat4.model = 0
    repeat5.model = 0
    repeat6.model = 0
    repeat7.model = 0
    repeat8.model = 0
    repeat9.model = 0
    repeat10.model = 0
    repeat11.model = 0
    repeat12.model = 0
    repeat13.model = 0
    repeat14.model = 0
    times = 0
}
```

```
        if(pgeSupport === "Y") {
            colId2.x = 437
            colId2.y = 206
            rowId1.x = 435
            rowId1.y = 368
            colId3.x = 230
            colId3.y = 368
            rowId2.x = 247
            rowId2.y = 69
            colId4.x = 558
            colId4.y = 91
            rowId3.x = 555
            rowId3.y = 437
            colId5.x = 87
            colId5.y = 437
            rowId4.x = 99
            rowId4.y = 46
            colId6.x = 616
            colId6.y = 69
            rowId5.x = 614
            rowId5.y = 459
            colId7.x = 2
            colId7.y = 461
            rowId6.x = 13
            rowId6.y = 24
        }
    }
}
```