МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

В. М. РЯБЕНЬКИЙ
О. О. УШКАРЕНКО
МАХМУД М. С. АЛЬ-СУОД

МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ, ЕЛЕКТРОТЕХНІЧНИХ ТА ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

Навчальний посібник

Рекомендовано Вченою радою НУК

Миколаїв
«Іліон»
2024

УДК 621.3:004.4
Р 97

А В Т О Р И :

В. М. РЯБЕНЬКИЙ, д-р техн. наук, проф., заслужений діяч науки і техніки України, завідувач кафедри програмованої електроніки, електротехніки і телекомунікацій НУК ім. адм. Макарова, Україна;

О. О. УШКАРЕНКО, д-р техн. наук, проф. кафедри програмованої електроніки, електротехніки і телекомунікацій НУК ім. адм. Макарова, Україна;

МАХМУД МОХАММАД САЛЕМ АЛЬ-СУОД, PhD, проф. інженерного відділу по електроенергетиці й механотроніці, Технічний університет Тафіла, Йорданія

Р Е Ц Е Н З Е Н Т И :

Г. В. ПАВЛОВ, д-р техн. наук, професор, проректор з наукової роботи НУК ім. адм. Макарова, м. Миколаїв;

А. В. ПЕРЕВЕРЗЄВ, д-р техн. наук, професор, проректор з наукової роботи Запорізького інституту економіки та інформаційних технологій, м. Запоріжжя;

Ю. П. КОНДРАТЕНКО, д-р техн. наук, професор, зав. кафедри інформаційних технологій Чорноморського національного університету ім. П. Могили, м. Миколаїв

Рекомендовано Вченою радою Національного університету кораблебудування імені адмірала Макарова як навчальний посібник для студентів вищих навчальних закладів

Рябенький В. М.

Р 97 Моделювання електронних, електротехнічних та електроенергетичних систем : навчальний посібник / В. М. Рябенький, О. О. Ушкаренко, Махмуд М. С. Аль-суод. — Миколаїв : Іліон, 2024. — 424 с.

ISBN 978-617-534-717-1

Описуються особливості моделювання електронних, електротехнічних та електроенергетичних систем в контексті практичного застосування розглянутих моделей. Наведено огляд сучасних методів моделювання, особливості моделювання мікропроцесорних систем керування. Значна увага приділяється питанням розробки програмного забезпечення для моделювання електричних та електронних кіл, використанню чисельних методів при вирішенні завдань моделювання. Наведено велику кількість прикладів моделей різноманітних електронних та електроенергетичних систем.

Для студентів спеціальностей 171 – «Електроніка», 172 – «Електронні комунікації та радіотехніка» і суміжних спеціальностей, а також всіх тих, хто цікавиться електротехнікою, електронікою і програмуванням.

УДК 621.3:004.4

© Рябенький В. М., Ушкаренко О. О., Аль-суод Махмуд М. С., 2024
© НУК ім. адм. Макарова, 2024

ISBN 978-617-534-717-1

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ 8

ВСТУП 9

Розділ 1
ОГЛЯД МЕТОДІВ ТА ЗАСОБІВ МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ ПРИСТРОЇВ ТА СИСТЕМ

1.1 Принципи системного підходу при моделюванні електронних систем 11
1.2 Загальна методика моделювання систем 15
1.3 Характеристики та моделі електронних схем 19
1.4 Інструментальні засоби моделювання електронних систем 23
1.5 Моделі компонентів і схем промислової електроніки 25
1.6 Моделювання електронних систем на компонентному рівні 28
1.7 Еквівалентні схеми елементів електронних систем 29
1.8 SPICE-моделі компонентів схем 32
1.9 Математичні моделі дискретних пристроїв 34
1.10 Автоматні алгоритми обробки інформаційних потоків в локальних мережах 43
1.11 Мови опису апаратури AHDL, VHDL та Verilog 48
1.12 Методи моделювання цифрових та аналогових пристроїв за допомогою SPICE-опису 51
1.13 Марківські моделі електронних систем 56
1.14 Імітаційне моделювання як метод проведення системних досліджень 59

Р о з д і л 2
МОДЕЛЮВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

2.1 Опис роботи з системою моделювання Proteus 65
2.2 Модель лабораторного стенду для дослідження мікроконтролерів 74
2.3 Моделювання багатопроцесорного режиму зв'язку 78
2.3.1 Організація зв'язку між мікроконтролерами по послідовному інтерфейсу UART 78
2.3.2 Організація обміну даними між мікроконтролерами по інтерфейсу SPI 83
2.4 Основи моделювання пристроїв Arduino 86
2.5 Моделі систем керування серводвигуном та кроковим двигуном 94
2.6 Моделювання мікропроцесорних систем збору і обробки даних 101
2.6.1 Моделювання системи з інфрачервоним датчиком руху HC-SR501 101
2.6.2 Моделювання роботи ультразвукового датчика відстані HC-SR04 107
2.6.3 Моделювання мікропроцесорної системи з датчиком освітленості 110
2.7 Покрокове моделювання та налагодження програм в AVR Studio 112
2.8 Моделювання алгоритмів цифрової обробки сигналів 118

Р о з д і л 3
МОДЕЛЮВАННЯ АПАРАТНО-ПРОГРАМНИХ ЗАСОБІВ ВІДОБРАЖЕННЯ ІНФОРМАЦІЇ

3.1 Моделювання систем керування напівпровідниковими індикаторами 137
3.2 Модель системи керування напівпровідниковим семисегментним індикатором 141
3.2.1 Мультиплексне управління індикатором 143

3.2.2 Моделювання системи керування матричним індикатором ... 145
3.2.3 Модель системи керування семисегментними індикаторами з регістрами зсуву ... 148
3.3 Моделювання пристрою «біжучий рядок» ... 151
3.4 Моделювання системи керування алфавітно-цифровим рідкокристалічним індикатором ... 155
3.5 Моделювання РКІ з послідовним інтерфейсом ... 157
3.6 Модель мікропроцесорної системи керування графічним РКІ ... 159
3.7 Спеціалізовані програмні засоби для моделювання роботи РКІ ... 169

Розділ 4
РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ СХЕМ

4.1 Програмні засоби моделювання електричних кіл постійного струму ... 171
4.2 Розробка програм для параметричного аналізу лінійних кіл змінного струму ... 180
4.3 Програмні засоби для модельних досліджень резонансних явищ в електронних системах ... 187
4.4 Розробка програмних засобів для моделювання трифазних кіл ... 194
4.5 Програмні засоби для моделювання активних фільтрів ... 210
4.6 Використання чисельних методів при моделюванні перехідних процесів в електронних системах ... 227
4.6.1 Програмна реалізація явного і неявного методів чисельного інтегрування Ейлера ... 230
4.6.2 Програмна реалізація методу Рунге-Кутта ... 235
4.7 Програмні засоби моделювання електронних систем з напівпровідниковими елементами ... 242

Р о з д і л 5
ІМІТАЦІЙНЕ ТА КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ ПРОЦЕСІВ В СУДНОВИХ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМАХ

5.1 Моделювання процесів комутації навантаження в СЕЕС ... 254

5.2 Використання ланцюгів Маркова для моделювання та аналізу режимів роботи СЕЕС ... 255

5.3 Модельні дослідження впливу характеристик інформаційного каналу на якість керування СЕЕС ... 271

5.4 Оцінка впливу процесу комутації навантаження на завантаженість каналу зв'язку ... 278

5.5 Оцінка ефективності використання паралельних обчислень у завданнях управління ... 281

5.6 Моделювання динамічних та аварійних режимів в СЕЕС ... 283

5.7 Комп'ютерна модель СЕЕС для дослідження процесу синхронізації генераторів ... 296

5.8 Модельні дослідження регуляторів для стабілізації коефіцієнту потужності в СЕЕС ... 300

5.9 Моделювання роботи споживачів електроенергії і силових агрегатів ... 311

Р о з д і л 6
ВИКОРИСТАННЯ UML ПРИ ВИРІШЕННІ ЗАВДАНЬ МОДЕЛЮВАННЯ АВТОМАТИЗОВАНИХ СИСТЕМ КЕРУВАННЯ

6.1 Ієрархічна модель автоматизованої системи керування судновою електроенергетичною системою ... 318

6.2 Статична UML-модель системи керування ... 324

6.3 Розробка UML-моделі інформаційних потоків ... 328

6.4 Використання UML-FB в проектуванні та моделюванні систем керування автономними електростанціями ... 331

6.5 Концептуальна модель системи моніторингу за станом СЕЕС 341
6.6 Методика створення програмного забезпечення для систем автоматизації електроенергетичних установок 345
6.7 Статична модель інтерфейсу користувача 351
6.8 Динамічна модель та аналіз повідомлень 360

Р о з д і л 7
МОДЕЛЬНІ ДОСЛІДЖЕННЯ ЯКОСТІ РОБОТИ ПРОГРАМНО-АПАРАТНИХ ЗАСОБІВ МОНІТОРИНГУ ТА КЕРУВАННЯ СЕЕС

7.1 Формальний опис елементів автоматизованих робочих місць оператора електроенергетичної системи 364
7.2 Використання UML-діаграм станів для опису алгоритмів роботи компонентів програмного забезпечення 371
7.3 Моделювання процесів взаємодії компонентів програмного забезпечення за допомогою паттернових мереж 394
7.4 Мультикомп'ютерна модель для дослідження процесів управління СЕЕС 398
7.5 Фізична модель суднової електроенергетичної системи та її автоматизованої системи керування 409
7.6 Використання фізичної моделі СЕЕС для підготовки фахівців з експлуатації суднового електрообладнання 413

СПИСОК ЛІТЕРАТУРИ 419

СПИСОК УМОВНИХ СКОРОЧЕНЬ

АД – асинхронний двигун
АЕЕС – автономна електроенергетична система
АРМ – автоматизоване робоче місце
АСК – автоматизована система керування
АФ – активний фільтр
АФЧХ – амплітудно-фазова частотна характеристика
АЦП – аналого-цифровий перетворювач
АЧХ – амплітудно-частотна характеристика
ВАХ – вольт-амперна характеристика
ГРЩ – головний розподільний щит
ДГА – дизель-генераторний агрегат
ДПФ – дискретне перетворення Фур'є
ЕОМ – електронно-обчислювальна машина
ЕРП – електророзподільний пристрій
ЕРС – електрорушійна сила
КЗ – коротке замикання
ККД – коефіцієнт корисної дії
КПФ – комплексна передатна функція
КС – комбінаційна схема
КСУ – комп'ютеризована система управління
ЛАЧХ – логарифмічна амплітудно-частотна характеристика
ЛФЧХ – логарифмічна фазова частотна характеристика
МВП – метод вузлових потенціалів
МК – мікроконтролер
МРС – магніторушійна сила
НВЧ – надвисокочастотні
ОА – обслуговуючий апарат
ОЗП – оперативний запам'ятовуючий пристрій
ПЗ – програмне забезпечення
ПЗП – постійний запам'ятовуючий пристрій
ПП – перехідний процес
РКІ – рідкокристалічний індикатор
САПР – система автоматизованого проектування
САУ – система автоматичного управління
СЕЕС – суднова електроенергетична система
СМО – система масового обслуговування
УАПП – універсальний асинхронний приймач/передавач
ФЧХ – фазо-частотна характеристика
ЦА – цифровий автомат
HDL – мова опису апаратури
HMI – людино-машинний інтерфейс
UML – уніфікована мова моделювання

ВСТУП

Моделювання як інструмент наукових досліджень у галузі електроніки, електротехніки та електроенергетики застосовується останнім часом дедалі ширше. У літературних джерелах описуються різні інструменти моделювання та об'єкти моделювання. При моделюванні електронних пристроїв та систем, пристроїв автономної, у тому числі суднової електроенергетики, виникають проблеми, пов'язані з точністю математичних розрахунків, узгодженням фізичного та модельного часу, організацією обміну даними між різними програмами, у яких моделюються об'єкти управління та їх системи керування.

У популярних програмах для моделювання, наприклад Matlab, міститься велика кількість моделей пристроїв, які легко адаптуються до конкретних параметрів і дозволяють вирішувати найрізноманітніші завдання. Можливість підключення додаткових апаратних та програмних модулів значно розширює можливості середовища моделювання для дослідження як традиційних, так і досить складних процесів. Але все ж таки можливості його не безмежні, і для багатьох практичних завдань його використання супроводжується значними труднощами.

Одна з проблем моделювання процесів в електроенергетиці полягає у відсутності можливості використання мікропроцесорних систем зі своїми алгоритмами управління, що обмежує коло завдань, що вирішуються. Багато програм дозволяють моделювати аналогові, цифрові та мікропроцесорні системи, а результати моделювання відображаються за допомогою численних віртуальних приладів, які є в цих програмах. Однак у них відсутня можливість передачі результатів моделювання до зовнішнього середовища для подальшого використання. У програмі для моделювання електронних систем Proteus ці недоліки усунуті. З використанням спеціальних комунікаційних модулів є можливість виконати моделювання мікропроцесорних систем керування та перевірити їхню роботу на моделі, яка була створена в іншій програмі, наприклад MatLab. Використання комунікаційних можливостей цих програм дозволяє створювати мультикомп'ютерні моделі з мікропроцесорними системами управління об'єктами автономної, в тому числі й суднової електроенергетики, зокрема дизель-генераторними агрегатами. Подібний підхід дозволяє розробникам перенести програмне забезпечення мікропроцесорних систем управління в реальні пристрої без необхідності його зміни, що зменшує час розробки та дозволяє заздалегідь виявити всі помилки програмного забезпечення, виходячи з результатів моделювання. Зв'язок між комп'ютерами організується за одним із стандартних інтерфейсів (RS232, Ethernet), що також дозволяє провести дослідження інформаційних потоків, оцінити вплив затримок передачі інформаційно-керуючих пакетів на якість керування.

Постійно зростаюча складність керованих людиною машин та механізмів, об'єктів та процесів вимагають нових методів та засобів для прове-

дення моделювання. Система Matlab Simulink разом із системою моделювання Proteus відкриває можливості створення гібридних моделей, і дозволяє моделювати практично будь-які процеси, що відбуваються під час роботи електронного та електротехнічного обладнання, що дає можливість досліджувати роботу систем керування та виконувати їх оптимізацію до встановлення на реальному об'єкті. При розгляді складних об'єктів управління, таких як суднові електроенергетичні системи, моделювання постає як один із найважливіших етапів. Створення мультикомп'ютерних моделей дозволяє об'єднати як силову електроніку, і мікропроцесорні системи управління, що відкриває широкі можливості моделювання як енергетичних процесів, а й процесів управління.

У навчальному посібнику розглядаються всі зазначені аспекти моделювання електронних, електротехнічних та електроенергетичних систем. При цьому значна увага приділяється практичним питанням вирішення завдань моделювання. Наведено велику кількість прикладів моделювання складних електронних, електротехнічних та електроенергетичних систем, розробки та моделювання компонентів програмного забезпечення автоматизованих систем керування, опису розподілених систем керування за допомогою уніфікованої мови моделювання UML, моделюванню інформаційних каналів зв'язку між електронними пристроями та системами.

Автори висловлюють щиру подяку к.т.н. Дорогань О. І., к.т.н. Нгуєну Ван Тханю, к.т.н. До Ань Туану, к.т.н. Аднану Сулейману, к.т.н. Язіду Джамалу Ісмаїлу Альшайху, Халеду Омару Ганнаму, Воскобоєнко В. І., які в різні роки навчались на кафедрі програмованої електроніки, електротехніки і телекомунікацій НУК ім. адм. Макарова та допомогали авторам цього навчального посібника в проведенні наукових досліджень в галузі моделювання електронних, електротехнічних та електроенергетичних систем, результати яких опубліковано в [59–67].

Навчальний посібник призначений для студентів спеціальностей 171 – «Електроніка», 172 – «Електронні комунікації та радіотехніка» і суміжних спеціальностей, а також всіх тих, хто цікавиться електротехнікою, електронікою і програмуванням.

РОЗДІЛ 1. ОГЛЯД МЕТОДІВ ТА ЗАСОБІВ МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ ПРИСТРОЇВ ТА СИСТЕМ

Моделювання є найбільш ефективним способом дослідження складних систем різного призначення – технічних, економічних, екологічних, соціальних, інформаційних – як на етапі їх проектування, так і в процесі експлуатації. Можливості моделювання систем далеко не вичерпані, тому постійно з'являються найновіші методи та технології моделювання. Створення моделі – кропіткий і творчий процес, що вимагає від дослідника не тільки глибоких теоретичних знань з різних математичних та технічних дисциплін, але й творчого підходу до розв'язання задач. Для освоєння мистецтва моделювання систем потрібно володіти знаннями з вищої математики, теорії ймовірностей, математичної статистики, теорії інженерного експерименту, системного аналізу і уміти програмувати. Досконале володіння універсальною мовою програмування, знання методів об'єктно-орієнтованого програмування необхідне для дослідника, що не тільки створює моделі систем, але й створює системи моделювання.

1.1. Принципи системного підходу при моделюванні електронних систем

Завданням системного аналізу є формування опису системи, що відповідає меті дослідження системи [1]. Опис системи складається з опису:

– набору вхідних змінних системи з вказанням їх основних характеристик;

– набору вихідних змінних системи, визначення яких забезпечує досягнення цілі дослідження;

– границь системи з вказанням того, що являється для системи її зовнішнім середовищем;

– елементів системи з вказанням їх основних властивостей;

– зв'язків між елементами системи.

Системний підхід до дослідження систем означає, що дослідник вивчає функціонування системи в цілому, не концентруючи свою увагу на окремих її частинах. Оснований системний підхід на визнанні факту, що навіть найліпше функціонування окремих підсистем та елементів системи не гарантує найліпшого функціонування всієї системи в цілому, оскільки завжди існує взаємодія між частинами системи. Всім системам притаманні характеристики, що являються перепоною на шляху до поліпшення функціонування системи [Шеннон]:

– змінюваність – жодна реальна система не являється статичною продовж тривалого проміжку часу, тому модель, що будується, має обмежений термін користування;

– наявність оточуючого середовища – в моделі має бути передбачений вплив зовнішнього середовища, який часто має випадковий характер;

– противоінтуїтивна поведінка – виникає як результат того, що інколи наслідок проявляється пізніше причини;

– тенденція до погіршення характеристик функціонування – окремі частини системи зношуються, в результаті чого погіршується їх робота, що в свою чергу призводить до непередбачуваних наслідків;

– взаємозалежність – всі частини системи залежать одна від одної, внаслідок чого погіршення характеристик функціонування однієї частини системи неминуче впливає на характеристики функціонування інших частин системи;

– організація – існує ієрархія підсистем, що підкоряється цільовому призначенню системи.

Будь-яка система об'єктивна і в той же час суб'єктивна з точки зору вибору границь системи та її елементів. Одному й тому ж самому об'єктивному процесу дослідник може поставити у відповідність різні системи і тільки досвід дослідника, його інтуїція і здатність творчо мислити допомагає здійснити вибір між багатьма варіантами і виконати дослідження системи найліпшим способом.

При створенні математичних моделей використання системного підходу припускає виділення декількох рівнів абстракції при описі об'єкту (електронної системи). Об'єднання рівнів, близьких по характеру математичного апарату, що використовується, призводить до утворення кількох рівнів в ієрархії функціональних моделей. В якості прикладу моделей різного рівня, які описують один і той самий процес, розглянемо приклад нагрівання елемента електронної схеми.

Якщо елемент електронної схеми розглядати як єдину систему, що нагрівається під дією теплового потоку (наприклад, радіатор, встановлений на інтегральний стабілізатор або транзистор в корпусі TO220), то процес можна описати за допомогою простого рівняння балансу:

$$T = T_0 + \frac{Q}{C_p \rho}, \tag{1.1}$$

де T – кінцева температура тіла, T_0 – початкова температура тіла, C_p, $\rangle$ – теплоємність та густина матеріалу тіла, Q – кількість тепла, що отримана тілом по відношенню до його об'єму. Будемо вважати модель (1.1) моделлю верхнього рівня, самого загального рівня абстракції.

Припустимо, потрібно знайти не лише кінцеву температуру об'єкта, але й зміну в часі. Тоді слід перейти до більш детального опису процесу, наприклад, в такому вигляді:

$$\frac{dT}{d\tau} = \frac{1}{C_p \rho} q(\tau), \tag{1.2}$$

де $q(\tau)$ – тепловий потік, віднесений до одиниці об'єму тіла, τ – час.

Для рішення цього рівняння слід вказати закон зміни теплового потоку $q(\tau)$ та задати початкову температуру тіла:

$$T\Big|_{\tau=0} = T_0.$$

Отримане рішення дозволить визначити зміну температури тіла у часі.

В математичних моделях (1.1) та (1.2) елемент електронної схеми розглядається як єдине ціле, і в них не входять просторові координати.

Якщо важливо провести аналіз зміни температури не тільки у часі, але й в різних точках простору, то слід перейти до ще більшої деталізації процесу. Приймемо, що тіло має форму довгого тонкого стрижня. Якщо знехтувати усіма розмірами стрижня окрім довжини, модель можна представити у вигляді наступного рівняння:

$$\frac{\delta T}{\delta \tau} = \frac{\lambda}{C_p \rho}\left(\frac{\delta^2 T}{\delta x^2}\right). \tag{1.3}$$

яке слід доповнити початковими та граничними умовами, що уточнюють перетікання процесу, наприклад такими:

$$T\Big|_{x=0} = f_1(\tau),$$

$$T\Big|_{x=1} = f_2(\tau),$$

$$T\Big|_{\tau=0} = \varphi(x),$$

де λ – теплопровідність матеріалу, l – довжина стрижня, x – просторова координата.

Рішення даної моделі дозволить визначити температуру в будь-якій точці стрижня в будь-який момент часу.

Проаналізуємо, як змінюється вигляд рівнянь математичної моделі процесу при переході від одного рівня абстракції до іншого.

Модель процесу, що представлена на верхньому рівні алгебраїчним рівнянням (1.1), ускладнюється на другому рівні абстракції та приймає вигляд диференціального рівняння (1.2). Перехід до третього, найбільш детального рівня, призводить до необхідності використовувати диференційне рівняння з частковими похідними (1.3). Однак за рахунок ускладнення моделі можна отримати додаткову інформацію про процес. Якщо в першому випадку можна визначити лише кінцеву температуру елемента електронної схеми, що моделюється, то в другому є можливість прослідкувати за процесом у часі, вважаючи, що температура всього тіла однакова. Модель третього рівня вже дозволяє досліджувати розподілення температури і в часі, і в просторі.

В даному прикладі наведено значно спрощений підхід до опису процесу. Так, не розглядається віддача тепла нагрітим елементом в оточуюче середовище. В моделі (1.3) процес розглядається лише по одній координаті,

а реальні елементи мають кінцеві розміри по всім просторовим координатам. Можна добавити й інші умови, які не враховані в прикладі. Врахування цих умов призведе до появи нових членів в рівняннях та значно ускладнить їх. Однак ускладнення математичної моделі робить її більш адекватною, більш наближеною до реальності, хоча і погіршує її економічність.

При моделюванні технічних об'єктів, в тому числі електронних схем та систем, розглянуті моделі та рівні абстракції називаються наступним чином. Модель виду (1.3) називають моделлю мікрорівня, виду (1.2) – макрорівня, виду (1.1) – мегарівня. При цьому характерним є наступне:

1. На мікрорівні абстракції використовують математичні моделі, що описують фізичний стан та процеси у суцільних середовищах. Для моделювання використовують апарат математичної фізики. Особливістю цих математичних моделей є відображення процесів, що протікають в неперервних просторі та часі. Вони представляють собою системи диференційних рівнянь. Такі математичні моделі часто називають моделями з розподіленими параметрами, оскільки в них параметри залежать від координат точок простору. Якщо в таких рівняннях час як незалежна змінна відсутня, то вони описують стаціонарний процес і називаються стаціонарними.

2. На макрорівні виконується дискретизація простору по функціональній ознаці. Тобто виділяються окремі зони, в яких процес можна вважати таким, що не залежить від просторових координат. Математичні моделі на цьому рівні представлені у вигляді систем диференційних рівнянь, де в якості незалежної змінної виступає лише час. Такі моделі називають моделями із зосередженими параметрами. При розгляді стаціонарного процесу на даному рівні математичні моделі отримують вигляд систем алгебраїчних рівнянь. Математичні моделі цього рівня є універсальними і придатними для дослідження як динамічних, так і статичних режимів.

3. На мегарівні за допомогою подальшого абстрагування від характеру фізичних процесів вдається ще більше спростити модель. Типовими моделями цього рівня є балансові співвідношення у вигляді систем алгебраїчних рівнянь.

Моделі, побудовані із застосуванням системного підходу, отримали назву системних моделей. Опис системи разом із вказанням цілі та задачі дослідження складає сутність концептуальної моделі системи. Назва «концептуальна» походить від латинського слова conceptio, що означає «сприйняття».

Умовно можна виділити такі етапи створення концептуальної моделі системи:

– Визначення цілі дослідження системи (орієнтація).

– Вибір рівня деталізації системи (стратифікація).

– Визначення елементів системи (деталізація).

– Визначення впливу зовнішнього середовища (локалізація).

– Визначення зв'язків між елементами системи та із зовнішнім середовищем (структуризація).

1.2. Загальна методика моделювання систем

Процес моделювання складається з кількох етапів [1]. На першому етапі дослідник визначає мету та задачу моделювання. На другому етапі, виходячи з мети та задачі моделювання, дослідник приступає до вербального опису системи. Опис набору змінних моделі, разом із описом структури системи та формулюванням цілі та задачі дослідження складає концептуальну модель системи. Виходячи з концептуальної моделі системи та з огляду на вибір інструментальних засобів, дослідник робить вибір теоретичної бази, на основі якої буде побудована модель системи. Обравши теоретичну базу моделювання, дослідник має описати систему, що розглядається, обраними елементами формального опису і визначити для них усі необхідні параметри. Формальне представлення системи має вигляд схеми, в якій указані зв'язки між елементами системи та зв'язки із зовнішнім середовищем і указані параметри елементів системи. У формальній моделі міститься також інформація, яким чином будуть знайдені вихідні змінні моделі в результаті моделювання.

На третьому етапі дослідник приступає до створення моделі. Спочатку виконується реалізація моделі за допомогою обраного програмного забезпечення. Потім виконується верифікація моделі, тобто перевірка алгоритму моделювання на відповідність задуму моделювання. Наприклад, змінюють значення вхідних змінних і спостерігають як модель реагує на таке змінювання. Якщо реакція моделі відповідає логіці її функціонування, то модель вважається правильною. Завершується створення моделі перевіркою адекватності моделі, що полягає у порівнянні значень вихідних змінних об'єкта, що моделюється, і моделі при однакових значеннях вхідних змінних. Очевидно, що таку перевірку можна здійснити тільки, якщо відомі деякі значення вхідних і вихідних змінних досліджуваного об'єкта.

Четвертий етап – це дослідження моделі. Результати моделювання стають корисними, якщо проведене змістовне дослідження моделі відповідно до цілі моделювання. Експерименти, що проводяться з моделлю, мають бути спочатку сплановані, потім – проведені, і наприкінці – статистично оброблені.

Аналіз результатів моделювання складається з оцінки точності результатів моделювання, оцінки стійкості результатів моделювання та оцінки чутливості результатів моделювання.

Формування висновків та пропозицій є завершальний етап моделювання, на якому підводяться підсумки та висловлюються думки щодо напрямів подальшого дослідження об'єкта моделювання.

Звісно, що процес моделювання може бути представлений етапами тільки у звіті про результати моделювання. У ході моделювання дослідникові доводиться неодноразово повертатись до попередніх етапів і уточнювати постановку задачі, формальний опис моделі, алгоритм реалізації або план проведення експериментів поступового наближаючись до мети.

На рис. 1.1 представлений процес моделювання з урахуванням можливих повернень до попередніх етапів моделювання.

Рис. 1.1.

Будь-яка електронна система складається з більш простих підсистем або елементів. Залежності, що описують функціонування елементарних підсистем, які є компонентами єдиної технічної системи, називають компонентними рівняннями. Розглянемо три рівняння, якими описуються основні процеси в електричному колі:

$$i = \frac{u}{R}, \quad i = C\frac{du}{d\tau}, \quad u = L\frac{di}{d\tau},$$

R – опір, C – ємність, L – індуктивність.

Перше з цих рівнянь описує процес розсіювання електричної енергії (її перетворення в тепло). Друге та третє рівняння описують процеси накопичення енергії в потенційній та кінетичній формах. Струм та напруга виступають в цих рівняннях фазовими змінними.

Можна сказати, що в електричній схемі існує три види найпростіших елементів – типу R, типу C і типу L. Сполученням цих найпростіших елементів, а також джерел фазових змінних, можна отримати математичну модель електричного кола.

Функціонування будь-якої системи може бути розглянуто як перетворення матерії (енергії або речовини). В системі будь-якої природи, в загальному випадку, можна виділити елементи трьох типів:

− розсіюють енергію;

− накопичують енергію в кінетичній формі;

− накопичують енергію в потенційній формі.

Фазові змінні характеризують стан процесу, а їх зміна у часі описує перехідні процеси. Однак є важливі відмінності у властивостях вказаних фазових змінних. Одна частина з них описує інтенсивні характеристики системи, що розглядається. Тому їх можна назвати потенціалами, або фазовими змінними потенціального типу. Типовим представником цієї групи для електронних систем можна назвати напругу *u*. Основною їх властивістю є те, що під дією різниці потенціалів виникають потоки енергії. У свою чергу, потоки описують екстенсивні характеристики систем та описуються змінними струменевого типу. Так, різниця напруг викликає електричний струм.

При синтезі математичної моделі системи, що складається з окремих елементів, недостатньо лише компонентних рівнянь. Важливо також знати, як ці елементи з'єднані між собою. Відомо, що при послідовному з'єднанні двох елементів струм в них однаковий, а загальна напруга дорівнює сумі напруг на кожному з елементів. При паралельному з'єднанні навпаки – напруги на елементах рівні, а загальний струм дорівнює сумі окремих струмів. Це витікає з відомих законів Кірхгофа. Для систем різної фізичної природи діють однакові закономірності в поведінці фазових змінних при різному з'єднанні елементів:

1. При послідовному з'єднанні двох елементів системи фазові змінні струменевого типу в них рівні, а змінні потенціального типу додаються, утворюючи сумарну величину.

2. При паралельному з'єднанні елементів, що утворюють систему, додаються фазові змінні струменевого типу, а потенціальні змінні рівні.

Для синтезу математичної моделі система розбивається на елементи. Для кожного елемента встановлюються відповідні компонентні рівняння. Вони доповнюються топологічними рівняннями зв'язків між елементами. Після цього до моделі можуть бути додані джерела фазових змінних.

Розглянемо приклад створення моделі електричного кола. В кабельну лінію, що має опір *r* та ємність *C*, подається імпульс амплітудою *U*. Визначити, через який час амплітуда імпульсу на виході лінії досягне 95% від *U*. Опором навантаження знехтувати.

Еквівалентна схема кабельної лінії приведена на рис. 1.2

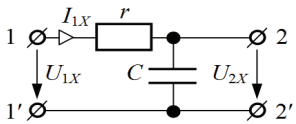


Рис. 1.2.

Запишемо компонентні рівняння:

$$i_r = \frac{u_r}{r},$$

$$i_C = C\frac{du_C}{dt}.$$

Топологічні рівняння:

$$u_{1X} = u_r + u_C,$$

$$u_C = u_{2X},$$

$$i_{1X} = i_r = i_C.$$

Можна записати:

$$C\frac{du_{2X}}{dt} = \frac{u_r}{r} = \frac{1}{r}(u_{1X} - u_{2X}).$$

Після перетворення отримуємо диференційне рівняння першого порядку, яке складає математичну модель подачі напруги по заданій кабельній лінії:

$$C\frac{du_{2X}}{dt} + \frac{u_{2X}}{r} = \frac{u_{1X}}{r}.$$

Отримане рівняння можна вирішити при відповідних початкових умовах: $u_{ВХ}=0$, при $0<t$.

Рішення рівняння:

$$u_{2X} = U(1 - e^{-\frac{t}{rC}}).$$

При $u_{2X}=0{,}95U$ отримаємо:

$$0{,}95U = U(1 - e^{-\frac{t}{rC}}),$$

$$t = -rC \cdot \ln 0{,}05.$$

Зміну напруги імпульсу на виході лінії демонструє графік на рис. 1.3.

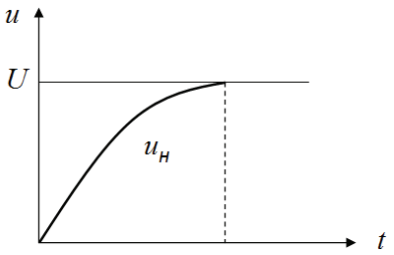


Рис. 1.3.

1.3. Характеристики та моделі електронних схем

Аналіз і розрахунок електронних схем здійснюється на основі моделювання реальних компонентів, тобто заміни їх деякими моделями, які з тим або іншим ступенем точності описують поведінку елементів схеми в різних режимах, тобто на основі створення математичних моделей компонентів.

Математичну модель елемента можна розглядати як деякий оператор P, що ставить у відповідність системі внутрішніх параметрів компонента або схеми $X=X_1, X_2,..., X_n$ сукупність пов'язаних з ними зовнішніх параметрів $Y=Y_1, Y_2,..., Y_n$. Зміст внутрішніх і зовнішніх параметрів, їх фізичний зміст змінюється в залежності від призначення моделі.

Для моделей компонентів зовнішніми параметрами є зазвичай струми і напруги, якщо модель призначена для розрахунку схем. Для розрахунку самого компонента зовнішніми параметрами виступають зазвичай густина струму, розподіл заряду, а також робочі управляючі дії.

Для моделей схем внутрішніми параметрами є електричні параметри компонентів, з яких вона складається, зовнішніми параметрами – струми, напруги, потужності, частотні та інші характеристики. Зв'язок між зовнішніми і внутрішніми параметрами здійснюється з використанням законів Кірхгофа (для консервативних систем) і законів Максвелла (для систем відкритих, де можливі втрати на випромінювання, що не враховуються законами Кирхгофа).

Вихідні параметри моделі – це показники, що характеризують функціональні, експлуатаційні, конструкторсько-технологічні, економічні та інші характеристики проектованого об'єкта. До таких показників можуть відноситися коефіцієнти передачі, маса і габарити проектованого об'єкта, надійність, вартість і т.п.

Поняття внутрішніх і вихідних параметрів інваріантні, так як при моделюванні на більш високому рівні вихідні параметри можуть стати внутрішніми і навпаки. Наприклад, опір резистора є внутрішнім параметром при моделюванні підсилювального пристрою, компонентом якого він є. Але цей опір буде вихідним параметром при моделюванні самого резистора.

Математичні моделі елементів електронних систем можуть бути класифіковані по ряду ознак:

1. За характером відображуваних процесів виділяють:

– статичні;

– динамічні моделі.

2. За способом представлення моделі розрізняють:

– аналітичні;

– графічні;

– табличні.

Аналітичні моделі визначають прилад або компонент у вигляді рівнянь, що описують його ВАХ, або у формі диференціальних рівнянь, що описують перехідні процеси в схемі і характеризують інерційність елемента.

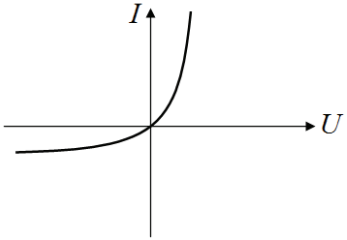
Графічні моделі дозволяють представити компоненти у вигляді графіків ВАХ або у вигляді еквівалентних схем заміщення.

Табличні моделі дозволяють представити схему або елемент у вигляді цифрових таблиць, отриманих в ході експериментального дослідження об'єкта моделювання та відповідних графіків експериментальних ВАХ. Табличні моделі використовують зазвичай в тому випадку, якщо аналітичну модель побудувати важко внаслідок складної залежності. Іноді при складних функціональних аналітичних залежностях для ВАХ їх свідомо табулюють, якщо це дозволяє обсяг пам'яті ЕОМ, і створюють таким чином можливо менш точну, але більш зручну модель. На рис. 1.4 представлені приклади опису моделі напівпровідникового діода: а – аналітичний; б – графічний, а в табл. 1.1 – табличний.

$$I = I_0\left(e^{\frac{U}{\varphi_T}} - 1\right),$$

$$I_0 = q \cdot S \cdot \left(\frac{D_p p_{n_0}}{L_p} + \frac{D_n p_{p_0}}{L_n}\right)$$

а



б

Рис. 1.4.

На рис. 1.4, а, q – заряд електрону, S – площа p-n-переходу, p_{n0}, n_{p0} – концентрації неосновних носіїв в областях n і p, відповідно; D_p, D_n – коефіцієнти дифузії дірок та електронів.

Таблиця 1.1.

$U_{ПР}$, В	-6	-4	-2	0	2	4	6	8
$I_{ПР}$, мА	-3	-2	-1	0	1,4	5	9	13

Точність моделі відносно невисока, що визначається прийнятими припущеннями. Реальна ВАХ діода відрізняється від ідеальної наявністю ділянки рекомбінаційної складової прямого струму, термогенераційної складової зворотного струму, струмами витоку, явищами пробою на зворотній гілці ВАХ, нарешті, наявністю опору базової області p-n-переходу і ємностями p-n-переходу, котрі характеризують інерційність процесів в приладі.

3. За характером залежностей моделі діляться на:

– лінійні;

– кусково-лінійні;

– нелінійні.

Є особливий клас кусково-лінійних моделей, нелінійність яких проявляється в обмеженій кількості точок стику лінійних ділянок. Нелінійні моделі виявляються більш точними, але і більш складними. На рис. 1.5 представлені графічні моделі біполярного транзистора: а – нелінійна модель; б – кусково-лінійна модель; в – лінійна модель.

Рис. 1.5.

4. За діапазоном робочих сигналів моделі класифікуються на:
– моделі великого сигналу;
– малосигнальні.

Малосигнальні моделі, як правило, представляють собою лінійні моделі; моделі для великого сигналу враховують нелінійність характеристик активних і пасивних елементів схеми (біполярних і польових підсилюючих приладів). На рис. 1.6 представлена динамічна модель польового транзистора для малого сигналу. Зазвичай при роботі в режимі підсилювача точка спокою знаходиться на пологій ділянці вихідної ВАХ при нормальному включенні транзистора (аналогічно активні нормальній області біполярного транзистора).

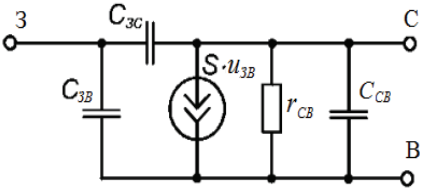


Рис. 1.6.

В області низьких частот модель описується всього двома параметрами – крутизною S і вихідним опором $r_{си}$. Ці параметри легко визначаються по вихідній ВАХ транзистора.

$$S = \left.\frac{\Delta I_C}{\Delta U_{ЗВ}}\right|_{U_{СВ}=\text{const}} = \frac{i_C}{u_{ЗВ}},$$

$$r_{СВ} = \left.\frac{\Delta U_{СВ}}{\Delta I_C}\right|_{U_{ЗВ}=\text{const}} = \frac{u_C}{i_C}.$$

5. За діапазоном робочих частот виділяють:

– низькочастотні;

– високочастотні;

– надвисокочастотні (НВЧ).

Низькочастотні моделі не враховують інерційність компонентів моделі, тому низькочастотні моделі використовують для розрахунку схем по постійному струму (в статичному режимі).

Високочастотні моделі – моделі більш високого рівня, вони враховують крім особливостей статичного режиму інерційність компонентів. Тому такі моделі доповнюють системою диференціальних рівнянь, яка враховує інерційність компонентів, або еквівалентними схемами реальних приладів на високих частотах – індуктивностями і ємностями виводів, інерційностями, що описують фізичні процеси в компонентах (наприклад, накопичення заряду), ємностями областей структур і т.п. Наприклад, вольтамперна характеристика реального резистора на постійному струмі не відрізняється від ідеальної. При створенні високочастотної моделі необхідно враховувати особливості виготовлення елемента. За рахунок наявності індуктивності виводів і резистивного шару опір резисторів на високій частоті стає комплексним, набуває індуктивний характер:

$$X_R = R + j\omega L \ .$$

З підвищенням частоти виявляються також ємнісні властивості резистора – в основному ємність між його виводами (рис. 1.7).

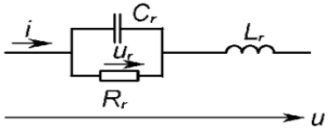
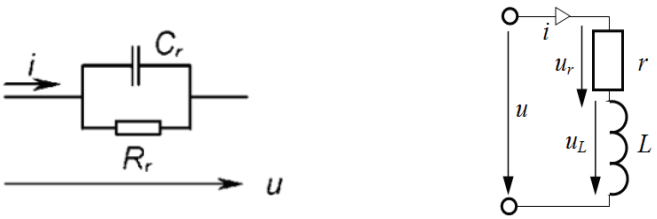


Рис. 1.7.

Приведену еквівалентну схему можна описати системою 2-х диференціальних рівнянь (аналітична модель):

$$\begin{cases} u = u_r + L_r \dfrac{di}{dt}, \\ i = \dfrac{u_r}{R_r} + C_r \dfrac{du_r}{dt}. \end{cases}$$

Тут C_r, L_r – величини паразитних параметрів, u_r – напруга на активній складовій резистора. В залежності від співвідношення параметрів еквівалентна схема резистора може бути спрощена. Для високоомних резисторів активний опір значно більше хвильового, можна знехтувати його індуктивною складовою (рис. 1.8, а). Для низькоомних резисторів можна знехтувати ємнісною складовою комплексного опору (рис. 1.8, б).



а б

Рис. 1.8.

Особливість низькочастотних і високочастотних моделей полягає в тому, що вони виконуються на зосереджених елементах і тому для цих моделей справедливі закони Кірхгофа.

НВЧ моделі відрізняються від високочастотних моделей урахуванням просторових і часових координат, тому для аналізу і розрахунку НВЧ-схем необхідно використовувати рівняння Максвела.

Рівняння математичної моделі можуть пов'язувати деякі фізичні характеристики компонентів, які повністю характеризують стан об'єкта, але не є вихідними або внутрішніми параметрами моделі (наприклад, струми і напруги в радіоелектронних пристроях, внутрішніми параметрами яких є номінали елементів електричних схем, а вихідними параметрами – вихідна потужність, коефіцієнт передачі тощо). Такі характеристики називають фазовими змінними.

1.4. Інструментальні засоби моделювання електронних систем

Віртуальне моделювання електронних схем дозволяє значно швидше відпрацювати ту чи іншу схему, тобто, підібрати складові електронного пристрою, не виготовляючи реального робочого зразка.

При віртуальному моделюванні очевидна ще одна перевага: навіть коли допущена критична помилка в побудові електронної схеми – це не призведе до пошкодження вимірювальних приладів або руйнування радіоелементів.

Таким чином, відпрацювавши різні варіанти електронної схеми на комп'ютері, можна вибрати найбільш прийнятний та приступити до виготовлення діючого зразка електронного пристрою.

Найчастіше для моделювання використовуються такі програми, як: Electronics Workbench, Multisim, OrCAR, MicroCap, Proteus та багато інших.

Для віртуального моделювання на комп'ютері користувач повинен знати склад електронної схеми, призначення радіоелементів, фізичну суть процесів, що проходять в схемі, тобто, підготовлений теоретично в межах програми курсу електроніки.

Проектування електронної схеми розпочинається введенням її (побудовою) в комп'ютер в програмі. Слід відзначити, що схема повинна бути

визначена, а її елементи попередньо розраховані. Після введення схеми в комп'ютер можливо провести численні зміни величин елементів, джерел живлення, вхідних сигналів в ту чи іншу сторону від розрахованих величин. Результати віртуального моделювання відображаються на багатоканальному осцилографі, віртуальних приладах, а на характерографі показана АЧХ фільтра (рис. 1.9).

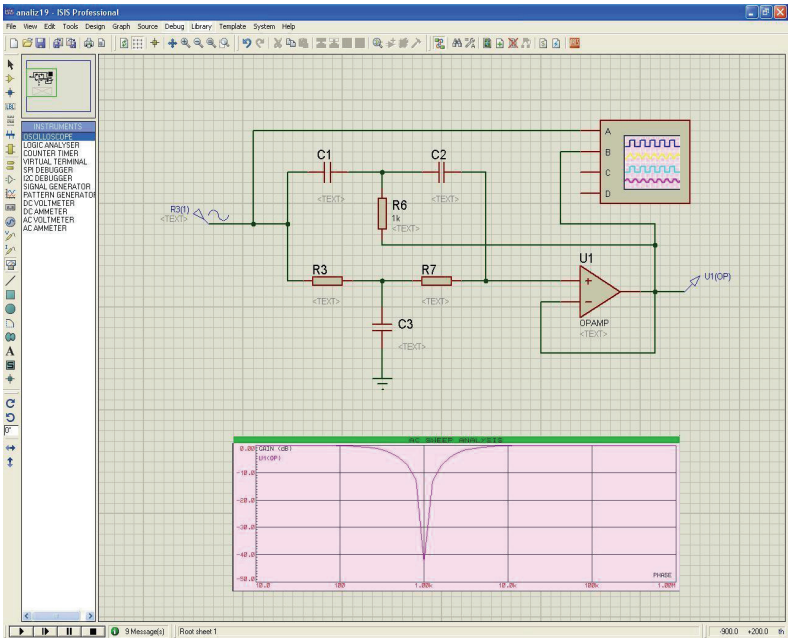


Рис. 1.9.

Необхідно сказати, що практичний зразок електронного пристрою побудований за віртуальною моделлю, може мати деякі відхилення параметрів від розрахованих. Головна причина в даному разі – зміни в часі параметрів резисторів, конденсаторів, котушок індуктивності, транзисторів та ін.

Принципи роботи з різними програмами моделювання електронних схем схожі. Після запуску програми відкриється вікно для створення схеми. У вікні розміщені значки параметрів: джерела живлення, резистори, котушки індуктивності, конденсатори, транзистори, діоди та ін., тобто панель компонентів. Також у вікні з'являються «іконки» приладів – це генератори, осцилографи, мультиметри, вимірювачі характеристик електронних схем та низка інших приладів – панель інструментів (Instruments). Активізація в панелі інструментів «іконок» дозволяє вивести той або інший прилад на поле, де буде створюватися схема електронного пристрою. Подвійне натискання лівою кнопкою «іконки» інструмента призведе до появи на екрані передньої панелі приладу з різноманітними кнопками керування.

Операції з компонентами виконуються подібно, причому дворазове натискання «іконки» необхідного компонента відкриває діалогове вікно, де можна вибрати будь-яку величину компонента, або зі списку числових величин встановити фіксоване числове значення компонента (у випадку, якщо компонент вибирався віртуальний).

Подвійне натискання лівою кнопкою миші на символах джерел живлення, струму, коливань приведе до відкриття діалогового вікна, де можливо змінити одиниці вимірювання, які відрізняються від встановлених в режимі «за замовчуванням».

При складанні електронної схеми пристрою спочатку на полі розміщують всі елементи схеми: резистори, конденсатори, діоди, котушки індуктивності, транзистори, мікросхеми, джерела живлення та сигналів, вимірювальні прилади. Згідно принципової схеми встановлюються всі фізичні величини елементів, вимірювальні прилади вмикаються для вимірювання відповідних електричних величин.

Надалі розміщують деталі схеми та прилади так, щоб не було накладання приладів на деталі схеми, звести до мінімуму кількість перетинів провідників. Схема повинна бути обов'язково заземлена.

1.5. Моделі компонентів і схем промислової електроніки

Побудова моделей – схем заміщення реальних компонентів електричних ланцюгів, що відображають з достатньою повнотою зв'язки між струмами і напругами на його виводах, є неоднозначним завданням. При достатньому розумінні фізичних процесів в об'єкті моделювання схема заміщення може в повному обсязі виражати співвідношення, що відображають його внутрішню структуру. Однак подібна фізична модель – навіть для порівняно простих об'єктів зазвичай містить велику кількість елементів. Прості моделі – так звані макромоделі – можна отримати моделюванням співвідношень між струмами і напругами на окремих виводах системи, що визначаються експериментально. Це найбільш зручно при побудові статичних моделей компонентів при незмінних в часі струмах і напругах. Для врахування динамічних властивостей об'єкта, що визначають співвідношення між змінними в часі струмами і напругами, до такої макромоделі додають паразитні ємності між окремими виводами модельованого об'єкта й індуктивності виводів. Характер подібної динамічної моделі і число елементів, що входять до неї, істотно залежать від швидкості зміни струмів і напруг (при періодичних процесах – від діапазону частот). Так, для правильного опису поведінки елементів при високих частотах може знадобитися модель з розподіленими параметрами.

Іншим важливим аспектом вибору схеми заміщення є рівень або діапазон зміни струмів і напруг, для якого призначена модель. Глобальна модель описує поведінку об'єкта при будь-яких можливих струмах і напругах на його виводах. Вона, як правило, містить нелінійні елементи. Локальну модель використовують для опису роботи об'єкта в обмеженому діапазоні

зміни струмів і напруг, коли можлива лінеаризація характеристик нелінійних елементів – заміна їх дотичними в робочій точці. Така модель містить тільки лінійні елементи. Подібний режим роботи, при якому зміни в часі складових струмів і напруг малі порівняно з їх постійними значеннями в робочій точці (малосигнальний режим), часто спеціально забезпечується в ланцюгах з нелінійними елементами, в яких необхідно зберегти форму змінної складової сигналу при його перетворенні в ланцюзі.

Однак при виборі схеми заміщення не слід прагнути врахувати всі фактори (включаючи і малоістотні в даних конкретних умовах). У ряді випадків використання моделі, побудованої з урахуванням малоістотних факторів, не просто ускладнює проблему аналізу, але й призводить до принципових ускладнень при отриманні рішення. Тому при моделюванні компонентів електронних схем доцільно обмежуватися найпростішою схемою заміщення, що відбиває лише найбільш істотні сторони досліджуваного процесу.

Загальна методика отримання математичної моделі елементу (ММЕ) включає в себе виконання наступних процедур:

1. Визначення властивостей елемента, які повинна відображати модель. Включення до переліку таких розкритих властивостей, оцінка яких не вимагається для прийняття проектних рішень на певному етапі проектування, призводить до ускладнення моделі та нераціонального витрачання ресурсів САПР.

2. Збір вихідної інформації про обрані властивості об’єкта. Використовуються знання про закономірності функціонування об’єкта, довідкові дані, плануються і реалізуються необхідні експериментальні дослідження і т.д.

3. Отримання структури моделі, тобто математичних виразів і рівнянь, що описують в загальному вигляді відношення між фазовими змінними та параметрами об’єкта. Іноді замість системи рівнянь модель зручно представляти на деякій формальній графічній мові (наприклад, мовою графів або еквівалентних схем), яка забезпечує однозначний перехід на мову математичних формул.

4. Розрахунок числових значень параметрів ММЕ для заданого елемента або групи елементів.

Для оцінки властивостей транзисторів поряд з їх характеристиками використовують параметри. Розрізняють дві групи параметрів: первинні та вторинні.

Первинні – це власні параметри транзистора, що характеризують його фізичні властивості і не залежать від схеми ввімкнення транзистора:

r_e – диференційний опір емітерного переходу в прямому напрямку;

$r_б$ – об’ємний опір бази;

$r_к$ – диференційний опір колекторного переходу в зворотному напрямку;

$C_e, C_к$ – ємності емітерного і колекторного переходів. Вплив цих ємностей на коефіцієнт підсилення на низьких частотах незначний.

Недоліком використання первинних параметрів є те, що їх неможливо виміряти безпосередньо за допомогою приладів, тому що точки для підключення вимірювальних приладів знаходяться в структурі транзистора.

Суть *вторинних* параметрів можливо пояснити, якщо представити транзистор як чотириполюсник, що має дві вхідні клеми та дві вихідні. Вхідні величини – це I_1, U_1 (вхідний струм та напруга), вихідні $-I_2, U_2$. В чотириполюснику прийнято, що сигнали, тобто прирости $\Delta I_1, \Delta U_1, \Delta I_2$ та ΔU_2 незначні. Ці величини взаємно пов'язані. Дві величини приймають як незалежні, дві інші будуть залежними, змінними. Для них складається система з двох рівнянь де вони зв'язані з незалежними величинами через коефіцієнти – які називаються системою параметрів. Це система Z-параметрів (має розмірність опору); *Y*-параметрів (розмірність провідності); і *h*-параметрів (змішана система). Остання використовується найчастіше. Щоб знайти *h*-параметри, складається система рівнянь, де незалежними змінними є $\Delta I_1, \Delta U_2$:

$$\left.\begin{aligned}\Delta U_1 &= h_{11}\Delta I_1 + h_{12}\Delta U_2\\ \Delta I_2 &= h_{21}\Delta I_1 + h_{22}\Delta U_2\end{aligned}\right\}. \qquad (1.4)$$

В системі рівнянь (1.4) чотири параметри $h_{11}, h_{22}, h_{21}, h_{12}$ з різною розмірністю. В залежності від схеми вмикання параметри транзистора будуть мати відповідні позначення, наприклад, $h_{21\text{е}}$, $h_{21\text{б}}$, $h_{21\text{к}}$.

$$h_{11} = \frac{\Delta U_1}{\Delta I_1},$$

при $U_2 = const$, вхідний опір транзистора [Ом].

$$h_{22} = \frac{\Delta I_2}{\Delta U_2},$$

при $I_1 = const$, вихідна провідність [Сименс].

$$h_{12} = \frac{\Delta U_1}{\Delta U_2},$$

при $I_1 = const$, коефіцієнт внутрішнього зворотного зв'язку.

$$h_{21} = \frac{\Delta I_2}{\Delta I_1},$$

при $U_2 = const$, коефіцієнт передачі струму.

На рис. 1.10 показано, як розраховуються h-параметри з використанням статичних характеристик транзистора.

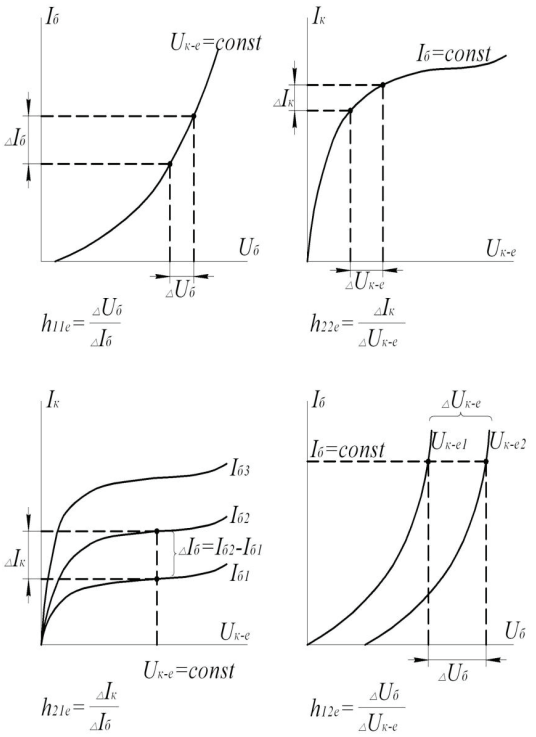


Рис. 1.10.

Використовуючи розраховані величини *h*-параметрів, можна знайти фізичні (внутрішні) параметри транзистора:

$$r_б = h_{11e} - \frac{h_{12e} \cdot (1 + h_{12e})}{h_{22e}},$$

$$r_e = \frac{h_{12e}}{h_{22e}},$$

$$r_k = \frac{1 + h_{21e}}{h_{22e}}.$$

1.6. Моделювання електронних систем на компонентному рівні

У загальному випадку об'єднання окремих електронних елементів в систему представляє собою висхідну гілку проектної процедури. У цьому зв'язку рівень складності синтезу структури системи визначається глибиною деталізації її компонентів. Дійсно, якщо в якості найпростіших елеме-

нтів системи вибрати транзистори, резистори і конденсатори, то число можливих варіантів їх об'єднання в систему виявляється незрівнянно більше аналогічних варіантів, що відповідають рівню операційний підсилювач, резистор і конденсатор. Збільшення числа можливих варіантів вирішення конкретного завдання може підвищити кількість перспективних за сукупністю критеріїв якості схемних конфігурацій. У той же час очевидна функціональна повнота в силу надзвичайно великої кількості структурних та інших перетворень, пов'язаних з процесом математичних перетворень, помітно підвищує ступінь ризику.

Розумним компромісом в становищі, що склалося є стратегія декомпозиції загальної задачі синтезу функціональних блоків, виділення набору базисних структур, принцип побудови яких базується на досягненнях базових технологічних обмежень.

Найпростішими базисними структурами є одиночні каскади, диференціальні підсилювачі і, в першу чергу, операційні підсилювачі, а також резистивні і ємнісні двополюсники. При побудові пристроїв, що можуть пере налаштовуватися в процесі роботи, до складу таких структур повинні входити керуючі елементи, що забезпечують цілеспрямовану зміну їх параметрів. Добре відпрацьовані технологічні процеси в мікроелектроніці дозволяють створювати як пасивні, так і активні керуючі багатополюсники, що реалізують принцип змінної крутизни.

1.7. Еквівалентні схеми елементів електронних систем

Однією із основних задач електроніки є розрахунок електричних схем, тобто отримання докладної інформації про процеси, що відбуваються у цій схемі. Однак розрахувати довільну схему, що складається з реальних елементів майже неможливо.

Поняття еквівалентна схема допомагає поєднати реальні елементи та їх ідеальні моделі. Еквівалентна схема представляє коло, що складається тільки з ідеальних компонентів, а також може відображати паразитні параметри реальної схеми: паразитні ємності, паразитні індуктивності і т.д.

Для складання еквівалентних схем використовують ідеальні елементи:

– Резистор. Ідеальний резистор характеризується тільки своїм електричним імпедансом. Паразитна індуктивність, паразитна ємність, а також опір виводів рівні нулю.

– Конденсатор. Ідеальний конденсатор характеризується тільки своєю електричною ємністю. Паразитна індуктивність, втрати, а також опір виводів рівню нулю.

– Котушка індуктивності. Ідеальна котушка індуктивності характеризується тільки своєю електричною індуктивністю. Паразитна ємність, втрати, а також опір виводів рівню нулю.

– Генератор напруги. Ідеальний генератор напруги характеризується тільки своєю електричною напругою. Внутрішній опір і опір виводів рівні нулю.

– Генератор струму. Ідеальний генератор струму характеризується тільки своїм електричним струмом. Внутрішній опір рівний нескінченності, а опір виводів та інші втрати рівні нулю.

Додатково приймається, що всі елементи еквівалентної схеми з'єднані між собою з допомогою ідеальних провідників. Тобто паразитна ємність, індуктивність і опір провідників рівні нулю.

Статична характеристика діода моделюється джерелом I_D, струм якого змінюється по закону:

$$I_D = I_0(e^{U_D/nVt} - 1).$$

Тут Vt – температурний потенціал p-n-переходу. Додатковий параметр n називається коефіцієнтом емісії. Для більшості діодів n=1. Нелінійна модель діода, яка використовується в SPICE, показана на рис. 1.11, а.

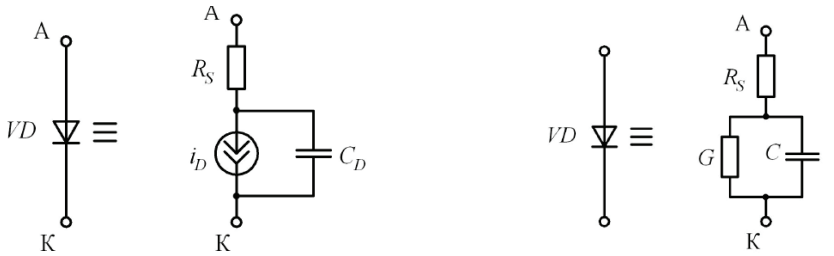


Рис. 1.11.

Динамічні характеристики діода враховують нелінійний ємнісний елемент C_D. Ємність C_D дорівнює сумі дифузійної ємності C_d і ємності переходу C_j:

$$C_D = C_d + C_j.$$

Резистор R_S враховує об'ємний опір області бази і емітера. Для малопотужних діодів його величина становить кілька ом.

При розрахунку частотних характеристик використовується лінеаризована схема заміщення діода, показана на рис. 1.11, б. В лінеаризованій схемі заміщення G – диференціальна провідність ВАХ діода в околиці робочої точки, C – диференціальна ємність.

Розглянемо еквівалентні схеми біполярного транзистора. Найбільш відомою моделлю біполярного транзистора є модель Еберса-Мола. Найпростіший варіант цієї моделі, названий інжекційною, показаний на рис. 1.12, а.

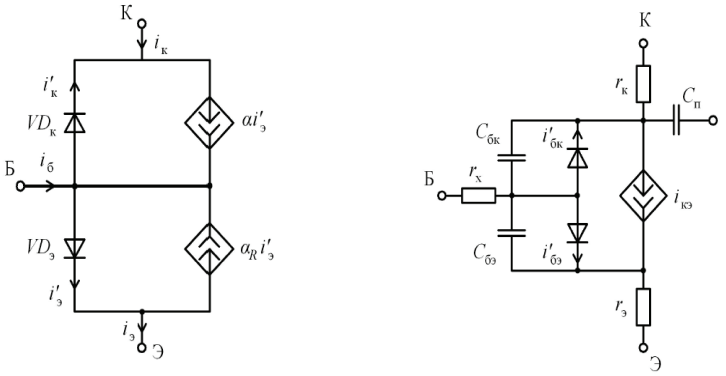


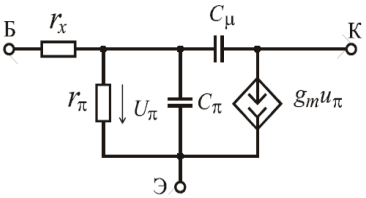
Рис. 1.12.

На рис. 1.12, а, α – коефіцієнт передачі струму емітера в активному режимі, α_R – коефіцієнт передачі колекторного струму в інверсному режимі. Модель Еберса-Мола дозволяє аналізувати біполярний транзистор в будь-якому з чотирьох режимів: активному, насичення, інверсному і відсічення.

Схема заміщення, показана на рис. 1.12, а, характеризує лише активну область транзистора. Модель, показана на рис. 1.12, б, доповнена резисторами, які враховують опір пасивних областей бази і колектора. Компоненти $C_{бк}$ та $C_{бэ}$ враховують ємності колекторного і емітерного переходів. Модель Еберса-Мола не враховує деякі ефекти, що спостерігаються в реальних приладах. Один з таких ефектів – залежність коефіцієнтів підсилення струму β_R і β_F від величини струму колектора. Такі ефекти враховує більш точна (хоча і більш складна) модель Гумеля-Пуна. Вибір моделі біполярного транзистора здійснюється в SPICE автоматично. Модель Гумеля-Пуна спрощується до моделі Еберса-Мола, якщо явно не заданий ряд параметрів.

Лінійна схема заміщення біполярного транзистора, яка використовується для аналізу в режимі малого сигналу, показана на рис. 1.13.

Рис. 1.13.



Резистор r_x враховує опір базового шару. Величина цього опору залежить від типу транзистора і положення робочої точки і може змінюватися від одиниць до декількох десятків Ом. Конденсатор $C_μ$ враховує ємність зміщеного в зворотному напрямку колекторного переходу. Конденсатор $C_μ$ враховує ємність, пов'язану з накопиченням неосновних носіїв в базі, і

ємність, зумовлену просторовим зарядом в області емітерного переходу. У більшості випадків C_μ становить від декількох пікофарад до декількох десятків пікофарад. Ємність C_μ не перевищує декількох пікофарад.

1.8. SPICE-моделі компонентів схем

Електронні компоненти в програмах комп'ютерного моделювання представляються у вигляді схем заміщення або моделей. Достовірність результатів моделювання залежить від того, наскільки точно модель враховує характеристики реального електронного приладу. Зрозуміло, не можна за допомогою програми комп'ютерного моделювання дослідити результат дії будь-якого ефекту, властивого електронного приладу, якщо цей ефект не враховується в його моделі. Моделі елементів, які є у сучасних програмах схемотехнічного моделювання, спочатку розроблялися для програми SPICE, тому їх часто називають SPICE-моделями. Створення таких моделей було тривалим процесом, в якому брало участь багато висококваліфікованих фахівців. SPICE має вбудовані моделі для більшості електронних компонентів: діодів, біполярних транзисторів, польових транзисторів з керуючим *p*-*n*-переходом, МОП-транзисторів, ліній передачі з розподіленими параметрами, пов'язаних індуктивних котушок і т.д. Аналогові інтегральні схеми, такі як операційні підсилювачі або компаратори, представляються підсхемами, які називають макромоделями.

Принципи моделювання схеми мовою SPICE:

1. Побудова моделі – файлу, що містить символічні імена компонентів схеми, інформацію про їх зв'язки. На рис. 1.14 представлений приклад моделі схеми в програмі LTSpice.

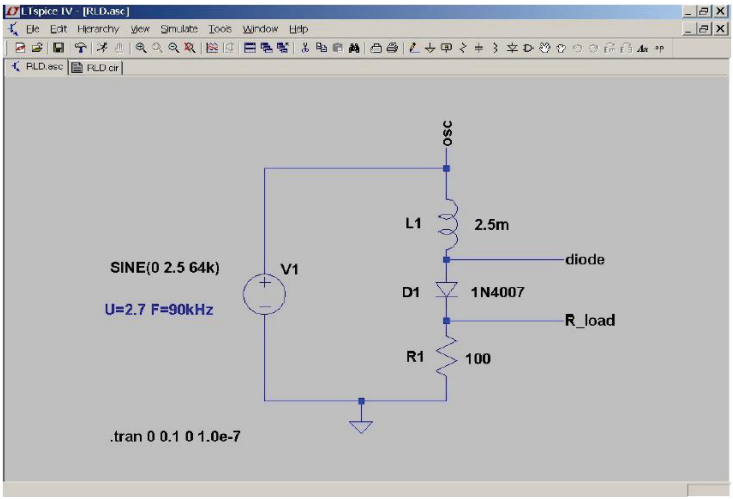


Рис. 1.14.

2. Трансляція файлу моделі в систему рівнянь. На рис. 1.15 представлено вікно зі SPICE-описом схеми.

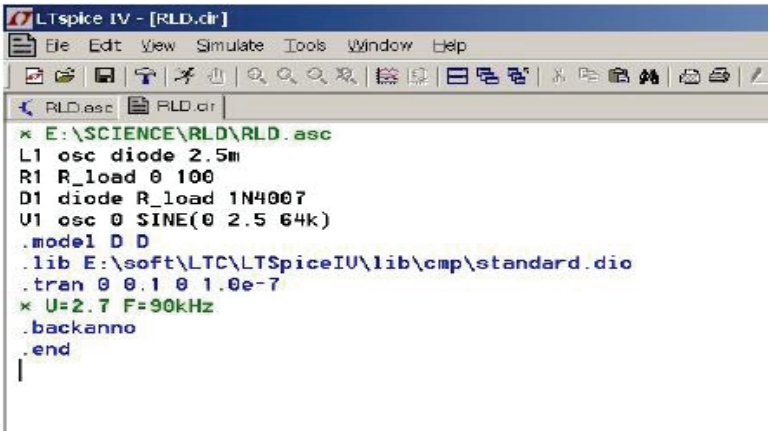


Рис. 1.15.

SPICE-опис елементів електронних схем починається з ключового слова .model. Наприклад, модель напівпровідникового діода може бути описана наступним чином:

.model 1N4004 D(IS=3.699E-09 RS=1.756E-02 +N=1.774 XTI=3.0 EG=1.110 CJO=1.732E-11 +M=0.3353 VJ=0.3905 FC=0.5 BV=400+IBV=1.0E-03)

Для опису частини схеми використовується ключове слово .subckt. У наступному описі показаний приклад використання підсхеми (макромоделі), а також моделей компонентів.

```
.subckt 4N27 1 2 3 4 5
R1 N003 2 2
D1 1 N003 LD
G1 3 5 N003 2 .582m
C1 1 2 18p
Q1 3 5 4 [4] NP
.model LD D(Is=1e-20 Cjo=18p)
.model NP NPN(Bf=610 Vaf=140
Ikf=15m Rc=1 Cjc=19p Cje=7p Cjs=7p
C2=1e-15)
.ends 4N27
```

Проведення перевірки SPICE моделей є необхідною і важливою частиною проектування схем. Від точності введених параметрів залежить правильність і коректність роботи проектованої схеми.

3. Моделювання (чисельне рішення модельних рівнянь) і виведення результатів. На рис. 1.16 представлені результати моделювання схеми.

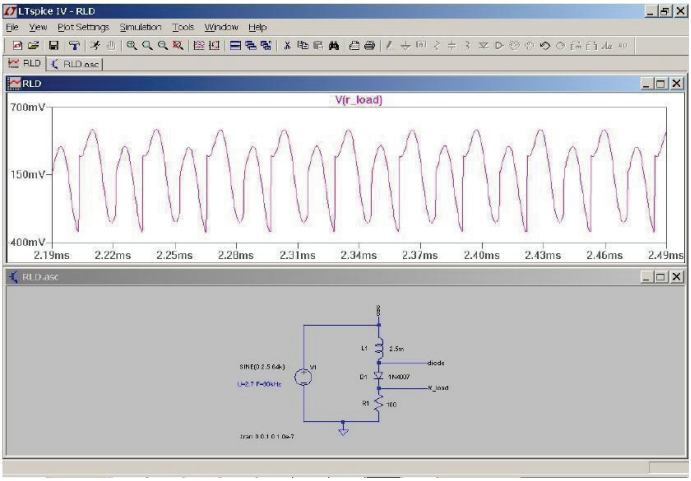


Рис. 1.16.

Бібліотека моделей електронних компонентів постійно розширюється і вдосконалюється. Фірми-виробники приділяють велику увагу розробці нових моделей електронних пристроїв. Сучасні програми схемотехнічного моделювання мають вбудовані редактори моделей, що дозволяють коригувати параметри моделей або створювати нові. Однак самостійне створення моделі компонента представляє дуже непросту задачу. Для цього потрібні висока кваліфікація і детальне знання принципів роботи пристрою. Паспортні дані пристроїв не завжди містять достатню інформацію для визначення всіх параметрів моделі.

Для отримання SPICE-моделей нових компонентів краще звернутися на Web-сайти фірм-розробників електронних компонентів.

1.9. Математичні моделі дискретних пристроїв

Аналіз дискретних пристроїв на функціонально-логічному рівні потрібен насамперед при проектуванні пристроїв обчислювальної техніки та цифрової автоматики. Проектування систем на функціонально-логічному і системному рівнях засноване на застосуванні дискретних математичних моделей. Тут додатково к припущенням, прийнятим при аналізі аналогових пристроїв, використовують дискретизацію сигналів, причому базовим є двозначне подання сигналів. Зручно цими двома можливими значеннями сигналів вважати «істину» (лог. 1) і «лож» (лог. 0), а самі сигнали розглядати як булеві величини. Тоді для моделювання можна використовувати апарат математичної логіки.

Елементами цифрових пристроїв на функціонально-логічному рівні служать елементи, що виконують логічні функції і функції зберігання інформації (рис. 1.17). Найпростішими елементами є диз'юнктор, кон'юнктор, інвертор, що реалізують відповідно операції диз'юнкції (АБО)

$$y = a \text{ OR } b,$$

кон'юнкції (І)

$$y = a \text{ AND } b,$$

інверсію (НІ)

$$y = \text{NOT } a,$$

де y – вихідний сигнал, a і b – вхідні сигнали.

Рис. 1.17.

Кількість входів може бути і більше двох. Математичні моделі пристроїв представляють собою систему математичних моделей елементів, що входять в пристрій, при ототожненні сигналів, які відповідають одному і тому ж з'єднанню елементів. Розрізняють синхронні і асинхронні моделі.

Синхронна модель являє собою систему логічних рівнянь. В ній відсутня така змінна як час. Синхронні моделі використовують для аналізу усталених станів. Прикладом синхронної моделі може служити наступна система рівнянь:

$$B = \text{NOT (R AND C)},$$
$$Q = \text{NOT (B OR P)},$$
$$P = \text{NOT (A AND Q)},$$
$$A = \text{(S OR C)}.$$

Асинхронні моделі відображають не тільки логічні функції, але і часові затримки в поширенні сигналів. Асинхронна модель логічного елемента має вигляд:

$$y\,(t+t_з) = f\,(X(t)),$$

де $t_з$ – затримка сигналу в елементі; f – логічна функція.

Запис означає, що вихідний сигнал y приймає значення логічної функції, відповідне значенням аргументів $X(t)$, в момент часу $t+t_з$. Отже, асинхронні моделі можна використовувати для аналізу динамічних процесів в логічних схемах.

Терміни синхронна і асинхронна моделі можна пояснити орієнтованістю цих моделей на синхронні і асинхронні схеми відповідно. У синхронних схемах передача сигналів між цифровими блоками відбувається тільки при подачі на спеціальний вхід тактових (синхронізуючих) імпульсів. Частота тактових імпульсів вибирається такою, щоб до моменту приходу синхроім-

пульсу перехідні процеси від попередніх передач сигналів фактично закінчилися. Отже, в синхронних схемах розрахунок затримок не актуальний, а швидкодія пристрою визначається заданням тактової частоти.

Асинхронні моделі можна використовувати не тільки для виявлення принципових помилок в схемній реалізації заданих функцій. З їх допомогою можна виявляти місця в схемах, небезпечні з точки зору виникнення в них перешкод і збоїв. Ситуації, пов'язані з потенційною небезпекою виникнення перешкод і збоїв, називають ризиками збою.

Розрізняють статичний та динамічний ризики збою. Статичний ризик збою ілюструє ситуація на рис. 1.18, якщо на два входи елемента І можуть приходити перепади сигналів в протилежних напрямках, як це показано на рис. 1.18, б. Якщо замість ідеального випадку, коли обидва перепади приходять одночасно, перепади внаслідок розкиду затримок прийдуть неодночасно, причому так, як показано на рис. 1.18, б, то на виході елемента з'являється імпульс перешкоди, який може спотворити роботу всього пристрою. Для усунення таких ризиків збою потрібно вміти їх виявляти. З цією метою застосовують тризначне синхронне моделювання.

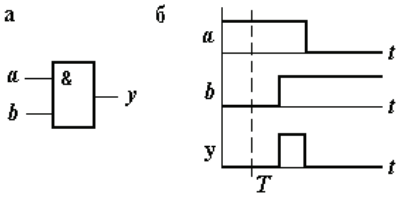


Рис. 1.18.

При цьому трьома можливими значеннями сигналів є 0, 1, і *X*, причому значення *X* інтерпретується як невизначеність.

Динамічний ризик збою ілюструють схема і часові діаграми на рис. 1.19. Збій виражається в появі замість одного перепаду на виході, що має місце при правильному функціонуванні, декількох перепадів.

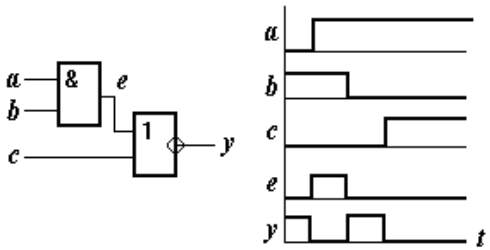


Рис. 1.19.

При моделюванні в підсистемах функціонально-логічного проектування приймаються ті самі допущення, що і при моделюванні аналогових

систем на верхніх рівнях. Крім того, модельований об'єкт представляється сукупністю взаємопов'язаних логічних елементів, стани яких характеризуються змінними, приймаючими значення в кінцевій множині. Безперервний час t замінюється дискретною послідовністю моментів часу $t_к$, при цьому тривалість такту $\Delta t = t_{k+1} - t_k$. Отже, математичною моделлю об'єкта є кінцевий автомат.

Теорія автоматів найбільш тісно пов'язана з теорією алгоритмів. Це пояснюється тим, що автомат перетворює дискретну інформацію по кроках в дискретні моменти часу і формує результуючу інформацію по крокам заданого алгоритму. Ці перетворення можливі з використанням як апаратних, так і програмних засобів (**SWITCH-технологія**).

Автомат можна представити як чорний ящик, на який подаються вхідні сигнали, знімаються вихідні і який може мати визначені внутрішні стани.

Математичною моделлю цифрового автомата є абстрактний автомат, в якому враховуються його вхідні та вихідні сигнали, а також внутрішні стани. Будемо розглядати детерміновані автомати, які завжди мають початковий стан Q_0, з якого починають працювати при дії вхідних сигналів і при повторі перебору вхідних сигналів повторюють послідовність станів і вихідних сигналів.

Абстрактний автомат задається множиною внутрішніх станів $Q = \{ Q_1, Q_2, \ldots, Q_m \}$ (алфавітом станів), множиною вхідних сигналів $X = \{ X_1, X_2, \ldots, X_p \}$ (вхідним алфавітом), множиною вихідних сигналів $Y = \{ Y_1, Y_2, \ldots, Y_k \}$ (вихідним алфавітом) і початковим станом Q_0. Перехід з одного стану в інший визначається функцією переходів f_p, що визначає стан автомата Q_s, в який він переходить з попереднього стану Q_m при дії сигналу X_p:

$$Q_s = f_p (Q_m, X_p).$$

Значення виходів автомата задається функцією виходів λ, що залежить від стану автомата Q_m і вхідного сигналу X_p:

$$Y_k = \lambda(Q_m, X_p).$$

Абстрактний автомат працює в дискретному часі, який задається цілими позитивними числами $t = 0, 1, 2, \ldots$

У кожний момент дискретного часу t, який зветься тактом, автомат перебуває у деякому стані $Q(t)$ з множини станів автомата Q.

У початковий момент часу ($t = 0$) він завжди знаходиться в стані $Q(0) = Q_0$. Вважається, що реакція автомата на вхідні сигнали не залежить від інтервалів часу між тактовими моментами часу.

У момент t, знаходячись у стані $Q(t)$, автомат сприймає на своєму вході сигнали, які називаються буквами (літерами) вхідного алфавіту $X_p \in X$. У відповідності до функції переходів f_p, він перейде у стан $Q_s(t)$, що описується алфавітом станів, тобто $Q_s(t) \in Q$.

Аналогічно, у відповідності з функцією виходів λ, на виході отримується сигнал Y_k, де $Y_k \in Y$.

У практиці використання цифрових автоматів можна виділити невелику кількість типових алгоритмів їх функціонування. Найбільшого розповсюдження набули два типи автоматів – автомати Мілі і Мура.

Закон функціонування автомата Мілі задається рівняннями:

$$\begin{cases} Q(t+1) = f_p\left[Q(t), X(t)\right]; \\ Y_k(t) = \lambda\left[Q(t), X(t)\right]; \qquad t = 0,\ 1,\ 2,\ \ldots \end{cases} \tag{1.5}$$

Закон функціонування автомата Мура описується рівняннями:

$$\begin{cases} Q(t+1) = f_p\left[Q(t),\ X(t)\right]; \\ Y_k(t) = \lambda\left[Q(t)\right]; \qquad t = 0,\ 1,\ 2,\ \ldots \end{cases} \tag{1.6}$$

Різниця між автоматами Мілі і Мура полягає лише в тому, що вихідний сигнал останнього залежить тільки від внутрішнього стану, у той час як в автоматі Мілі значення вихідного сигналу залежить також і від вхідного сигналу.

Структура автомата Мілі зображена на рис. 1.20.

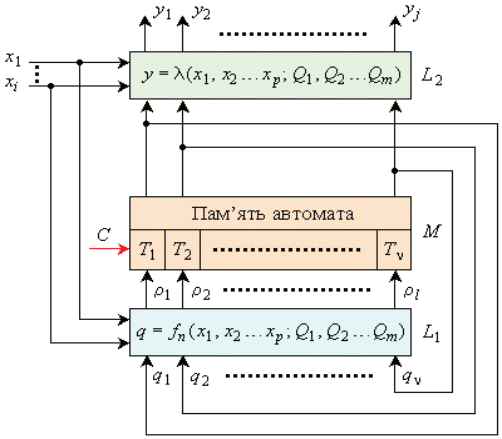


Рис. 1.20.

Автомат складається з набору елементарних автоматів (тригерів T_1, T_2, ..., T_n), стани яких q_1, q_2, …, q_n в кожний момент часу визначають внутрішній стан автомата в цілому. Під дією вхідних сигналів $x_1 \ldots x_i$, які подаються в дискретні моменти часу, відбувається формування сигналів $\rho_1 \ldots \rho_l$, що забезпечують послідовне перемикання тригерів $T_1 \ldots T_n$. Таким шляхом відбувається послідовна зміна станів автомата. Оскільки комбінаційна схема

L_1 формує керуючі сигнали тригерів в залежності від значень виходів тригерів у момент подачі тактового сигналу (для синхронних схем) або в моменти подачі вхідних сигналів (для асинхронних), то значення виходів тригерів у послідуючий момент часу і, відповідно, стан автомата однозначно визначається вхідними сигналами і станом автомата в попередній момент часу.

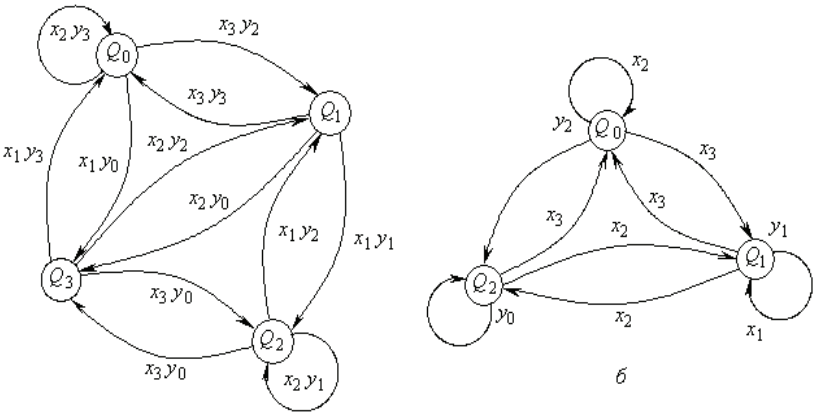
Сигнали керування тригерами $\rho_1 \ldots \rho_l$ формуються комбінаційним пристроєм L_1, структура якого визначає функцію переходів автомата. Функція виходів реалізується комбінаційним пристроєм L_2, який формує сигнали $y_1 \ldots y_j$ як функції виходів пам'яті (для автоматів Мура) або як функції виходів пам'яті і вхідних сигналів (для автоматів Мілі).

У практиці аналізу і синтезу цифрових автоматів використовують різні способи опису їх роботи. Найбільш поширеними є табличний (рис. 1.21, а) і графічний способи (рис. 1.21, б).

Табл. 5.2

Q / X	Q_0	Q_1	Q_2	Q_3
X_1	Q_3	Q_2	Q_1	Q_0
X_2	Q_0	Q_3	Q_2	Q_1
X_3	Q_1	Q_0	Q_3	Q_2

а)



б)

Рис. 1.21.

В кожній клітці таблиці переходів записується стан, в який переходить автомат з попереднього стану (стану, що стоїть у заголовку стовпця) при дії відповідного вхідного сигналу. Так, наприклад, якщо автомат знаходиться у стані Q_0, то при дії сигналу X_1 він перейде в стан Q_3; при дії сигналу X_2 залишиться в стані Q_0, а при дії сигналу X_3 перейде в стан Q_1.

Граф-схеми широко використовуються як при аналізі, так і при синтезі автоматів, а також при переході від словесного до формалізованого їх опису.

За допомогою таблиць переходів і виходів, як і за допомогою граф-схеми завжди можна знайти вихідну реакцію автомата на будь-яке вхідне слово, що належить множині X.

Однією з форм зображення автомата є його дерево переходів і виходів. Дерево може мати декілька ярусів, у кожному з яких за допомогою гілок показуються можливі переходи, починаючи з нульового стану. На рис. 1.22 приводиться приклад дерева переходів, що відповідає таблиці на рис. 1.21, а.

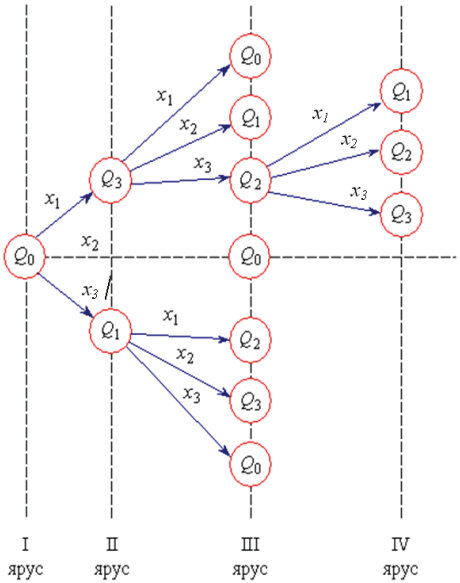


Рис. 1.22.

Скінченні автомати можуть бути синхронними, зміна станів яких відбувається в тактові моменти часу, що задаються зовнішнім генератором. В асинхронних автоматах зміна станів відбувається внаслідок дії вхідних сигналів без затримок. Тому асинхронні автомати вважаються більш швидко-

діючими і знаходять використання у швидкодіючих інформаційних пристроях вимірювання і керування різноманітними процесами, де необхідна миттєва реакція на зміну вхідних сигналів.

Як мови алгоритмізації та програмування в системах логічного керування залежно від типів керуючих обчислювальних пристроїв застосовуються: алгоритмічні мови високого рівня (C, Pascal), алгоритмічні мови низького рівня (асемблер), і спеціалізовані мови (наприклад, на базі ступінчастих і функціональних схем).

Існує два підходи алгоритмізації електронних систем цього класу. У першому з них вважається, що відомий алгоритм функціонування об'єкта керування і потрібно по ньому синтезувати алгоритм логічного керування, що забезпечує задане поводження. У другому підході враховується інформація про стан об'єкта керування. Таким чином, перша стратегія базується на понятті «стан», а друга – на понятті «подія».

Ні той, ні інший різновид керування в загальному випадку не є вичерпним, однак в SWITCH-технології пропонується зробити поняття «стан» первинним, а як мова специфікації для опису алгоритмів застосовувати графи переходів – пропонується представляти програму як систему взаємодіючих скінчених автоматів, що описуються графами переходів. Інакше кажучи, спочатку описується поводження керуючого пристрою, а не його структура, а побудова алгоритмів і програм починається з формування станів.

Об'єкти керування, як і система керування, можуть бути реалізовані за допомогою розглянутої технології. Розглянемо приклад.

Нехай автомат Мура заданий графом переходів на рис. 1.23.

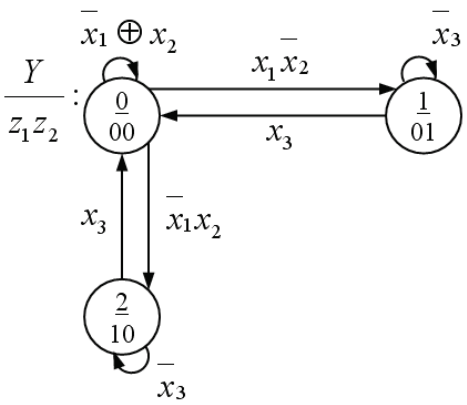


Рис. 1.23.

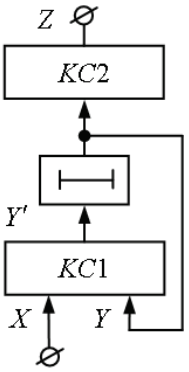


Рис. 1.24.

Для реалізації автомата можна застосувати модель автомата Мура першого роду (рис. 1.24).

Реалізуємо цю модель, застосовуючи дві конструкції **switch**, перша з яких відповідає комбінаційній схемі КС2, а друга – комбінаційній схемі КС1, охопленої зворотним зв'язком:

```
switch(Y)
{
case 0: z1=0; z2=0; break;
case 0: z1=0; z2=1; break;
case 0: z1=1; z2=0; break;
}
        switch(Y)
{
case 0: if ( x̄1 ⊕ x2) Y=0;
          if (x1 & x̄2 ) Y=1;
          if ( x̄1 & x2) Y=2;
break;
case 1: if (x3) Y=0;
                  if ( x̄3 ) Y=1;
break;
case 2: if (x3) Y=0;
          if ( x̄3 ) Y=2;
break;
}
```

Є можливість об'єднати обидві конструкції **switch** в одну конструкцію, утворивши узагальнену програмну модель автомата Мура першого роду:

```
switch (Y)
{
case 0: zl=0; z2=0;
if (xl & x̄2 ) Y=l;
if ( x̄1 & x2) Y=2;
break;
case 1: /*zl=0;*/ z2=l;
if (x3) Y=0;
break;
case 2: zl=l; /*z2=0;*/
if (x3) Y=0;
break;
}.
```

Ця модель повністю ізоморфна графу переходів, по якому вона побудована.

За допомогою кінцевих автоматів описуються вузли та елементи комп'ютерів, систем контролю, регулювання та керування, системи часової та просторової комутації в телекомунікаційних системах. Однак цей підхід не може бути використаний для опису процесів прийняття рішень і процесів в динамічних системах з наявністю стохастичних елементів.

На системному рівні проектування дискретних систем також поширені моделі систем масового обслуговування (СМО). Для таких моделей характерно те, що в них відображаються об'єкти двох типів – заявки на обслуговування та обслуговуючі апарати (ОА). При проектуванні обчислювальних систем заявками є розв'язувані завдання, а обслуговуючими апаратами – апаратне і програмне забезпечення обчислювальної системи. Заявка може перебувати в стані «обслуговування» або «очікування», а обслуговуючий апарат – в стані «вільний» чи «зайнятий». Стан СМО характеризується станами її ОА і заявок. Зміна станів називається подією. Моделі СМО використовуються для дослідження процесів, що відбуваються в цій системі при подачі на входи потоків заявок. Ці процеси представляються послідовностями подій. За результатами дослідження визначаються найбільш важливі вихідні параметри системи: продуктивність, пропускна здатність, ймовірність і середній час рішення задач, коефіцієнти завантаження обладнання.

Поява паралельних і конвеєрних систем, необхідність моделювати процеси функціонування не тільки апаратних, але і програмних засобів призвело до появи класу дискретних математичних моделей, що називаються мережами Петрі. Мережі Петрі можна використовувати для моделювання на функціонально-логічному і системному рівнях проектування широкого кола систем і мереж.

Мережі Петрі – ефективний інструмент розробки самих САПР. Ці мережі можуть служити моделями алгоритмів функціонування різних пристроїв дискретної автоматики.

1.10. Автоматні алгоритми обробки інформаційних потоків в локальних мережах

Одним з важливих завдань на сучасному промисловому підприємстві стає інтеграція між автоматизованими системами управління виробництвом та автоматизованими системами управління технологічними процесами. Розподілені системи управління, що працюють у реальному масштабі часу, потребують жорстких та точних гарантій якості обслуговування. Коректна побудова мережі та класифікація потоків з розбиттям на невелику кількість пріоритетних класів можуть виявитися достатніми для забезпечення нормальної роботи системи управління. У результаті, правильне виділення ресурсів на періоди пікового навантаження або аварійної ситуації, разом із захистом від потоків з нижчим пріоритетом, забезпечує необхідний рівень обслуговування. Оскільки механізми пріоритетного обслуговування набули широкого поширення зовсім недавно, необхідно мати апарат, що дозволяє моделювати роботу мереж з таким обладнанням. Взаємодія подій великих асинхронних системах має складну динамічну структуру. Ці взаємодії описуються простіше, якщо вказувати не безпосередні зв'язки між подіями, а ті ситуації, у яких ця подія може реалізуватися. Для цих завдань зручно використовувати мережі Петрі.

Прагнення оптимізації управління, скорочення часу реакції зміни зовнішнього середовища неминуче призводять до необхідності стандартизації різнорідних систем, що використовуються на підприємствах, їх відкритості і можливості простої взаємодії між ними. Необхідно враховувати дедалі частіше використання мереж передачі даних про стан виробничого процесу. Інформаційні потоки вимагають створення спеціалізованої інфраструктури, яка дозволяла б здійснювати оперативну обробку інформації, що надходить. При передачі пакетів через мережу необхідно контролювати такі параметри, як затримка, коливання затримки, втрати пакетів та інші. Багато виробничих процесів вимагають управління в реальному часі, і для керуючих потоків затримка передачі має бути мінімальною. Разом з тим, в мережі є потоки, допустимий час обслуговування яких може коливатися у широкому діапазоні. Тому під час вирішення завдання управління потоками можна розділити їх на пріоритети.

Пріоритизація має на увазі наявність засобів, за допомогою яких певні пакети отримують право на кращу обробку порівняно з іншими. Найчастіше у маршрутизаторах застосовуються такі алгоритми обробки черг:

– традиційний алгоритм FIFO;

– пріоритетне обслуговування;

– зважені черги, що настроюються.

В мережах Петрі події і умови представлені абстрактними символами з двох непересічних алфавітів, що називаються множиною переходів і множиною позицій. Переходи зображуються бар'єрами, а позиції – кружками. Мережа Петрі [1] – це набір $N=<S,T,F,W,M_0>$, де S – непуста множина елементів мережі, що називаються позиціями; T – непуста множина елементів мережі, що називається переходами; $F \subseteq S \times T \cup T \times S$ – відношення інцидентності, W і M_0 – кратність дуг і початкова розмітка.

Мережа Петрі складається з графічної частини *GT*, що дає структурний опис системи, та описової частини *BT*, що відноситься до опису поведінки окремих підсистем та елементів. *GT* складається з позицій, які можуть бути маркованими, переходів та напрямлених дуг, за допомогою яких реалізується зв'язок між позиціями та переходами [1].

S, T і F – кінцеві множини, переходів і дуг. При цьому $F \subseteq (S \times T) \cup (T \times S)$. З включення $s \in S$, $t \in T$ і $(s,t) \in F$ випливає, що s з'єднано з t. Запис $mar(s) \in N$ вказує кількість міток, що знаходяться в позиції s. Позиція $s \in S$ називається маркованою, коли $mar(s) \geq 1$.

Мережа Петрі дозволяє моделювати складні паралельні процеси і часто використовується для моделювання мереж обробки інформації.

Моделювання всієї обчислювальної мережі мережами Петрі через їх можливу складність є дуже непростою. За допомогою мереж Петрі пропонується моделювати критичні елементи комутації та обробки потоків. В результаті такого підходу є можливість досліджувати роботу як окремих механізмів у мережі, і всієї мережі загалом.

Розподіл на класи виконується виходячи з очікуваної інтенсивності потоків, вимог до якості обслуговування потоків. Ухвалення рішення про розміщення найпростішого потоку в конкретний клас обслуговування проводиться на підставі близькості нечітких оцінок їх обсягів, фізичного розташування приймачів та генераторів пакетів.

Для оцінки якості обслуговування потоку за методами нечіткої логіки можна виділити такі лінгвістичні змінні:

– чутливість трафіка до затримок *Fz*;

– передбачуваність швидкості *W*;

– чутливість до втрат пакетів *Y*.

Крім цього, для оцінки потрібно визначити такі лінгвістичні змінні:

– швидкість передавання *V*;

– розподіл інтенсивності потоку у часі *p*.

Результатом нечіткої продукції правил стає віднесення аналізованого потоку до конкретного класу або класів обслуговування з нечітким значенням приналежності до класів.

Надання ресурсів мережі з урахуванням ступеня важливості пакетів є однією з найважливіших функцій диференційованих послуг. Класифікація пакетів забезпечує найважливішим програмам пріоритетне обслуговування їх потоків.

На рис. 1.25 представлено модель розділу потоків при класифікації. *S*1 – позиція класифікації потоку; *S*2, *S*3, *S*4, *S*5 – позиції класифікованих потоків; *t*1, *t*2, *t*3, *t*4 – переходи класифікації потоку.

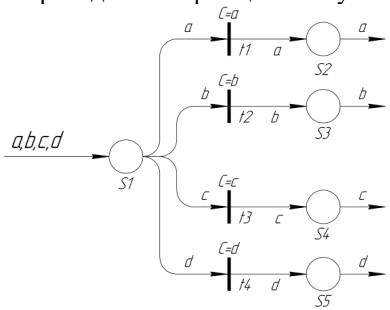


Рис. 1.25.

Опис роботи елемента має вигляд:

$K(S1)=1$; $K(S2)=1$; $K(S3)=1$; $K(S4)=1$; $K(S5)=1$;

$G(t1):M(S2).C=a \in N$;

$G(t2):M(S3).C=b \in N$;

$G(t3):M(S4).C=c \in N$;

$G(t4):M(S5).C=d \in N$;

де $\{a, b, c, d\}$ – множина класів обслуговування.

Механізм пріоритетного обслуговування черг передбачає наявність кількох вхідних підчерг: підчерги з високим, середнім, звичайним та низьким пріоритетом обслуговування. Пакети, що належать черзі з високим пріоритетом обслуговування, передаються першими. Коли високопріоритетна черга виявиться порожньою, почнеться передача пакетів з наступної за пріоритетом обслуговування черги. Передача пакетів черги із середнім пріоритетом обслуговування не почнеться доти, доки не будуть оброблені всі пакети високопріоритетної черги.

Якщо під час перевантаження мережі передачі потоків необхідним є вирішення критично важливих завдань, тоді буде виділена вся доступна смуга пропускання каналу. Механізм пріоритетного обслуговування черг у будь-якому разі забезпечує передачу високопріоритетної інформації, навіть якщо для цього доведеться пожертвувати керуючими потоками мережі [2]. На рис. 1.26 представлено модель пріоритетного обслуговування.

*S*1, *S*2, *S*3, *S*4 – позиції класифікованих потоків

*S*5 – позиція загального каналу

*t*1, *t*2, *t*3, *t*4 – переходи пріоритетного обслуговування

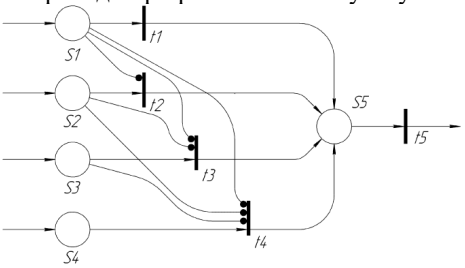


Рис. 1.26.

Опис роботи елементу має вигляд:

$K(S1)=1$; $K(S2)=1$; $K(S3)=1$; $K(S4)=1$; $K(S5)=1$;

$G(t2)$:$mar(S1)=0$;

$G(t3)$:$(mar(S1)=0) \wedge (mar(S2)=0)$;

$G(t4)$:$(mar(S1)=0) \wedge (mar(S2)=0) \wedge (mar(S3)=0)$.

Алгоритм зважених черг, що настроюються, дозволяє надати всім класам потоків певний мінімум пропускної спроможності або гарантувати деякі вимоги до затримок.

Як і при пріоритетному обслуговуванні, потоки діляться на кілька класів, для кожного з яких ведеться окрема черга пакетів. Але з кожною чергою пов'язується не її пріоритет, а відсоток пропускної спроможності вихідного інтерфейсу, який гарантується даному класу потоків при перевантаженнях. На рис. 1.27 представлено модель зважених черг, що налаштовуються.

*S*1, *S*2, *S*3, *S*4 – вхідні позиції буферів для кожного класу обслуговування;

*S*5 – позиція вихідного буфера;

*t*1, *t*2, *t*3, *t*4 – переходи обробки буферів;

*t*5 – перехід вихідного буфера;

*t*6, *t*7, *t*8, *t*9 – переходи механізму обробки зважених черг.

У межах черги пакетів обробляються відповідно до механізму FIFO. FIFO є механізм обслуговування черг, відповідно до якого порядок постановки пакетів у чергу збігається з порядком їх вилучення з черги для обробки (передачі).

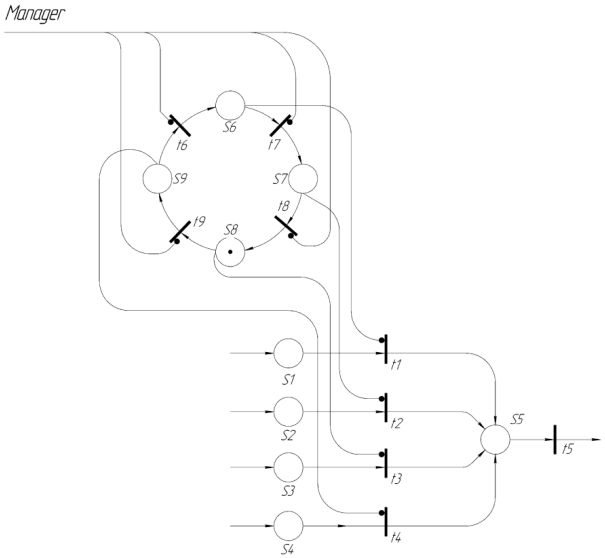


Рис. 1.27.

Механізм FIFO є найпоширенішим механізмом обслуговування черг, що застосовується у маршрутизаторах. Його реалізація відрізняється своєю простотою. Проте механізм FIFO немає засобів диференціювання потоків, відповідно, може виділити пріоритетні потоки. Крім того, механізм FIFO не здатний забезпечити рівномірне обслуговування потоків з однаковим пріоритетом та захистити їх від придушення потоками з нерівномірною інтенсивністю. Відповідно до механізму FIFO потоки обслуговуються майже пропорційно до їх інтенсивності. Наведена схема обслуговування черг не є рівномірною, оскільки вона допускає переважання потоків високої інтенсивності над іншими потоками.

Кінцева мета спеціальної теорії мереж Петрі – автоматичний аналіз властивостей мереж, їх автоматичний синтез та перетворення, на підставі чого можуть бути побудовані практичні алгоритми аналізу, синтезу та перетворень дискретних систем, що моделюються мережами.

Основна перевага мереж Петрі полягає у можливості відображення як однієї компоненти взаємодії кількох паралельно-послідовних процесів, а їх недолік у тому, що вони не описують у явному вигляді поведінку – динаміку зміни станів. Тим не менш, мережі Петрі є гнучким апаратом моделювання, який дозволяє повною мірою промоделювати комплекс механізмів обробки потоків у вузлах мережі, що застосовуються в сучасних пристроях комутації. Їх використання дозволяє будувати компактні та наочні моделі процесів.

1.11. Мови опису апаратури AHDL, VHDL та Verilog

При розробці цифрових пристроїв широке поширення знайшли програмовані логічні інтегральні схеми, які дуже вдало доповнили мікропроцесорні засоби. Це зручна елементна база, яка використовується в системах цифрової обробки сигналів, телекомунікації, для побудови різноманітних систем керування та ін. Існує можливість створювати на кристалі спеціалізовані процесорні ядра, що дозволяє вирішувати будь-які науково-практичні задачі. Однак схемна реалізація подібних систем – задача досить складна, а іноді і неможлива. Для вирішення подібних задач може бути використана мова опису апаратури (HDL – Hardware Description Language).

Мова опису апаратури – це спеціалізована формальна комп'ютерна мова, що використовується для проектування структури, дизайну та роботи електронної мікросхеми та її моделювання. Вона дає можливість автоматично аналізувати, імітувати та тестувати створюваний пристрій. Компілятор мусить забезпечувати переведення програми, написаної на будь-якій з мов опису апаратури на низькорівневу специфікацію фізичних електронних компонентів з ціллю створити мікросхему.

До переваг використання мови проектування апаратури над схемним проектуванням можна віднести наступні:

– можливість проектування пристроїв, для яких створення схем неможливе через надмірну складність продукту, наприклад при розробці сучасних процесорів;

– такий опис зазвичай об'єднує як структурну, так і функціональну складову;

– проект на мові проектування апаратури портативний та універсальний, тобто легко переноситься на іншу елементну базу і може використовуватися в іншому проекті. Як результат, він стає довгоживучим, оскільки не залежить від «старіння» конкретних мікросхем;

– мови опису апаратури гарантують високу надійність розробленого пристрою та забезпечують самодокументування.

Мови проектування апаратури дають можливість інженерам працювати на вищому рівні абстракції, що дозволяє збільшити кількість елементів розробленого пристрою.

Мова опису апаратури виглядає дуже схоже на мову програмування, наприклад С, оскільки її структура складається з таких самих текстових виразів. Також мови проектування апаратури дають можливість описувати специфікації для апаратного забезпечення, що можуть виконуватися. Це дає ілюзію наявності мови програмування, хоча насправді їх відносять до мов проектування чи моделювання.

Першою причиною цього є те, що, на відміну від мов програмування, що не завжди мають в собі ознаки паралелізму, інструкції мов проектування апаратури завжди виконуються паралельно. Іншою важливою їх складовою є опис синхросигналу, що є особливістю проектування апаратного забезпечення. Мови опису апаратури можуть використовуватися для створення пристрою у структурній, поведінковій формах чи на рівні регістрових передач з такою ж функціональністю.

В загальному випадку опис пристрою за допомогою однієї з HDL-мов представляє собою текст, що зберігається в одному або декількох файлах, які в сукупності складають уявлення розробника про проект. Тексти описів на більшості мов проектування цифрових пристроїв по складу синтаксичних конструкцій схожі із традиційними мовами програмування. Як правило, створюючи програму розробник абстрагується від конкретної фізичної реалізації, виділяючи насамперед алгоритм функціонування проектованого пристрою. На основі моделі може бути побудований фізичний пристрій. В сучасних умовах основну роботу з інтерпретації моделі у фізичний об'єкт виконує комп'ютер. Як буде продемонстровано, сучасні мови дозволяють будувати моделі з різним ступенем наближення до реалізації – від опису закону функціонування до детального представлення проекту на рівні вентилів.

Вхідна інформація в дискретних пристроях – цифрові сигнали, які повинні надходити в систему через контакти і далі рухатися по ланцюгах до блоків, що виконують ті або інші перетворення. Джерелу сигналу, який подається на деякий ланцюг, в HDL-мовах зіставляється оператор, який присвоює значення змінній. Ця змінна представляє сигнал на цій лінії зв'язку. Такі оператори називаються драйверами. Основним типом в мовах є дані сигнального типу, по властивостях близькі до звичайних логічних даних. Основною відмінністю даних у HDL-мовах від логічних даних мов програмування є набір припустимих значень. Крім того враховується той факт, що сигнал має часові характеристики, зокрема, зміна значення змінної може відбуватися не відразу після присвоєння нової величини. Дані можуть поєднуватися в масиви, вектори. Крім того, застосовується ряд службових типів даних. Найбільш важливими з них є дані арифметичного типу.

Модель, що відображає об'єкт проектування у формі правил перетворення вхідних даних у вихідні, називається поведінковою. Структурна модель описує проект у вигляді сукупності модулів, кожен з яких реалізує певну частину завдання, і набору зв'язків між ними. Структурна модель є загальним способом створення складних проектів, опис яких доцільно

виконати по ієрархічному принципу. В програмному модулі вищого рівня ієрархії присутні оголошення портів і внутрішніх зв'язків, а також декларації входжень модулів. Це ілюструє приклад асинхронного передавача. Як правило, першим етапом проектування є створення поведінкової моделі вищого рівня ієрархії. Часто може знадобитися кілька корекцій до одержання відповідності поведінкової моделі заданим вимогам. В процесі реалізації деякого проекту повна декомпозиція потрібна не завжди. На деякому етапі може виявитися, що сукупність операторів, яка відтворюється деяким модулем, присутня у бібліотеці, що поставляється із САПР. В такому випадку досить у відповідному структурному тілі посилатися на цей бібліотечний модуль. Загальна структура проекту зберігається і в цьому випадку. Опис модуля, який підключається, поведінковий або структурний, від розробника може бути приховано.

Однією з найважливіших проектних процедур є тестування. Проектування передбачає об'єднання найпростіших модулів в більш складну структуру. Тестування цих модулів дозволить виявити помилки в алгоритмах їх роботи на ранніх етапах реалізації проекту і у перспективі зменшити час на налагодження пристрою. Тестування розроблених модулів зручно виконувати за допомогою редактора часових діаграм.

Вибір стилю програмування багато в чому визначається досвідом розробника, але потрібно мати на увазі, що стиль може істотно впливати на апаратну реалізацію. Послідовний стиль опису відповідає традиційним підходам до написання комп'ютерних програм. Розташування і порядок виконання операторів в тексті опису відповідає порядку проходження даних через них. Паралельний стиль припускає асинхронне, одночасне виконання операторів в реалізованому пристрої. Блок, що реалізує деякий оператор, безпосередньо реагує на виконання дії будь-яким із його попередників. Для представлення паралельних процесів використовуються такі форми запису, які передбачають виконання кожного оператора після виконання хоча б однієї з ініціюючих подій. Порядок запису паралельних операторів байдужий. Припустимі зворотні зв'язки і петлі. Потоковий стиль відрізняється тим, що оператор виконується після готовності всіх попередників даного оператора. В чистому вигляді потоковий принцип досить складний в реалізації. Автоматний стиль спирається не на модель передачі даних, а на модель перемикання станів. Він заснований на визначенні деякої множини станів проектованого пристрою і правил переходу з одного стану в інший залежно від вхідних сигналів. Кожному стану або переходу відповідає певний набір дій. Такий підхід найбільш ефективний при описі пристроїв, що характеризуються циклічним виконанням однотипних послідовностей перетворень, зокрема керуючих пристроїв, процесорних модулів. Стиль створення проекту і використані засоби мови визначаються структурними особливостями проектованого пристрою. В реальних проектах, як правило, присутні структурні фрагменти різних типів, які описані різними способами (текстовий, графічний, у вигляді часових діаграм) і стилями програмування.

Відомі два протилежних підходи до побудови систем моделювання дискретних пристроїв – наскрізне моделювання і моделювання по подіям. При наскрізному моделюванні час ділиться на кванти, тривалість яких вибирається як найбільший час затримки компонента. Недоліки такого моделювання – нераціональні витрати машинного часу, викликані потребою в дискретизації часу і необхідність відтворення на кожному кроці поведінки всіх компонентів, у тому числі і тих, на яких не відбувається зміна сигналу. Інші системи моделювання використовують подійний підхід, або дискретну модель подій. На кожному кроці моделювання переобчислюються стани тільки тих компонентів, на вході яких у цей момент відбуваються зміни. Зміни логічних змінних називають подіями. Будь-яка подія може викликати ланцюжок інших подій. Важливою характеристикою методу моделювання цифрових пристроїв є кількість помітних станів сигналу. Кожному стану сигналу зіставляється індивідуальний символ. Сукупність символів становить алфавіт моделювання. Кожний стан специфічно сприймається приймачем сигналів, тому в системі моделювання визначається набір правил перетворення сигналів типовими цифровими елементами. Найпростіший алфавіт – двійковий, який складається з 0 та 1. Функціонування елементів описується за правилами алгебри логіки. Однак неможливо описати шинну логіку, яка має високоімпедансний стан Z. Досить розповсюджений алфавіт із чотирьох символів {'0', 'X', '1', 'Z'}. X позначає невизначений стан. Такий сигнал присвоюється виходу логічного елемента під час перехідного процесу. Невизначений стан приймає тригер після подачі синхронізуючого сигналу при забороненій комбінації сигналів на інформаційних входах.

Verilog HDL – це мова опису апаратури, яка використовується для опису цифрових систем, наприклад комп'ютерних модулів. Компілятори Verilog, поряд з VHDL-компіляторами, включені в більшість САПР інтегральних схем. Синтаксис мови Verilog нагадує синтаксис мови програмування C. У Verilog менша в порівнянні з VHDL кількість службових слів, що спрощує його вивчення. Verilog дозволяє ефективно виконувати опис і проводити моделювання та синтез цифрових схем завдяки вбудованим примітивам, засобам часового контролю, моделюванню затримок сигналів від входу до виходу. Крім того, через свої розширені можливості VHDL програє Verilog по ефективності, тобто для опису однієї і тієї ж конструкції на Verilog потрібно в кілька разів менше символів, чим в VHDL.

1.12. Методи моделювання цифрових та аналогових пристроїв за допомогою SPICE-опису

Пік досліджень з розробки алгоритмів комп'ютерного аналізу електронних схем припав на 70–80 роки. Підсумком цих досліджень є сучасні програми моделювання електронних кіл: PSpice, MicroCAP, Multisim, Circuit Maker та ін. Більшість з них є модифікацією програми схемотехнічного моделювання SPICE, розробленої в середині 70-х років. Ця програма де-факто стала еталонною програмою аналізу аналогових пристроїв. Моделі

електронних компонентів, розроблені для SPICE, використовуються в більшості інших програм.

Мова SPICE призначена для опису електричних ланцюгів різної складності, і вона використовується для розрахунку схем у часовій і частотній областях, а також в статичному режимі. Ці типи розрахунків використовуються в інженерній практиці найбільш часто. При проведенні моделювання всі елементи схеми замінюються їх математичними моделями. Таким чином, SPICE-моделі є повними.

Процедура моделювання електронних схем в програмі PSpice складається з трьох етапів:

1. Створення принципової схеми.

2. Моделювання.

3. Представлення результатів моделювання в зручній для користувача формі.

Базовий набір елементів PSpice включає резистори, конденсатори, індуктивні котушки, діоди, біполярні транзистори, польові транзистори з керуючим *p-n*-переходом та ізольованим затвором, довгі лінії, джерела напруги і струму різної форми. Аналогові інтегральні схеми, а також деякі компоненти (тиристори, деякі види польових транзисторів) представляються підсхемами, параметри яких задає користувач. Базовий набір елементів PSpice містить також цифрові функціональні блоки, які виконують логічні операції. Цифрові елементи можуть реалізовуватися користувачем на основі функціональних блоків.

Блок-схема, що ілюструє функціонування і зв'язок окремих модулів програми, показана на рис. 1.28.

Основні види аналізу, виконувані SPICE:

1. Аналіз нелінійних резистивних ланцюгів постійного струму;

2. Аналіз лінійних ланцюгів в частотній області;

3. Аналіз нелінійних динамічних ланцюгів при дії сигналів довільної форми.

Решта видів аналізу, такі як статистичний аналіз, розрахунок чутливості або режим варіації параметрів є похідними і базуються на трьох перерахованих режимах.

Аналіз нелінійних резистивних ланцюгів постійного струму виконується в наступних випадках:

1. При визначенні робочої точки нелінійного ланцюга.

2. При розрахунку частотних характеристик. В цьому режимі попередньо виконується розрахунок резистивної підсхеми, визначається робоча точка і будується лінійна схема заміщення.

3. При розрахунку перехідних процесів. В цьому режимі робоча точка визначається на кожному часовому кроці.

4. В режимі варіації параметрів. Розрахунок робочої точки виконується для кожного нового значення змінними параметрами.

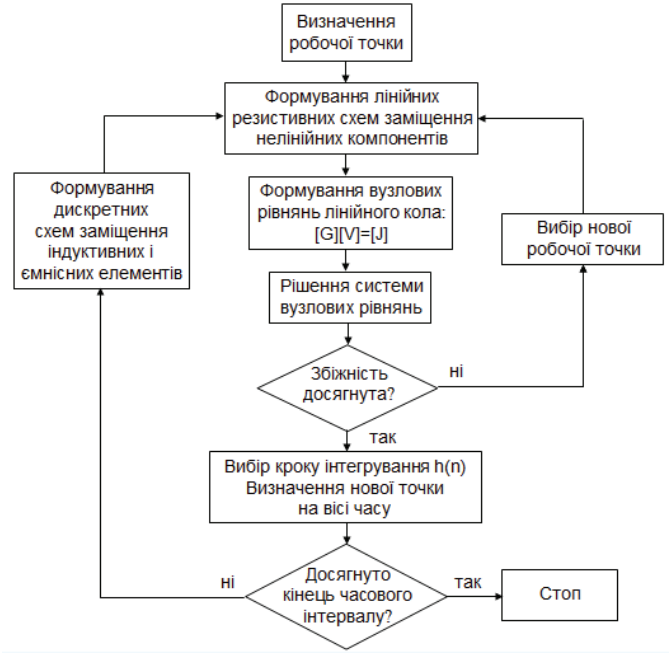


Рис. 1.28.

При аналізі резистивних ланцюгів індуктивні і ємнісні елементи виключаються. Ємнісний елемент замінюється розривом, а індуктивний елемент закорочується. Для аналізу лінійного резистивного кола використовується модифікований метод вузлових потенціалів. При аналізі нелінійних резистивних ланцюгів використовується ітеративна процедура, яка представляє собою схемотехнічну інтерпретацію методу Ньютона-Рафсона. На кожній ітерації формуються вузлові рівняння еквівалентного лінійного резистивного ланцюга. В результаті рішення цієї системи рівнянь визначається уточнене значення робочої точки.

Розрахунок частотних характеристик проводиться в два етапи. Спочатку автоматично визначається робоча точка нелінійного резистивного ланцюга. Для цього виконується аналіз резистивного ланцюга, в якій виключені індуктивні і ємнісні елементи. Потім нелінійні компоненти замінюються лінеаризованими моделями з параметрами, відповідними робочій точці. Після цього виконується розрахунок частотних характеристик.

Для розрахунку перехідних процесів використовуються неявні методи чисельного інтегрування – методи трапецій, Гіра другого порядку або неявний метод Ейлера. За замовчуванням використовується метод трапецій. Максимальний крок інтегрування вибирається користувачем або встановлюється автоматично. SPICE автоматично вибирає крок інтегрування, що забезпечує оптимальне співвідношення між точністю і часом розрахунку.

Крім розглянутих базових методів аналізу частотних і часових характеристик програма SPICE дає розробнику ще один потужний інструмент дослідження електронних схем – параметричний аналіз. В цьому режимі користувач має можливість проводити багатоваріантний аналіз частотних або часових характеристик. Результатом такого аналізу є сімейства кривих, що наочно показують, як впливає зміна того чи іншого параметра на характеристики схеми. На кожному кроці варіації параметрів можуть виконуватися різні види аналізу. Варіюватися можуть напруги і струми джерел, температура компонентів, параметри моделей, глобальні параметри.

Елементи реальних ланцюгів завжди мають випадкові відхилення від номінальних значень. Програма PSpice дозволяє моделювати характеристики електронних ланцюгів з урахуванням статистичного розкиду параметрів компонентів. В ході одного циклу статистичного аналізу ланцюг може моделюватися кілька сотень разів. При цьому кожен раз моделювання здійснюється з новим набором параметрів, що задаються випадковим чином.

Відхилення значень елементів від номінальних задаються за допомогою генератора випадкових чисел. Випадкове значення параметра x розраховується за формулою:

$$x = x_{\text{ном}}(1 + \vartheta\Delta),$$

де $x_{\text{ном}}$ – номінальне значення параметра, вказане користувачем; ϑ – відносний розкид параметра x; ∇ – центрована випадкова величина, що приймає значення на відрізку (–1, +1).

Статистичні випробування за методом Монте-Карло можуть проводитися при розрахунку режиму по постійному струму, аналізі перехідних процесів або розрахунку частотних характеристик. Проведення статистичного аналізу за методом Монте-Карло дозволяє визначити багато важливий характеристик електронних схем. Наприклад, за величиною відхилення частотних характеристик можна судити про чутливість ланцюга.

Мова SPICE може бути використана для моделювання цифрових вузлів з використанням тільки електричних сигналів. Це потребує використання повних SPICE-моделей мікросхем. Основна складність, яка виникає при повному моделюванні внутрішньої електричної схеми мікросхеми при автоматизованому проектуванні, пов'язана з розмірністю обчислювальної задачі і високим порядком систем рівнянь, що описують цифровий вузол. Вже для мікросхем середнього ступеня інтеграції обсяг обчислень стає невиправдано великим.

Одним з методів, що дозволяють істотно скоротити розмірність задачі, є використання макромоделей. Проте їх вдається побудувати тільки для дуже обмеженого класу цифрових елементів – тригерів і логічних елементів.

Ще однією трудністю, що виникає на шляху повного схемотехнічного моделювання цифрового вузла на основі SPICE-моделей складових його елементів, є відсутність у вільному доступі принципових електричних схем, тим більше з номіналами пасивних і характеристиками активних елементів.

Навпаки, більшість великих фірм намагається зберегти цю інформацію в таємниці (особливо для нових розробок). У технічній документації на мікросхеми вони лише іноді приводять структурну схему, яка не дає практично нічого для складання SPICE-моделей.

Тому в даний час виробники мікросхем практично припинили розповсюдження їх SPICE-моделей. Все більший розвиток отримує новий підхід до моделювання мікросхеми, заснований на IBIS-описі (I/O Buffers Information Specification, або інформаційна специфікація буферів вводу/ виводу).

При формуванні загальних принципів побудови IBIS-специфікації і виборі параметрів моделей вхідних і вихідних буферів була врахована необхідність можливості роботи з цим форматом САПР різного призначення, введена система пріоритету параметрів і передбачена різна глибина опису моделей. Практично всі документи, пов'язані з IBIS, знаходяться у відкритому доступі в мережі Інтернет.

Сутність IBIS полягає в тому, що в специфікації наводиться опис властивостей тільки вхідних і вихідних буферів цифрової інтегральної схеми, причому під буфером (термін широко використовується в стандарті) розуміється частина схеми, безпосередньо пов'язана з тим чи іншим сигнальним виводом. Поширення сигналів всередині мікросхеми не моделюється. Самі буфери представляються у вигляді схем заміщення, які показані на рис. 1.29 і рис. 1.30.

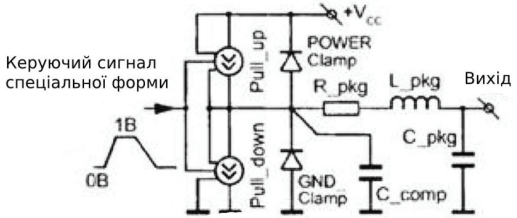


Рис. 1.29.

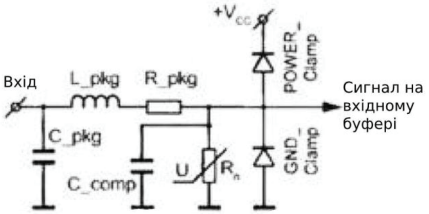


Рис. 1.30.

Для того щоб вхідні і вихідні буфери функціонували взаємопов'язано, необхідно вхідні сигнали привести до нормованого виду. Схема, за допомо-

гою якої обробляється вхідний сигнал і формується керуючий вихідний, називається внутрішньою логікою, або, згідно з прийнятою в IBIS термінологією, логікою рівня і активації.

Моделювання з використанням IBIS-моделей проводять на етапі проектування друкованих плат. При його виконанні всі мікросхеми замінюються сукупністю IBIS-моделей їх буферів, проводиться аналіз того, як поширюється сигнал в друкованому вузлі від мікросхеми до мікросхеми, і оцінюються спотворення сигналів при розповсюдженні в елементах друкованого монтажу. Зміна логічного стану вихідних буферів забезпечується безпосередньо програмою моделювання і не залежить від логічного стану на вході.

Однак IBIS-моделі можна використовувати і при моделюванні з використанням мови SPICE і систем на його основі. В цьому випадку внутрішню частину схеми замінюють такою схемою заміщення, щоб вона виконувала потрібні для користувача логічні функції, а вхідні і вихідні буфери – їх IBIS-моделями. Це дозволяє істотно розширити область застосування IBIS-моделей і отримати можливість моделювати ті властивості цифрового вузла, які цікавлять розробника. Моделювання можна провести кілька разів, щоб оцінити, як функціонує той чи інший цифровий вузол.

В даний час спостерігаються дві важливі тенденції в області IBIS. Перша з них – це масовий перехід від SPICE до IBIS-моделей мікросхем. Друга полягає в тому, що IBIS в своєму розвитку все більше зближується з VHDL та іншими подібними мовами. Файли з IBIS-описом мікросхем можуть бути доповнені VHDL-описом логіки її функціонування.

1.13. Марківські моделі електронних систем

Для формального опису складних систем часто використовують стохастичні математичні моделі, які враховують вплив випадкових факторів. Детерміновані моделі, які не враховують випадкові фактори, є окремим випадком більш загальних стохастичних моделей. При стохастичному моделюванні найчастіше використовуються ймовірнісні автомати, які дозволяють дослідити процеси в дискретному часі, та системи масового обслуговування, які дозволяють дослідити процеси в неперервному часі.

При проектуванні засобів обчислювальної техніки широке використання знаходять марківські моделі. Функціонування широкого класу електронних систем можна представити як процес переходу з одного стану в інший. Наприклад, процес функціонування комп'ютера характеризується тим, що в кожний момент часу обробкою інформації зайняті ті або інші блоки. Процес проходження інформації по блокам ЕОМ можна розглядати як процес переходу системи з одного стану в інший.

При дослідженні неперервних та дискретних випадкових ланцюгів зазвичай використовують графічне представлення функціонування системи.

Граф станів системи представляє собою сукупність вершин, що відображають можливі стани системи і сукупність гілок, що відображають можливі переходи системи з одного стану в інший.

Розглянемо приклад. Центральний процесор мультипрограмної комп'ютерної системи в будь-який момент часу виконує або пріоритетну програму, або фонову програму, або знаходиться в стані очікування. Тривалість знаходження системи в кожному стані кратна тривалості кроку Δt. Визначити коефіцієнт використання процесора, якщо задані вірогідності переходів з одного стану в інший.

Позначимо 1 – стан, в якому обслуговується пріоритетна програма; 2 – стан, в якому обслуговується фонова програма; 3 – стан простою.

Нехай матриця перехідних ймовірностей має вигляд:

$$|p| = \begin{vmatrix} 0,7 & 0,2 & 0,1 \\ 0,8 & 0,1 & 0,1 \\ 0,8 & 0,05 & 0,15 \end{vmatrix}.$$

На рис. 1.31 зображено граф функціонування системи з станами S_1, S_2, S_3 .

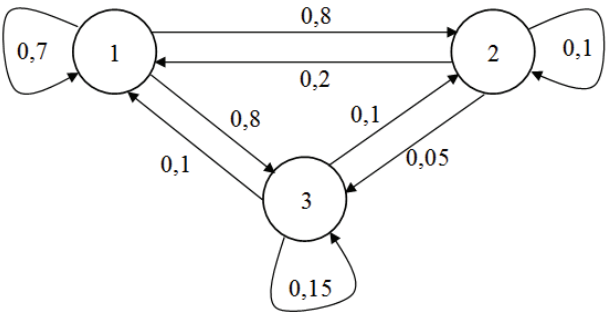


Рис. 1.31.

Для стаціонарного режиму можна скласти систему рівнянь:

$$\begin{cases} S_1 = 0,7S_1 + 0,8S_2 + 0,8S_3; \\ S_2 = 0,2S_1 + 0,1S_2 + 0,05S_3; \\ S_3 = 0,1S_1 + 0,1S_2 + 0,15S_3. \end{cases}$$

Умова нормування ймовірностей:

$$S_1 + S_2 + S_3 = 1.$$

В результаті рішення системи рівнянь отримуються ймовірності станів в усталеному режимі:

$$S_1 = 0,73;\ S_2 = 0,17;\ S_3 = 0,1.$$

Ймовірність простою процесора S_3=0,1.

Коефіцієнт використання процесора $K_B=1–S_3=0,9$, при цьому на обслуговування пріоритетної програми витрачається 73% часу, а на обслуговування фонової програми 17%.

Розглянемо приклад моделювання електронної системи з використанням безперервних Марківських ланцюгів.

Двохпроцесорна обчислювальна система призначена для обробки простішого потоку завдань, що надходять з інтенсивністю λ. Інтенсивність рішення завдань першим процесором дорівнює μ_1, другим – μ_2, причому $\mu_1>\mu_2$. Завдання приймається на обслуговування процесором, який має більшу продуктивність. Якщо обидва процесори зайняті, користувач отримує відмову. Визначити в усталеному режимі ймовірність відмови, коефіцієнти завантаження процесорів K_1 та K_2.

Розглянемо можливі стани системи, які визначаються станами процесорів:

00 – обидва процесори в стані очікування завдання;

10 – перший процесор зайнятий рішенням задач, другий вільний;

01 – другий процесор зайнятий рішенням задач, перший вільний;

11 – обидва процесори зайняті рішенням задач.

Граф функціонування системи з переходами λ і μ має вигляд, представлений на рис. 1.32.

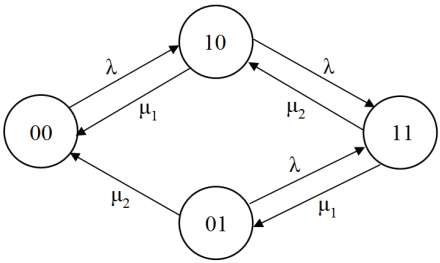


Рис. 1.32.

По графу можна записати систему лінійних диференційних рівнянь А.М. Колмогорова:

$$\begin{cases} \dfrac{dS_{00}(t)}{dt} = -\lambda S_{00}(t) + \mu_1 S_{10}(t) + \mu_2 S_{01}(t); \\ \dfrac{dS_{10}(t)}{dt} = \lambda S_{00}(t) - (\mu_1 + \lambda) S_{10}(t) + \mu_2 S_{11}(t); \\ \dfrac{dS_{01}(t)}{dt} = -(\lambda + \mu_2) S_{01}(t) + \mu_1 S_{11}(t); \\ \dfrac{dS_{11}(t)}{dt} = \lambda S_{10}(t) + \lambda S_{01}(t) - (\mu_1 + \mu_2) S_{11}(t); \\ S_{00}(t) + S_{10}(t) + S_{01}(t) + S_{11}(t) = 1. \end{cases}$$

Початкові умови задані вектором ймовірностей:

$$S_{00}(0)=1;\ S_{10}(0)=0;\ S_{01}(0)=0;\ S_{11}(0)=0.$$

Рішення цієї системи при заданих початкових умовах дозволяє визначити ймовірності станів як функції часу, які, в свою чергу, дозволяють визначити потрібні характеристики обчислювальної системи.

Оскільки марківський процес, що описує роботу обчислювальної системи, є ергодичним, існує стаціонарний режим, при якому вірогідності станів прагнуть до постійних величин. При цьому система диференційних рівнянь Колмогорова перетворюється в систему лінійних рівнянь:

$$\begin{cases} -\lambda S_{00}+\mu_1 S_{10}+\mu_2 S_{01}=0; \\ \lambda S_{00}-(\mu_1+\lambda)S_{10}+\mu_2 S_{11}=0; \\ -(\lambda+\mu_2)S_{01}+\mu_1 S_{11}=0; \\ S_{00}+S_{10}+S_{01}+S_{11}=1. \end{cases}$$

Останнє рівняння замінено умовою нормування ймовірностей.

Ймовірність відмови співпадає з ймовірністю стану S_{11}, в якому обидва процесори зайняті. Коефіцієнти завантаження K_1 та K_2 процесорів представляють собою ймовірності знаходження відповідних процесорів в зайнятому стані:

$$K_1=S_{10}+S_{11},\ K_2=S_{01}+S_{11}.$$

1.14. Імітаційне моделювання як метод проведення системних досліджень

Імітаційне моделювання полягає в логіко-аналітичній імітації функціонування системи. При цьому для детермінованих систем визначаються зміни їх стану у часі під впливом зовнішніх дій. Імітаційне моделювання – це метод дослідження, який заснований на тому, що аналізована динамічна система замінюється імітатором і з ним проводяться експерименти для отримання інформації про поведінку системи. Роль імітатора зазвичай виконує комп'ютерна програма.

На відміну від аналітичних моделей, які припускають наявність математичного опису процесів, що перетікають в оригіналі і які зазвичай будуються при жорстких обмеженнях на параметри, імітаційні моделі є більш універсальними та можуть бути побудовані при відсутності математичного опису оригіналу. Ідея імітаційного моделювання досить проста і полягає в тому, що будується деякий алгоритм поведінки підсистем та окремих елементів систем у часі. Цей алгоритм може бути реалізований у вигляді програми для ЕОМ. Багаторазово виконуючи моделювання з використанням імітаційної моделі в умовах випадкових потоків подій на вході і в самій системі, можна накопичувати статистичну інформацію про зміни станів імітаційної моделі. Статистична обробка цієї інформації дозволяє отримати статистичні оцінки показники ефективності.

Однак на відміну від аналітичної моделі, імітаційні моделі мають принципову методичну похибку, яка суттєво залежить від обсягу вибірки, і, відповідно, часу спостереження за імітаційною моделлю.

Розглянемо приклад. В якості системи для моделювання виступає однопроцесорна система оперативної обробки, на вхід якої поступає випадковий потік заявок, а можливість створення черги відсутня. Тривалість обслуговування запиту є випадковою величиною. Потрібно визначити параметри роботи системи.

Якщо припустити, що потоки подій є простішими, то можна використовувати відому модель одноканальної системи масового обслуговування без черги (рис. 1.33).

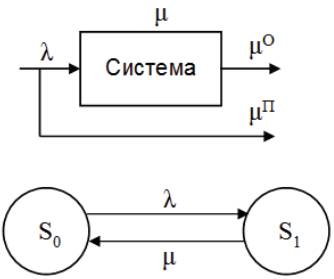


Рис. 1.33.

При цьому використовуються наступні залежності для розрахунку ймовірностей станів:

$$p_0 = \frac{\mu}{\lambda + \mu},$$

$$p_1 = 1 - p_0 = \frac{\lambda}{\lambda + \mu}.$$

Однак в цьому випадку робиться багато спрощень відносно характеристик вхідного потоку заяв та характеристик обслуговування.

Розглянемо процедуру імітаційного моделювання цієї системи. Спеціальним чином генерується вхідний потік заявок, при цьому момент надходження кожної заявки є випадковою подією. Якщо на момент надходження заявки система вільна, то вона поступає на обслуговування. Якщо система зайнята, то заявка покидає систему. Таким чином створюються вихідні потоки обслужених та втрачених заявок.

Блок-схема алгоритму програми імітації системи представлено на рис. 1.34.

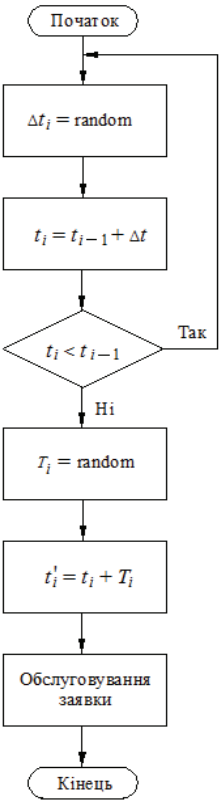


Рис. 1.34.

На рисунку позначено: t_i – момент надходження заявки, t'_i – момент завершення обслуговування, τ_i – тривалість обслуговування.

Імітаційне моделювання технічних та інформаційних систем, до яких також відносяться електронні системи, припускає використання наступних обставин.

Модель системи завжди представляє собою сукупність моделей елементів та їх функціональні зв'язки. Модель елемента системи – це зазвичай набори правил (алгоритмів) поведінки елемента по відношенню до вхідних впливів та змін станів елементу.

Якщо елемент відображає функціональну будову на тому або іншому рівні деталізації, то в простішому випадку він може знаходитися в робочому або не робочому стані. В робочому стані елемент може бути зайнятий виконанням операції, або бути вільним. До правил поведінки пристроїв обчис-

лювальної техніки відносяться правила вибору заявок з черги, реакція пристрою на надходження заявки, реакція пристрою на виникнення відмови в процесі обслуговування заявки та ін.

Таким чином, метою імітаційного моделювання є:

− відтворення з необхідною достовірністю поведінки окремих елементів системи в процесі реалізації функцій системи;

− накопичення статистичних даних про поведінку елементів;

− подальша обробка статистичних даних для отримання статистичних оцінок, кількісних характеристик, законів розподілення, показників ефективності.

При імітаційному моделюванні інформаційних та обчислювальних систем, які характеризуються стохастичною поведінкою, можливі наступні аспекти аналізу:

− при аналізі продуктивності випадковим є потік запитів на обслуговування. Випадковою також є трудомісткість запиту, що визначається кількістю операцій, які необхідні для виконання програми, що обробляє запит користувача;

− при аналізі надійності випадковими є процеси відмов елементів системи, а також інтервали часу, необхідні для ремонту відмовивших елементів.

Імітаційне моделювання інформаційних та обчислювальних систем може використовувати різні рівні абстракції:

− системне моделювання – моделювання на рівні системи (аналіз продуктивності окремих елементів та системи в цілому);

− моделювання на рівні архітектури ЕОМ та регістрів (аналіз ефективності системи команд, аналіз коректності мікропрограм);

− моделювання на рівні логічних схем;

− схемотехнічне моделювання.

При імітаційному моделюванні інформаційних об’єктів в основному використовується системний рівень абстракції, при якому досліджується такий важливий показник, як продуктивність.

Імітаційна модель системного рівня передбачає опис функціонування системи в термінах активності, подій або процесів на деякій мові програмування. Компактність та час розробки моделі покращуються при використанні спеціальних мов та пакетів імітаційного моделювання, хоча імітаційна модель може бути побудована на будь-якій універсальній мові програмування.

Основні етапи імітаційного моделювання:

1. Постановка завдання. На цьому етапі виконується вибір властивостей системи, які потрібно відобразити в імітаційній моделі та не враховуються ті властивості, які на даному етапі дослідження можна вважати несуттєвими. Джерелом інформації зазвичай виступає науково-технічна та довідкова література, результати експериментальних досліджень.

2. Формування та побудова концептуальної моделі. При дослідженні характеристик інформаційних та обчислювальних систем концептуальною моделлю зазвичай виступає система масового обслуговування. Об'єктами моделі в такому випадку можуть бути:

– джерела заявок;

– канали обслуговування;

– черги з обмеженою або нескінченною кількістю місць.

Далі на основі типових схем імітаційного моделювання створюються алгоритми, які формалізують реакцію системи на події, що відбуваються.

1. Вибір мови імітаційного моделювання та побудова моделі.

2. Тестування імітаційної моделі. На цьому етапі виконується перевірка коректності функціонування імітаційної моделі зазвичай з використанням контрольних прикладів.

3. Планування експерименту. Метою етапу є побудова плану експерименту, що забезпечує отримання максимальної інформації про об'єкт з мінімальними затратами ресурсів ЕОМ.

4. Модельний експеримент. На цьому етапі відбувається накопичення результатів моделювання та їх статистична обробка.

5. Інтерпретація результатів модельного експерименту. Виконується аналіз отриманих результатів, робляться висновки про їх достовірність та адекватність, визначаються області використання отриманих даних.

Для опису імітаційного процесу важливим є питання вибору мови програмування. Можливо описати модель на будь-якій з розповсюджених універсальних мов програмування – Паскаль, С++, Фортран, Basic та ін. Однак найбільш ефективним засобом є спеціальні імітаційні мови, більшість з яких ефективно використовуються, особливо за кордоном, де жоден великий проект не реалізується без перевірки на імітаційній моделі. Найбільш відомі мови GPSS, GASP, SIMSCRIPT, DYNAMO, які реалізують різні підходи до моделювання. Одним з найбільш поширених э мова GPSS, яка призначена для моделювання систем з інформаційними потоками. Можна описувати систему та керувати маршрутами проходженням через об'єкти системи спеціальних динамічних елементів – транзактів. Транзакт може розумітися як динамічна одиниця інформаційного потоку, що може переміщуватися від об'єкта до об'єкта та імітувати послідовність обслуговування, яку отримає транзакт за час його перебування в системі.

Необхідно відзначити ряд проблем, що виникають в процесі моделювання систем.

1. Пошук «золотої середини» між спрощенням та складністю системи. Якщо модель занадто проста і в ній не враховано деякі суттєві фактори, то висока вірогідність отримання помилкових даних. Якщо модель занадто складна та в ній враховано фактори, які мають незначний вплив на систему, то зростають витрати на створення такої моделі та існує вірогідність помилки в логічній структурі моделі.

2. Проблема відтворення випадкових впливів оточуючого середовища. Більшість систем є стохастичними, і при їх моделюванні необхідне якісне відтворення випадковості, інакше результати, отримані на моделі, не будуть відповідати дійсності.

3. Оцінка якості моделі та отриманих з її допомогою результатів. Адекватність моделей може бути оцінена методом експертних оцінок, порівнянням з іншими моделями, які підтвердили свою достовірність.

РОЗДІЛ 2. МОДЕЛЮВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

2.1. Опис роботи з системою моделювання Proteus

Існує велика кількість програм для моделювання електронних пристроїв, які відрізняються між собою складністю інтерфейсу, наявністю бібліотек різних компонентів, точністю результатів моделювання. Одним з найпоширеніших пакетів є Electronic WorkBench, що має зручний інтерфейс, велику кількість бібліотек елементів. Однак істотне обмеження полягає в неможливості моделювання схем, в яких є мікропроцесори та програмовані логічні матриці. Аналогічне можна сказати й про систему MicroCap. Також істотним недоліком цих програм є неможливість створення друкованої плати для схеми, що моделюється. Система автоматизованого проектування PCAD призначена для розробки схем і друкованих плат, однак немає засобів моделювання.

САПР Proteus дозволяє моделювати та налагоджувати складні пристрої, у яких може міститися декілька мікроконтролерів одночасно, причому різних сімейств. У бібліотеці є мікроконтролери AVR, 8051, Microchip PIC10, PIC16, PIC18, Philips ARM7, Motorola MC68HC11. Для моделювання мікроконтролера у вікні його параметрів потрібно вказати шлях до файлу прошивки формату «.hex», що може бути створений будь-яким компілятором (IAR, CodeVision, ICC, WinAVR), а також вказати тактову частоту. Крім того, є велика кількість різноманітних віртуальних приладів: осцилограф, логічний аналізатор, лічильник імпульсів, вимірник частоти, відлагоджувачі шин SPI та I^2C, термінал, генератор складних сигналів та інше. У генераторі сигналів можна вказати шлях до WAVE-файлу, інформація з якого буде використана під час моделювання. На виході цього блоку буде напруга, що відповідає даним у файлі. За допомогою цих інструментів можна отримати всю інформацію про роботу схеми. Дані й результати роботи віртуально створеного пристрою можуть бути збережені на комп'ютері тв доступні в цифровому вигляді для подальшого аналізу в пакетах Excel або MATLAB.

Істотною перевагою Proteus є можливість створення власних компонентів. Для цього на сайті виробника треба знайти SPICE-модель і додати її в потрібний корпус. SPICE-модель можна взяти з бібліотек інших пакетів моделювання. Наприклад, у системі PacDesigner фірми Lattice Semiconductors, що призначена для створення систем на базі програмованих аналогових інтегральних схем (ПАІС), є можливість створення SPICE-моделі розробленої системи на ПАІС. З підвищенням складності схеми, що моделюється, зростає різниця між часом моделювання та реальним часом. На рис. 2.1 представлене головне вікно програми.

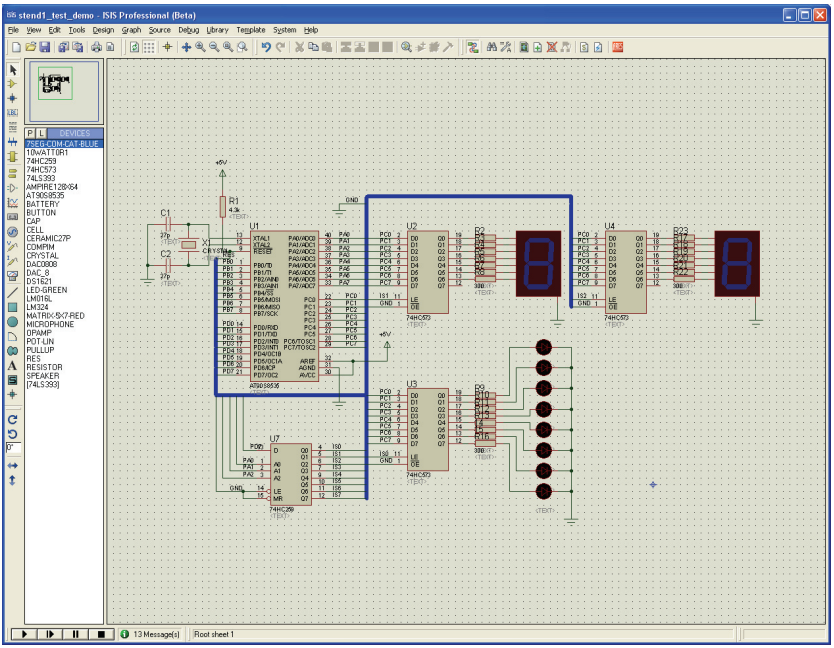


Рис. 2.1.

У лівій верхній області екрана можна побачити міні макет сторінки, а трохи нижче панель DEVICES – компоненти проекту (рис. 2.2). У цьому вікні відображаються всі елементи, що використовуються в схемі.

Рис. 2.2.

Кнопка з буквою «P» відкриває форму пошуку компонента в бібліотеках Proteus для додавання в схему. Кнопка з буквою «L» відкриває менеджер бібліотек – з його допомогою можна підключати нові бібліотеки компонентів.

У бібліотеці Proteus є моделі електромеханічних пристроїв: електродвигуни із квадратурним енкодером зворотного зв'язку, сервоприводи, крокові двигуни, безщітковий трьохобмотковий електродвигун. У цих моделях

можна встановлювати інерційність якоря з підключеними до нього механізмами, навантаження, опір й індуктивність пристрою, номінальна напруга живлення та швидкість обертів без навантаження. Моделі інтерактивних кнопок, перемикачів, ламп, світлодіодів, рідкокристалічних індикаторів, сенсорних панелей, змінних резисторів, дозволяють змінювати їхні параметри прямо в процесі роботи пристрою, що моделюється.

Для додавання компонента в схему необхідно натиснути на кнопку з буквою «P». Відкриється діалогове вікно пошуку й вибору компонентів «Pick Devices», показане на рис. 2.3.

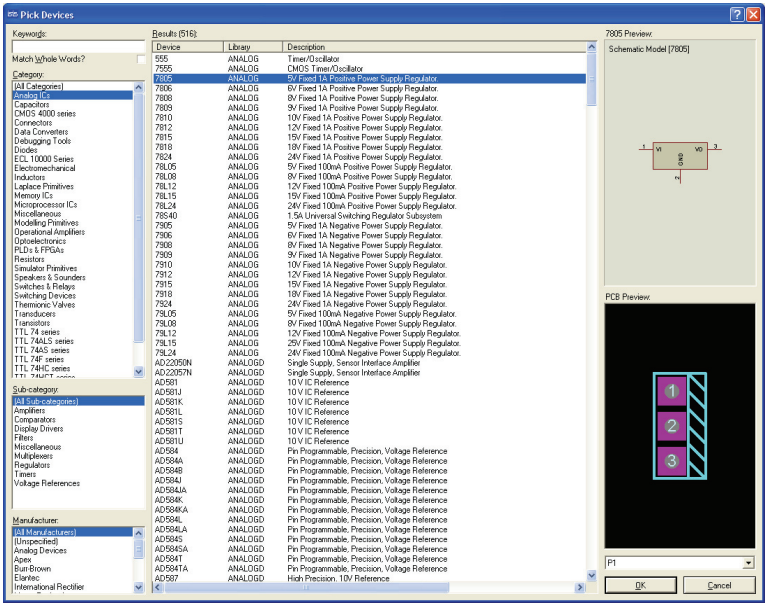


Рис. 2.3.

У полі «Keywords» можна ввести назву потрібного компонента, і якщо такий елемент є, то в правій частині діалогового вікна з'явиться його схематичне зображення, а також зображення посадкового місця для друкованої плати.

У полі «Category» відображається список груп електронних компонентів по їхньому призначенню: аналогові мікросхеми, перетворювачі даних, діоди, транзистори, елементи оптоелектроніки, мікропроцесори, лампи, реле й перемикачі, та інші. Виділивши окрему категорію, у поле «Results» з'явиться весь список доступних компонентів. У полі «Sub-category» можна виділити окремий клас елементів, що дозволить швидше знайти необхідний елемент. Наприклад, для категорії «Діоди» є підкатегорії: мостові випрямлячі, діоди Шотки, стабілітрони, варикапи, тунельні діоди, випрямні діоди.

Помістити компонент на схему можна, натиснувши на кнопку «ОК», а потім перемістити курсор в потрібне місце на аркуші схеми і натиснути ліву кнопку миші. Компонент появиться на схемі.

Компонент на схему можна додати й так: виділити, скопіювати та вставити. Тільки в цьому випадку потрібно вручну привласнити йому порядковий номер замість знаку питання «?».

Виділити компонент або будь-який елемент схеми можна, натиснувши по ньому правою кнопкою миші. При цьому він стане червоного кольору й одночасно будуть виділені всі провідники, що підходять до компоненту – вони виділяться червоними кольорами до першого вузла. Скасувати виділення всіх виділених компонентів можна натисканням правої кнопки миші в порожньому місці схеми. Убрати компонент або будь-який елемент зі схеми можна двома натисканнями правої кнопки миші по ньому.

У табл. 2.1 приводиться опис кнопок панелі інструментів.

Таблиця 2.1.

	L1	Відкрити панель DEVICES – компоненти проекту та пошук нових
	L2	Поставити вузол з'єднання провідників вручну
	L3	Дати назву провіднику – однойменні провідники з'єднані
	L4	Додати текст у довільне місце схеми
	L5	Прокласти шину
	L6	Створити підсхему
	L7	Натискання по компоненту відразу відкриває вікно редагування його властивостей
	L8	TERMINALS – живлення, земля, між блочні з'єднання, виводи
	L9	Додати вивід до створюваного компонента
	L10	Графічне відображення, збереження й аналіз результатів моделювання
	L11	Запис у файл і відтворення даних
	L12	Генератори напруг, струмів і вивід їх з файлів даних
	L13	Вказати точку виміру напруги на провіднику
	L14	Вказати точку виміру струму на провіднику
	L15	Virtual Instruments – вимірювальні прилади
	L16	Прокладка провідників на схемі, провідники можна називати за допомогою L3

Змінювати зображення виділеного компонента на схемі дозволяють кнопки:

– Повернути компонент на 90 градусів по годинній стрілці

– Повернути компонент на 90 градусів проти годинної стрілки

– Поле для введення кута повороту

– Дзеркальний переворот відносно вертикалі

– Дзеркальний переворот відносно горизонталі

Для керування процесом моделювання використовується панель, розташована в нижній частині головного вікна програми (рис. 2.4.).

Рис. 2.4.

Ця панель має 4 кнопки:

– «Пуск» – запуск симуляції або продовження припиненої симуляції;

– «Крок» – виконати мінімальний крок по програмі мікроконтролера (одна інструкція на асемблері). Цією кнопкою також можна почати симуляцію;

– «Пауза» – пауза симуляції. Можна продовжити кнопками «Пуск» або «Крок»;

– «Стоп» – зупинка симуляції. Після цього симуляція почнеться спочатку кнопками «Пуск» або «Крок».

Створіть схему, показану на рис. 2.5.

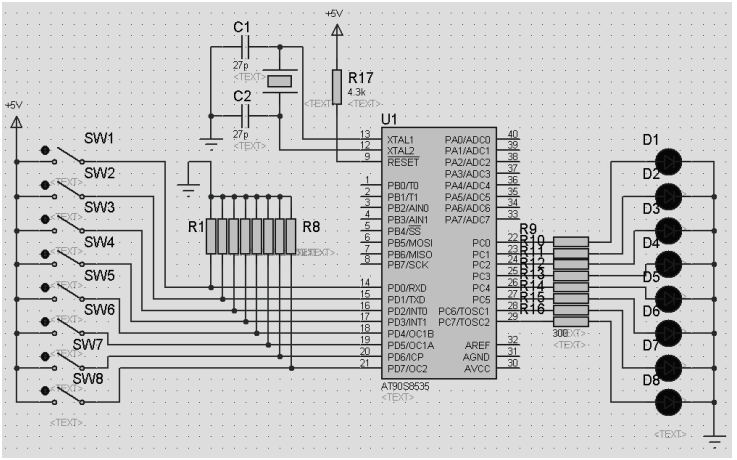


Рис. 2.5.

Виділить мікроконтролер натиснувши по ньому правою кнопкою миші. Мікроконтролер і підключені до нього провідники підсвітяться червоними кольорами. Відкрийте панель редагування властивостей компонента (Edit Component) натиснувши по виділеному мікроконтролеру лівою

кнопкою миші. У діалоговому вікні, що з'явилося (рис. 2.6) натискання по кнопці «Hidden Pins» (сховані виводи) відкриє додаткове меню, у якому буде показано, як за замовчуванням називаються вузли (провідники) схеми до яких підключені живлення VCC і загальний провідник GND. Зміна цих назв може знадобитися при живленні мікроконтролера або інших компонентів різними напругами або від різних джерел.

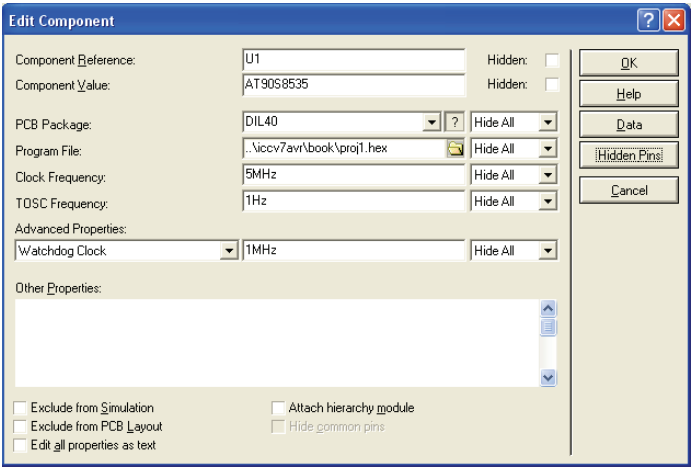


Рис. 2.6.

У полі «Program File» потрібно вказати шлях до файлу прошивки мікроконтролера:

«.cof» – файл якщо буде виконуватися налагодження програми, що написана на мові Сі;

«.hex» – файл прошивки.

У компіляторі ImageCraft необхідно створити новий проект, і відкомпілювати програму, текст якої представлений нижче:

```
#include "io8535v.h"          // підключення заголовочного файлу
void main()                   // початок програми
{
 char data;                   // оголошення змінної
 DDRC=0b11111111;             // налаштування порту С на вихід
 while(1)                     // вічний цикл
 {
  data=PIND;                  // зчитування значення з порту D
  data=data+23;               // до зчитаного значення додається 23
  PORTC=data;                 // результат виводиться в порт С
 }
}
```

Наведена програма зчитує значення з порту D мікроконтролера, додає до цього значення число 23 і виводить результат у порт C.

Після компіляції проекту створюється файл із розширенням «.hex», шлях до якого необхідно вказати в діалоговому вікні налаштування властивостей мікроконтролера.

Також необхідно вказати частоту, на якій буде працювати мікроконтролер (поле Clock frequency). У наведеному прикладі частота дорівнює 5 МГц.

Щоб побачити всі налаштування компонента в панелі редагування властивостей, потрібно поставити галочку навпроти мітки «Edit All Properties As Text» (рис. 2.7).

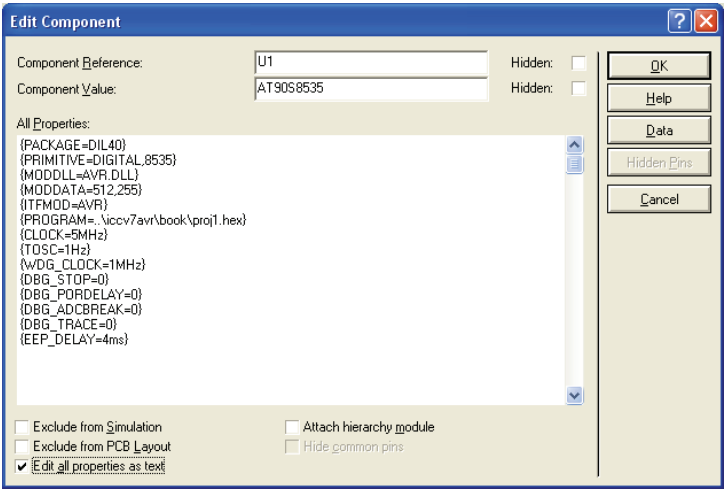


Рис. 2.7.

Запуск процесу моделювання виконується натисканням на кнопку «Пуск», що розташована на панелі керування процесом моделювання. Після цього на світлодіодах можна спостерігати стан виводів мікроконтролера. При натисканні на перемикачі, стан виводів буде змінюватися. У процесі моделювання біля кожного виводу мікроконтролера з'являється кольорова крапка. Якщо на ніжці мікросхеми встановлений рівень логічного 0, крапка має синій колір. Якщо ж на виводі присутній сигнал високого рівня, то крапка буде червоного кольору.

В Proteus є можливість покрокового виконання програми мікроконтролера в режимі налагодження. Для запуску режиму налагодження з меню Debug (рис. 2.8) потрібно вибрати команду Start/Restart Debugging.

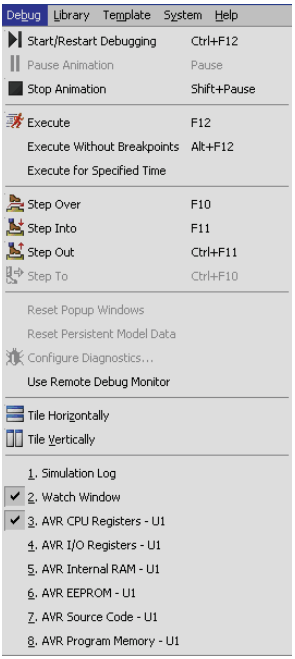


Рис. 2.8.

У нижній частині цього меню розташовується перелік вікон, у яких можна побачити інформацію про стан регістрів. На рис. 2.9 показане вікно спостереження за станами регістрів введення/виведення мікроконтролера.

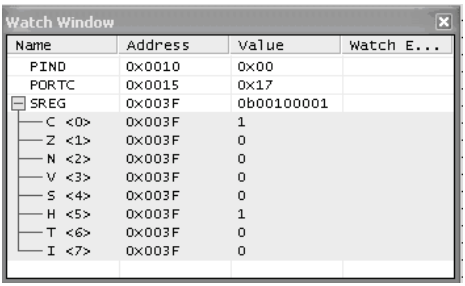


Рис. 2.9.

Для додавання регістра у вікно для спостереження за його станом, необхідно викликати контекстне меню в цьому вікні (рис. 2.10) шляхом натискання правої кнопки миші. Потім вибрати команду «Add Items By Name». У діалоговому вікні, що з'явилося, подвійне натискання лівої кнопки миші на назві регістра призведе до додавання його у вікно спостереження.

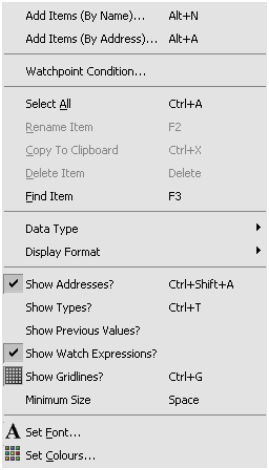


Рис. 2.10.

На рис. 2.11 представлене вікно станів внутрішніх регістрів мікроконтролера. У процесі покрокового налагодження стан регістрів буде змінюватися відповідно до алгоритму роботи програми, і ці зміни можна побачити в цьому вікні.

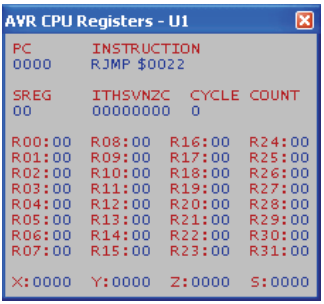


Рис. 2.11.

При натисканні клавіш F10 або F11 виконується перехід до наступної команди програми.

Перевагою пакета Proteus є те, що немає необхідності вносити зміни в розроблені програми для мікроконтролера при їх переносі на реальний пристрій. Система моделювання Proteus дозволяє редагувати та досліджувати програми без особливих матеріальних витрат, додаткових приладів і лабораторних стендів. Найбільша перевага – це додаткові елементи, які дозволяють моделювати роботу мікроконтролера зі світлодіодними індикаторами, рідкокристалічними дисплеями, клавіатурою, кроковим двигуном, датчиками з різними інтерфейсами, програмованою логікою, іншими мікроконтролерами ті зовнішніми пристроями.

2.2. Модель лабораторного стенду для дослідження мікроконтролерів

Існує велика кількість програм для моделювання електронних пристроїв, які відрізняються між собою складністю інтерфейсу, наявністю бібліотек різних компонентів, точністю результатів моделювання. Одним з найпоширеніших пакетів є Electronic WorkBench, який має зручний інтерфейс, велику кількість бібліотек елементів. Однак суттєвим обмеженням є неможливість моделювання схем, в яких є мікропроцесори, програмовані логічні матриці. Теж саме відноситься до системи MicroCap. Також суттєвим недоліком цих програм є неможливість створення друкованої плати для схеми, що моделюється. Система автоматизованого проектування PCAD призначена для розробки схем та друкованих плат, однак немає засобів моделювання.

САПР Proteus дозволяє досить достовірно моделювати та налагоджувати складні пристрої, в яких може міститися декілька мікроконтролерів одночасно, причому різних сімейств. В бібліотеці наявні мікроконтролери AVR, 8051, Microchip PIC10, PIC16, PIC18, Philips ARM7, Motorola MC68HC11. Для моделювання мікроконтролера у вікні його параметрів потрібно вказати шлях до файлу прошивки формату .hex, який може бути створений будь яким компілятором (IAR, CodeVision, ICC, WinAVR), а також вказати тактову частоту. Крім того, є велика кількість різноманітних віртуальних приладів: осцилограф, логічний аналізатор, лічильник імпульсів, вимірювач частоти, налагоджувачі шин SPI та I2C, термінал, генератор складних сигналів та інші. В генераторі сигналів можна вказати шлях до WAVE-файлу, інформація з якого буде використана під час моделювання. На виході цього блоку буде напруга, що відповідає даним у файлі. За допомогою цих інструментів можна отримати всі дані про роботу схеми. Дані і результати роботи віртуально створеного пристрою можуть бути збережені на комп'ютері і доступні в цифровому вигляді для подальшого аналізу в пакетах Excel або MATLAB.

Суттєвою перевагою Proteus є можливість створення власних компонентів. Для цього на сайті виробника треба знайти SPICE-модель та додати її в потрібний корпус. SPICE-модель можна взяти з бібліотек інших пакетів симуляції. Є моделі електромеханічних пристроїв: електродвигуни з квадратурним енкодером зворотного зв'язку, сервоприводи – servo – с налаштуваннями, крокові двигуни, безщітковий трьохобмотковий електродвигун. В цих моделях можна встановлювати інерційність якоря з підключеними до нього механізмами, навантаження, опір та індуктивність пристрою, номінальну напругу живлення та швидкість роботи без навантаження. Моделі інтерактивних кнопок, перемикачів, ламп, світлодіодів, рідкокристалічних індикаторів, сенсорних панелей, змінних резисторів дозволяють змінювати їх параметри прямо в процесі роботи пристрою, що моделюється.

Для вивчення мікропроцесорної техніки зручно використовувати спеціалізовані стенди. На рис. 2.12 представлений зовнішній вигляд стенду,

який розроблений на кафедрі Теоретичної електротехніки та електронних систем Національного університету кораблебудування ім. адм. Макарова та використовується у навчально-дослідній роботі.

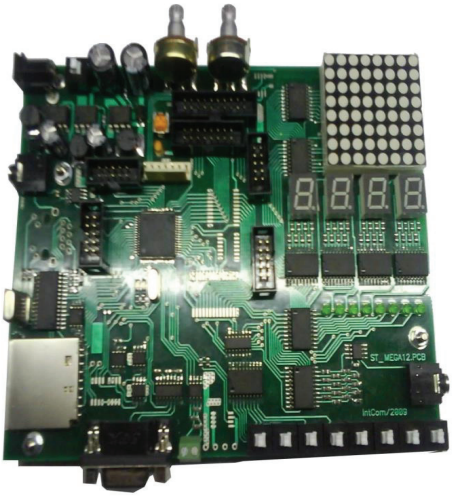


Рис. 2.12.

Лабораторний стенд на базі мікроконтролера ATMEGA128 має в своєму складі:

– мікроконтролер ATMEGA128;

– цифровий датчик температури DS18B20 з інтерфейсом 1-Wire;

– цифровий датчик температури DS1621 з інтерфейсом I2C;

– Flash-пам'ять з інтерфейсом SPI, роз'єм для карт SD-MMC;

– інтерфейс RS-232, RS-485;

– Ethernet-контролер;

– 4-семисегментні індикатори, 8 світлодіодів, 8 кнопок з фіксацією положення;

– інтерфейси підключення графічного та алфавітно-цифрового рідкокристалічних індикаторів;

– інтерфейс клавіатури 4×4;

– 10-бітний аналого-цифровий перетворювач;

– матричний індикатор 8×8.

В пакеті Proteus створена точна модель такого стенду, за допомогою якої можна перевіряти та налагоджувати програми для мікроконтролерів. На рис. 2.13 представлена схема моделі стенду.

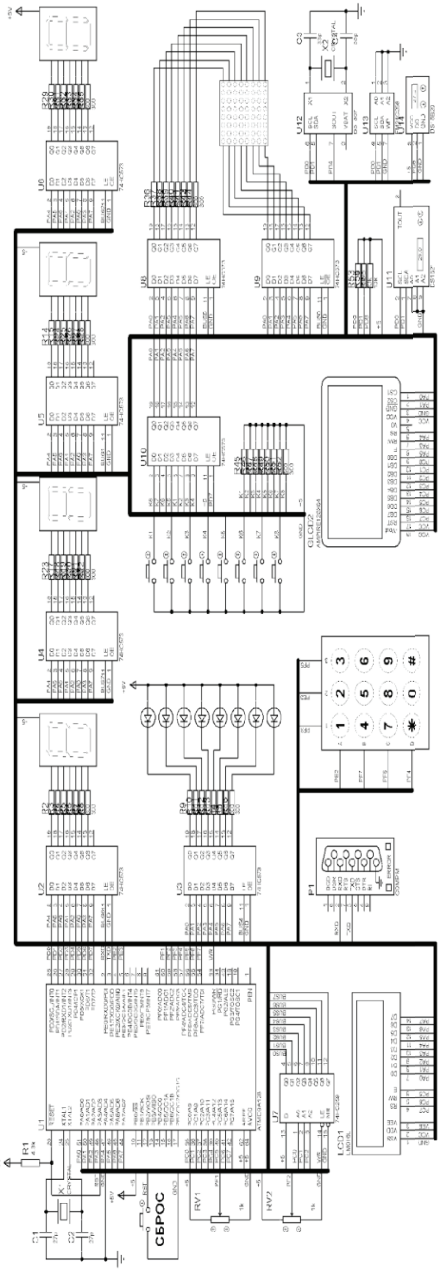


Рис. 2.13.

Основним елементом стенду є мікроконтролер ATMEGA128. Він має велику кількість периферійних пристроїв тому може бути використаний в різних системах. Стенд має засоби для відображення даних: вісім світлодіодів, чотири семисегментні індикатори, які під'єднані до порту А мікроконтролера. Кожен індикатор керується окремим паралельним регістром 74HC573. Дані в регістр записуються при наявності високого рівня на вході LE. Керування регістрами відбувається за допомогою дешифратора двійкового коду в десятковий (мікросхема 74HC259). Входи дешифратора під'єднані до порту С (PC0, PC1, PC2). Для того, щоб керувати визначеним індикатором, потрібно вивести в порт С його адрес, а потім в порт А мікроконтролера вивести дані. В стенді також є матричний індикатор розмірами 8×8, алфавітно-цифровий рідкокристалічний індикатор 2×16 (на базі контролера Hitachi HD44780), а також графічний рідкокристалевий індикатор розмірами 128×64 точок. На ці індикатори може виводитися різноманітна інформація, як текстова, так і графічна, створюватися анімація, меню користувача. Система команд рідкокристалічних індикаторів повністю відповідає їх реальним зразкам. За допомогою інтерактивних кнопок є можливість в процесі моделювання формувати на входах мікроконтролера зміну сигналів. Це зручно при створенні систем, керування якими буде здійснювати оператор. Матричний індикатор дозволяє вивчати принципи створення знаків, динамічної індикації. Модель стенду містить компонент COMPIM, який дозволяє віртуальному пристрою підключитися до реального COM-порту комп'ютера. В моделі можна не використовувати перетворювач рівнів RS232 типу MAX232. Також в схемі присутні датчики температури з інтерфейсами 1-Wire (DS1820) та I2C (DS1607), мікросхема годинника реального часу з інтерфейсом I2C, карта пам'яті SD/MMC з інтерфейсом SPI. При необхідності можна добавити в модель ще кілька датчиків та об'єднати їх в мережу.

Таким чином з'являється можливість створення та відлагодження програмних засобів ще до того, як виготовлений реальний пристрій. Симулятор Proteus дозволяє редагувати та досліджувати програми без особливих матеріальних затрат, додаткових приладів та лабораторних стендів. Найбільша перевага – це додаткові елементи, що дозволяють моделювати роботу мікроконтролера зі світлодіодними індикаторами, рідкокристалічними дисплеями, клавіатурою, кроковим двигуном, датчиками з різними інтерфейсами, програмованою логікою, іншими мікроконтролерами та зовнішніми пристроями. Однак слід зауважити, що віртуальні стенди не переслідують мету повної заміни реальних стендів.

2.3. Моделювання багатопроцесорного режиму зв'язку

2.3.1. Організація зв'язку між мікроконтролерами по послідовному інтерфейсу UART

Установка біта багатопроцесорного режиму зв'язку MPCM у регістрі UCSRA мікроконтролера активізує функцію фільтрації вхідних посилок приймачем УАПП. Посилки, які не містять інформації про адресу ігноруються і не переміщуються в буфер приймання. Це дозволяє істотно зменшити кількість вхідних посилок багатопроцесорних системах, зв'язок між процесорами в яких організований через одну послідовну шину. Значення біта MPCM не виконує ніякого впливу на роботу передавача, але при цьому передавач повинен бути використаний інакше, якщо використовується режим багатопроцесорного зв'язку.

Якщо приймач налаштовується на прийом посилок з 5...8 бітами даних, то перший стоп-біт дозволяє відрізнити призначення прийнятих даних: адреса або дані. Якщо приймач налаштований на прийом 9 біт даних, то значення 9-го біта (RXB8) використовується для ідентифікації адреси або даних. Якщо ідентифікатор типу посилки (перший стоп-біт або 9-ий біт даних) дорівнює 1, то в посилці утримується адреса. У протилежному випадку в посилці передані дані.

Режим багатопроцесорного зв'язку дозволяє декільком підлеглим мікроконтролерам приймати дані від одного ведучого. При цьому підлеглі мікроконтролери при першій адресній посилці визначають, до якого мікроконтролера адресується ведучий. Якщо один з підлеглих мікроконтролерів виявляє свою адресу, то наступні посилки даних він буде приймати в нормальному режимі, а інші підлеглі мікроконтролери ці дані ігнорують доти, поки не буде виявлена наступна адресна посилка.

Якщо мікроконтролер діє як ведучий, то він може використовувати 9-бітний формат даних у посилці (UCSZ = 7). 9-ий біт даних (TXB8) установлюється при передачі адресної посилки (TXB8 = 1) і скидається при передачі посилки даних (TXB = 0). У цьому випадку підлеглі мікроконтролери також повинні встановлювати 9-бітний формат.

Для обміну даними в багатопроцесорному режимі зв'язку необхідно використовувати наступні процедури:

1. Всі підлеглі мікроконтролери (МК) переводяться в багатопроцесорний режим зв'язку (MPCM =1 в UCSRA).

2. Ведучий МК відправляє адресну посилку, а всі підлеглі приймають і зчитують цю посилку. У підлеглих МК прапор RXC у регістрі UCSRA встановлюється як зазвичай.

3. Кожен підлеглий МК зчитує регістр UDR і визначає до кого адресується ведучий МК. Адресований МК повинен очистити біт MPCM в UCSRA, у противному випадку він очікує наступного адресного байта і зберігає налаштування MPCM.

4. Адресований МК приймає всі дані до наступної адресної посилки. Інші підлеглі МК, у яких біт MPCM залишився встановленим будуть ігнорувати посилки даних.

5. Після прийому адресованим МК останньої посилки даних, встановлюється біт MPCM і очікується прийом нової адресної посилки від ведучого МК. Далі процес повторюється з пункту 2.

Використовувати 5.8 – розрядні формати даних можливо, але незручно, тому що приймач повинен перемикатися між *n* та *n*+1 форматами посилки. При використанні 5...8 – розрядних даних у посилці передавач повинен використати два стоп-біти, оскільки перший стоп-біт буде задіяний для індикації типу посилки.

В наступному прикладі виконується розробка модель мережі з двох мікроконтролерів, що з'єднані між собою по інтерфейсу RS232 (рис. 2.14). Розробити простий протокол обміну даними між ними та написати програму для кожного з мікроконтролерів.

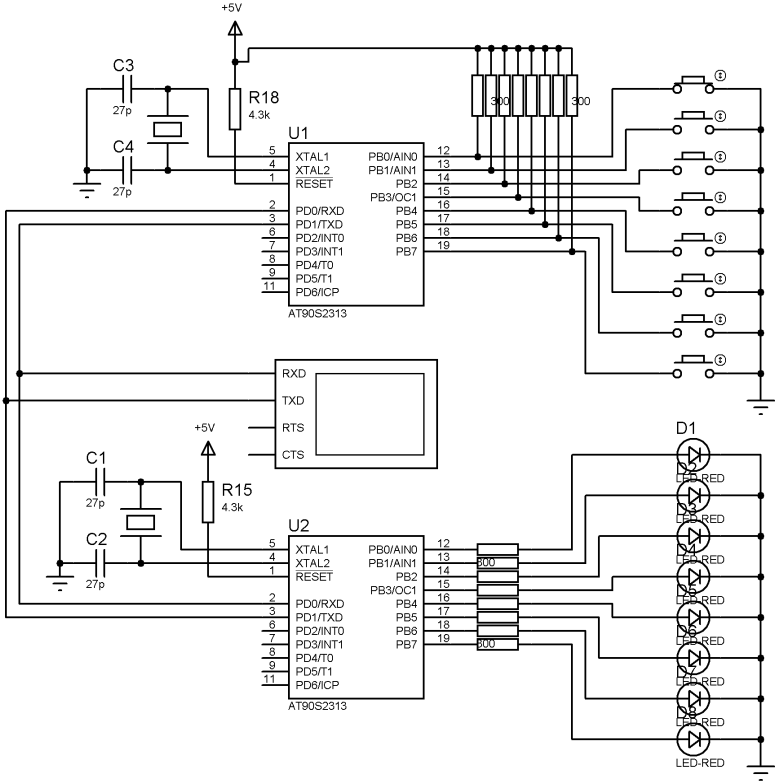


Рис. 2.14.

Для ведучого мікроконтролера текст програми має вигляд:

```
#include <io2313v.h>          // підключення заготовочних файлів
#include <macros.h>
void delay(int ms);           // прототип функції часової затримки
void init_cpu(void);          // функція ініціалізації мікроконтролера
char status;                  // додаткові глобальні змінні
char bcnt;
char rbytes[4];

// обробник переривання по прийманню даних по UART
#pragma interrupt_handler uart0_rx_isr:8
void uart0_rx_isr(void)
{
 char d;                  // додаткова змінна
 d=UDR;                   // зчитування даних з регістру даних
 if(bcnt<2)               // перевірка значення змінної
 {
  rbytes[bcnt]=d;         // запис даних у масив (буфер приймання)
  bcnt++;                 // інкремент значення змінної
 }
 else if(bcnt==2)         // якщо значення змінної дорівнює 2
 {
  rbytes[bcnt]=d;         // запис даних у масив (буфер приймання)
  // перевірка прийнятих даних на відповідність формату
  if((rbytes[0]=='Y') && (rbytes[1]=='F') && (rbytes[2]=='R')) status=0;
  bcnt=0;
  delay(100);             // затримка 100 мс
 }
 else bcnt=0;             // ініціалізація змінної
}

void delay(int ms)        // функція часової затримки
{
 int c1, c2;              // оголошення додаткових змінних
 for(c1=0;c1<ms;c1++)     // в циклі виконується «пуста» операція
  for(c2=0;c2<970;c2++) NOP();
}

void init_cpu(void)       // функція ініціалізації контролера
{
 PORTB = 0;
 DDRB  = 0;
 PORTD = 0;
 DDRD  = 0;
 UBRR = 0x33;             // установка швидкості 9600 біт/с
 UCR  = 0x98;             // дозвіл роботи передавача та приймача
 status=0;                // онулення змінних
 bcnt=0;
 SEI();                   // дозвіл обробки переривань
}
```

```
void main()                    // головна функція програми
{
 char data[4]={'A', 'N',0,'!'};
 int i;
 init_cpu();                   // виклик функції ініціалізації мікроконтролера
 while(1)
 {
 if(status==0)          // якщо значення змінної дорівнює 0
{
data[2]=~PINB;          // зчитування даних з порту
for(i=0;i<4;i++)
{
 UDR=data[i];           // передача даних по UART
 while(!(USR&(1<<UDRE)));          // завершення передачі чергового байта
 }
 status=1;              // присвоєння змінній значення 1
} } }
```

Перший мікроконтролер формує інформаційний пакет для передачі даних. Перші два байти, що передаються, є ASCII-кодами символів A та N, і дозволяють ідентифікувати початок пакету даних. Третім байтом передаються дані про стан кнопок, що підключені до порту B мікроконтролера. Четвертий байт є завершаючим, і за допомогою нього визначається кінець пакету. В циклі почергово передаються всі байти, після чого змінній status присвоюється значення 1. Ведений мікроконтролер після прийому даних надсилає відповідь, що підтверджує отримання даних. Якщо ведений мікроконтролер прийняв дані і надіслав відповідь, то у головному мікроконтролері змінна status приймає значення 0, і формується новий пакет даних для відправлення. Текст програми для веденого мікроконтролера має вигляд:

```
#include <io2313v.h>           // підключення заголовних файлів
#include <macros.h>
void delay(int ms);            // прототип функції часової затримки
void init_cpu(void);           // функція ініціалізації мікроконтролера
char bcnt;                     // оголошення глобальних змінних
char rbytes[4];
// обробник переривання по прийманню символу
#pragma interrupt_handler uart0_rx_isr:8
void uart0_rx_isr(void)
{
 char d, i, data[3]={'Y', 'F', 'R'};        // оголошення масиву символів
 d=UDR;                        // зчитування даних з регістра приймача
 if(bcnt<3)                    // якщо значення змінної менше 3
 {
  rbytes[bcnt]=d;              // запис даних у масив (буфер приймання)
  bcnt++;                      // інкремент значення змінної
 }
 else if(bcnt==3)              // якщо умова виконується
 {
  rbytes[bcnt]=d;              // запис даних у масив (буфер приймання)
```

```
 // перевірка прийнятого пакета на відповідність протоколу
 if((rbytes[0]=='A') && (rbytes[1]=='N') && (rbytes[3]=='!')) PORTB=rbytes[2];
        for(i=0;i<3;i++)
        {
        UDR=data[i];          // передача відповіді у ведучий мікроконтролер
        while(!(USR&(1<<TXC)));  // очікування завершення передачі байта даних
        delay(2);             // затримка 2 мс
        }
 bcnt=0;                      // ініціалізація змінної
 }
 else bcnt=0;
}

void delay(int ms)            // функція часової затримки
{
 int c1, c2;                  // оголошення додаткових змінних
 for(c1=0;c1<ms;c1++)
  for(c2=0;c2<970;c2++) NOP();      // в циклі виконується «пуста» операція
}

void init_cpu(void)
{
 PORTB = 0;
 DDRB  = 0xFF;                // налаштування порту B на вихід
 PORTD = 0;
 DDRD  = 0;
 UBRR = 0x33;                 // установка швидкості 9600 біт/с
 UCR  = 0x98;                 // дозвіл роботи передавача та приймача
 bcnt=0;                      // обнуління змінних
 SEI();                       // дозвіл обробки переривань
}

void main()                   // головна функція програми
{
 init_cpu();                  // виклик функції ініціалізації мікроконтролера
 while(1);                    // вічний цикл
}
```

В наведеній програмі використовується переривання по прийманню даних з UART. Глобальна змінна bcnt використовується як лічильник прийнятих байт. Якщо значення прийнятих байт менше 4, то кожний наступний байт записується в масив. Після прийому 4-х байт виконується їх перевірка відповідності формату пакета. У разі вдалої перевірки на порт мікроконтролера виводиться прийнятий байт і на світло діодах відображається стан кнопок ведучого мікроконтролера. Також формується пакет-відповідь у ведучий мікроконтролер, що підтверджує прийом даних.

2.3.2. Організація обміну даними між мікроконтролерами по інтерфейсу SPI

На сьогоднішній день при побудові автоматизованих систем управління виробничими процесами все частіше постають завдання, які не можуть бути ефективно вирішені системами з одним мікроконтролером або мікропроцесором, що зумовлює зацікавленість замовників у створенні багатопроцесорних систем з розподілом виконуваних функцій. Найпривабливішими для виробництва залишаються мікроконтролерні мережі типу «ведучий – ведений». Більшість таких мереж мають відкритий протокол як на фізичному, так і на канальному рівні, відрізняються простотою його реалізації, і, що саме головне, дані протоколи легко адаптуються до різних застосувань за рахунок наявності в подібних системах властивостей нарощуваності та адаптованості.

Інтерфейс SPI часто використовується для обміну даними між мікроконтролерами. Структурна схема типового застосування представлена на рис. 2.15.

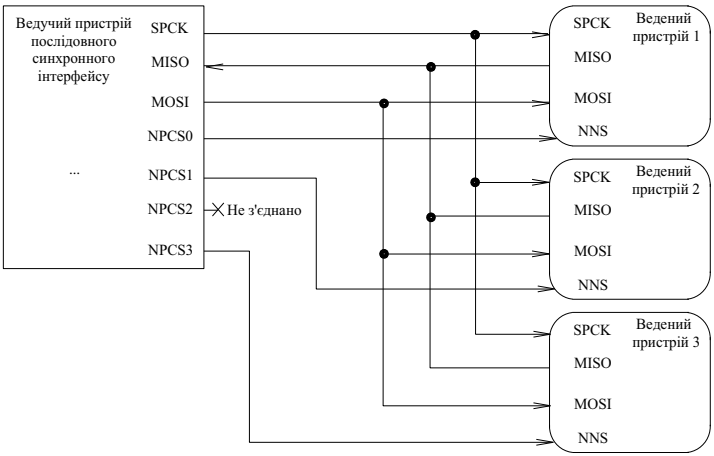


Рис. 2.15.

На рис. 2.16 представлена схема сполучення 2-х мікроконтролерів AT90S8535 по інтерфейсу SPI.

Далі наведено програми для ведучого і веденого мікроконтролерів AT90S8535 в моделі мікропроцесорної мережі (рис. 2.16), які реалізують наступний алгоритм: ведучий мікроконтролер (DD1) виконує інкремент змінної-лічильника, і кожні 200 мс передає значення цієї змінної по інтерфейсу SPI. Прийняте значення виводиться в порт С. Ведений мікроконтролер виводить у порт С прийняте по SPI значення. Прийняте число інвертується і передається назад ведучому мікроконтролеру.

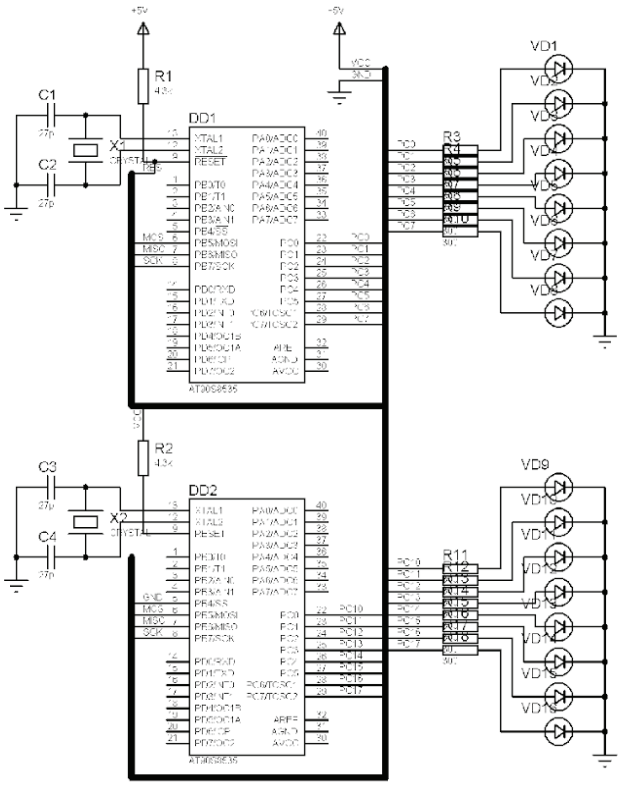


Рис. 2.16.

Запускаємо середовище розробки ImageCraft і створюємо новий проект із назвою Master (див. розділ 2). У налаштуваннях проекту вибираємо контролер AT90S8535. Текст програми для мікроконтролера DD1, що працює в режимі ведучого, представлений нижче.

```
 #include <io8535v.h>          // підключення заголовних файлів
 #include <macros.h>
// функція ініціалізації мікроконтролера
 void init_cpu()
{
 DDRA=0x00;          // порт A налаштований на вхід
 DDRB=0b10110000; // лінії SS, MOSI, SCK налаштовані на вихід
 DDRC=0xFF;          // порт C налаштований на вихід
 DDRD=0x00;          // порт D налаштований на вхід
 SPCR=0x53;                  // налаштування SPI: SPI включений у режимі
                     // ведучого, дільник тактової частоти 128
}
```

```
void delay_ms(int ms)          // функція затримки
{
 int c1, c2;                   // додаткові змінні
 for(c1=0;c1<ms;c1++)          // контролер виконує «порожню» операцію
 for(c2=0;c2<970;c2++) NOP();
}

void main()                    // головна функція
{
 char counter=0;               // оголошення змінної-лічильника
 char rdata;                   // додаткова змінна
 init_cpu();                   // виклик функції ініціалізації контролера
 while(1)                      // вічний цикл
 {
  SPDR=counter;                // передача значення змінної counter
  while(!(SPSR&0x80));         // очікування завершення передачі
  rdata=SPDR;                  // зчитування з регістра даних SPI
  PORTC=rdata;                 // вивід зчитаного значення в порт C
  counter++;                   // інкремент значення змінної
  delay_ms(200);               // виклик функції затримки
 }
}
```

Після компіляції проекту буде створений файл прошивки мікроконтролера. Створюємо новий проект із назвою Slave. Текст програми для мікроконтролера DD2, що працює в режимі веденого, представлений нижче.

```
#include <io8535v.h>           // підключення заголовних файлів
#include <macros.h>
// підпрограма обробки переривання
#pragma interrupt_handler spi_stc_isr:11
void spi_stc_isr(void)
{
char data;                     // оголошення додаткової змінної
data=SPDR;                     // зчитування прийнятого значення з регістра
                               // даних SPI
PORTC=data;          // вивід даних у порт C
SPDR=~data;                    // запис у регістр даних інвертованого
                               // значення
}
// функція ініціалізації мікроконтролера
void init_cpu()
{
DDRA=0;                        // порт A налаштований на вхід
DDRB=0b01000000; // вивід MISO налаштований на вихід
DDRC=0xFF;           // порт C налаштований на вихід
DDRD=0;                        // порт D налаштований на вхід
SPCR=0xC3;                     // налаштування SPI: SPI включений в режимі
                     // веденого, дільник тактової частоти 128
                               // дозволене переривання по прийому
```

```
SEI();                  // дозвіл обробки переривань
}
void main()             // головна функція
{
char c;                 // оголошення додаткової змінної
init_cpu();             // виклик функції ініціалізації контролера
while(1);               // вічний цикл
}
```

2.4. Основи моделювання пристроїв Arduino

Мова програмування пристроїв Arduino заснована на C/C++. Вона проста в освоєнні, і на даний момент Arduino – це, мабуть, найпростіший спосіб програмування пристроїв на мікроконтролерах.

Програмувати Arduino можна без наявності самого Arduino. Для цього використовується середовище моделювання Proteus. В стандартних бібліотеках середовища моделювання електронних систем Proteus. відсутня бібліотека для роботи з Arduino. Тому перед початком роботи її слід завантажити з сайту:

https://www.theengineeringprojects.com/2015/12/arduino-uno-library-proteus.html

На рис. 2.17 представлено фрагмент сторінки цього сайту та позначено посилання, натиснувши на яке можна завантажити бібліотеку Arduino для Proteus.

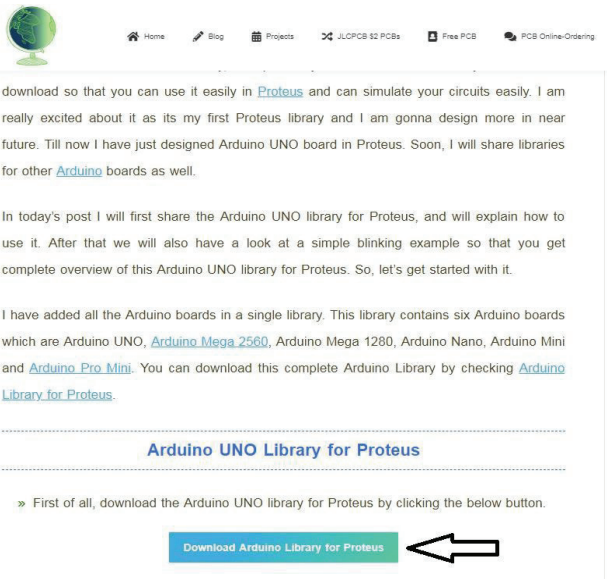


Рис. 2.17.

Також на цьому сайті можна завантажити бібліотеки різноманітних модулів та датчиків для Arduino: ультразвуковий та інфрачервоний сенсор, Bluetooth-модуль, приймач GPS, GSM-радіомодем, датчики газу, полум'я та інші. Додаткові модулі розширюють функціональність Arduino для управління різними пристроями, отримання даних і т.д. Плата розширення WiFi використовується для з'єднання з бездротовими мережами стандарту 802.11. Плата розширення Xbee Shield забезпечує за допомогою модуля Maxstream Xbee Zigbee бездротовий зв'язок з кількома пристроями Arduino в радіусі до 35 метрів (в приміщенні) і до 90 метрів (поза приміщенням). Плата розширення Motor Shield забезпечує управління двигунами постійного струму і зчитування даних з датчиків положення. Плата розширення Ethernet Shield забезпечує підключення до Інтернету. На рис. 2.18 представлено деякі доступні моделі датчиків та модулів розширення для Proteus.

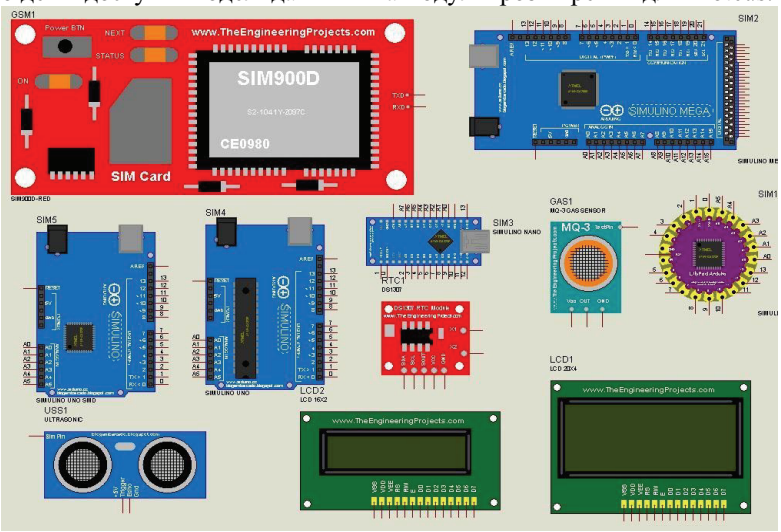


Рис. 2.18.

Після завантаження архіву з файлами моделей його необхідно розпакувати. В результаті в каталозі, в який відбувається розпаковка архіву, з'явиться 2 файли: ArduinoUnoTEP.LIB та ArduinoUnoTEP.IDX. Ці файли необхідно скопіювати в каталог з бібліотеками Proteus, який називається Library. Детальну інструкцію по підключенню в Proteus бібліотек різних модулів для роботи з Arduino можна знайти за посиланням: http://www.instructables.com/id/How-to-add-Arduino-Library-in-to-Proteus-7-8/

Для моделювання роботи Arduino в середовищі Proteus спочатку потрібно скласти схему. Для цього необхідно запустити середовище моделювання Proteus, і в головному вікні програми натиснути кнопку з літерою «P» для додавання елемента в схему, як показано на рис. 2.19.

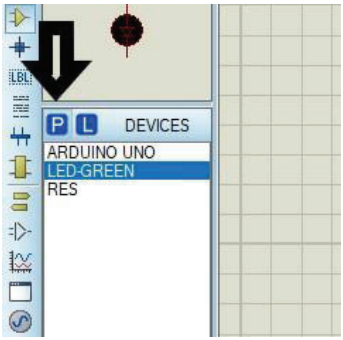


Рис. 2.19.

У вікні додавання електронного компонента, що з’явиться, в списку категорій ліворуч потрібно обрати категорію Arduino, а в списку праворуч модель пристрою, як показано на рис. 2.20. В даному випадку це Arduino UNO.

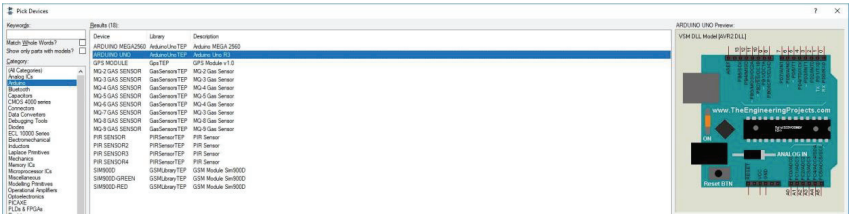


Рис. 2.20.

Аналогічне потрібно виконати для додавання в схему світлодіода та резистора. Пришвидшити процес пошуку елемента в бібліотеці можна шляхом введення його назви в поле Keywords у вікні додавання компонента (рис. 2.20). Це поле вводу знаходиться зверху ліворуч. Для пошуку резистора потрібно ввести RES, а для пошуку світлодіода потрібно ввести LED-GREEN.

Для додавання в схему джерела живлення та загального провідника потрібно на робочому полі середовища Proteus натиснути праву кнопку миші, і з контекстного меню, що з’явиться, вибрати пункт Place – Terminal – GROUND, як показано на рис. 2.21.

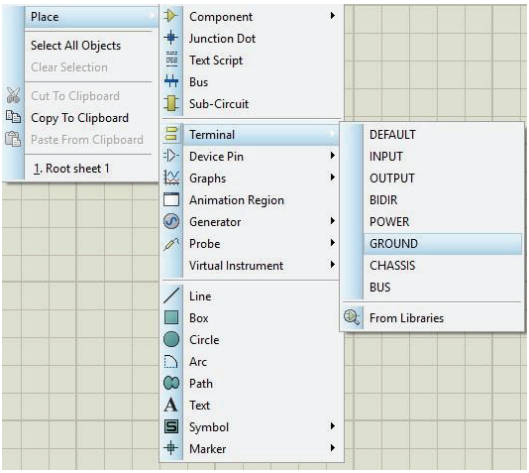


Рис. 2.21.

Далі ці елементи потрібно розташувати на робочому полі та з'єднати, як показано на рис. 2.22.

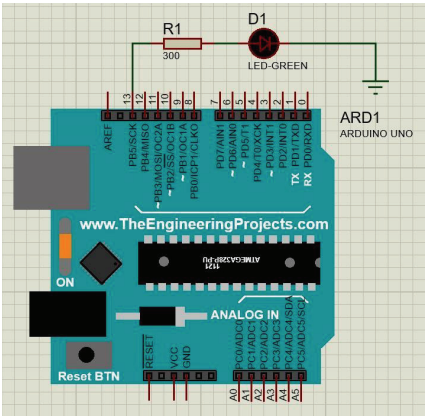


Рис. 2.22.

Наступний крок, який необхідно зробити, це вказати файл «прошивки» мікроконтролера з відкомпільованою програмою. Цей файл має розширення .hex. За замовчуванням при компіляції програми в середовищі Arduino такий файл не створюється. Тому потрібно дещо змінити налаштування програми. В меню Файл потрібно вибрати пункт Налаштування. У вікні, що з'явиться, потрібно встановити позначку Компіляція, як показано на рис. 2.23.

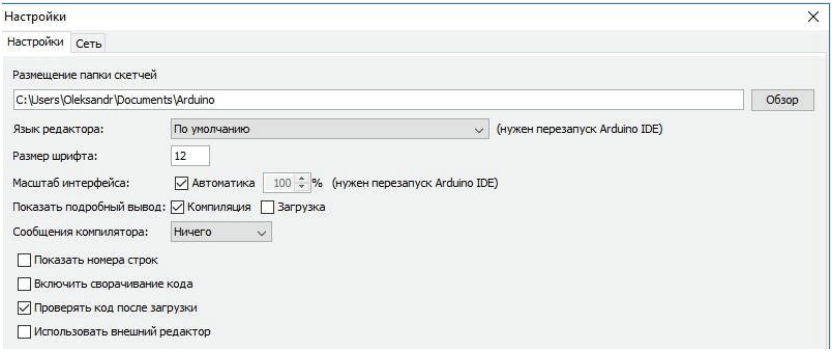


Рис. 2.23.

Тепер, після компіляції програми у програмному забезпеченні Arduino, у вікні повідомлень у нижній частині вікна програми з'явиться додаткова інформація, а саме інформація про місце розташування файлу «прошивки». В даному випадку це файл «Blink.ino.hex», що можна побачити на рис. 2.24.

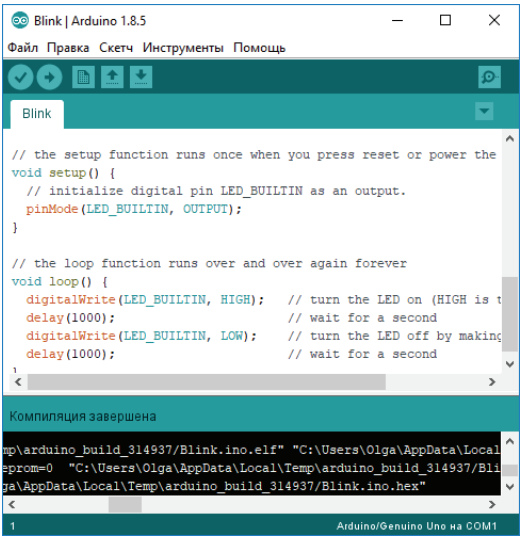


Рис. 2.24.

З цього вікна можна скопіювати весь рядок, який вказує шлях розташування файлу на диску, і вставити його в поле вводу Program File у вікні властивостей компонента, яке можна визвати подвійним натисканням лівої кнопки миші на самому компоненті схеми. На рис. 2.25 представлене вікно властивостей моделі плати Arduino UNO в середовищі Proteus, де потрібно вказати шлях до файлу «прошивки», що має розширення .hex.

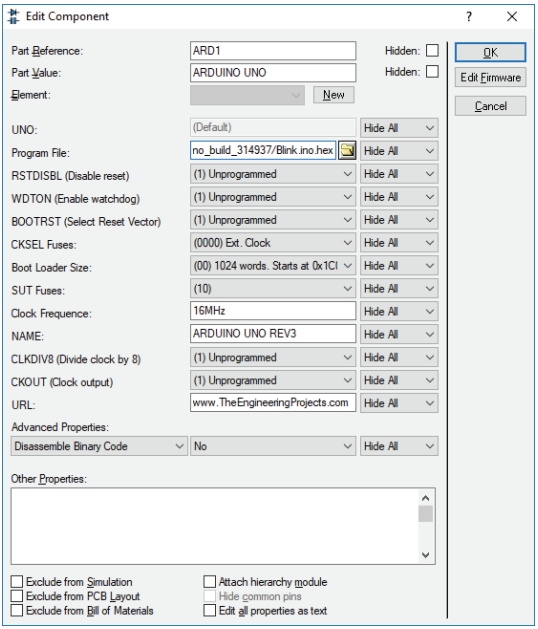


Рис. 2.25.

Тепер схему можна моделювати. Для цього потрібно натиснути на кнопку Run the simulation із зображенням стрілки, яка розташована в правому нижньому куті головного вікна програми (перша кнопка на рис. 2.26).

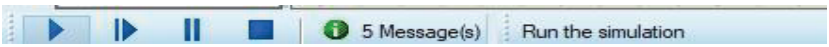
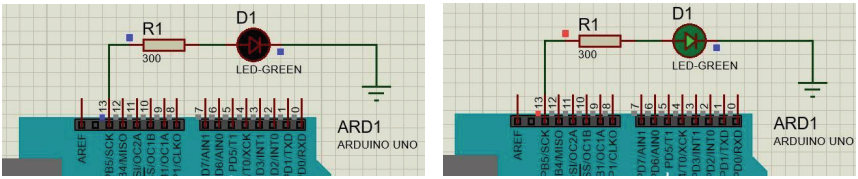


Рис. 2.26.

Після запуску процесу моделювання у вікні Proteus можна спостерігати зміну стану світлодіода, яка відбувається з періодом 1 секунда. На рис. 2.27, а, б, представлено два різних стани світлодіода в процесі моделювання роботи схеми.



а б

Рис. 2.27.

Текст програми, що виконує блимання світлодіодом з періодом 1 секунда, представлено нижче.

```
void setup() {
  // налаштування виводу LED_BUILTIN на вихід.
  pinMode(LED_BUILTIN, OUTPUT);
}

// цикл виконується весь час, доки є живлення пристрою
void loop() {
  // увімкнути світлодіод (на виводі високий рівень напруги HIGH)
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);              // виклик функції затримки 1 секунда = 1000 мілісекунд
  // вимкнути світлодіод (на виводі низький рівень напруги LOW)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);              // виклик функції затримки 1 секунда = 1000 мілісекунд
}
```

Для перевірки роботи програми на реальному пристрої потрібно зібрати схему, що представлена на рис. 2.28. Для завантаження програми в контролер потрібно натиснути кнопку «Загрузить», яка розташована на панелі інструментів середовища розробки Arduino IDE. Також не слід забувати в меню «Інструменти» головного вікна програми вказати послідовний комунікаційний порт, до якого підключено контролер Arduino. Це може бути порт COM3, COM4 або інший. На початку розділу було описано, яким чином можна дізнатися номер порту.

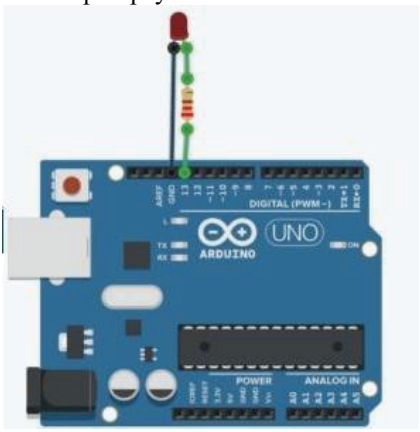


Рис. 2.28.

Функція setup() викликається, коли стартує скетч. Використовується для ініціалізації змінних, визначення режимів роботи виводів, запуску використовуваних бібліотек і т.д. Функція setup запускається тільки один раз, після кожної подачі живлення або скидання плати Arduino.

Після виклику функції setup(), яка ініціалізує і встановлює початкові значення, функція loop() робить те, що означає її назва – виконується в циклі, дозволяючи програмі здійснювати обчислення і реагувати на них. Її потрібно використовувати для управління платою Arduino. Саму в функції loop() розташовується основна логіка програми.

Функція digitalWrite() подає HIGH (високий рівень, логічна 1) або LOW (низький рівень, логічний 0) значення на цифровий вхід/вихід (pin).

Якщо вхід/вихід (pin) був встановлений в режим вихід (OUTPUT) функцією pinMode(), то для значення HIGH напруги на відповідному вході/виході (pin) буде 5В (3,3В для 3,3V плат), і 0В (земля) для LOW.

Якщо вхід/вихід (pin) був встановлений в режим вхід (INPUT), то функція digitalWrite() зі значенням HIGH буде активувати внутрішній 20K навантажувальний резистор. Подача LOW, в свою чергу, відключає цей резистор. Якщо світлодіод працює, але яскравість низька, можливо необхідно встановити режим «вихід» (OUTPUT) функцією pinMode().

Синтаксис: digitalWrite(pin, value);

Параметри: pin: номер вхід/вихід (pin); value: значення HIGH або LOW.

Функція delay() зупиняє виконання програми на задану в параметрі кількість мілісекунд (1000 мілісекунд в 1 секунді). Більшість активності плати зупиняється функцією delay(). Проте робота переривань не зупиняється, триває запис послідовно (serial) переданих даних на RX порту, ШІМ сигнал (analogWrite) продовжує генеруватися на портах.

Синтаксис ; (крапка з комою) використовується для позначення кінця оператора. Забута в кінці рядка крапка з комою призводить до помилки компіляції. Текст помилки може бути або видимим і посилатися на пропущену крапку з комою, або ні. Якщо зустрічається незрозуміла або схожа на нелогічну помилка компіляції, одним з перших дій повинна бути перевірка пропущених точок з комою в коді, безпосередньо в попередньому рядку, в якій компілятор видав попередження.

Фігурні дужки {} (також називаються просто «дужки») – важливий елемент мови програмування Сі. Вони використовуються в кількох різних конструкціях, наведених нижче, і це може іноді збивати з пантелику початківців.

Відкриваюча дужка «{» повинна завжди супроводжуватися закриваючою дужкою «}». Ця умова відома як парність (симетричність) фігурних дужок. Arduino IDE (інтегроване середовище розробника) включає відповідний інструмент для перевірки парності дужок. Досить виділити дужку, або навіть поставити курсор відразу ж за дужку, як буде підсвічена її логічна пара.

Програмісти-початківці або програмісти, які перейшли на Сі з Бейсика, часто вважають використання фігурних дужок лякаючим. Зрештою, одні і ті ж фігурні дужки замінюють оператор RETURN в підпрограмах (функціях), оператор ENDIF в умовних циклах і оператор NEXT в циклах FOR.

Оскільки використання фігурних дужок настільки багатогранно, рекомендованою практикою програмування буде друкувати закриваючу фігурну дужку відразу після того, як надрукована відкриваюча дужка, коли вставляється конструкція, для якої потрібно використовувати фігурні дужки. Потім курсор слід повернути в позицію між фігурними дужками і почати вводити оператори.

Непарні дужки можуть часто приводити до прихованих, незрозумілих помилок компіляції, які складно відстежити у великій програмі. Через їх всебічне використання, дужки також неймовірно важливі в синтаксичній правильності програми і переміщення дужки на один або два рядки часто призводять до значного впливу на логіку програми.

Коментарі – це рядки в програмі, які використовуються для інформування про те, як працює програма. Вони ігноруються компілятором і не експортуються в процесор, таким чином, вони не займають місце в пам'яті мікроконтролера.

Коментарі призначені тільки для того, щоб допомогти зрозуміти (або згадати), як працює програма, або пояснити це іншим. Є два способи позначити рядок як коментар:

```
x = 5;  // це коментар в одному рядку. Все після подвійного слеша – коментар
        // до кінця рядка
/ * Це багаторядковий коментар – використовуйте його для
коментування цілих фрагментів коду
if (gwb == 0) {
// коментар в рядку припустимий всередині багаторядкового коментаря
// але не інший багатостроковий коментар
}
// не слід забувати «закривати» коментарі – вони повинні бути парними
* /
```

Під час експериментів з кодом, коментування частин програми – відповідний спосіб видалення рядків, в яких можуть бути помилки. Так рядки в коді залишаються, але перетворюються в коментарі, і компілятор просто ігнорує їх. Це може бути особливо корисно при локалізації проблеми, або коли не виходить скомпілювати програму, а повідомлення про помилку при компіляції приховано або не містить змісту.

2.5. Моделі систем керування серводвигуном та кроковим двигуном

Наступний приклад демонструє систему керування положенням ротора сервопривода в залежності від освітленості. Цей приклад може бути використаний для автоматичного керування положенням жалюзі у будинку в залежності від рівня освітленості в приміщенні. На рис. 2.29 представлено принципову схему. Для наочності в схемі також є вольтметр, на якому відображається значення напруги, що поступає на аналоговий вхід (вхід АЦП)

Arduino. При зміні освітленості датчика, яким виступає фоторезистор. Його опір зменшується, і, відповідно, зменшується напруга, що надходить на вхід A0.

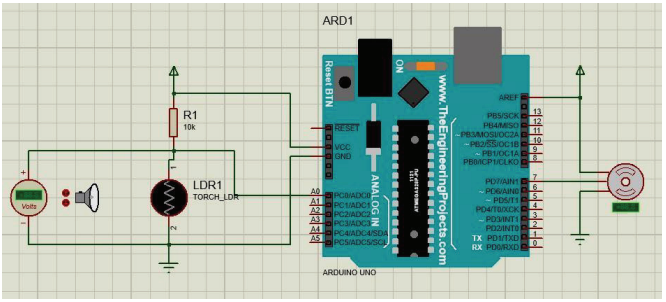


Рис. 2.29.

Текст програми керування положенням ротора сервопривода в залежності від освітлення представлено нижче.

```
#include <Servo.h> // підключення бібліотеки керування сервоприводом
const int servoPin = 7; // номер виводу, до якого підключено сервопривод
const int sensorPin = A0; // вхід, до якого підключений датчик освітлення

int sensorValue = 0; // змінна для зберігання даних, зчитаних з датчика
Servo myservo; // об'єкт для керування сервоприводом

void setup() {
  myservo.attach(servoPin); // ініціалізація сервопривода шляхом вказівки,
                            // до якого виводу плати Arduino він підключений
}

void loop() {
 int angle;       // змінна для зберігання кута повороту ротора сервопривода
 sensorValue = analogRead(sensorPin); // зчитування даних з датчика освітленості
 angle = sensorValue*0.176; // перетворення апаратних одиниць в кут повороту
 myservo.write(angle);  // встановлення сервоприводу в потрібне положення
                        // в залежності від освітленості датчика
 delay(50);             // затримка 50 мс
}
```

В програмі використовується коефіцієнт 0,176 для перетворення апаратних одиниць, що отримуються після аналого-цифрового перетворення сигналу з датчику (максимальне значення 1023) в кут сервопривода (максимальне значення 180). Після компіляції програми можна виконати моделювання роботи схеми і перевірити коректність алгоритму. На рис. 2.30 представлено вигляд схеми під час моделювання за умови мінімальної освітленості датчика. Положення ротора сервоприводу складає 90 градусів.

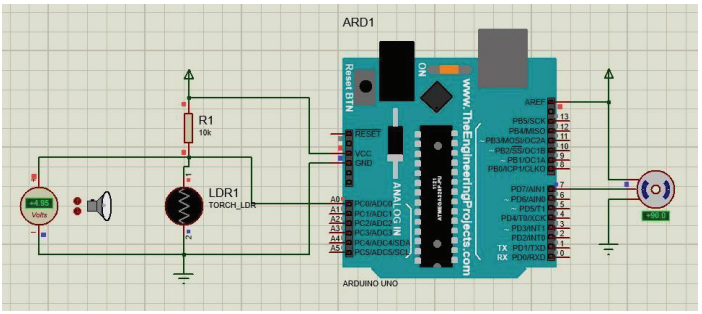


Рис. 2.30.

На рис. 2.31 представлена схема показаний випадок, коли освітленість датчика більша, ніж в попередньому випадку. Видно, що ротор сервоприводу змінив положення і зараз кут становить 50,7 градусів.

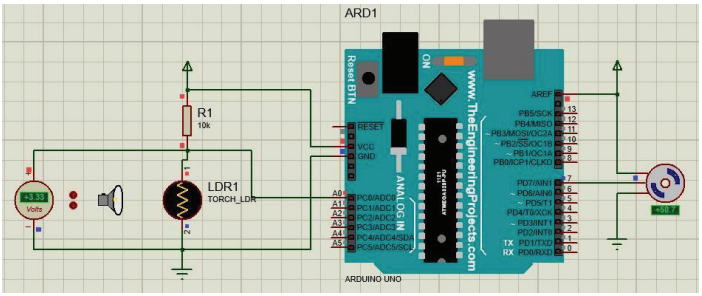


Рис. 2.31.

Бібліотека функцій Servo для Arduino контролера надає набір функцій для управління сервоприводами. Стандартні сервоприводи дозволяють повертати привід на визначений кут, зазвичай від 0 до 180 градусів. Деякі сервоприводи дозволяють здійснювати повні оберти на заданій швидкості.

Бібліотека Servo дозволяє одночасно управляти 12 сервоприводами на більшості плат Arduino і 48 на Arduino Mega. На контролерах, відмінних від Mega, використання бібліотеки виключає можливість використовувати виходи 9 і 10 в режимі ШІМ навіть якщо привід не підключений до цих виводів. На платі Mega можуть бути використані до 12 сервоприводів без втрати функціоналу ШІМ.

У загальному випадку сервопривод підключається 3-ма проводами: живлення, земля і сигнальний. Зазвичай живлення – червоний провід і може бути підключений до виводу +5В на платі Arduino. Чорний провід (земля) підключається до виводу GND плати Arduino. Сигнальний провідник, який зазвичай має жовтий колір, підключається до цифрового виводу контролера

Arduino. Слід зазначити, що потужні сервоприводи можуть створювати велике навантаження. В цьому випадку він повинен живитися окремо (не через вихід +5В контролера Arduino).

Функція Servo.attach() підключає Servo до зазначеного виходу, з якого здійснюється управління приводом. Функція має наступний синтаксис:

```
servo.attach(pin);
servo.attach(pin, min, max);
```

Параметри функції: servo – змінна типу Servo; pin – номер виходу, до якого підключається servo і з якого здійснюється управління приводом; min (опціональний параметр) – ширина імпульсу в мікросекундах, що відповідає мінімальному (кут 0 градусів) положенню сервоприводу (за замовчуванням 544); max (опціональний параметр) – ширина імпульсу в мікросекундах, який відповідає максимальному (кут 180 градусів) положенню сервоприводу. Опціональний параметр означає, що вказувати значення цього параметра при виклику функції не обов'язково. Якщо опціональний параметр відсутній, то використовується значення за замовчуванням.

Функція Servo.write() передає значення для управління приводом. Для стандартного сервоприводу це кут повороту. Для приводу постійного обертання функція задає швидкість обертання (0 – для максимальної швидкості обертання в одну сторону, 180 – для максимальної швидкості в іншу сторону і 90 для нерухомого стану).

Функція має наступний синтаксис:

```
servo.write(angle);
```

Параметри функції: servo – змінна типу Servo; angle – значення кута повороту сервопривода від 0 до 180.

Бібліотека Stepper надає зручний інтерфейс управління біполярними і уніполярними кроковими двигунами. Для управління кроковим двигуном, в залежності від його типу (біполярний або уніполярний) і обраного способу підключення, знадобляться деякі додаткові електронні компоненти.

На рис. 2.32 представлено схему підключення уніполярного крокового двигуна до контролера Arduino. Безпосередньо підключати кроковий двигун до контролера неможна, оскільки навантажувальна здатність виводів недостатня для безпосереднього керування двигуном. В цій схемі використовується спеціальна мікросхема ULN2004A, яка представляє собою набір транзисторів Дарлінгтона та має виходи типу відкритий колектор. В даному випадку мікросхема відіграє роль підсилювача потужності. Сигнали керування, що поступають на вхід мікросхеми від контролера, змінюють режим роботи виводів мікросхеми і струм протікає по визначених обмотках крокового двигуна, тим самим обертаючи його ротор.

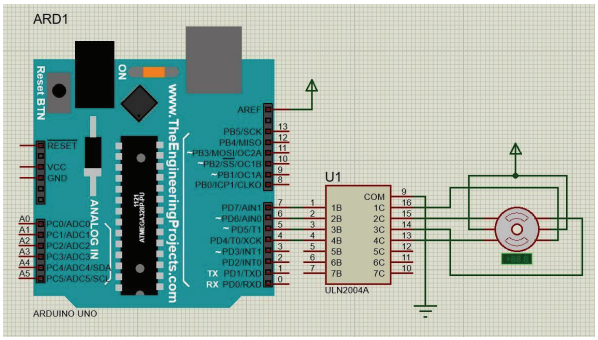


Рис. 2.32.

На рис. 2.33 представлено вікно налаштувань моделі крокового двигуна в Proteus. Напруга живлення крокового двигуна складає 5 В. Кількість кроків для здійснення одного повного оберту дорівнює 72, тобто один крок складає 5 градусів. Максимальна швидкість обертання ротора крокового двигуна складає 100 обертів на хвилину.

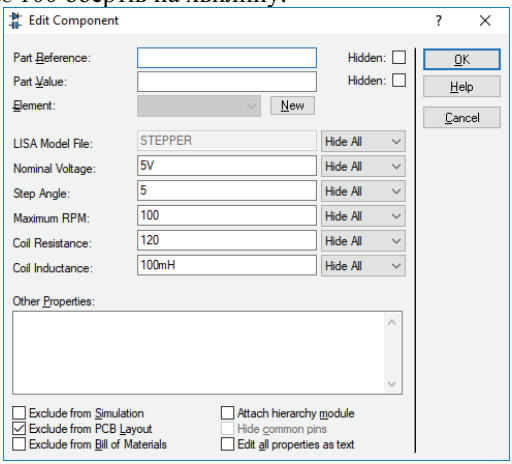


Рис. 2.33.

Текст програми, що забезпечує обертання крокового двигуна, представлений нижче.

```
#include <Stepper.h> // підключення бібліотеки для роботи з кроковим двигуном
// створення об’єкта для керування кроковим двигуном
// в якості параметрів передаються значення кількості кроків на 1 оберт та
// номери виводів контролера Arduino, до яких підключено обмотки двигуна
Stepper mystepper = Stepper(72, 4, 5, 6, 7);

void setup() {
```

```
 mystepper.setSpeed(1); // ця функція встановлює швидкість обертання 1 об/хв.
}

void loop() {
 mystepper.step(1);   // здійснити оберт на 1 крок
// затримка 100 мс щоб завершилися всі перехідні процеси в обмотках
 delay(100);
}
```

Функція-конструктор Stepper створює новий об'єкт класу Stepper. Цей об'єкт приєднаний до одного крокового двигуна, підключеного до контролера Arduino. Конструктор слід використовувати при оголошенні змінної класу Stepper, зазвичай на самому початку, поза setup() і loop(). Кількість параметрів залежить від способу підключення – 2 або 4 виводи використовується для керування кроковим двигуном. Синтаксис конструктора має наступний вигляд:

```
Stepper (steps, pin1, pin2);
Stepper (steps, pin1, pin2, pin3, pin4);
```

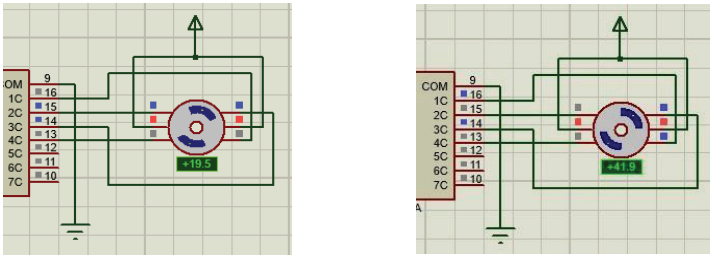
Параметри конструктора об'єкта Stepper наступні: steps – кількість кроків в повному обороті крокового двигуна, який використовується. Якщо в документації до двигуна вказано кут одного кроку, то слід розділити 360° на цей кут, що дасть кількість кроків. Так, наприклад, якщо один крок складає 3,6°, то кількість кроків на один повний оберт буде 100 (360° / 3,6° = 100 кроків); pin1, pin2: номери двох виходів, які використовуються для управління кроковим двигуном; pin3, pin4 (опціональні параметри) – два додаткові виходи для випадку використання 4-х провідної схеми підключення крокового двигуна до контролера. Значення, що повертається конструктором – новий екземпляр об'єкта класу Stepper.

Функція Stepper.step(steps) обертає кроковий двигун на певну кількість кроків, на швидкості, заданою функцією setSpeed(). Ця функція є блокуючою, тобто вона очікує закінчення обертання двигуна, перш ніж передати управління в наступний рядок коду. Наприклад, якщо швидкість обертання встановлена на 1 RPM (1 оберт в хвилину), і викликана функція step(100) на двигуні з повним оборотом в 100 кроків, то виконання цієї функції займає цілу хвилину. Правильніше організовувати управління так, щоб швидкість була висока, а за один виклик step() робилося лише кілька кроків, для уникнення тривалого блокування виконання коду скетчу. Функція має наступні параметри: steps (кількість кроків) – позитивне число викликає обертання в один бік, негативне значення – в протилежний. Функція не повертає жодного значення.

Функція Stepper.setSpeed(rpms) встановлює швидкість обертання в обертах на хвилину (RPMs – rotations per minute). Дана опція не примушує двигун обертатися, а лише встановлює швидкість обертання, яка буде використана при виконанні функції step(). Параметри функції: rpms – швидкість,

з якою буде відбуватися обертання крокового двигуна, виражається в обертах на хвилину. Значення має бути позитивним цілим. Опис бібліотеки Stepper для керування кроковими двигунами можна знайти за посиланням: http://arduino.ru/Reference/Library/Stepper

Результати моделювання роботи системи керування кроковим двигуном представлено на рис. 2.34, а, б. На рисунках видно, що кут повороту крокового двигуна змінюється і поступово збільшується.



а б

Рис. 2.34.

На рис. 2.35 представлена осцилограма сигналів керування, що поступають на кроковий двигун і змушують його обертатися.

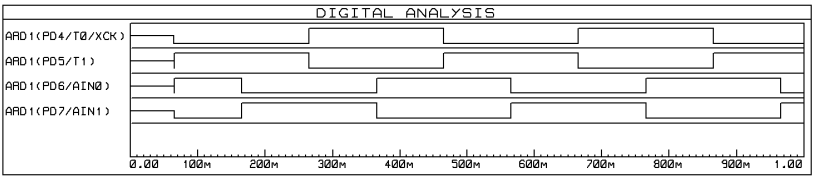


Рис. 2.35.

Ще один приклад схеми системи керування кроковим двигуном представлено на рис. 2.36.

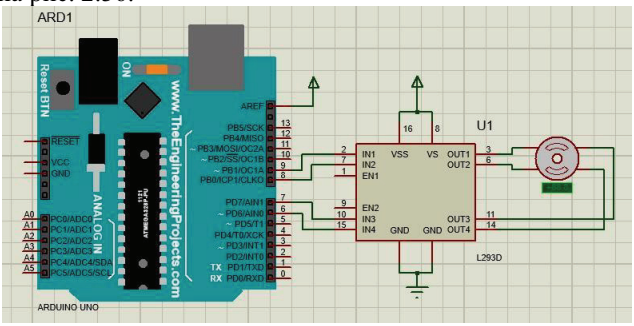


Рис. 2.36.

В даній схемі використовується спеціальна мікросхема-драйвер L293D, яка також виконує функцію підсилювача потужності, адже навантажувальна здатність виводів контролера Arduino недостатня для безпосереднього керування уніполярним кроковим двигуном. Текст програми представлено нижче.

```
#include <Stepper.h> // підключення бібліотеки для роботи з кроковим двигуном
// створення об'єкта для керування кроковим двигуном
// в якості параметрів передаються значення кількості кроків на 1 оберт та
// номери виводів контролера Arduino, до яких підключено обмотки двигуна
Stepper mystepper = Stepper(72, 6, 7, 9, 8);

void setup() {
 mystepper.setSpeed(1); // ця функція встановлює швидкість обертання 1 об/хв.
}

void loop() {
 mystepper.step(1);   // здійснити оберт на 1 крок
 delay(100);     // затримка 100 мс щоб завершилися всі перехідні процеси в обмотках
}
```

2.6. Моделювання мікропроцесорних систем збору і обробки даних

2.6.1. Моделювання системи з інфрачервоним датчиком руху HC-SR501

Для з'єднання з мікроконтролером, або безпосередньо з реле, у датчика HC-SR501 є три виводи – живлення, загальний провідник та вихідний сигнал. Підключається датчик до контролера Arduino за схемою, що представлена на рис. 2.37.

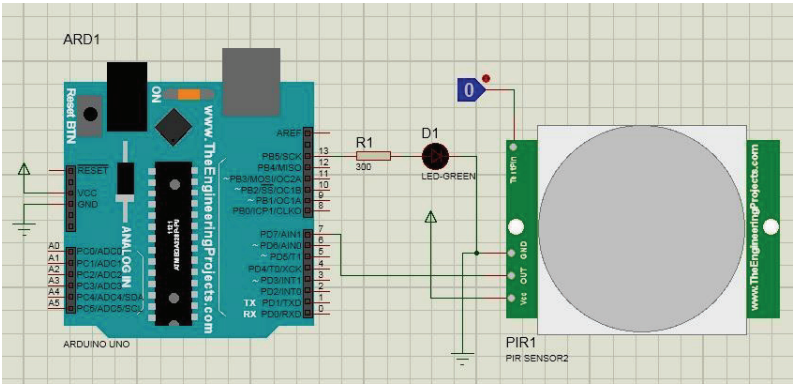


Рис. 2.37.

В схемі використовується модель контролера Arduino UNO та модель інфрачервоного датчика руху HC-SR501. Моделі для Proteus можна знайти на сайті за посиланням:

https://www.theengineeringprojects.com/2015/12/arduino-library-proteus-simulation.html

Завантаживши з сйту заархівовані файли моделей для Proteus, архіви потрібно розпакувати, і всі файли, які в них містилися, скопіювати в каталог LIBRARY, який, в свою чергу, розташований в каталозі, в який було встановлено Protues. За замовчуванням шлях до цього каталогу наступний: C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY

Далі, запустивши середовище моделювання Proteus, у вікні додавання компонента потрібно виділити категорію Arduino і розмістити елементи на робочому полі. На рис. 2.38 показане вікно додавання компонента в схему.

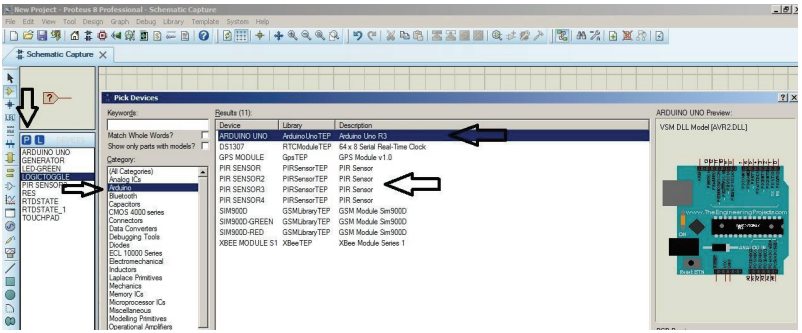


Рис. 2.38.

Також для моделювання роботи схеми потрібен світлодіод LED-GREEN, який буде загорятися при спрацюванні датчика, та струмообмежуючий резистор. Для моделювання виявлення руху модель датчика має додатковий вхід TestPin. Сигнал логічної одиниці на цьому вході призводить до спрацювання датчика. Для моделювання цього процесу в схемі використано елемент LOGICTOGGLE, який формує на виході лог.1 при натисканні на ньому при моделюванні роботи схеми в режимі реального часу. Також у вікні властивостей моделі інфрачервоного датчика руху потрібно вказати шлях до файлу PIRSensorTEP.hex, який також містився разом з файлом моделі в архіві при скачуванні. На рис. 2.39 показане вікно властивостей датчика руху. Шлях до файлу PIRSensorTEP.hex потрібно ввести в поле Program File.

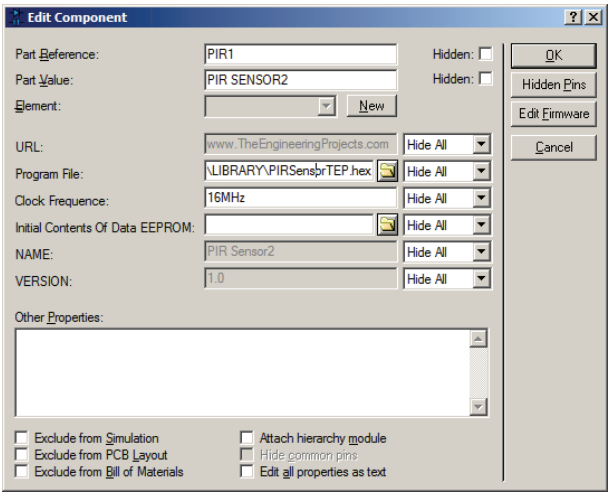


Рис. 2.39.

Скетч для перевірки роботи схеми представлено нижче.

```
const int movPin = 7; // номер виводу, до якого підключений датчик руху
const int ledPin = 13; // номер виводу, до якого підключений світлодіод

void setup() {
   pinMode(movPin, INPUT);          // налаштування виводу номер 7 на вхід
   pinMode(ledPin, OUTPUT);         // налаштування виводу номер 13 на вихід
}

void loop(){
   int value = digitalRead(movPin); // зчитування значення на вході 7 в змінну
   digitalWrite(ledPin, value); // запис вмісту змінної value на вихід 13
   delay(100);  // часова затримка 100 мс
}
```

Для того, щоб після компіляції проекту було створено файл hex, в якому міститься «прошивка» мікроконтролера, у середовищі розробки програмного забезпечення Arduino IDE потрібно в меню Файл вибрати пункт налаштування. У вікні, що з’явиться, необхідно встановити позначку Компіляція, як показано на рис. 2.40.

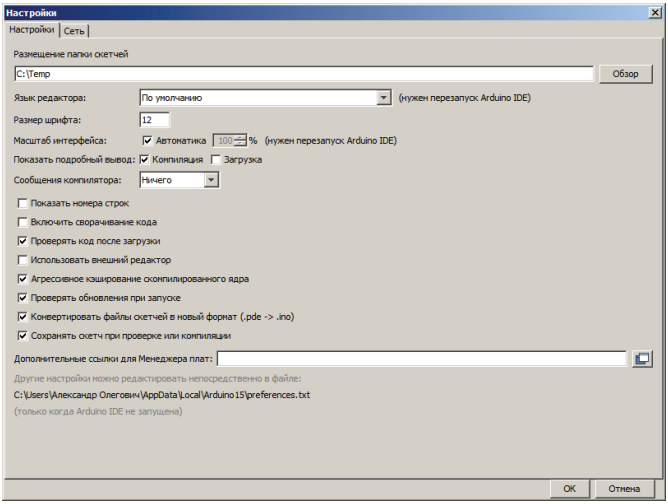


Рис. 2.40.

Тепер після компіляції проекту буде створено файл з розширенням hex. Шлях до цього файлу можна знайти у вікні повідомлень середовища Arduino (рис. 2.41). Шлях до сього файлу потрібно скопіювати в буфер обміну та вставити у поле Program File у вікні властивостей моделі контролера Arduino в Proteus, як показано на рис. 2.42.

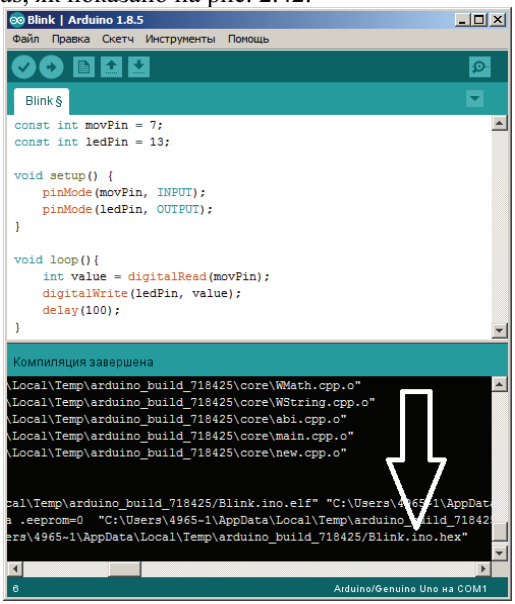


Рис. 2.41.

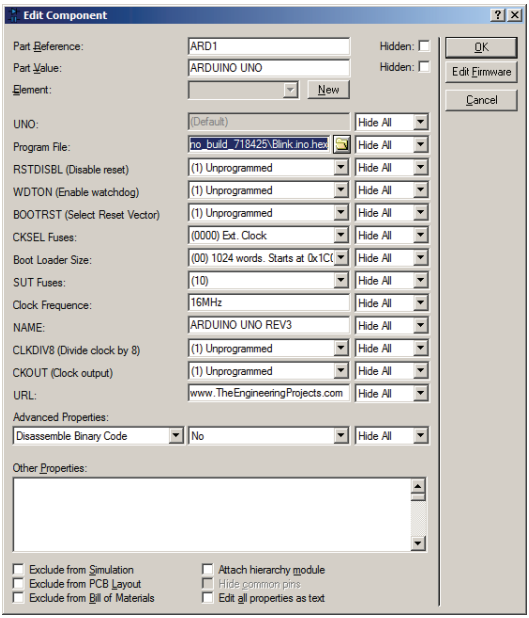


Рис. 2.42.

Запустивши процес моделювання можна перевірити роботу схеми та програми. Натиснувши один раз на елементі LOGICTOGGLE датчик спрацьовує, і на його виході з'явиться сигнал високого рівня. Цей сигнал буде зчитаний контролером та виведений на вивід, до якого підключений світлодіод. Тому світлодіод змінить свій стан і загориться. Через деякий проміжок часу світлодіод згасне. Рис. 2.43 демонструє схему в режимі моделювання.

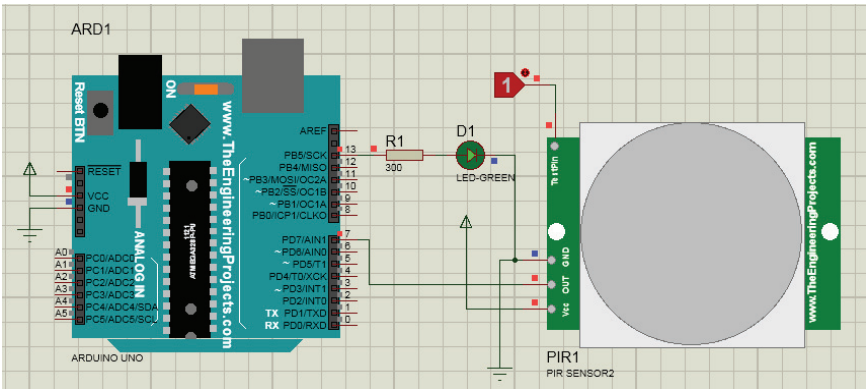


Рис. 2.43.

Цей проект можна розвинути, зробивши систему автоматичного вмикання світла при виявленні руху. Оскільки лампа розжарювання живиться від 220 В, тому для керування нею використовується модуль реле. При виявленні руху за допомогою інфрачервоного датчика контролер Arduino ввімкне реле, що призведе до вмикання лампи розжарювання. Через деякий час, при відсутності руху в полі зору датчика, на його виході з'явиться сигнал низького рівня, і контролер Arduino вимкне реле, і, відповідно, освітлення.

Скетч для системи автоматичного керування освітленням при виявлені руху представлено нижче.

```
const int sensorPin = 2; // номер виводу, до якого підключено датчик руху
const int relayPin = 3; // номер виводу, до якого підключено реле

void setup() {
  pinMode(sensorPin, INPUT); // налаштування виводу номер  2 на вхід
  pinMode(relayPin, OUTPUT); // налаштування виводу номер 3 на вихід
}

void loop(){
  int value = digitalRead(sensorPin); // зчитування дискретного сигналу з датчика
  if (value == HIGH) // якщо датчик формує високий рівень (є рух)
    digitalWrite(relayPin, HIGH);   // увімкнути реле
  else                              // інакше
    digitalWrite(relayPin, LOW);   // вимкнути реле
}
```

Для перевірки роботи програми в Proteus потрібно зібрати схему, представлену на рис. 2.44.

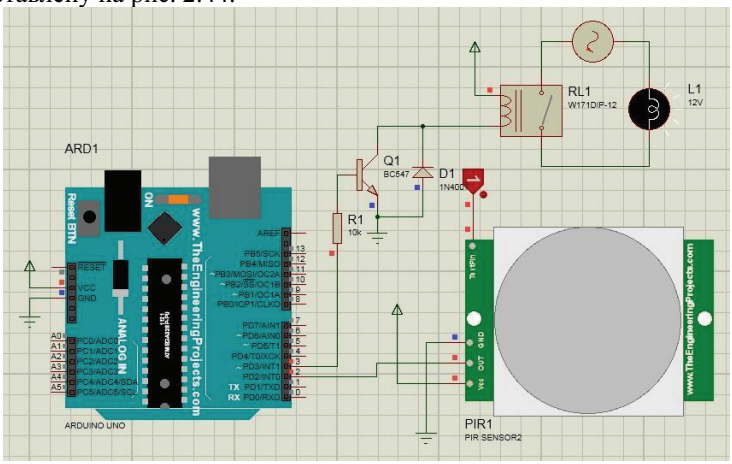


Рис. 2.44.

Запустивши процес моделювання роботи схеми і сформувавши високий рівень сигналу на вході TestPin датчика руху можна побачити, як лампа загориться. Моделювання дозволяє зробити висновки про коректність роботи схеми та програми.

2.6.2. Моделювання роботи ультразвукового датчика відстані HC-SR04

Ультразвуковий датчик HC-SR04, може служити датчиком відстані для робота, завдяки якому він зможе визначати дистанцію до об'єктів, об'їжджати перешкоди, або будувати карту приміщення. Його можна також використовувати в якості датчика для сигналізації, що спрацьовує при наближенні об'єктів. В даному прикладі ультразвуковий датчик HC-SR04 визначає відстань і виводить отримані значення у вікно серійного монітора в середовищі Arduino IDE. Якщо відстань до перешкоди менше 50 см, загоряється світлодіод. Основна мета даного прикладу – допомогти зрозуміти принцип дії датчика для подальшого використання в більш складних проектах. На рис. 2.45 представлена принципова схема системи вимірювання відстані до перешкоди.

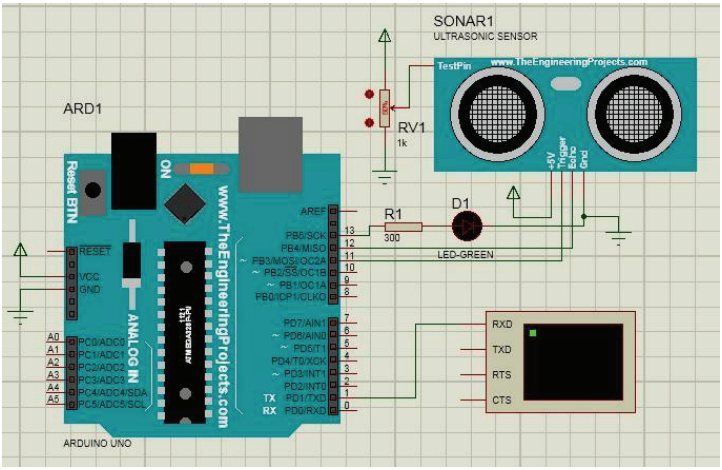


Рис. 2.45.

Текст програми представлено нижче.

```
int trigPin = 11;    // номер виводу, до якого підключено вхід Trig датчика
int echoPin = 12;  // номер виводу, до якого підключено вихід датчика
int ledPin = 13;    // номер виводу, до якого підключено світлодіод
// оголошення додаткових змінних для зберігання даних про виміряну відстань
long duration, cm, inches;

// у цій функції відбувається налаштування портів та швидкості передачі
// даних по послідовному інтерфейсу – ініціалізація контролера
```

```
void setup() {
Serial.begin (9600); // налаштування швидкості передачі даних 9600 біт/с
pinMode(trigPin, OUTPUT); // вивід номер 11 налаштовується на вихід
pinMode(echoPin, INPUT); // вивід номер 12 налаштовується на вхід
pinMode(ledPin, OUTPUT); // вивід номер 13 налаштовується на вихід
}
// в цій функції реалізовано основний алгоритм роботи програми
void loop()
{
// Датчик спрацьовує і генерує імпульси шириною 10 мкс або більше
// Генерується короткий LOW імпульс, щоб забезпечити імпульс HIGH:
digitalWrite(trigPin, LOW); // встановлення низького рівня на виході 11
delayMicroseconds(5); // формування часової затримки тривалістю 5 мкс
digitalWrite(trigPin, HIGH); // встановлення високого рівня на виході 11
delayMicroseconds(10); // формування часової затримки тривалістю 10 мкс
digitalWrite(trigPin, LOW); // встановлення низького рівня на виході 11
// Зчитуємо дані з ультразвукового датчика: значення HIGH, яке
// залежить від тривалості (в мікросекундах) між відправленням
// акустичної хвилі і її прийманням на локаторі відлуння.
pinMode(echoPin, INPUT); // вивід номер 12 налаштовується на вхід
duration = pulseIn(echoPin, HIGH); // зчитування тривалості сигналу на вході 12
cm = (duration/2) / 29.1; // перетворення значення виміряного часу у сантиметри
inches = (duration/2) / 74; // перетворення значення виміряного часу у дюйми
if(cm<50) // якщо відстань менше 50 см
{
 digitalWrite(ledPin, HIGH);  // встановити високий рівень на виході 13
}
else                              // інакше
{
 digitalWrite(ledPin, LOW);  // встановити низький рівень на виході 13
}
Serial.print(inches);    // передавання по послідовному порту значення дюймів
Serial.print("in, ");    // передавання текстового рядка «in»
Serial.print(cm);        // передавання по послідовному порту значення сантиметрів
Serial.print("cm");      // передавання текстового рядка «cm»
Serial.println();        // передавання символу переходу на новий рядок
delay(250);              // часова затримка 250 мс
}
```

Після компіляції програми можна промоделювати роботу схеми. На рис. 2.46 представлена схема в режимі моделювання. За допомогою віртуального термінала можна спостерігати значення відстані, що вимірюється датчиком. Модель датчика має додатковий вхід TestPin. Змінюючи напругу, яка подається на цей вхід, можна моделювати процес вимірювання датчиком різних відстаней. В даному випадку найпростіший спосіб змінювати напругу на цьому вході – це використовувати змінний опір. Якщо відстань стане менше 50 см, то загориться світлодіод.

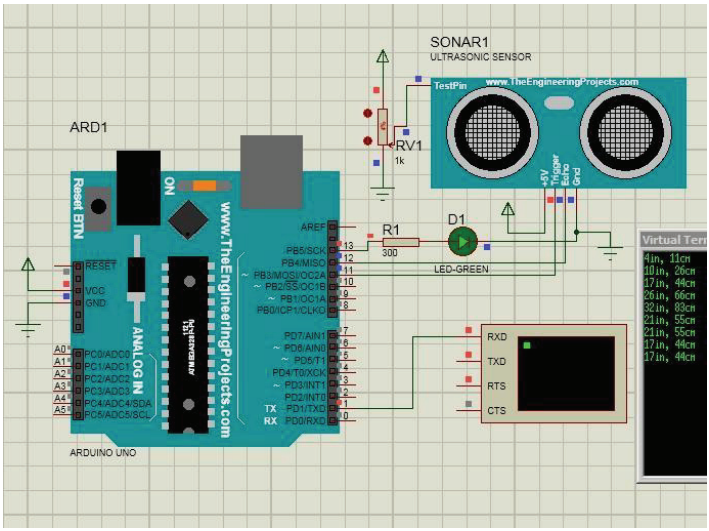


Рис. 2.46.

В програмі для вимірювання тривалості імпульсу було використано функцію pulseIn(). Ця функція зчитує довжину сигналу на заданому порту (HIGH або LOW). Наприклад, якщо виконується зчитування HIGH функцією pulseIn(), функція чекає поки на заданому порту не з'явиться високий рівень сигналу HIGH. Коли HIGH отримано, включається таймер, який буде зупинений коли на порту входу/виходу буде низький рівень сигналу LOW. Функція pulseIn() повертає довжину сигналу в мікросекундах. Функція повертає 0, якщо протягом заданого часу (таймаута) сигнал на відповідному виводі ні був зафіксований.

При використанні функції можливі деякі похибки в вимірювання довгих сигналів. Функція може вимірювати сигнали довжиною від 10 мікросекунд до 3 хвилин. Функція має наступний синтаксис:

```
pulseIn(pin, value);
pulseIn(pin, value, timeout);
```

Параметри, що передаються в функцію: pin – номер виводу, на якому буде очікуватися сигнал; value – тип очікуваного сигналу – HIGH або LOW; timeout (опціональний параметр) – час очікування сигналу (таймаут) в мікросекундах (за замовчуванням одна секунда).

Значення, що повертається – тривалість сигналу в мікросекундах або 0, якщо сигнал не був отриманий до закінчення часу очікування.

2.6.3. Моделювання мікропроцесорної системи з датчиком освітленості

Модуль датчика освітленості призначений для використання в автоматиці, яка керує включенням освітлення, або в приладах, що контролюють рівень освітленості. Світлочутливий елемент датчика – фоторезистор, який виступає за плату. При зміні освітленості від яскравого світла до темряви його опір змінюється від сотень Ом до декількох МОм. Це фіксує електроніка модуля і змінює стан виходів. Вихідний струм до 15 мА на контакті D0 (ТТЛ) дозволяє використовувати датчик освітленості спільно з модулем реле, який має на вході елементи управління реле і не потребують великих вхідних струмів керування.

Наступний приклад демонструє використання датчика освітленості для керування яскравістю світлодіода. Принципова схема пристрою представлена на рис. 2.47.

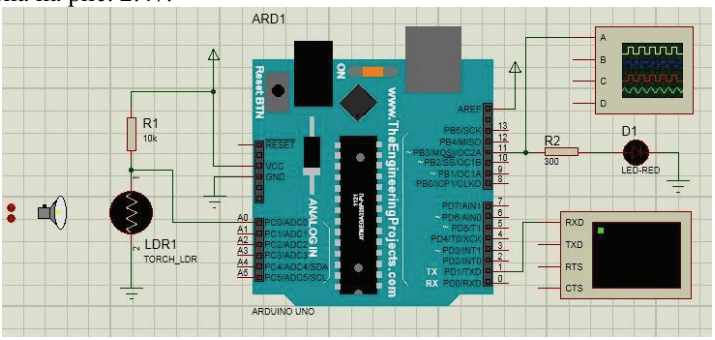


Рис. 2.47.

У цьому скетчі відбувається зчитування аналогового значення з виходу датчика освітленості для керування яскравістю світлодіода. Чим менша освітленість датчика, тим яскравіше буде світити світлодіод. Текст програми представлено нижче.

```
int photocellPin = A0; // датчик освітленості підключений до виходу A0 контролера
int photocellReading; // оголошення змінної для зчитування в неї даних з датчика
int LEDpin = 11; // номер виводу, до якого підключений світлодіод
int LEDbrightness; // оголошення змінної, в якій зберігатиметься значення яскравості
void setup(void) {
// налаштування швидкості передачі даних по послідовному інтерфейсу
// інформація передається з контролера в комп'ютер на швидкості 9600 біт/с
Serial.begin(9600);
}

void loop(void) {
photocellReading = analogRead(photocellPin); // зчитування даних з виходу датчика
Serial.print("Analog reading = "); // відправлення текстового рядка в комп'ютер
Serial.println(photocellReading); // відправлення виміряних значень в комп'ютер
```

```
// світлодіод світить яскравіше якщо рівень освітленості на датчику зменшується
// це означає, що потрібно інвертувати зчитане значення від 0-1023 до 1023-0
photocellReading = 1023 - photocellReading;
// тепер потрібно перетворити діапазон 0-1023 в 0-255, оскільки саме такий діапазон
// використовує функція analogWrite
LEDbrightness = map(photocellReading, 0, 1023, 0, 255);
analogWrite(LEDpin, LEDbrightness); // встановлення значення яскравості світлодіода
delay(100);      // часова затримка 100 мс
}
```

Після компіляції програми і запуску схеми на моделювання у вікні віртуального монітора можна бачити інформацію про поточне виміряне значення освітленості, а за допомогою осцилографа спостерігати ШІМ-сигнал керування яскравістю світлодіода. На рис. 2.48 представлена схема в режимі моделювання.

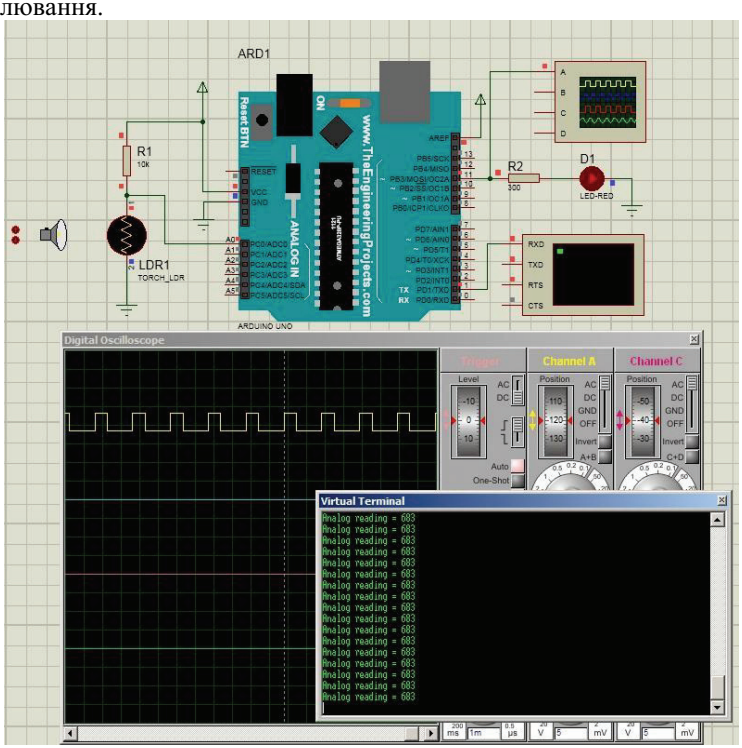


Рис. 2.48.

Фоторезистори досить активно застосовуються в найрізноманітніших системах. Найпоширеніший варіант застосування – ліхтарі вуличного освітлення. Якщо на вулиці стає темно, то вогні включаються автоматично. з Використовуючи фоторезистор можна зробити економну лампочку для будинку, що включається не за розкладом, а в залежності від освітлення. На

базі датчика освітленості можна зробити навіть охоронну систему, яка буде спрацьовувати одразу після того, як закриту шафу або сейф відкрили і на датчик падає світло.

2.7. Покрокове моделювання та налагодження програм в AVR Studio

При старті AVR Studio відразу відкривається вікно майстра створення нового проекту. Натиснувши на кнопку Open і вказавши шлях до файлу з розширенням COFF, з'явиться вікно, показане на рис. 2.49. У цьому вікні ліворуч необхідно вибрати обладнання, з яким буде виконуватися робота. Для моделювання та налагодження програм необхідно вибрати пункт AVR Simulator і вказати модель мікроконтролера, для якого була написана програма.

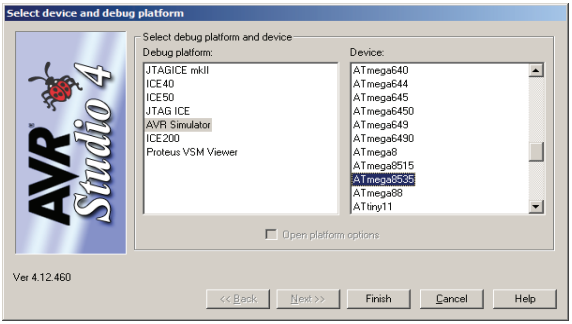


Рис. 2.49.

Після натискання на кнопку Finish з'являється головне вікно програми (рис. 2.50).

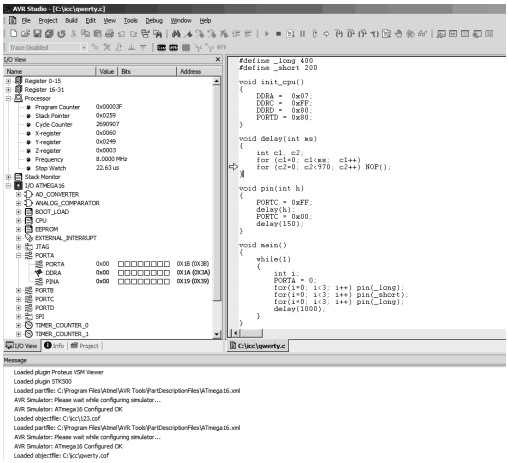


Рис. 2.50.

AVR Studio має дуже гнучкий інтерфейс, і зовнішній вигляд може сильно відрізнятися від варіанту, як показано на малюнку. Головна панель програми AVR Studio розділена на три основних вікна. Зліва розташоване вікно ресурсів мікроконтролера. Воно надає повну інформацію про поточний проект. За замовчуванням це вікно містить в собі три вкладки. Перша вкладка називається «Info». Вона містить довідкову інформацію про мікроконтролер, що використовується, а саме опис векторів переривань, опис виводів для різних корпусів та короткий опис регістрів.

Наступна вкладка називається «Project». Вона містить повну інформацію про поточний проект. Інформація представлена у вигляді дерева. Різні гілки цього дерева описують всі вихідні і результуючі файли проекту, всі мітки, процедури і файли.

Остання вкладка вікна називається «I/O View» (перегляд вводу/виведення). Це найкорисніша вкладка. На ній можна побачити, що відбувається з мікроконтролером: стан біт регістрів керування периферійними пристроями, регістрів порту, регістрів конфігурації. Вся інформація також представлена у вигляді дерева. Кожна гілка цього дерева – це окремий елемент. Якщо який-небудь елемент складається з інших елементів, то його можна розкрити і побачити ці елементи.

Якщо яка-небудь гілка може бути розкрита, то в своєму корені вона має квадратик з «+» всередині. Подвійне натискання на цій позначці розкриває гілку. Повторне подвійне натискання закриває розкриту гілку.

У центрі розташоване вікно з кодом програми. Внизу, під вікном коду, вікно повідомлень компілятора. Всі помилки, попередження, а також інша інформація, відображаються у цьому вікні.

На рис. 2.51 для наочності розкрито гілки всіх портів вводу/виводу і регістри, пов’язані з EEPROM. На рисунку можна бачити:

– повний склад керуючих регістрів для кожного з пристроїв;

– їх назви та адреси;

– склад та назву кожного біту (якщо біти мають свої назви).

У процесі налагодження в цьому вікні можна побачити не тільки назву і склад всіх ресурсів, але і їх вміст. Вміст буде відображатися як у шістнадцятковому вигляді, так і шляхом затемнення квадратиків, що відображають окремі біти конкретних регістрів.

Затемнений квадратик означає, що біт дорівнює одиниці. Світлий квадратик говорить про те, що біт дорівнює нулю. Також можна оперативно змінювати цей вміст прямо в цьому вікні. Для зміни значення біта досить натискання лівою кнопкою миші у відповідному квадратику.

Вікно порту має три різні регістри вводу-виводу (Port – регістр даних, Pin – ніжки порту, DDR – Data Direction Registers (регістри напрямку передачі даних)), які звичайно зв’язані з портом. Ніжки порту використовуються тільки для зчитування, в той час як регістр даних і регістри напрямку передачі даних – для зчитування/запису.

Pin не являється регістром, по цьому адресу здійснюється доступ до фізичних значень кожного із виводів порту. При зчитуванні з регістру даних Port, зчитуються дані з «регістру-защіпку», при зчитуванні з Pin зчитуються логічні значення, які знаходяться на виводах порту. Біти регістру DDR вибирають напрямок передачі даних. Якщо біт встановлений (1), вивід сконфігурований як вихід. Якщо біт скинутий (0) – вивід сконфігурований як вхід.

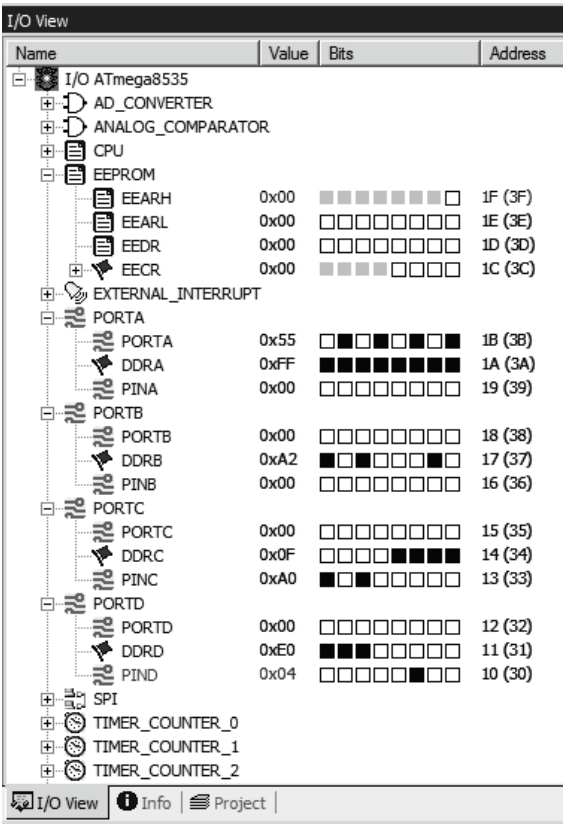


Рис. 2.51.

В центральному вікні вдображається код програми. Кожна наступна команда, яка буде виконуватись, вдмчена жовтим маркером. Можна вдкрити вкно регстрв (меню View → Registers), вкно для спостереження за змнними (View → Processor), вкно пам'яті (View → Memory). Вкно регстрв складається з 32-х регстрв і має вигляд, як показано на рис. 2.52.

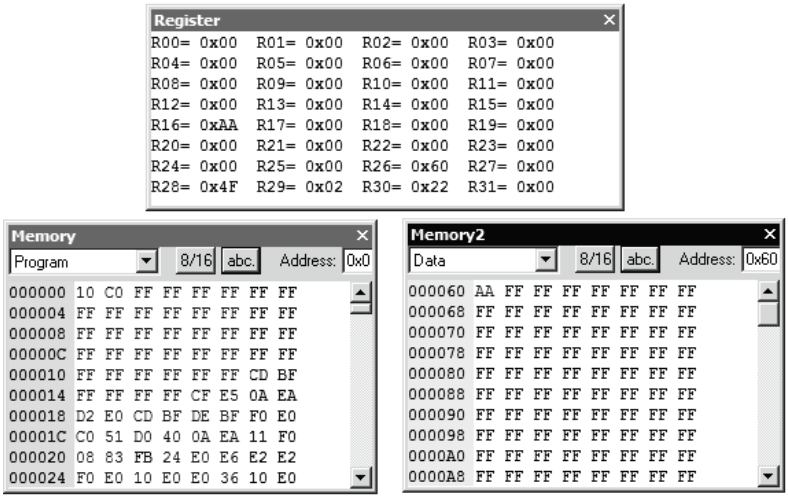


Рис. 2.52.

Вміст регістрів можуть змінюватись після виконання команди. Значення регістрів можна змінювати самостійно. Для цього потрібно поставити курсор на значення регістру і нажати двічі ліву кнопку миші з невеликою паузою між натисканням кнопки. Після цього вписати нове значення регістру у шістнадцятирічному коді.

Вікно пам'яті використовується користувачем для огляду і модифікації змісту змінних пам'яті, які використовуються у мітках. Одне й теж вікно використовується для огляду різних типів пам'яті: пам'яті даних, програмної пам'яті, пам'яті вводу-виводу і енергонезалежної пам'яті EEPROM. Тип пам'яті, який потрібно продивитись, може бути змінений в меню вибору пам'яті, яке знаходиться в верхньому лівому куті вікна пам'яті. Коли створюється нове вікно пам'яті, пам'ять даних (Data memory) створює стандартний тип пам'яті.

Адресація і зміст пам'яті завжди відображаються в 16-річному коді. Користувач може увімкнути огляд змісту пам'яті в символах ASCII, а також продивитись зміст пам'яті в 8-річному коді. Користувач може змінити зміст пам'яті подвійним натисканням кнопки миші (double click).

Вікно процесора вказує всю необхідну інформацію про команду, яка виконується в даний момент (рис. 2.53).

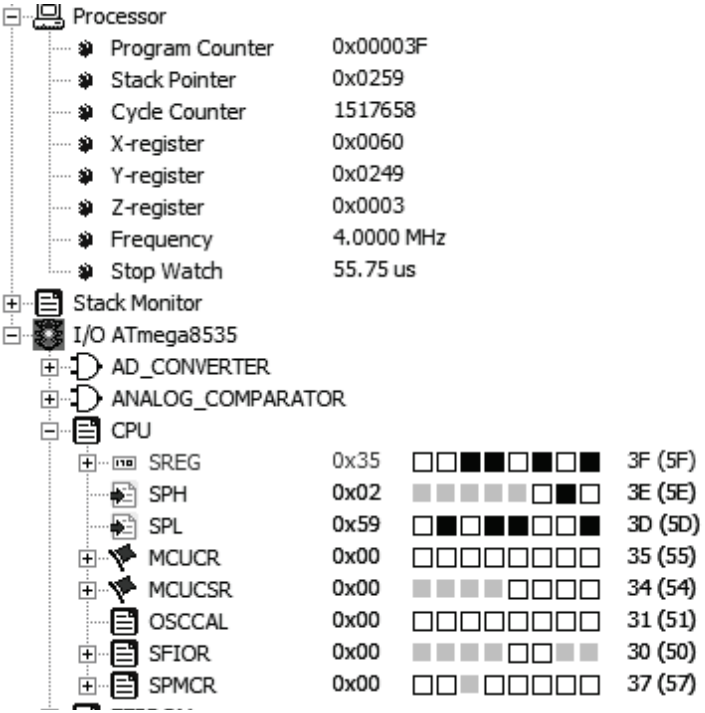


Рис. 2.53.

Програмний лічильник (Program Counter) вказує адресу команди, яка буде виконуватись. Він вказує адрес в 16-річному коді і цей код може бути змінений після зупинки команди. Коли значення програмного лічильника змінюється, поточна команда відкидається. Після зміни значення програмного лічильника програма переходить на наступну адресу.

Стековий вказівник (Stack Pointer) зберігає поточне значення стекового вказівника, які розташовані в області вводу-виводу. Якщо мітка має апаратний стек замість базового стеку SRAM, вона вказується в полі стекового вказівника. Значення стекового вказівника може бути змінено після того, як зупинено виконання команди.

Лічильник циклів (Cycle Counter) дає інформацію про кількість циклів тактового генератора, які пройшли після останнього скидання. Значення лічильника циклів записується в десятковому коді і може бути змінений після зупинки команди.

Прапорці (Flags) показують поточне значення бітів статус регістру (Status register). Коли виконання команди зупинено, ці біти можуть бути змінені натисканням кнопки миші на визначений біт. Відмічений прапорець означає, що відповідний біт статус регістру має значення 1.

Команди керування виконанням програми використовуються для керування виконанням програми. Усі вони мають меню, так називаємі гарячі кнопки виклику і панель наладки (Debug toolbar).

Команда **Go** (пуск)викликається в меню Debug→Go(F5), або на панелі наладки натисканням кнопки . Вона запускає виконання програми. Виконується до тих пір, доки користувач не зупинить виконання або доки не дійде до крапки переривання. Команда **Break** (зупинити) викликається в меню Debug→Break(Ctrl+F5), або на панелі наладки натисканням кнопки . Зупиняє виконання програми. Після зупинки програми вся інформація в усіх вікнах відновлюється.

Команда **Trace info** (відслідкувати) викликається в меню Debug→Trace info (F11), або на панелі наладки натисканням . Ця команда виконує одну команду програми. За допомогою неї можна по командам перевірити роботу програми.

Команда **Step over** викликається в меню Debug→Step Over (F10), або на панелі наладки натисканням кнопки . При виклиці цієї команди програма буде виконуватись до тих пір, доки не дійде до команди виклику підпрограми чи функції (якщо така є), також це стосується крапок переривання.

Команда **Step out** викликається на панелі наладки натисканням кнопки . Якщо курсор знаходиться в функції чи підпрограмі, то при натисканні даної команди програма буде виконуватись доти, доки не дійде до команди виходу з функції чи підпрограми. Це також стосується крапок переривання.

Команда **Goto cursor** викликається в меню Debug →Run to cursor(F7), або на панелі наладки натисканням кнопки . Якщо в програмі в якомусь місці поставити курсор і викликати цю команду, то програма виконується доти, доки не дійде до курсору. Команда припинить виконуватись, коли також дійде до крапки переривання.

Команда **Reset** викликається в меню Debug→Reset (Shift F5), або на панелі наладки натисканням кнопки . При виклиці цієї команди курсор переходить на начало програми, значення статус-регістру, портів, процесору, таймера-лічильника обнуляються.

Для вибору контролера заходимо в меню Debug →AVR Simulator Option (рис.2.54).

В меню Device вибираємо потрібний контролер. В меню Frequency задаємо частоту роботи мікроконтролера. У вікні праворуч можна побачити характеристики вибраного мікроконтролера та максимальну частоту його роботи.

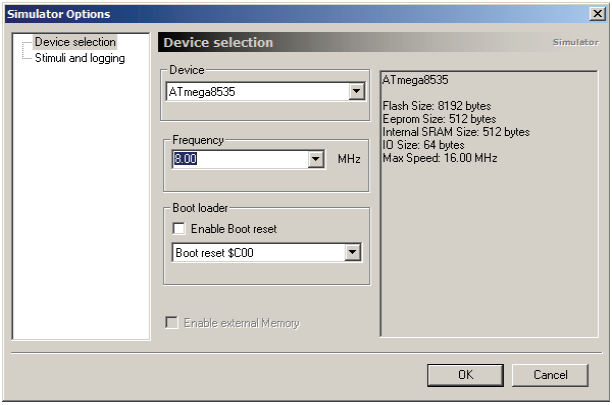


Рис. 2.54.

2.8. Моделювання алгоритмів цифрової обробки сигналів

В даному параграфі розглядаються питання створення алгоритмів цифрової обробки сигналів на прикладі цифрових фільтрів для мікроконтролерів STM32 та моделювання результатів їх роботи у середовищі проектування Keil uVision. Для розуміння матеріалу цього параметра читач має бути знайомим з архітектурою мікроконтролерів ARM, бібліотекою HAL, середовищем розробки CubeMX, та мати базові знання з цифрової обробки сигналів.

Для створення нового проекту у середовищі Keil uVision необхідно в меню Project вибрати команду New uVision Project. Відкриється стандартне вікно, к якому потрібно вказати шлях до каталогу, в якому буде збережено файли проекту. В даному прикладі на диску було створено каталог з назвою noise_remover та вказано шлях до нього. Після цього у новому вікні, що відкриється, потрібно вказати модель мікроконтролера, для якого виконується розробка програми. В цьому прикладі використовується мікроконтролер STM32F411RE, який встановлено на платі Nucleo. Далі, у вікні що відкриється (рис. 2.55), необхідно підключити деякі бібліотеки: CMSIS CORE, CMSIS DSP, Device Startup, STM32Cube Framework (API) STM32CubeMX. Якщо деякі пункти будуть виділені жовтим кольором, слід натиснути на кнопку Resolve.

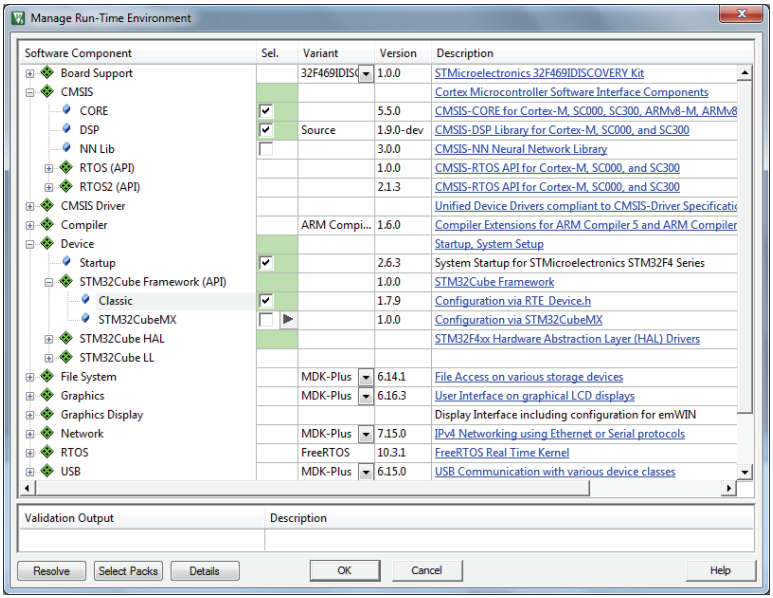


Рис. 2.55.

Натиснувши на кнопку OK відкриється середовище розробки. Відразу можна перейменувати вузли дерева проекту так, як показано на рис. 2.56.

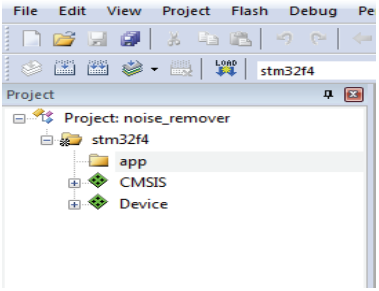


Рис. 2.56.

В процесі розробки програмного забезпечення може знадобитися встановити нові пакети. Зробити це можна відкривши вікно Pack Installer шляхом натискання на відповідну кнопку на панелі інструментів (рис. 2.57).

Рис. 2.57.

У вікно Pack Installer (рис. 2.58) в правій частині відображено список пакетів, які вже встановлено або які можна встановити. Наприклад, для того, щоб встановити пакет для роботи з операційною системою реального часу Free RTOS, навпроти цієї назви потрібно натиснути на кнопку Install. На рис. 2.58 показано, що цей пакет вже встановлено, тому кнопка неактивна з написом Up to Date.

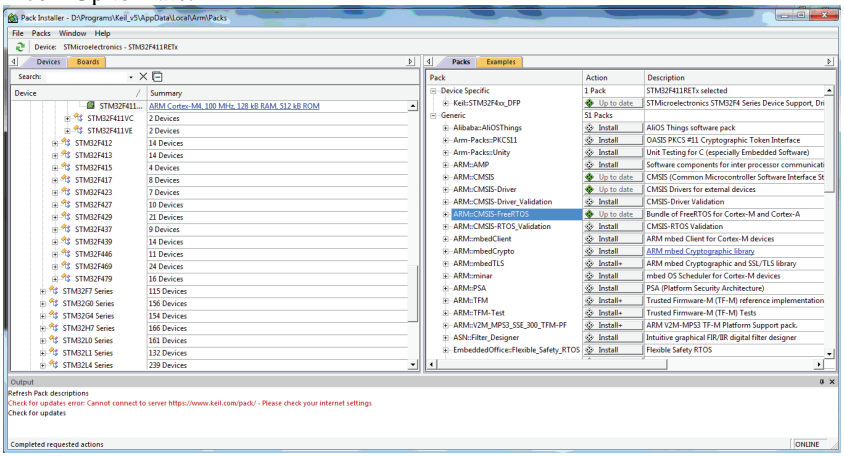


Рис. 2.58.

Для реалізації алгоритмів цифрової обробки сигналів необхідно, щоб мікроконтролер працював на високій частоті. Налаштувати частоту тактових сигналів можна за допомогою середовища CubeMX. Створивши новий проект для мікроконтролера STM32F411RE (рис. 2.59), можна перейти до налаштування частоти тактового сигналу.

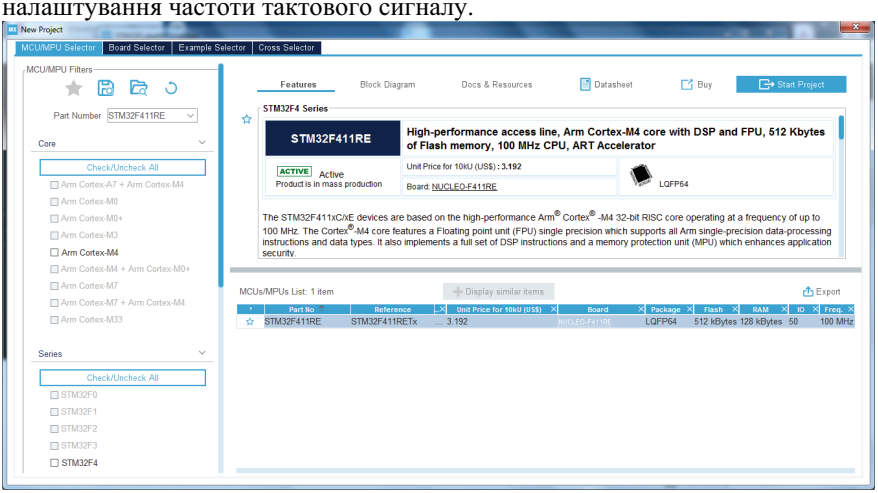
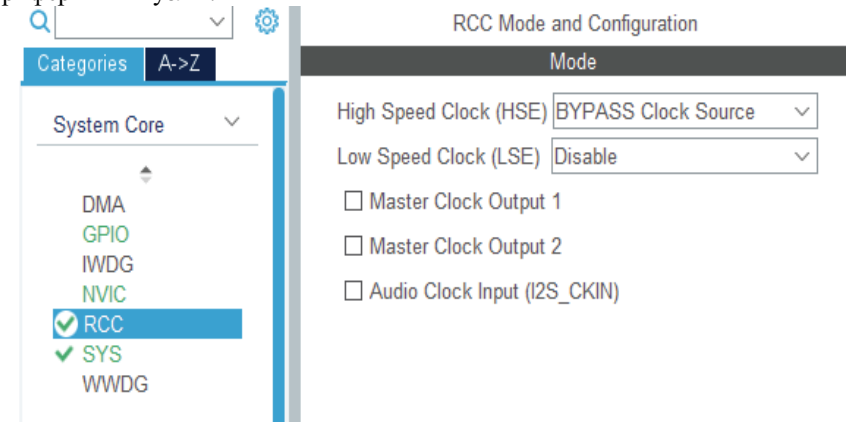
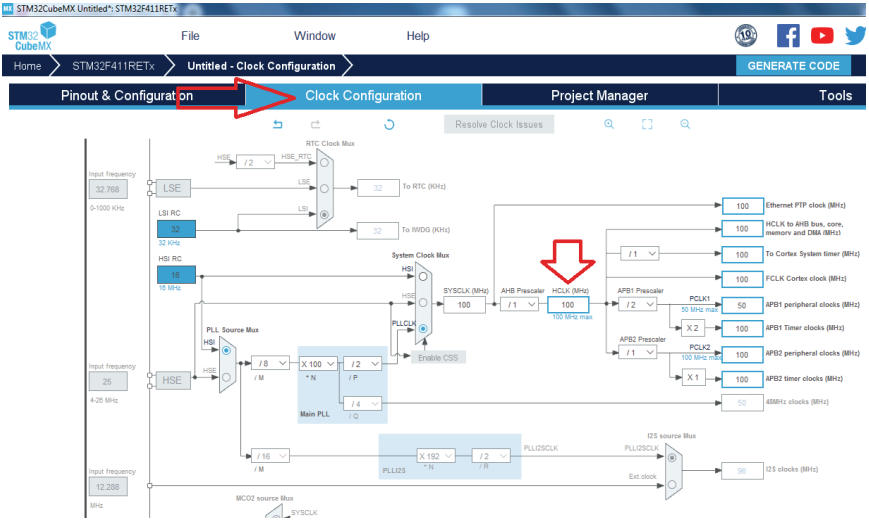


Рис. 2.59.

На вкладці Clock Configuration у полі HCLK потрібно ввести значення бажано частоти, як показано на рис. 2.60. В даному випадку встановлено максимальне значення 100 МГц. Натиснувши Enter програмне середовище автоматично змінить значення частот тактових сигналів для всіх інших периферійних вузлів.



а



б

Рис. 2.60.

Тепер можна згенерувати код для середовища Keil uVision. Для цього на вкладці Project Manager потрібно виконати налаштування, як показано на рис. 2.61, та натиснути на кнопку Generate Code.

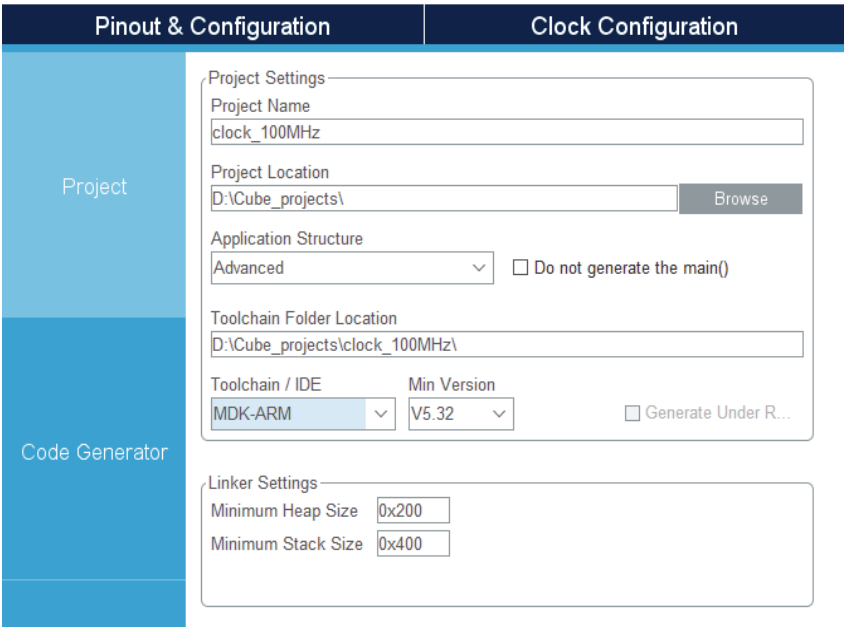


Рис. 2.61.

Відкривши проект у середовищі Keil uVision можна перевірити значення частоти тактового сигналу. Для цього потрібно оголосити глобальну змінну, та викликати функцію, що повертає значення частоти, і записати це значення в змінну. Наступний фрагмент коду (файл main.c) демонструє це.

```
uint32_t freq;

int main(void)
{
 HAL_Init();
 SystemClock_Config();

        freq = HAL_RCC_GetHCLKFreq();

 while (1)
 {
 }
}
```

Коментарі, що автоматично були створені середовищем CubeMX в наведеному фрагменті коду видалені. Підключивши плату Nucleo до компью-

тера, відкомпілювавши проект та запустивши відлагоджувач (Debugger) можна побачити значення, що зберігається в цій змінній після виклику функції HAL_RCC_GetHCLKFreq().

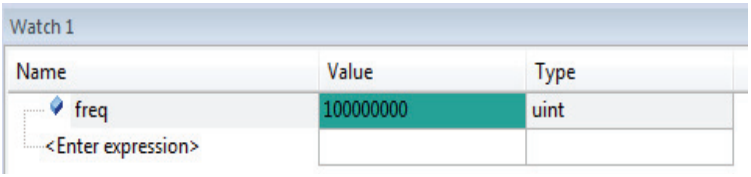


Рис. 2.62.

Тепер в новому проекті, що був створений раніше, потрібно додати файл вихідного коду main.c. Для цього викликавши контекстне меню для вузла app потрібно вибрати команду Add New Item to Group 'app'. У вікні, що з'явиться (рис. 2.63), вибрати тип файлу та вказати його назву.

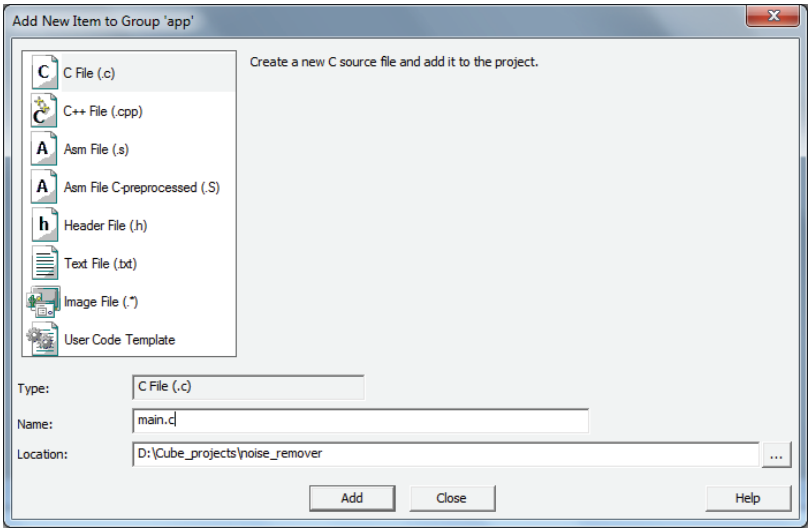


Рис. 2.63.

Тепер слід підключити заголовний файл. Спростити цей процес можна шляхом натискання правої кнопки миші в редакторі та виборі з відповідного меню потрібного заголовного файлу, як показано на рис. 2.64.

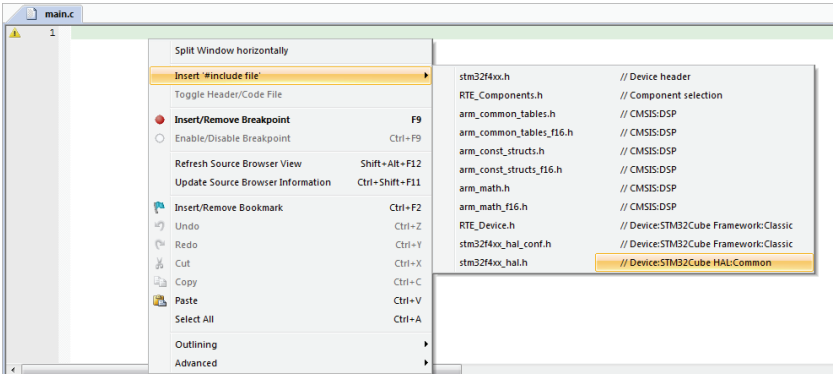


Рис. 2.64.

Код програми, що міститься у файлі main.c, представлено далі.

```
#include "stm32f4xx_hal.h"            // Keil::Device:STM32Cube HAL:Common

uint32_t freq;

void SystemClock_Config(void);
void SysTick_Handler(void);

int main(void)
{
 HAL_Init();

 SystemClock_Config();

freq = HAL_RCC_GetHCLKFreq();

 while (1)
 {
 }
}

void SysTick_Handler(void)
{
 HAL_IncTick();
        HAL_SYSTICK_IRQHandler();
}

void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

 /** Configure the main internal regulator output voltage
 */
```

```
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    //Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                        |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
   // Error_Handler();
  }
}
```

Перед завантаженням програми в мікроконтролер слід впевнитися, що у вікні налаштувань (Options for Target) вибраний ST Link Debugger (рис. 2.65).

Таким чином можна використовувати CubeMX для пришвидшення процесу налаштування мікроконтролера.

Для налаштування логічного аналізатора, який доступний в режимі відлагодження (Debugger), у вікні Options for Target потрібно вибрати команду Debug Settings. Відкриється вікно, показане на рис. 2.66, в якому потрібно встановити позначку Trace Enable.

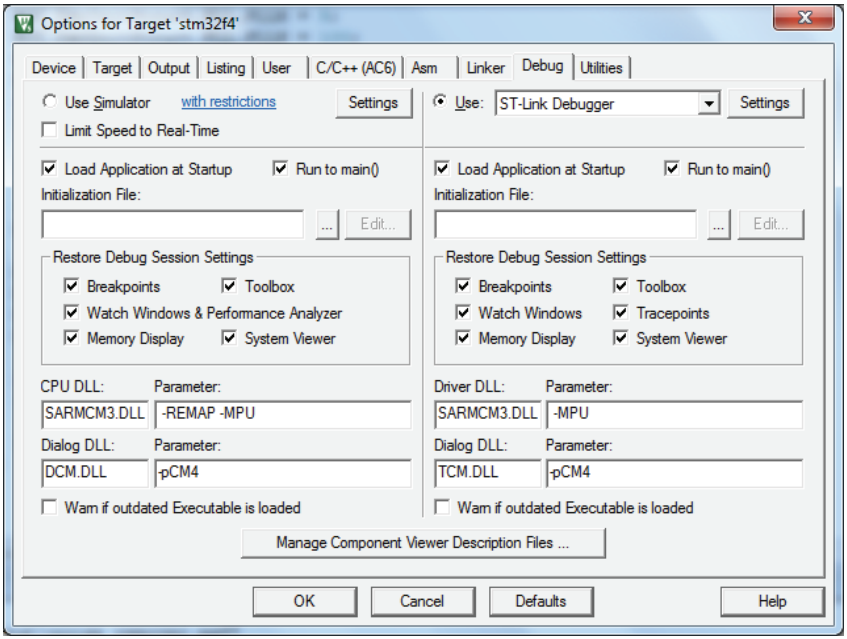


Рис. 2.65.

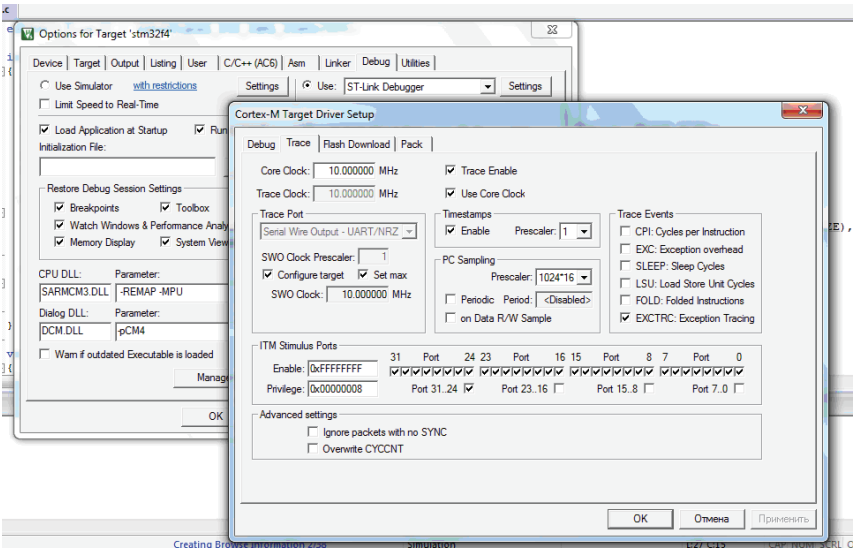


Рис. 2.66.

В цьому вікні слід впевнитися, що значення Core Clock відповідає частоті тактового сигналу, який було налаштовано – в даному випадку це 100 МГц. Це потрібно для того, щоб у вікні логічного аналізатора відображалися правильні значення часу.

Тепер в проект потрібно додати ще один файл з назвою waveform.c, в який скопіювати масив числових значень, що описують сигнал з шумом.

```
#include "arm_math.h"

float32_t inputSignal_f32_1kHz_15kHz[320] =
{
+0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
-0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
-0.8660254038f, -0.4619397663f, -1.3194792169f, -1.1827865776f, -0.5000000000f, -
1.1827865776f, -1.3194792169f, -0.4619397663f,
-0.8660254038f, -1.2552931065f, -0.3535533906f, -0.4174197128f, -1.0000000000f, -
0.1913417162f, +0.0947343455f, -0.5924659585f,
+0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
+0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
-0.8660254038f, -0.4619397663f, -1.3194792169f, -1.1827865776f, -0.5000000000f, -
1.1827865776f, -1.3194792169f, -0.4619397663f,
-0.8660254038f, -1.2552931065f, -0.3535533906f, -0.4174197128f, -1.0000000000f, -
0.1913417162f, +0.0947343455f, -0.5924659585f,
+0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
+0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
-0.8660254038f, -0.4619397663f, -1.3194792169f, -1.1827865776f, -0.5000000000f, -
1.1827865776f, -1.3194792169f, -0.4619397663f,
-0.8660254038f, -1.2552931065f, -0.3535533906f, -0.4174197128f, -1.0000000000f, -
0.1913417162f, +0.0947343455f, -0.5924659585f,
-0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
```

```
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
-0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
-0.8660254038f, -0.4619397663f, -1.3194792169f, -1.1827865776f, -0.5000000000f, -
1.1827865776f, -1.3194792169f, -0.4619397663f,
-0.8660254038f, -1.2552931065f, -0.3535533906f, -0.4174197128f, -1.0000000000f, -
0.1913417162f, +0.0947343455f, -0.5924659585f,
+0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
+0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
-0.8660254038f, -0.4619397663f, -1.3194792169f, -1.1827865776f, -0.5000000000f, -
1.1827865776f, -1.3194792169f, -0.4619397663f,
-0.8660254038f, -1.2552931065f, -0.3535533906f, -0.4174197128f, -1.0000000000f, -
0.1913417162f, +0.0947343455f, -0.5924659585f,
-0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
+0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
-0.8660254038f, -0.4619397663f, -1.3194792169f, -1.1827865776f, -0.5000000000f, -
1.1827865776f, -1.3194792169f, -0.4619397663f,
-0.8660254038f, -1.2552931065f, -0.3535533906f, -0.4174197128f, -1.0000000000f, -
0.1913417162f, +0.0947343455f, -0.5924659585f,
-0.0000000000f, +0.5924659585f, -0.0947343455f, +0.1913417162f, +1.0000000000f,
+0.4174197128f, +0.3535533906f, +1.2552931065f,
+0.8660254038f, +0.4619397663f, +1.3194792169f, +1.1827865776f, +0.5000000000f,
+1.1827865776f, +1.3194792169f, +0.4619397663f,
+0.8660254038f, +1.2552931065f, +0.3535533906f, +0.4174197128f, +1.0000000000f,
+0.1913417162f, -0.0947343455f, +0.5924659585f,
+0.0000000000f, -0.5924659585f, +0.0947343455f, -0.1913417162f, -1.0000000000f, -
0.4174197128f, -0.3535533906f, -1.2552931065f,
};
```

Для того, щоб цей масив був доступний у файлі main.c, потрібно додати ключове слово extern:

```
extern float32_t inputSignal_f32_1kHz_15kHz[320];
```

Також в файл main.c слід додати заголовний файл #include «arm_math.h».

У вікні налаштувань проекту потбібно вказати, яка саме реалізація бібліотеки буде використовуватися (в даному прикладі для мікроконтролерів з ядром Cortex M4, отже ARM_MATH_CM4). На рис. 2.67 показано відповідне вікно налаштувань.

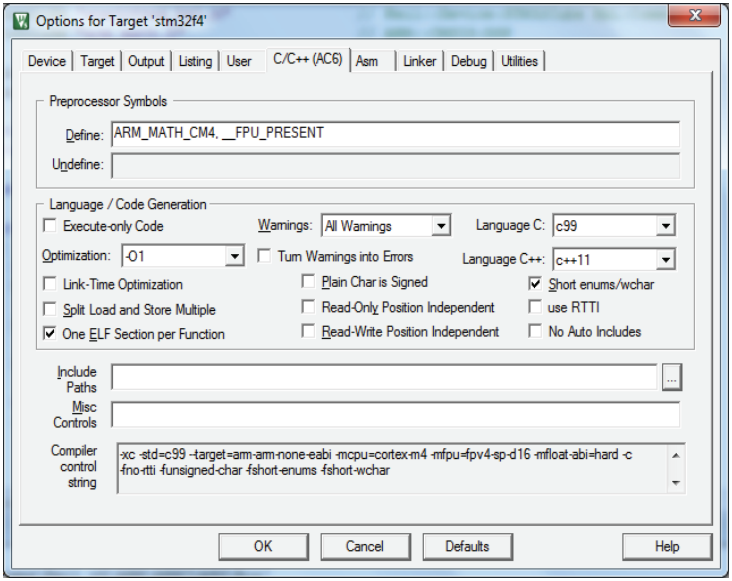


Рис. 2.67.

Вибраний для цього мікроконтролер підтримує операції з плавоючою комою на апаратному рівні, тому для оптимізації роботи також слід вказати параметр __FPU_PRESENT, як це зроблено у вікні налаштувань.

Фільтрація сигналу відбуватиметься з використанням цифрового нерекурсивного фільтру зі скінченною імпульсною характеристикою (СІХ-фільтру). Коефіцієнти фільтру були заздалегідь розраховані. На початку файлу main.c потрібно додати масив, що містить коефіцієнти цифрового фільтру:

```
const float32_t firCoeffs32[29] = {
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f,
+0.0085302217f, -0.0000000000f, -0.0173976984f,
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f,
+0.2229246956f, +0.2504960933f, +0.2229246956f,
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -
0.0173976984f, -0.0000000000f, +0.0085302217f,
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};
```

Для того, щоб зберігати отримані значення на виході фільтру (відфільтровани сигнал), потрібно також створити масив.

```
float32_t outputSig[320];
```

У вхідному сигналі міститься 320 значень, тому масив вихідних значень також містить 320 елементів. Для роботи фільтру також потрібен проміжний масив:

```
#define BLOCK_SIZE 32
float32_t firStateF32[320 + BLOCK_SIZE - 1];
```

Ініціалізація фільтру відбувається наступним чином:

```
arm_fir_init_f32(&_1kHz_15kHz_sig, 320, (float32_t *)&firCoeffs32[0], &firStateF32[0],
BLOCK_SIZE);
```

Весь текст програми представлено далі.

```
\#include "stm32f4xx_hal.h"            // Keil::Device:STM32Cube HAL:Common
#include "arm_math.h"                 // ARM::CMSIS:DSP

#define SIG_LENGTH 320
#define BLOCK_SIZE 32
#define NUM_TAPS 29

float32_t sigOut;
float32_t sigIn;
float32_t blockNum = SIG_LENGTH / BLOCK_SIZE;

const float32_t firCoeffs32[NUM_TAPS] = {
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f,
+0.0085302217f, -0.0000000000f, -0.0173976984f,
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f,
+0.2229246956f, +0.2504960933f, +0.2229246956f,
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -
0.0173976984f, -0.0000000000f, +0.0085302217f,
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};

float32_t outputSignal[SIG_LENGTH];
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];
extern  float32_t inputSignal_f32_1kHz_15kHz[SIG_LENGTH];

void printSignals(float32_t * signalIn, float32_t * signalOut, uint32_t length);
void SystemClock_Config(void);
void SysTick_Handler(void);

int main(void)
{
```

```
int i;

  HAL_Init();
  SystemClock_Config();

arm_fir_instance_f32 _1kHz_15kHz_sig;
arm_fir_init_f32(&_1kHz_15kHz_sig, NUM_TAPS, (float32_t*)&firCoeffs32[0],
&firStateF32[0], BLOCK_SIZE);

for(i = 0; i<blockNum; i++)
{
 arm_fir_f32(&_1kHz_15kHz_sig, (float32_t*)&inputSignal_f32_1kHz_15kHz[0] + (i *
BLOCK_SIZE), &outputSignal[0] + (i * BLOCK_SIZE), BLOCK_SIZE);
}


  while (1)
  {
          printSignals(&inputSignal_f32_1kHz_15kHz[0], &outputSignal[0], SIG_LENGTH);
  }
}

void SysTick_Handler(void)
{
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    //Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                        |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
   // Error_Handler();
  }
}

void printSignals(float32_t * signalIn, float32_t * signalOut, uint32_t length)
{
        uint32_t i;
        for(i = 0; i < length; i++)
        {
                sigIn = signalIn[i];
                HAL_Delay(1);
                sigOut = signalOut[i];
                HAL_Delay(1);
        }
}
```

В програмі використано символічні назви для числових значень – порядку фільтру (NUM_TAPS), довжині масиву, що містить вибірки сигналу (SIG_LENGTH) та розмір блоку даних (BLOCK_SIZE). Слід бути дуже уважним при використанні функції цифрової обробки сигналів. Для роботи фільтру потрібен додатковий масив, розмір якого дорівнює порядку фільтра плюс розмір блоку даних мінус 1, тобто firStateF32[BLOCK_SIZE + NUM_TAPS - 1].

Після виконання програми в масиві outputSignal буде відфільтрований сигнал. На рис. 2.68 представлено вікно логічного аналізатора з сигналом до фільтрації та після неї.

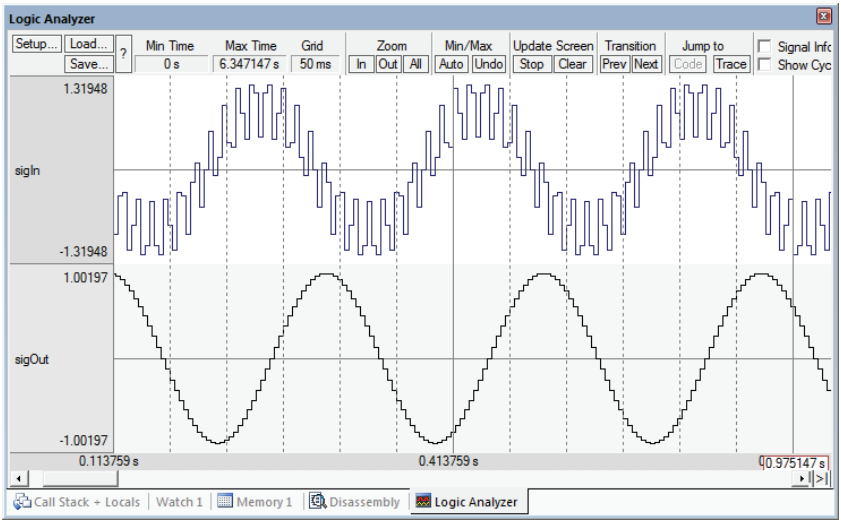


Рис. 2.68.

В наступному прикладі демонструється використання згортки для виконання фільтрації сигналу. Сигнал, що містить дві компоненти – низькочастотну та високочастотну, згортається з імпульсною характеристикою цифрового фільтру нижніх частот. В результаті цієї операції високочастотна компонента зникає. Текст програми представлено далі.

```
#include "stm32f4xx_hal.h"          // Keil::Device:STM32Cube HAL:Common
#include "arm_math.h"

#define SIG_LENGTH 320
#define NUM_TAPS 29

void SystemClock_Config(void);
void SysTick_Handler(void);
void BuildSin_Waveform(void);
void PrintSignal(void);

float32_t inputSignal_f32_1kHz_15kHz[SIG_LENGTH];
float32_t inputSignal;
float32_t outputSignal;
float32_t outputSignal_f32_1kHz[SIG_LENGTH + NUM_TAPS];

const float32_t firCoeffs32[NUM_TAPS] = {
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f,
+0.0085302217f, -0.0000000000f, -0.0173976984f,
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f,
+0.2229246956f, +0.2504960933f, +0.2229246956f,
```

```
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -
0.0173976984f, -0.0000000000f, +0.0085302217f,
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};

int main()
{
 HAL_Init();
 SystemClock_Config();

 BuildSin_Waveform();

arm_conv_f32(inputSignal_f32_1kHz_15kHz, SIG_LENGTH, firCoeffs32, NUM_TAPS,
outputSignal_f32_1kHz);

 while(1) {
  PrintSignal();
 }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    //Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                        |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
```

```
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
 {
  // Error_Handler();
 }
}

void SysTick_Handler(void)
{
 HAL_IncTick();
        HAL_SYSTICK_IRQHandler();
}

void BuildSin_Waveform(void)
{
 uint32_t i;
        for(i = 0; i < SIG_LENGTH; i++){
         inputSignal_f32_1kHz_15kHz[i] = (float32_t) sin(6.28 * i / 64) +
(float32_t) 0.33 * sin(6.28 * i / 4);
        }
}

void PrintSignal(void)
{
        uint32_t i;
  for(i = 0; i < SIG_LENGTH; i++){
         inputSignal = inputSignal_f32_1kHz_15kHz[i];
         outputSignal = outputSignal_f32_1kHz[i];
          HAL_Delay(10);
  }
}
```

Параметрами функції розрахунку згортки двох сигналів є перший сигнал у вигляді вказівника на масив, розмір першого масиву даних, вказівник на другий масив, що містить вибірки сигналу та його розмір, та вказівник на масив, де буде зберігатися результат згортки. При цьому розмір масиву для збереження результату повинен дорівнювати сумі розмірів першого та другого масивів.

Після компіляції проекту та завантаження програми в мікроконтролер необхідно перейти в режим відлагоджувача та додати змінні до логічного аналізатора.

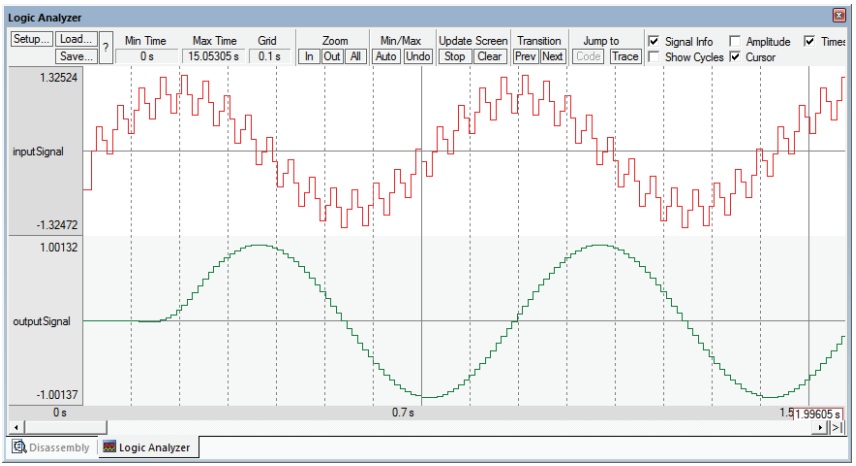


Рис. 2.69.

Оскільки другий сигнал в даному прикладі являє мобою імпульсну характеристику цифрового фільтру нижніх частот, тому операція згортки описує проходження сигналу через цифровий фільтр. Результатом згортки є відфільтрований сигнал, який містить лише низькочастотну складову.

РОЗДІЛ 3. МОДЕЛЮВАННЯ АПАРАТНО-ПРОГРАМНИХ ЗАСОБІВ ВІДОБРАЖЕННЯ ІНФОРМАЦІЇ

Більше 80% інформації людина сприймає органами зору. Тому візуальному сприйняттю інформації приділяється першочергова увага. Значні успіхи, досягнуті в області обчислювальної техніки і цифрових апаратних комплексів, що розробляються на її основі, стимулювали широкий фронт робіт зі створення електронних індикаторних пристроїв і систем.

Сфери застосування електронних індикаторів різноманітні і практично не обмежені. В даний час в апаратному забезпеченні різних електронних систем в якості елементів індикації широко використовуються напівпровідникові індикатори, рідкокристалічні панелі, газорозрядні, електролюмінесцентні індикатори, електронно-променеві трубки. Кожен з вказаних типів індикаторів, заснованих на використанні різних фізичних принципів, висуває певні вимоги до амплітуд управляючих напруг, виду струму, щільності розміщення на лицьових панелях приладів, зовнішньої освітленості. Вибір типу індикаторів диктується часто не лише оптимальним поєднанням вказаних технічних характеристик, але і вартістю, термінами розробки, станом серійного виробництва. Сучасні індикатори повинні забезпечувати впевнене зчитування інформації в умовах сильної і слабкої освітленості, мати високу яскравість і контрастність, мати високу чутливість до управляючих сигналів, малу споживану потужність, високу довговічність і надійність.

Застосування мікропроцесорних засобів дозволяє значно скоротити об'єми схем управління елементами індикації і підвищити надійність індикаторних пристроїв і систем за рахунок реалізації алгоритму управління не апаратними, а програмними засобами. В цьому розділі розглянуті питання моделювання електронних систем керування різноманітними засобами відображення інформації.

3.1. Моделювання систем керування напівпровідниковими індикаторами

Широтно-імпульсний метод (ШІМ) управління яскравістю заснований на скороченні часу протікання струму через світлодіоди індикаторів. При цьому знижується значення середнього прямого струм через світлодіоди, отже, знижується і яскравість їх світіння.

На рис. 3.1 представлена принципова схема моделі системи керування одиничним напівпровідниковим індикатором (світлодіодом). Світлодіод підключений катодом до виводу PB0 мікроконтролера, а анодом – до лінії живлення.

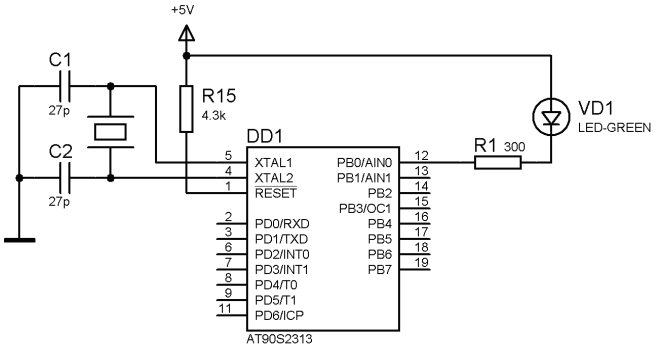


Рис. 3.1.

Програма для мікроконтролера виконує управління яскравістю світлодіода. Протягом 1 секунди яскравість світлодіода плавно знижується до мінімуму. Текст програми для мікроконтролера представлено нижче.

```
#include <io2313v.h>                    // підключення заголовочних файлів
#include <macros.h>
void delay(int ms)                      // функція затримки
{
 int c1, c2;                            // додаткові змінні
 for(c1=0; c1<ms; c1++)
  for(c2=0; c2<150; c2++) NOP();        // пуста операція
}
void main()                             // головна функція програми
{
 char i;                          // оголошення змінних
 char k=10;
 DDRB=0x01;                             // налаштування виводу PB0 на вихід
 while(1)                               // вічний цикл
 {
  for(i=0;i<7;i++)
  {
  PORTB=1;                              // виведення значення 1 в порт B
  delay(k);                             // затримка k мілісекунд
  PORTB=0;                              // виведення значення 0 в порт B
  delay(100-k);                         // затримка 100-k мілісекунд
  }
  if(k<90) k=k+10;     // якщо значення змінної k менше 90, до нього додається 10
  else k=10;           // інакше змінній k присвоюється значення 10
 }
}
```

На рис. 3.2 представлені результати моделювання роботи системи – епюра напруги, що пояснює роботу цієї схеми і алгоритм програми. При

зміні скважності імпульсів, зміниться середній струм, що протікає через світлодіод.

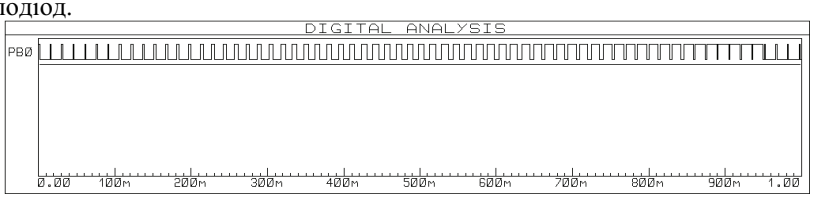


Рис. 3.2.

Зниження середнього прямого струму через сегменти викликає зниження світловіддачі напівпровідникового матеріалу індикатора, тобто регулювання яскравості індикатора.

Наступний приклад програми для мікроконтролера демонструє реалізацію алгоритму плавного запалення і згасання світлодіодів через один. Принципова схема мікроконтролерної моделі системи керування представлена на рис. 3.3. Аноди світлодіодів підключені до лінії живлення 5В, а катоди через струмообмежувальні резистори до виводів мікроконтролера. Таке схемотехнічне рішення використане з тієї причини, що мікроконтролер на своїх виводах може прийняти більший струм, ніж віддати. Таким чином, не відбувається перевантаження вихідних буферів мікроконтролера.

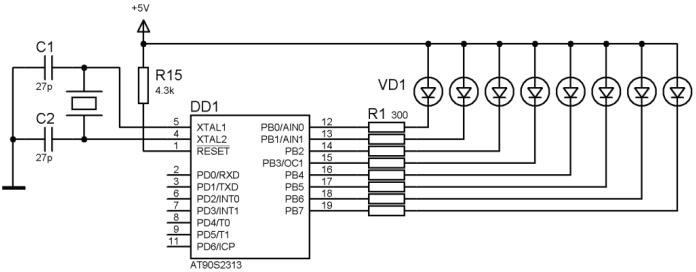


Рис. 3.3.

Текст програми для мікроконтролера представлено нижче

```
#include <io2313v.h>                    // підключення заголовочних файлів
#include <macros.h>
void delay(int ms)                      // функція затримки
{
 int t1,t2;                             // додаткові змінні
 for(t1=0;t1<ms;t1++)
 for(t2=0;t2<100;t2++) NOP();           // у циклі виконується «пуста» операція
}
void main()                             // головна функція програми
{
 char C=1;                              // оголошення змінної
 DDRB=0xFF;                  // налаштування порту B на вихід
```

```
 while(1)                        // вічний цикл
 {
  for(C=1;C<100;C++)             // плавне загоряння/згасання світлодіодів через один
  {
        PORTB=0b01010101;        // виведення значення 0x55 в порт B
        delay(C);                // затримка C мілісекунд
        PORTB=0b10101010;        // виведення значення 0xAA в порт B
        delay(100-C);            // затримка 100-C мілісекунд
        }
for(C=1;C<100;C++)
        {
        PORTB=0b01010101;        // виведення значення 0x55 в порт B
        delay(100-C);            // затримка 100-C мілісекунд
        PORTB=0b10101010;        // виведення значення 0xAA в порт B
        delay(C);                // затримка C мілісекунд
  }
 }
}
```

На рис. 3.4 представлена часова діаграма роботи пристрою. Світлодіоди, підключені до виводів PB0, PB2, PB4 і PB6 плавно згасають, а світлодіоди, підключені до виводів PB1, PB3, PB5 і PB7 плавно запалюються протягом 500 мс. Потім цикл повторюється знову.

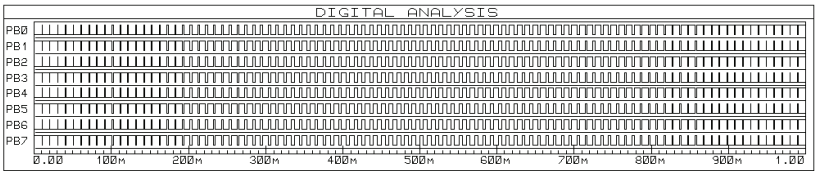
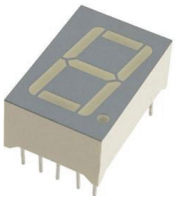


Рис. 3.4.

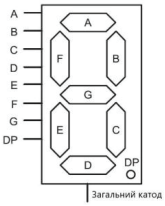
Схеми регулювання яскравості індикаторів з використанням генераторів широтно-модульованих імпульсів можуть бути різними. Проте будь-які варіанти такої схеми регулювання яскравості можуть використовуватися лише при обмеженій кількості індикаторів, оскільки одночасне включення-виключення великої кількості індикаторів викликає значні зміни струму джерела живлення. Боротьба з такими перешкодами в мікросхемній частині приладу часто викликає значні труднощі. Наприклад, найбільший струм індикатора ЗЛС324Б1 складає 140 мА і при одночасному регулюванні яскравості 15 індикаторів зміна струму складає більше 2А. Це необхідно враховувати при проектуванні джерела живлення і розводці провідників, або друкованого монтажу. Крім того, такий індикатор не можна безпосередньо під'єднати до виводів мікроконтролера, тому необхідно використовувати буферні елементи або транзисторні ключі.

3.2. Модель системи керування напівпровідниковим семисегментним індикатором

Розглянемо приклад використання масиву в програмі керування семисегментним індикатором. Семисегментний індикатор, як зрозуміло з його назви, складається з семи елементів індикації (сегментів), що включаються і виключаються окремо. Включаючи їх в різних комбінаціях, з них можна скласти зображення арабських цифр. Сегменти позначаються буквами від A до G; восьмий сегмент – десяткова кома, призначена для відображення дрібних чисел. На рис. 3.5, а, представлено зовнішній вигляд семисегментного індикатора, а на рис. 3.5, б, позначення його виводів.



а



б

Рис. 3.5.

В табл. 3.1 наводиться інформація про сегменти. Які потрібно увімкнути для того, щоб сформувати відповідну цифру. Наприклад, для цифри 7 треба включити сегменти A, B і C, тобто видати на виводи, до яких підключені ці сегменти, значення HIGH, а на інші LOW.

Таблиця 3.1.

Цифра	Сегменти
0	ABCDEF
1	BC
2	ABDEG
3	ABCDG
4	BCFG
5	ACDFG
6	ACDEFG
7	ABC
8	ABCDEFG
9	ABCDFG

На рис. 3.6 представлено модель пристрою для демонстрації програми керування семисегментним індикатором. В даному прикладі на індикатор послідовно виводяться цифри від 0 до 9 через проміжок часу 1 секунда.

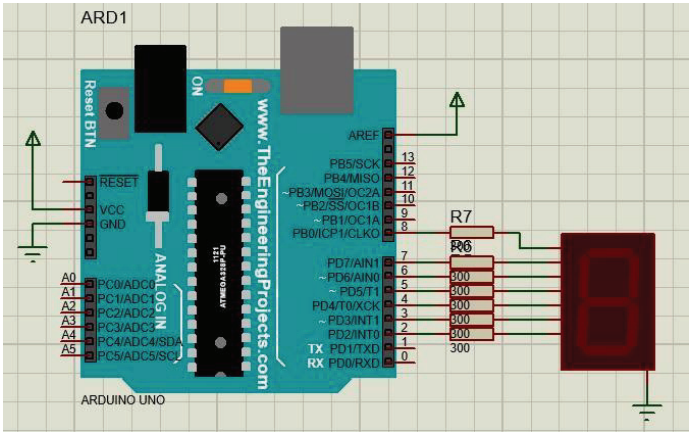


Рис. 3.6.

Текст програми представлено нижче.

```
// в масиві зберігаються номери виводів, до яких підключені сегменти
int pins[7]={8,7,6,5,4,3,2};
// в масиві зберігаються значення, що представляють кожну цифру
// якщо встановити виводи порту у відповідний стан 0 (LOW) або 1 (HIGH)
byte numbers[10] = { B11111100, B01100000, B11011010, B11110010, B01100110,
B10110110, B10111110, B11100000, B11111110, B11110110};
// змінна для зберігання значення
int number=0;

void setup() // функція ініціалізації контролера
{
// всі виводи контролера налаштовуються на вихід
// в циклі виконується послідовний доступ до кожного елемента масиву
for(int i=0;i<7;i++)
 pinMode(pins[i],OUTPUT); // налаштування відповідного виводу на вихід
}

void loop() // функція, в якій реалізовано основну логіку програми
{
 showNumber(number); // виклик функції відображення цифри на індикаторі
 delay(1000); // часова затримка 1 секунда
 number=(number+1)%10; // збільшення значення змінної на 1
 // залишок від ділення на 10 використовується для того, щоб не вийти за межі
 // масиву, адже залишок ділення на 10 не може перевищувати значення 9
}
// функція виводу цифри на семисегментний індикатор
void showNumber(int num)
{
 for(int i=0;i<7;i++)
 {
  if(bitRead(numbers[num],7-i)==HIGH) // якщо біт в масиві встановлений
```

```
 {
  digitalWrite(pins[i], HIGH); // увімкнути сегмент
}
else  // інакше
{
  digitalWrite(pins[i],LOW); // вимкнути сегмент
} // закриваюча дужка else
} // закриваюча дужка циклу for
} // закриваюча дужка тіла функції showNumber
```

В цій програмі було використано функцію bitRead(). Ця функція зчитує біт числа. Синтаксис функції:

```
bitRead(x, n);
```

Параметри, що передаються в функцію: x – номер біта, який зчитувати; n – який біт зчитувати, починаючи з 0 для найменш значущого (крайній праворуч) розряду. Функція повертає значення біта (0 або 1).

3.2.1. Мультиплексне управління індикатором

Мультиплексний режим рекомендується застосовувати для управління напівпровідниковими цифровими індикаторами з великою інформаційною ємністю. У таких пристроях використовуються в основному індикатори другої групи – індикатори монолітної конструкції, що мають управляючі виводи для однойменних сегментів декількох цифр, розміщених в одному корпусі. Мультиплексний режим управління напівпровідниковими цифровими індикаторами дозволяє знизити енергоспоживання індикаторів при збереженні їх характеристик яскравості.

На рис. 3.7 представлена модель схеми управління цифровими індикаторами в мультиплексному режимі, в якій використовується динамічна індикація.

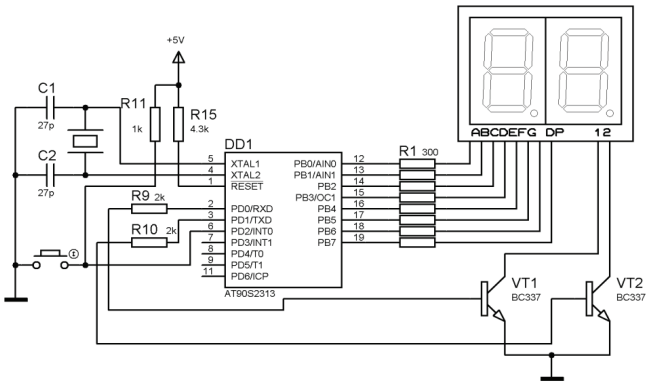


Рис. 3.7.

Модель представляє мобою схему лічильника натискань на кнопку. У схемі використаний двохрозрядний семисегментний індикатор зі спільним катодом. Оскільки сумарний струм загального катода індикатора перевищує максимально допустиме значення струму, що формується виводом мікроконтролера, в схемі використані біполярні транзистори, які працюють в ключовому режимі. При відкритому транзисторі VT1 відображається цифра на першому індикаторі, а при відкритому транзисторі VT2 – на другому.

У запропонованій схемі виводи однойменних сегментів всіх цифр сполучені паралельно і підключені до відповідних виходів мікроконтролера. Транзистори VT1 і VT2 виконують почергове підключення загального виводу кожного з сегментів до землі.

У першому такті мікроконтролер видає на порт B інформаційний код для відображення першої цифри. У другому такті мікроконтролер вимикає загальний вивід першої цифри, формуючи сигнал низького рівня на виводі PD0, тим самим замикаючи транзистор VT1, і підключає загальний вивід другої цифри шляхом відкриття транзистора VT2. Мікроконтролер по другому імпульсу встановлює на порту B код для відображення другої цифри, відключивши код першої. Потім цикл повторюється.

Час протікання струму через елемент, що світиться, обернено пропорційний до кількості цифр в керованому наборі. Отже, значення середнього прямого струму сегментів і яскравість їх світіння також скорочуються.

Текст програми представлено нижче.

```
#include <io2313v.h>                    // підключення заголовочних файлів
#include <macros.h>
char cnt;                               // змінна – лічильник
#pragma interrupt_handler int0_isr:2// обробник зовнішнього переривання
void int0_isr(void)
{
 // якщо значення лічильника менше 99, відбувається його інкремент
 if(cnt<99) cnt++;
 else cnt=0;                            // в протилежному випадку лічильник онулюється
}
void delay(int ms)                      // функція затримки
{
 int c1, c2;                            // додаткові змінні
 for(c1=0; c1<ms; c1++)
  for(c2=0; c2<970; c2++) NOP();        // у циклі виконується «пуста» операція
}
void main()                             // головна функція програми
{
// оголошення масиву з кодами для 7-сегментного індикатора
 char segment[10]={0b00111111, 0b00000110, 0b01011011, 0b01001111,
 0b01100110,0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111};
 DDRB=0b01111111;           // налаштування виводів PB0-PB6 на вихід
 DDRD=0b00000011;           // налаштування виводів PD0 і PD1 на вихід
 MCUCR = 0x02;                          // переривання виникає по спаду сигналу
```

```
GIMSK = 0x40;                    // дозвіл обробки переривання INT0
cnt=0;                           // обнулення лічильника
SEI();                           // дозвіл обробки переривань
while(1)                         // вічний цикл
{
 PORTD=1;                        // вмикання першого індикатора
 PORTB=~segment[cnt/10];         // виведення значення десятих на індикатор
 delay(20);                      // затримка 20 мс
 PORTD=2;                        // вмикання другого індикатора
 PORTB=~segment[cnt%10];         // виведення значення одиниць на індикатор
 delay(20);                      // затримка 20 мс
}
}
```

На рис. 3.8 приводиться часова діаграма, що пояснює роботу схеми і принцип динамічної індикації. Як видно з діаграми, на базах транзисторів сигнал формується в протифазі, що виключає виведення на обидва сегменти однакових значень.

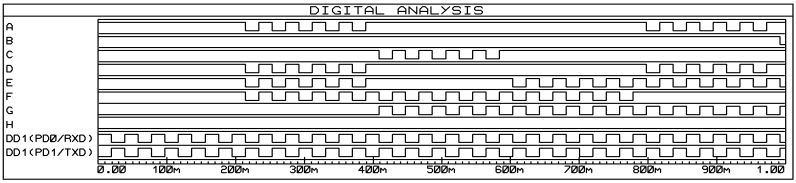


Рис. 3.8.

У наступному прикладі демонструється використання принципу динамічної індикації для багаторозрядних семисегментних індикаторів.

3.2.2. Моделювання системи керування матричним індикатором

Наступні приклади демонструють принципи управління матричними індикаторами зі стробуванням по стовпцях і по рядках.

Принципова схема моделі системи керування матричним індикатором, яка виводить на матричному індикаторі символ «А», представлена на рис. 3.9. Стробування виконується по стовпцях. Оскільки максимальний сумарний струм одного стовпця матричного індикатора складає 70 мА, і мікроконтролер на своїх виводах не може сформувати такий струм, в схемі використовуються транзистори. Низький рівень на базі транзистора включає відповідний стовпець. В кожний момент часу відкритий лише один транзистор.

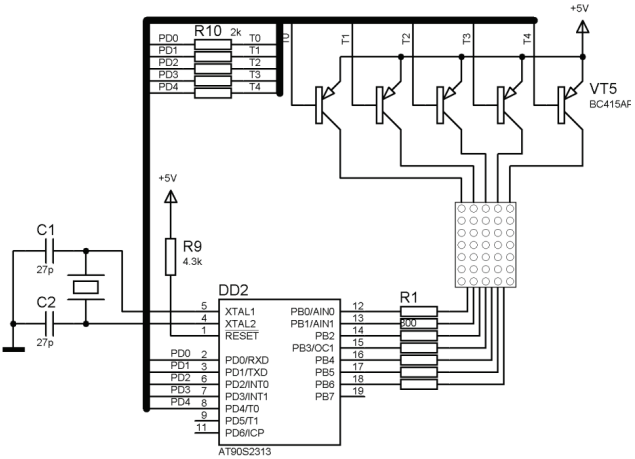


Рис. 3.9.

Текст програми для мікроконтролера представлено нижче.

```
#include <io2313v.h>                    // підключення заголовочних файлів
#include <macros.h>
void delay(int ms) {                    // функція затримки
 int c1, c2;                            // оголошення додаткових змінних
 for(c1=0; c1<ms; c1++)
  for(c2=0; c2<970; c2++) NOP();        // у циклі виконується «пуста» операція
}
void main() {                           // головна функція програми
 char c=0;                              // оголошення додаткової  змінної
 DDRB=0b01111111;           // налаштування порту B на вихід
 DDRD=0b00011111;           // налаштування виводів PD4-PD0 на вихід
 while(1)                               // вічний цикл
 {
 PORTD=0b00011110;
 PORTB=~0b01111100;
 delay(4);                              // затримка 4 мс
 PORTD=0b00011101;
 PORTB=~0b00010010;
 delay(4);                              // затримка 4 мс
 PORTD=0b00011011;
 PORTB=~0b00010001;
 delay(4);                              // затримка 4 мс
 PORTD=0b00010111;
 PORTB=~0b00010010;
 delay(4);                              // затримка 4 мс
 PORTD=0b00001111;
 PORTB=~0b01111100;
 delay(4);                              // затримка 4 мс
 } }
```

На рис. 3.10 представлена часова діаграма роботи пристрою.

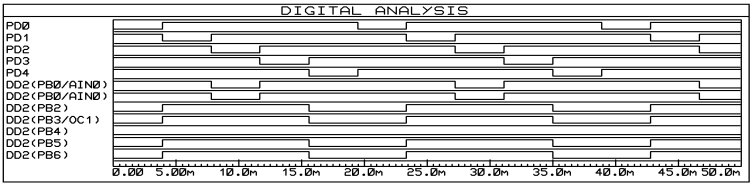


Рис. 3.10.

З часової діаграми видно, що в кожному такті лише на одному виводі порту D присутній низький логічний рівень, який управляє включенням відповідного стовпця. Відповідно до включеного стовпця, на порт B виводиться код, який формує символ, що відображається.

Принципова схема моделі системи керування, яка виводить на матричному індикаторі символ «Б» шляхом стробування по рядках, представлена на рис. 3.11. Оскільки максимальний сумарний струм одного стовпця матричного індикатора складає 50 мА, і мікроконтролер на своїх виводах не може сформувати такий струм, в схемі використовуються транзистори. Низький рівень на базі транзистора вмикає відповідний рядок. В кожен момент часу відкритий лише один транзистор.

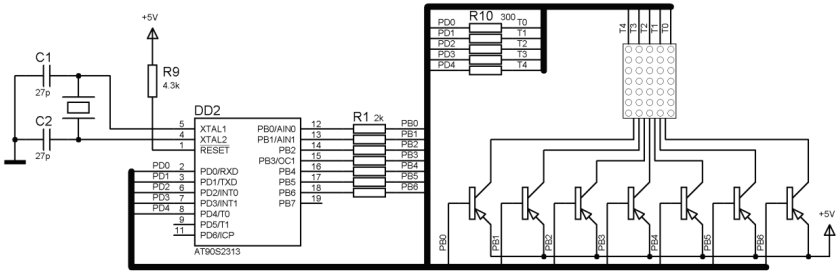


Рис. 3.11.

Текст програми для мікроконтролера представлено нижче.

```
#include <io2313v.h>                    // підключення заголовочних файлів
#include <macros.h>
void delay(int ms)                      // функція затримки
{
int c1, c2;                             // оголошення додаткових змінних
for(c1=0; c1<ms; c1++)
for(c2=0; c2<970; c2++) NOP();          // в циклі виконується «пуста» операця
}
void main()                             // головна функція програми
{
```

```
char i=0;
// в масиві зберігаються коди для формування символа
char symbol[7]={0b00000000, 0b00001111, 0b00001111, 0b00000001,
0b00001110, 0b00001110, 0b00000001};
DDRB=0b01111111;          // налаштування порту B на вихід
PORTB=0b01111111;                 // установка сигналу лог.1 на виводах порту B
DDRD=0b00011111;          // налаштування виводів PD4-PD0 на вихід
while(1)                          // вічний цикл
{
for(i=0;i<7;i++)          // у циклі почергово виводяться в порт елементи масиву
{
PORTD=symbol[i];
PORTB=~(1<<i);                    // на виводі порту B встановлюється сигнал лог.1.
delay(3);                         // затримка 3 мс
} } }
```

На рис. 3.12 представлена часова діаграма роботи пристрою

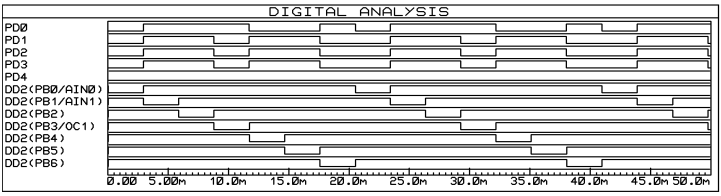


Рис. 3.12.

З часової діаграми видно, що в кожному такті лише на одному виводі порту B присутній низький логічний рівень, який управляє включенням відповідного рядка. Відповідно до включеної позиції, на порт D виводиться код, який формує символ, що відображається.

3.2.3. Модель системи керування семисегментними індикаторами з регістрами зсуву

Наступний приклад демонструє спосіб управління семисегментними індикаторами, при якому використовується мінімальна кількість виводів мікроконтролера.

На рис. 3.13 представлена модель лічильника імпульсів. Для зменшення кількості виводів мікроконтролера, необхідних для управління чотирма семисегментними індикаторами, використовуються регістри зсуву. Дані, що виводяться на індикатори, передаються від мікроконтролера в послідовному форматі, для чого потрібно всього 2 виводи – тактовий та інформаційний. Оновлення інформації на індикаторах відбувається за 32 такти.

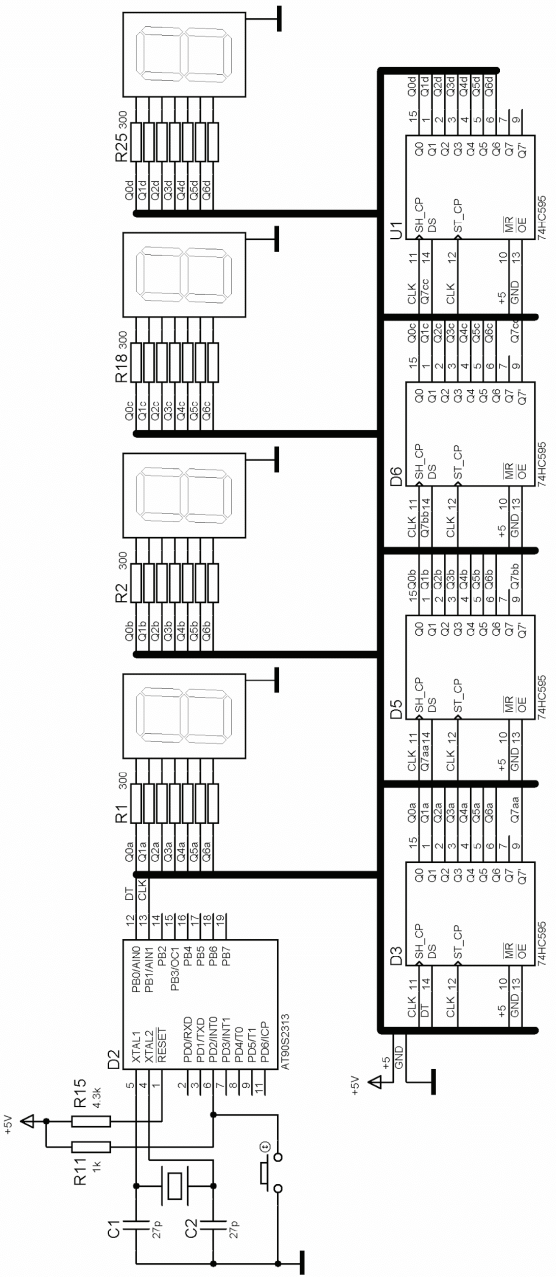

+5V
R15
4.3k
R11
1k
C1
27p
C2
27p
D2
AT90S2313
XTAL1
XTAL2
RESET
PB0/AIN0
PB1/AIN1
PB2
PB3/OC1
PB4
PB5
PB6
PB7
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/T0
PD5/T1
PD6/ICP
DT
CLK
R1 300
R2 300
R18 300
R25 300
D3
D5
D6
U1
74HC595
SH_CP
ST_CP
CLK
DS
MR
OE
GND
+5

Рис. 3.13.

Текст програми для мікроконтролера представлено нижче.

```
#include <io2313v.h>                    // підключення заголовочних файлів
#include <macros.h>
int cnt;                                // змінна – лічильник

#pragma interrupt_handler int0_isr:2 // обробник зовнішнього переривання
void int0_isr(void)
{
 // якщо значення лічильника менше 9999, відбувається його інкремент
 if(cnt<9999) cnt++;
 else cnt=0;                            // інакше – обнулення лічильника
}
void delay(int ms)                      // функція часової затримки
{
 int c1, c2;                            // оголошення додаткових змінних
 for(c1=0; c1<ms; c1++)                 // в циклі виконується «пуста» операція
  for(c2=0; c2<970; c2++) NOP();
}
void main()                             // головна функція програми
{
 char d[4]={0,0,0,0};                   // оголошення масиву из 4-х елементів
 char i, j, tmp;                        // оголошення додаткових змінних
 // оголошення масиву з кодами для 7-сегментного індикатора
 char segment[10]={0b00111111, 0b00000110, 0b01011011, 0b01001111,
0b01100110,0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111};
 DDRB=0b00000011;           // налаштування виводів PB0 і PB1 на вихід
 DDRD=0;                                // налаштування порту D на вихід
 MCUCR = 0x02;                          // обробка переривання за спадом імпульса
 GIMSK = 0x40;                          // дозвіл обробки зовнішнього переривання
 cnt=0;                                 // онулення лічильника
 SEI();                                 // дозвіл обробки переривань
 while(1)                               // вічний цикл
 {
  d[3]=segment[cnt/1000];               // в 0 елемент масиву записуються тисячі
  d[2]=segment[(cnt%1000)/100];     // в 1 елемент масиву записуються сотні
  d[1]=segment[((cnt%1000)%100)/10]; // в 2 елемент масиву записуються десятки
  d[0]=segment[((cnt%1000)%100)%10]; // в 3 елемент масиву записуються одиниці
PORTB|=0x02;                            // установка на виводі PB1 сигналу лог.1.
  PORTB&=~0x02;                         // установка на виводі PB1 сигналу лог.0.
  for(j=0;j<4;j++)                      // у циклі відбувається зсув значень
  {                                     // для передачі їх в послідовному форматі
  for(i=0;i<8;i++)
  {
   tmp=d[j]>>(7-i);                     // зсув значення елемента масиву
   tmp&=0x01;                   // виділення молодшого розряду
   PORTB=tmp;                           // виведення значення в порт B
   PORTB|=0x02;                         // установка на виводу PB1 сигналу лог.1
   delay(1);                            // затримка 1 мс
   PORTB&=~0x02;                        // установка на виводі PB1 сигналу лог.0
   delay(1);                            // затримка 1 мс
  }}
```

```
  PORTB|=0x02;                    // установка на виводі PB1 сигналу лог.1.
  PORTB&=~0x02;                   // установка на виводі PB1 сигналу лог.0.
  delay(200);                     // затримка 200 мс
}}
```

У програмі результатом виконання оператора «%» є залишок від ділення. Елементами масиву d[] будуть значення тисяч, сотень, десятків і одиниць. У циклі відбувається зсув кожного з елементів масиву, і молодший біт встановлюється на виводі PB0. Після цього на виводі PB1 формується тактовий імпульс, по фронту якого відбувається запис значення в регістр зсуву. Після 32 тактів на виводах регістрів встановлюються значення, відповідні тим, що відображається.

На рис. 3.14 представлена часова діаграма сигналів, що формуються мікроконтролером.

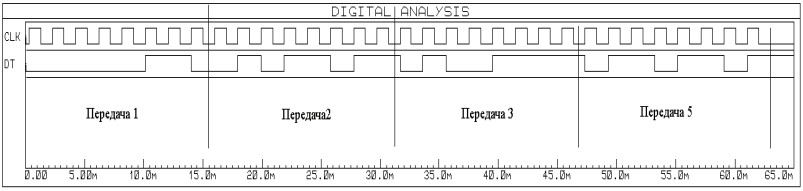


Рис. 3.14.

3.3. Моделювання пристрою «біжучий рядок»

Біжучі рядки на світлодіодах, як елемент інформаційного носія, можуть використовуватися як на фасаді будівель, так і всередині. Це дуже сучасний рекламний носій, що дозволяє людині отримувати цікаву інформацію у вигляді букво-цифрової послідовності, а також графіки.

Управління будь-яким світлодіодним рядком полягає в передачі спеціальної управляючої послідовності в ПЗП рядка. Управляюча послідовність – це спеціальний протокол, який дозволяє в певному порядку виводити час, температуру (тиск, вологість) та іншу інформацію на рядок, при цьому можуть використовуватися різні спецефекти (наприклад, вибігання по букві, поява тексту зверху внизу і так далі). Формування управляючої послідовності здійснюється за допомогою програмного забезпечення. Виникає також необхідність сполучення світлодіодного рядка з якимось іншим устаткуванням, що не є комп'ютером, наприклад мікроконтролером. При цьому принципи управління біжучим рядком залишаються однаковими. Передача інформаційної послідовності може здійснюватися через порти RS232, LPT, USB та ін. Використання багатоколірних світлодіодів дозволяє створювати кольорові зображення.

Для побудови засобів відображення інформації типу «біжучий рядок», використовуються регістри зсуву.

Пояснимо принцип формування символів. Передбачимо, що необхідно сформувати і вивести на біжучий рядок стилізований напис R1 (рис. 3.15).

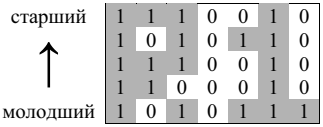


Рис. 3.15.

Для цього необхідно виконати таку послідовність дій:

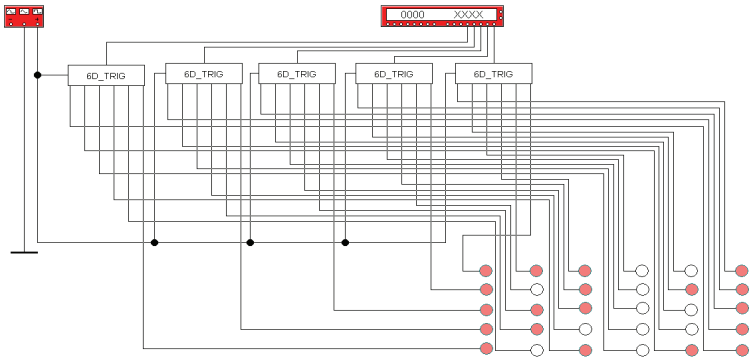
– на аркуші паперу вписати необхідні символи в матрицю, висота якої дорівнює кількості світлодіодів в біжучому рядку по вертикалі (замалювати потрібні клітинки);

– привласнити клітинкам, які виявилися зафарбованими, значення «1», а незафарбованим – значення «0»;

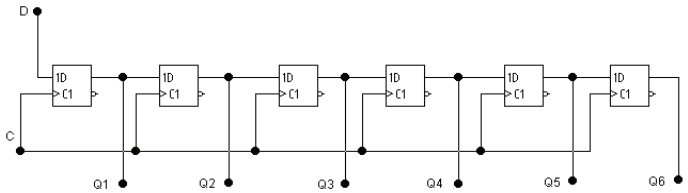
– нанести розрядну сітку;

– отримані числа розміщуються в масиві (в програмі для мікроконтролера або комп'ютера) для того, щоб він формував на своїх виходах в паралельному форматі потрібний двійковий код.

На рис. 3.16 представлена схема біжучого рядка, яка створена в Workbench. Для формування потрібної послідовності чисел використовується блок Word Generator (рис. 3.17).



а



б

Рис. 3.16.

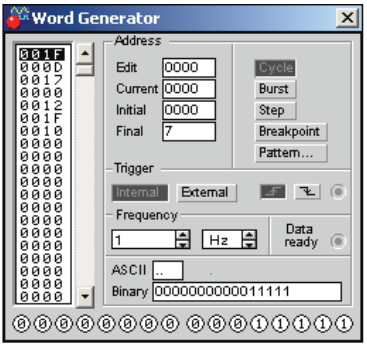


Рис. 3.17.

На рис. 3.18 представлена принципова схема біжучого рядка.

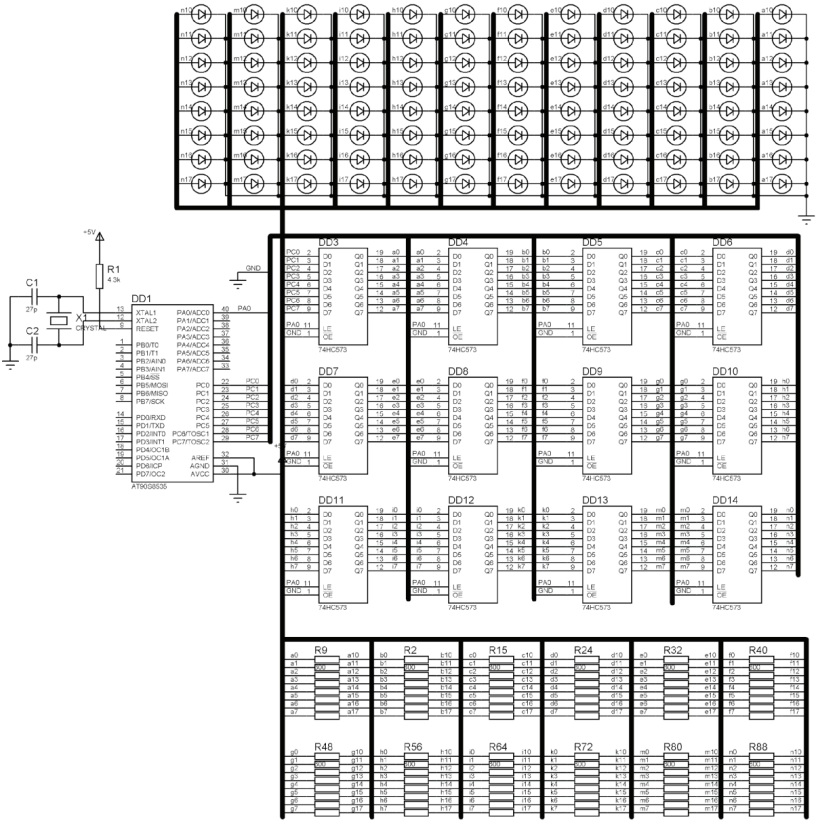


Рис. 3.18.

Порт С мікроконтролера сполучений з входом паралельного регістра DD4. До виходів паралельного регістра DD3 через струмообмежувальні резистори, підключена лінійка світлодіодів, а також входи регістра DD4. До виходів паралельного регістра DD4 також підключена лінійка світлодіодів, і входи регістра DD5, і т.д. Таким чином, всі паралельні регістри сполучені послідовно, і до виходу кожного з них підключені світлодіоди. При кожному наступному такті відбувається запис нового значення в регістри і створюється ефект руху.

Табл. 3.2.

1	1	1		1	1	1		1	1	1		1	1	1
1	0	1		1	0	0		1	0	0		0	1	0
1	0	1		1	0	0		1	0	0		0	1	0
1	0	1		1	1	1		1	1	1		0	1	0
1	0	1		1	0	0		1	0	0		0	1	0
1	0	1		1	0	0		1	0	0		0	1	0
1	0	1		1	0	0		1	0	0		0	1	0
1	0	1		1	1	1		1	1	1		0	1	0

Текст програми для мікроконтролера представлено нижче.

```
#include <io8535v.h>                    // підключення заголовочних файлів
#include <macros.h>
void delay(int ms)                       // функція затримки
{
 int c1, c2;                             // додаткові змінні
 for(c1=0; c1<ms; c1++)
  for(c2=0; c2<970; c2++) NOP();         // у циклі виконується «пуста» операція
}
void main()                              // головна функція програми
{
 char i;                          // додаткова змінна
 char text[12]={011111111, 0b00000001, 0b11111111, 0b11111111,
0b10001001, 0b10001001, 0b11111111, 0b10001001, 0b10001001,
0b00000001, 0b11111111, 0b00000001};    // в масиві зберігаються коди символів
 DDRA=0x01;                      // вивід PA0 налаштований на вихід
 DDRC=0xFF;                      // порт С налаштований на вихід
 while(1)                                // вічний цикл
 {
  for(i=0;i<12;i++)                      // у циклі виводяться в порт елементи масиву
  {
  PORTC=text[i];                         // виведення елементу масиву в порт
  delay(10);                             // часова затримка
  PORTA=1;                               // установка на виводі PA0 сигналу лог.1
  delay(10);                             // часова затримка
  PORTA=0;                               // установка на виводі PA0 сигналу лог.0
  delay(200);                            // затримка 200 мс
 }
 for(i=0;i<12;i++)
```

```
  {
  PORTC=0;                    // виведення в порт С значення 0 для очищення
табло
  PORTA=1;                    // установка на виході PA0 сигналу лог.1
  delay(10);                  // часова затримка
  PORTA=0;                    // виведення в порт А значення 0
  delay(200);                 // затримка 200 мс
  }
 delay(1000);                 // затримка 1 с
 }
}
```

На рис. 3.19 представлена часова діаграма роботи біжучого рядка.

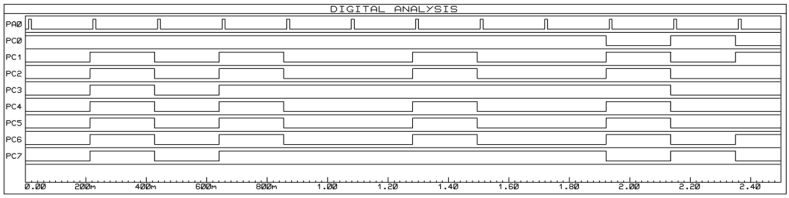


Рис. 3.19.

На рис. 3.20 представлений результат виконання програми. На світлодіодних індикаторах відображається рядок «TEEC».

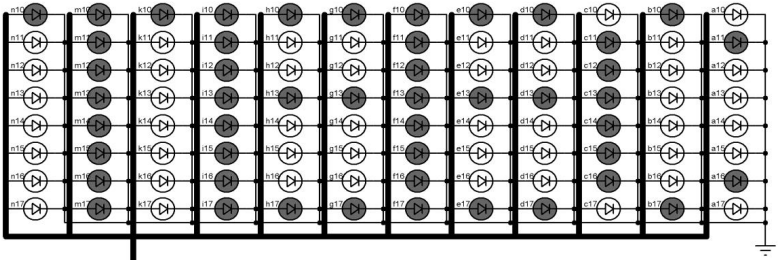


Рис. 3.20.

3.4. Моделювання системи керування алфавітно-цифровим рідкокристалічним індикатором

Наступний приклад демонструє принцип використання алфавітно-цифрового рідкокристалічного індикатора (РКІ) для виводу тексту та числових даних. Схема підключення РКІ до Arduino представлена на рис. 3.21.

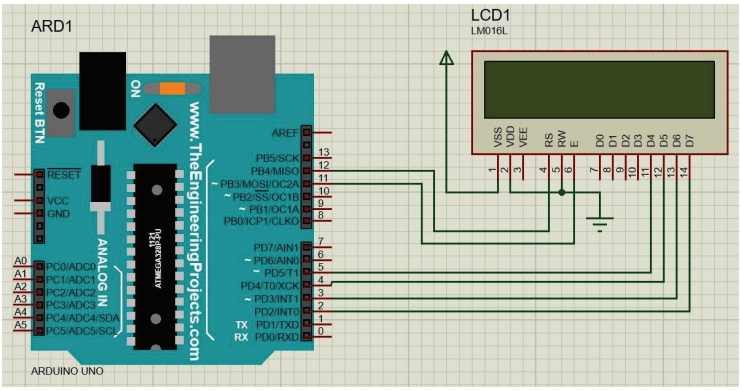


Рис. 3.21.

У верхній строчці РКІ виводиться текст «Hello, world!», а в нижній кількість секунд. Що минули з моменту вмикання пристрою (у випадку моделювання – з моменту початку моделювання роботи пристрою). Текст програми представлено нижче.

```
#include <LiquidCrystal.h> // підключення бібліотеки для роботи з РКІ

// ініціалізація РКІ шляхом вказівки, до яких виводів плати Arduino підключені
// відповідні виводи РКІ
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
// виклик функції ініціалізації РКІ
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // налаштування РКІ – режим виводу в 2 строки по 16 символів в кожній
  lcd.begin(16, 2);
  // вивід тексту "Hello, world!" на екран РКІ
  lcd.print("Hello, world!");
}

void loop() {
  // переведення курсору на другий рядок в початкову позицію
 // рядок 1 – це другий рядок. Оскільки нумерація починається з 0
  lcd.setCursor(0, 1);
  // вивід значення секунд з моменту вмикання пристрою
  lcd.print(millis() / 1000);
}
```

Після компіляції програми і запуску схеми на моделювання на екрані РКІ можна побачити відповідну інформацію, як показано на рис. 3.22.

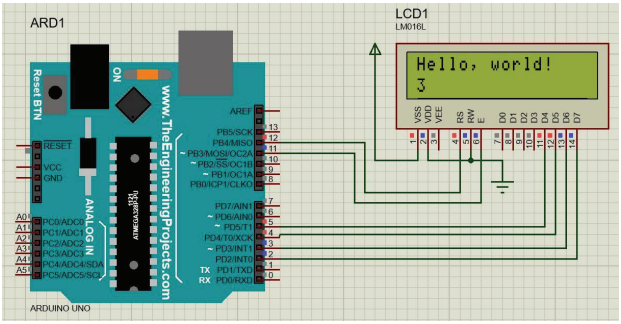


Рис. 3.22.

3.5. Моделювання РКІ з послідовним інтерфейсом

На рис. 3.23 представлена схема сполучення рідкокристалічного індикатора й мікроконтролера з використанням послідовного інтерфейсу.

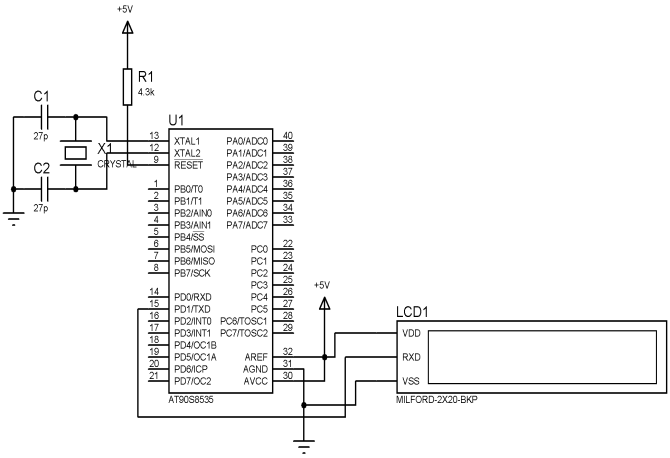


Рис. 3.23.

В наступній програмі для мікроконтролера у верхньому рядку РКІ виводиться статичний текст, а в нижньому рядку – значення лічильника. Управління РКІ виконується по послідовному інтерфейсу.

Текст програми для мікроконтролера представлено нижче.

```
#include <io8535v.h>                    // підключення заголовочних файлів
#include <macros.h>
#include <string.h>
void delay(int ms)                      // функція часової затримки
{
 int c1, c2;                            // додаткові змінні
```

```
 for(c1=0; c1<ms; c1++)
  for(c2=0; c2<970; c2++) NOP();              // у циклі виконується «пуста» операція
}
void init_cpu()                          // функція ініціалізації мікроконтролера
{
        UCR=0b00011000;                  // налаштування UART
        UBRR=51;                         // швидкість передачі 9600 біт/с
}
void main()                              // головна функція програми
{
char t;
int counter=0;
char text[]="LCD Display MILFORD";       // рядок для виведення на екран РКІ
char text1[]="Counter value: ";
init_cpu();                              // виклик функції ініціалізації РКІ
delay(500);                              // затримка 500 мс
while(1)                                 // вічний цикл
{
UDR=0xFE;                                // префікс команди: передаються команди
while(!(USR&(1<<UDRE)));           // очікування завершення передачі
UDR=0x0D;                                // команда відображення курсора
while(!(USR&(1<<UDRE)));           // очікування завершення передачі
for (t=0;t<strlen(text);t++)              // у циклі передається по буквам рядок
{
UDR=text[t];                             // передача елемента масиву
while(!(USR&(1<<UDRE)));           // очікування завершення передачі
}
for(counter=0; counter<10000; counter++)  // у циклі лічильник інкрементується
{
UDR=0xFE;                                // префікс команди: передаються команди
while(!(USR&(1<<UDRE)));           // очікування завершення передачі
UDR=0xC0;                                // переведення курсора на 2 рядок
while(!(USR&(1<<UDRE)));           // очікування завершення передачі
for (t=0;t<strlen(text1);t++)
{
UDR=text1[t];                            // виведення тексту
while(!(USR&(1<<UDRE)));           // очікування завершення передачі
}
 UDR=counter/1000+48;                    // виведення тисяч
 while(!(USR&(1<<UDRE)));                // очікування завершення передачі
 UDR=(counter%1000)/100+48;              // виведення сотень
 while(!(USR&(1<<UDRE)));                // очікування завершення передачі
 UDR=((counter%1000)%100)/10+48;         // виведення десятків
 while(!(USR&(1<<UDRE)));                // очікування завершення передачі
 UDR=((counter%1000)%100)%10+48;         // виведення одиниць
 while(!(USR&(1<<UDRE)));                // очікування завершення передачі
 delay(500);                             // пауза 500 мс
}}}
```

У функції init_cpu() виконується налаштування UART мікроконтролера. Швидкість обміну даними дорівнює 9600 біт/с. Для виведення статичного тексту в РКІ відправляються ASCII-коді символів, що виводяться. Після цього передається команда переведення курсора на другий рядок 0xC0. Для того, щоб вказати, що наступним байтом є команда, в модуль необхідно відправити префікс команди – число 0xFE. Для виведення числа на екран РКІ його необхідно заздалегідь перетворити в символи.

На рис. 3.24 показаний результат виконання програми.

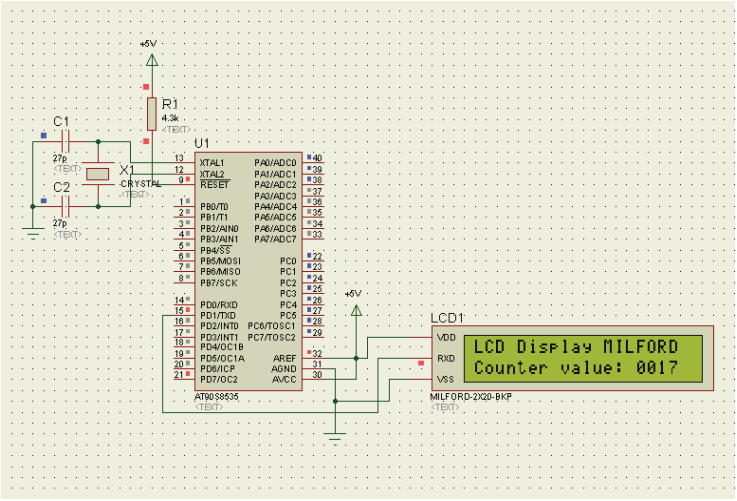


Рис. 3.24.

3.6. Модель мікропроцесорної системи керування графічним РКІ

Графічний РКІ MTG-12864 дозволяє приймати команди по шині DB7-DB0, записувати дані в ОЗП по 8-розрядній шині даних (DB7-DB0). Для початкової ініціалізації модуля потрібно подати сигнал RES. Після цього модуль нормально функціонує. Опис виводів модуля MTG-S12864BFGHSGY представлений в табл. 3.3.

Таблиця 3.3.

№ виводу	Назва виводу	Напрям даних	Опис
1	V_{dd}	–	Напруга живлення
2	V_{ss}	–	Земля
3	V_{ee}	–	Управління контрастністю
4-11	DB0-DB7	I/O	Шина даних з 3 станами
12	CS1	I	Вибір контролера лівої половинки дисплея. Активний рівень – низький

13	CS2	I	Вибір контролера правої половинки дисплея. Активний рівень – низький
14	RES	I	Встановлення сигналу в низький рівень ініціалізує графічний модуль
15	R/W	I	R/W=1 – дані можуть бути зчитані з модуля в контролер; R/W=0 – дані можуть бути записані в модуль
16	D/I	I	D/I=1 – дані на DB7-DB0 – дані для відображення D/I=0 – дані на DB7-DB0 – команда
17	E	I	Сигнал тактування. Якщо R/W=0, дані записуються по зрізу сигналу E. Якщо R/W=1, дані зчитуються, коли E приймає високий рівень
18	FGND	–	Земля
19	BKL_A	–	Живлення для підсвічування
20	BKL_K	–	

На рис. 3.25 представлена модель мікропроцесорної системи керування графічним РКІ MTG-12864.

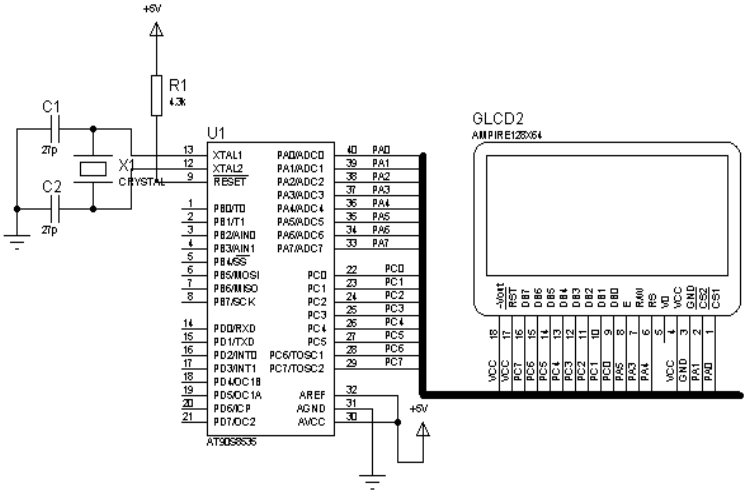


Рис. 3.25.

Модуль РКІ містить ОЗП для зберігання даних, які виводяться на РКІ, розміром 64×64×2 біт (по 64×64 біт на кристал). Для вибору потрібного кристалу використовуються сигнали CS1 і CS2. Далі, вся інформація відноситься до кожного кристалу. ОЗП розбитий на 8 сторінок розмірами 64×8 біт кожна. Кожній точці, яка світиться на РКІ, відповідає лог.1 у комірці ОЗП модуля.

Зчитування (запис) даних із (в) модуль виконується по сторінках (64×8 біт або 64×1 байт). Кожна сторінка представлена як 64 байти. Для зчитування або запису даних за заданою адресою необхідно заздалегідь встановити сторінку ОЗП і вибрати стовпчик в межах сторінки ОЗП. Це виконується командами «Set Page» і «Set Address» відповідно (табл. 4.10). Після цього можна записати, або прочитати байт даних. Модуль підтримує безперервну послідовність операцій зчитування і запису: після зчитування (запису) одного байта лічильник стовпчика автоматично збільшується на 1, і модуль готовий до нової операції зчитування (запису) за наступною адресою без попередньої установки сторінки ОЗП і адреси стовпчика. Лічильник стовпчика рахує в межах однієї сторінки. При досягненні лічильником адреси 63, наступним буде значення 0 і т. д.

Таблиця 3.4.

Команда	RD/WR	A0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Функція
Display ON/OFF	0	0	0	0	1	1	1	1	1	0/1	Вмикає або вимикає РКІ, незалежно від даних у екранному ОЗП й внутрішнього стану "1"- ввімкнути дисплей "0"- вимкнути дисплей
Display START Line	0	0	1	1	Display START Line (0...63)						Визначає строку ОЗП, яка буде відображатися у верхній строчці РКІ (Стартова строка РКІ)
Set Page	0	0	1	0	1	1	1	Page (0...7)			Встановлює сторінку ОЗП (стр. 0...7)
Set Address	0	0	0	1	Column address (0...63)						Визначає адресу стовпця
Status Read	1	0	BUSY	0	ON/OFF	RESET	0	0	0	0	Зчитування режиму стану BUSY 1 Модуль зайнятий внутрішньою обробкою BUSY 0 Модуль готовий до роботи з зовнішнім МП ON/OFF 1 РКІ ввімкнено ON/OFF 0 РКІ вимкнено RESET 1 Стан скидання RESET 0 Нормальний стан
Write Display Data	0	1	Write Data								Запис данних до ОЗП модуля
Read Display Data	1	1	Read Data								Зчитування данних з ОЗП модуля

Ці команди вибирають ОЗП за допомогою раніше заданої адреси, після чого адреса стовпця інкремінується

Модуль підтримує команду «Display START Line», яка встановлює номер верхнього рядка, що відображається. Номер може бути в інтервалі від 0 до 63, що відповідає інтервалу від першого рядка нульової сторінки ОЗП до останнього рядка сьомої сторінки ОЗП. Відразу за сьомою сторінкою відображається знову нульова сторінка ОЗП. Це дозволяє реалізувати плавний зсув інформації на РКІ по вертікалі шляхом зміни першого рядка, що

На відміну від алфавітно-цифрових рідкокристалічних модулів, в яких для виведення тексту достатньо було записати ASCII-коди символів в пам'ять, в графічних модулях така можливість відсутня. Тому завдання по

відображенню інформації (як текстової, так і графічної) вирішується програмним шляхом. Кожна сторінка пам'яті має 8 біт (точок) у висоту. При виведенні текстової інформації необхідно витримати проміжки між рядками для зручності сприйняття тексту (хоч і не обов'язково). У алфавітно-цифрових РКІ найчастіше розміри символу складають 5×7 точок. Розмір знакомісця дорівнює 6×8 точок. Нижній рядок і стовпчик праворуч завжди погашені. Це зроблено для формування відстані між символами. Для відображення одного знакомісця вказаних розмірів, необхідно записати в ОЗП модуля 6 байт даних, заздалегідь вказавши адресу сторінки і стовпця, де потрібно вивести символ (рис. 3.26). Всі дані, які утворюють шрифт, зручно оформити у вигляді двовимірного масиву, наприклад font[256][6]. У цьому масиві зберігатиметься графічна інформація про 256 символів ASCII, кожен з яких представлений 6 байтами. Управління виведенням текстової інформації значно полегшується, якщо індекс елементу масиву збігається з ASCII-кодом символу. У такому разі досить звернутися до елементу масиву за кодом символу з таблиці ASCII, і розробник отримує доступ до 6 байт даних, які власне і сформують графічне зображення при виведенні їх на індикатор.

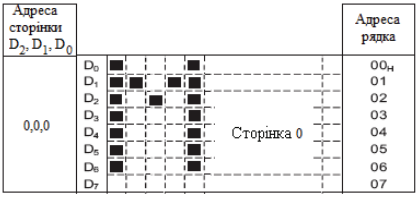


Рис. 3.26.

Крім тексту можна виводити на екран графічного РКІ різні зображення. Принцип формування зображення залишається таким самим, як і при виведенні текстової інформації. Необхідно сформувати масив даних в пам'яті контролера (Flash, EEPROM), а потім в циклі записати ці дані в індикатор. Звичайно, потрібно стежити за значенням стовпця, в який виводяться дані у поточний момент, і досягши значення 63, змінювати номер сторінки пам'яті. Крім того, можливе програмне формування графіки.

В наступному прикладі наведено код програми, в результаті виконання якої мікроконтролер формує сигнали керування РКІ модулем для виведення в правій частині екрану фотографії, а в лівій – текстової інформації. Принципова схема пристрою представлена на рис. 3.25.

Текст програми представлено нижче.

```
#include <io8535v.h>                    // підключення заголовочних файлів
#include <macros.h>                     // підключення бібліотеки макросів
#include <string.h>                     // бібліотека функцій для роботи з рядками
#define CS1() PORTA|=0x01// установка сигналу управління
// лівою половиною індикатора
```

```
#define NCS1() PORTA&=~0x01     // скидання сигналу управління
// лівою половиною індикатора
#define CS2() PORTA|=0x02// установка сигналу управління
// правою половиною індикатора
#define NCS2() PORTA&=~0x02     // скидання сигналу управління
// правою половиною індикатора
#define RES() PORTA|=0x04       // установка сигналу скидання RES
#define NRES() PORTA&=~0x04     // скидання сигналу RES
#define RW() PORTA|=0x08 // установка сигналу RW – зчитування/запис
#define NRW() PORTA&=~0x08      // скидання сигналу RW – зчитування/запис
#define DI() PORTA|=0x10        // установка сигналу DI – команда/дані
#define NDI() PORTA&=~0x10      // скидання сигналу DI – команда/дані
#define RS() DI()               // установка сигналу RS – команда/дані
#define NRS() NDI()             // скидання сигналу RS – команда/дані
#define E() PORTA|=0x20         // установка сигналу дозволу запису в РКІ
#define NE() PORTA&=~0x20       // скидання сигналу дозволу запису в РКІ
// в двомірному масиві зберігається інформація про шрифт
const char currfont[91][6]={{0,0,0,0,0,0},{0,6,95,6,0,0},{0,0,0,0,0,
0},{0,0,0,0,0,0},{36,74,255,82,36,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{
0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,160,96,0,0},{0,0,0,0,0,0},{0,0,96,96,0,0},{0,0,0,
0,0,0},{62,65,65,65,62},{0,4,2,127,0,0},{66,97,81,73,70,0},{34,65,73,73,54,0},{56,36,34,12
7,64,0},{47,73,73,73,49,0},{62,73,73,73,50,0},{65,33,17,9,7,0},{54,73,73,73,54,0},{38,73,7
3,73,62,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{
0,0,0,0,0,0},{124,18,17,18,124,0},{127,73,73,73,54,0},{62,65,65,65,34,0},{127,65,65,65,28,
0},{127,73,73,73,73,0},{127,9,9,9,1,0},{62,65,65,81,50,0},{127,8,8,8,127,0},{129,129,127,
129,129,0},{48,64,64,64,63,0},{127,8,20,34,65,0},{127,64,64,64,64,0},{127,2,4,2,127,0},{1
27,4,8,16,127,0},{62,65,65,65,62,0},{127,9,9,9,6,0},{62,65,73,81,62,0},{127,9,25,41,70,0},
{38,73,73,73,50,0},{1,1,127,1,1,0},{63,64,64,64,63,0},{31,32,64,32,31,0},{127,64,48,64,12
7,0},{99,20,8,20,99,0},{3,4,120,4,3,0},{105,89,73,77,75,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0
,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0},{56,68,68,36,120,0},{127,72,72,72,48,0},{56
,68,68,68,40,0},{56,68,68,36,127,0},{56,84,84,84,88,0},{8,126,9,1,6,0},{28,162,162,162,12
4,0},{127,8,8,8,112,0},{0,68,125,68,0,0},{0,32,64,64,61,0},{127,16,40,68,0,0},{0,127,64,0,
0,0},{124,4,124,4,120,0},{124,4,4,4,120,0},{56,68,68,68,56,0},{252,36,36,36,24,0},{24,36,
36,36,252,0},{124,4,4,4,8,0},{72,84,84,84,36,0},{4,63,68,64,32,0},{60,64,64,64,124,0},{28,
32,64,32,28,0},{60,64,56,64,60,0},{68,40,16,40,68,0},{28,160,160,160,124,0},{68,116,84,9
2,68,0}};
// в масиві, який розташований у Flash-пам’яті, зберігається інформація про
// зображення (фотографія)
const char image[512]={0,0,0,24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
64,128,128,160,224,224,224,216,176,247,239,14,13,13,27,19,7,15,15,31,31,51,32,0,0,0,0,0,0,
1,2,0,0,1,2,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,192,192,128,64,132,133,139,127,255,255
,255,255,255,127,123,115,118,96,98,68,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,3,134,142,22,12,28,29,25,27,19,7,135,135,135,7,7,7,7,6,26,28,12,140,28,1
42,138,200,204,232,248,248,248,248,240,240,240,224,224,224,240,224,224,64,128,128,192,
192,0,128,192,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,132,198,199,99,99,7,6,14,159,255,255,255
,255,255,254,252,252,248,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,2
55,255,255,255,255,254,248,251,223,255,7,7,15,15,6,0,0,0,0,0,0,224,224,192,128,224,224,0,
0,0,0,1,1,0,192,228,252,254,255,231,255,255,255,255,255,255,127,127,63,63,191,251,243,24
3,243,227,227,199,199,7,15,15,223,255,255,255,255,255,255,255,255,255,255,63,7,1,1,4,0,0,12,1
24,112,0,6,134,140,143,183,63,115,121,252,124,126,254,124,112,56,140,135,95,63,127,255,
1,1,1,1,1,0,48,56,62,31,159,207,231,103,179,243,217,249,224,244,248,252,248,243,255,255,
255,255,255,255,127,1,128,192,224,104,0,32,0,0,0,1,6,14,12,31,31,31,15,6,30,62,240,222,22
```

```
3,60,62,127,127,127,92,28,8,0,0,0,0,0,0,56,48,48,0,4,15,14,6,135,195,195,199,255,255,255,2
55,255,255,255,255,255,63,31,15,7,129,208,254,255,255,246,250,240,240,248,254,254,240,0,0,0,
0,16,24,120,123,127,63,127,255,39,6,4,0,0,0,0,0,0,128,240,112,184,136,144,216,152,224,240
,176,32,195,7,143,15,143,31,159,143,207,199,131,1,1,128,8,28,8,65,97,89,239,255,255,255,2
55,255,255,255,255,254,255};
 // оголошення прототипів функцій
void init_device(void);          // функція ініціалізації контролера
void write(void);                // функція запису команди або даних в РКІ
void clear_lcd(void);            // функція очищення екрану індикатора
void clear_row(char row, char half);          // функція очищення окремого рядка;
// передаються номер рядка (0-7) і номер кристалу (1-лівий, 2 – правий, 0 – обидва)
void init_lcd(void);             // функція ініціалізації індикатора
void delay_ms(int);              // функція затримки; параметр – затримка в мс
void hi_z(void);                 // функція переведення виводів в стан високого імпеданса
void delay_us(int);              // функція затримки; параметр – затримка в мкс
void out_string(char *str, char row, char stpos);     // функція виведення текстового рядка
void draw_flash(char half);   // функція, яка виводить на індикатор дані з масиву image
// реалізація функцій
void init_device(void)           // функція ініціалізації контролера
{
 PORTA=0x00;                     // виведення значення 0 в порт A
 DDRA=0b00111111;                // налаштування шести розрядів порту A на вихід
 PORTB=0x00;                     // виведення значення 0 в порт B
 PORTC=0x00;                     // виведення значення 0 в порт C
 DDRC=0xFF;           // налаштування виводів порту C на вихід
 PORTD=0x00;                     // виведення значення 0 в порт D
 DDRD=0x08;           // налаштування виводів PD3, PD4, PD5 на вихід
 }
 void write(void)                // функція запису команд/даних в індикатор
 {
 char t;                  // оголошення додаткової змінної
 E();                            // установка тактового сигналу
 for(t=0;t<6;t++);               // затримка
 NE();                           // скидання тактового сигналу
 for(t=0;t<6;t++);               // затримка
}
 void hi_z(void)                 // функція переведення виводів в стан високого імпеданса
 {
 NCS1();                         // скидання сигналу CS1
 NCS2();                         // скидання сигналу CS2
 NRS();                          // скидання сигналу RS
 PORTC=0;                        // виведення в порт значення 0
 DDRC=0;                         // переведення порту C в стан високого імпеданса
 }

 void clear_lcd(void)            // функція очищення екрану індикатора
 {
 char i;                  // додаткова змінна для використання в циклі
 for(i=0;i<8;i++)                // цикл виконується 8 разів
 clear_row(i,0);                 // виклик функції очищення рядків обох кристалів дисплея
(параметр 0)
 DDRC=0xFF;           // налаштування порту C на виведення даних
```

```
NRS();                          // скидання сигналу RS – передаються команди
PORTC=0b11000000;               // виклик команди Display Start Line
write();                        // запис команди в індикатор
hi_z();                         // переведення порту в стан високого імпеданса
}

void clear_row(char row, char half) // функція очищення окремого рядка індикатора
{
char i;                 // додаткова змінна для використання в циклі
PORTC=0;                        // виведення в порт С значення 0
DDRC=0xFF;              // налаштування порту С на виведення даних
NRS();                          // скидання сигнала RS – передаються команди
NRW();                          // скидання сигналу RW
if(half==1)                     // якщо вибрана ліва половинка екрану
{
NCS1();                         // скидання сигналу CS1
CS2();                          // установка сигналу CS2
}
else if(half==2)                // якщо вибрана права половина екрану
{
CS1();                          // установка сигналу CS1
NCS2();                         // скидання сигналу CS2
}
else                            // якщо вибрані обидві половинки екрану
{
NCS1();                         // скидання сигналу CS1
NCS2();                         // скидання сигналу CS2
}
row&=0x07;              // виділення трьох молодших біт змінної row  row|=0b10111000;
                // формування команди Set Page
PORTC=row;                      // виведення даних на порт С
write();                        // запис команди в індикатор
PORTC=0b01000000;               // передача команди Set address
write();                        // запис команди в модуль
RS();                           // установка сигналу RS – передаються дані
PORTC=0;                        // установка значення 0 на порт С
for(i=0;i<64;i++)               // цикл виконується 64 рази
{
 write();                       // запис 0 в пам'ять індикатора - очищення рядка
}
hi_z();                         // переведення порту в високоімпедансний стан
}

void init_lcd(void)             // функція ініціалізації РКІ
{
        char i;                 // додаткова змінна
        NRES();                 // скидання сигналу RES
        delay_us(500);// часова затримка 500 мкс
        RES();                  // установка сигналу RES
        delay_us(500);// часова затримка 500 мкс
        NCS1();                 // скидання сигналу CS1
        NCS2();                 // скидання сигналу CS2
```

```
        NRW();                  // скидання сигналу R/W
        NDI();                  // скидання сигналу D/I (RS) – передаються команди
        PORTC=0b00111111;       // виклик команди Display ON
        write();                // запись команди в індикатор
        PORTC=0b11000000;       // виклик команди Display StartLine
        write();                // запись команди в індикатор
        PORTC=0b10111000;       // виклик команди Set Page
        write();                // запис команд в модуль
        PORTC=0b01000000;       // виклик команди Set Address
        write();                // запис команди в індикатор
        hi_z();                 // переведення порту в стан високого імпеданса
}
void delay_ms(int ms)           // функція затримки; параметр – затримка в мс
 {
  char i,j;                     // додаткові змінні
  int k;
  for(k=0;k<ms;k++)             // у циклі контролер виконує «пусті» операції
        {
         for(i=0;i<20;i++)
         for(j=0;j<75;j++);
        }
}
void delay_us(int us)           // функція затримки; параметр – затримка в мкс
 {
 int k;                         // додаткова змінна
 for(k=0;k<=us;k++)             // у циклі контролер виконує «пусті» операції
        {
                NOP();          // пуста операція
        }
}
void out_string(char * str, char row, char stpos)    // функція виведення текстового рядка
 {
 char len=strlen(str);          // визначення довжини, яка виводиться текстового рядка
 char i,j;                      // змінні для роботи алгоритма
 char curpos=stpos;             // запам'ятовування стартової позиції
 clear_row(row,0);              // виклик функції очищення рядка
 PORTC=0;                       // виведення значення 0 в порт С
 DDRC=0xFF;         // налаштування порту С на вивід
 NRS();                         // скидання сигналу RS – передаються команди
 delay_us(2);                   // затримка 2 мкс
 NCS1();                        // скидання сигналу CS1
 NCS2();                        // скидання сигналу CS2
 row&=0x07;                     // виділення трьох молодших біт змінної row
 row=row|=0b10111000;           // формування команди Set Page
 PORTC=row;                     // виведення команди в порт
 write();                       // запис команди в порт
 CS2();                         // установка сигналу CS2
 stpos&=0b00111111;// виділення 6 молодших біт значення стартової позиції
 stpos|=0b01000000;             // формування команди Set Address
 PORTC=stpos;                   // виведення команди в порт
 write();                       // запис команди в порт
 RS();                          // установка сигналу RS – передаються дані
```

```
delay_us(2);                    // затримка 2 мс
for(i=0;i<len;i++)              // цикл виконується стільки разів, скільки букв в рядку
{
 for(j=0;j<6;j++)               // виводяться дані про символи з двовимірного масиву
 {
  PORTC=currfont[str[i]-32][j];       // установка значення j-елемента масиву на порту C
  write();                      // запис даних в РКІ
  curpos++;                     // збільшення на 1 внутрішнього лічильника
  if(curpos==64)                // якщо досягнуто край половинки екрану
  {
        CS1();                  // скидання сигналу CS1
        NCS2();                 // установка сигналу CS2
        NRS();                  // скидання сигналу RS – передаються команди
        PORTC=0b01000000;           // установка команди Set Address
        write();          // запис команди в РКІ
        RS();                   // установка сигналу RS – передаються дані
  }
 }
}
hi_z();                         // переведення порту в стан високого імпеданса
}

void draw_flash(char half)  // функція виведення на екран даних
{
char i=0;                       // додаткова змінна
char page=0;                    // змінна – номер сторінки пам’яті
int a=0;                        // додаткова змінна
char data;
DDRC=0xFF;          // налаштування порту C на вивід
if(half==1)                     // якщо значення змінної half дорівнює 1
{
CS2();                          // установка сигналу CS2
NCS1();                         // скидання сигналу CS1
}
else if(half==2)                // якщо значення змінної half дорівнює 2
{
NCS2();                         // скидання сигналу CS1
CS1();                          // установка сигналу CS2
}
else                            // якщо значення змінної half дорівнює 0
{
NCS2();                         // скидання сигналу CS2
NCS1();                         // скидання сигналу CS1
}
for(page=0;page<8;page++) // у циклі виконується запис даних в кожну сторінку пам’яті
{
NRS();                          // скидання сигналу RS – передається команда
PORTC=0b11000000;               // виведення команди Display Start Line
write();                        // запис команди в індикатор
PORTC=0b10111000|page;  // виведення команди Set Page
write();                        // запис команди в индикатор
PORTC=0b01000000;               // виведення команди Set Address
```

```
write();                    // запис команди в індикатор
for(i=0;i<64;i++)           // цикл повторюється 64 рази
{
 RS();                      // установка сигналу RS – запис даних
 PORTC=~image[a];           // установка даних на порту С
 write();                   // запис даних в індикатор
 a++;                       // інкремент змінної
}
}
hi_z();                     // переведення порту в стан високого імпеданса
}

void main(void)
{
char name[]="Photo on";     // оголошення змінних, в яких зберігається текстова
інформація
char sname[]="Graphic";     // кожна змінна – одновимірний масив
char surname[]="Display"; // елементи масиву – окремі букви
char ed1[]="Magister of";   // розмір кожного з масивів дорівнює кількості букв у слові
char ed2[]="electronics";
char vuz[]="TEES 2011";
init_device();              // виклик функції ініціалізації контролера
init_lcd();                 // виклик функції ініціалізації РКІ
clear_lcd();                // очищення екрану РКІ
out_string(name,0,0);// виклик функції виведення тексту
out_string(sname,1,0);      // виклик функції виведення тексту
out_string(surname,2,0);    // виклик функції виведення тексту
out_string(ed1,4,0);   // виклик функції виведення тексту; 4 рядок починаючи з позиції 0
out_string(ed2,5,0); // виклик функції виведення тексту; 5 рядок починаючи з позиції 0
out_string(vuz,7,0); // виклик функції виведення тексту; 7 рядок починаючи з позиції 0
draw_flash(2);          // виклик функції виведення зображення у правій частині екрану
while(1);               // зациклення мікроконтролера
}
```

Фотографія, яка відображається на екрані РКІ, зберігається у flash-пам'яті мікроконтролера. Розмір фотографії дорівнює 8×64=512 байт. Також в пам'яті мікроконтролера зберігається інформація про шрифт. Для одного символу потрібно 6 байт. З метою економії пам'яті можна зберігати в пам'яті лише ті символи, які відображатимуться. Формування на екрані РКІ фотографії відбувається таким чином. У циклі послідовно на екран виводяться значення з масиву. Коли значення індексу досягає 63, відбувається зміна адреси сторінки пам'яті, в яку записуються дані. У межах однієї сторінки немає необхідності установки зсуву по горизонталі, оскільки контролер РКІ автоматично збільшує значення лічильника зсуву на 1 після запису чергового значення.

Після компіляції проекту і програмування мікроконтролера, на екрані графічного РКІ з'явиться текстова інформація і зображення, як показано на рис. 3.27.

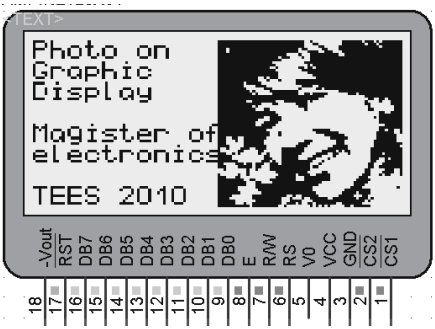


Рис. 3.27.

3.7. Спеціалізовані програмні засоби для моделювання роботи РКІ

Існують програми, за допомогою яких можна досліджувати алгоритми управління алфавітно-цифровими рідкокристалічними індикаторами. Однією з програм, яка створена для моделювання роботи дворядкового РКІ, є ***DjLCDSIM***. Цей стимулятор можна знайти в мережі Internet за адресою: www.geocities.com. Стимулятор створений у вигляді HTML-сторінки, вигляд якої представлений на рис. 3.28.

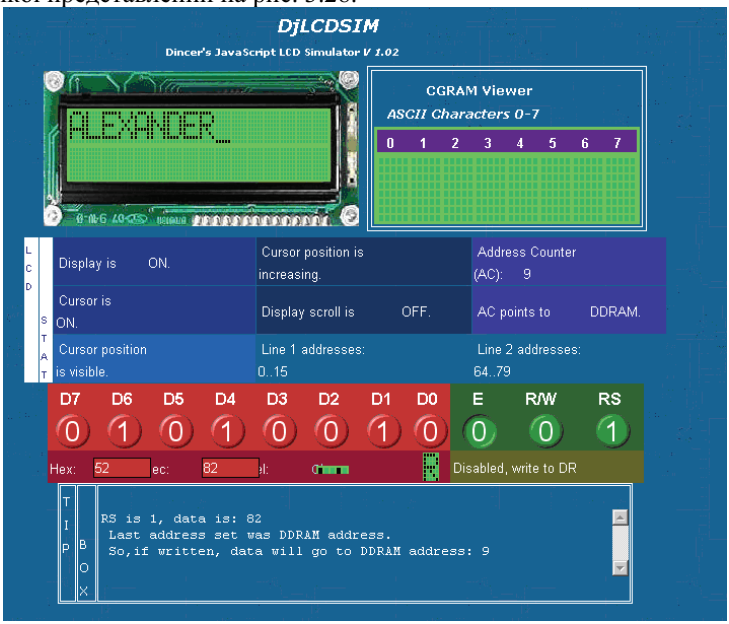


Рис. 3.28.

За допомогою цього симулятора можна вивчати різні режими роботи модуля. У нижній частині розташовані кнопки, за допомогою яких встановлюється рівень сигналів управління (E, R/W, RS), і шина даних (D0–D7). У текстовому полі з'являються підказки про команду, яка встановлена на шині. Також можна побачити значення біт регістра статусу: включений або виключений дисплей, значення лічильника адреси, режим відображення курсора і напрям його зсуву, чи включена прокрутка дисплея та інші. Також відображається вміст пам'яті CGRAM.

На рис. 3.29 представлено вигляд симулятора графічних РКІ.

Рис. 3.29.

За допомогою цього симулятора можна вивчати різні режими роботи графічного модуля. У нижній частині розташовані кнопки, за допомогою яких встановлюється рівень сигналів управління (E, R/W, RS, CSA, CSB) і шина даних (D0–D7). Ці кнопки дублюють відповідні виводи безпосередньо на самому індикаторі. У текстовому полі з'являються підказки про команду, яка встановлена на шині. Також можна побачити, яка половинка індикатора вибрана, поточне значення сторінки пам'яті і позицію курсора по горизонталі в межах сторінки. Послідовно формуючи команди й дані на шині даних, і записуючи їх в індикатор, на стилізованому зображенні індикатора можна побачити результат виконання переданих команд.

Загалом, використання подібного симулятора дозволяє значно прискорити налаштування програми, зрозуміти принципи управління РКІ, а також досліджувати роботу індикатора без необхідності програмування контролера.

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ СХЕМ

Одним з основних етапів проектування та моделювання систем, зокрема електронних систем, є параметричний аналіз. Параметр об'єкта – це величина, що характеризує деяку властивість об'єкта або режим його функціонування [32]. В контексті електронних систем об'єктом є електронна схема. Процедура параметричного аналізу полягає в дослідженні проектованої електронної системи, що спрямовано на отримання корисної інформації про її властивості. При цьому опис електронної системи, в загальному випадку, представляє собою математичну модель системи, яка може бути представлена у вигляді системи рівнянь або в іншому вигляді. Нестаціонарність характеристик та параметрів окремих елементів, що входять до складу електронної системи, впливає на характеристики всієї системи та може призвести до погіршення якості її функціонування при дії зовнішній факторів. Отже метою параметричного аналізу є перевірка працездатності електронної системи при зміні параметрів її елементів в деякому діапазоні, що може бути обумовлено дією зовнішніх факторів (наприклад, температури). Специфіка завдання параметричного аналізу електронних систем полягає у використанні спеціальних методів, відповідних структур і технічній реалізації таких систем. До них можна віднести дослідження області стійкості систем зі зворотним зв'язком, аналіз чутливості систем, представлених в частотної області, оцінка робастності систем та ін [32].

Розвиток інформаційних технологій та їх тісна інтеграція з електронікою дозволили реалізувати завдання проектування, моделювання та аналізу систем на якісно новому рівні, з використанням програмних засобів, що мають графічний інтерфейс користувача. Графічний інтерфейс програми для моделювання та аналізу електронних систем – це інструмент, який покликаний полегшити організацію досліджень, розуміння отриманих результатів моделювання та їх передавання для використання в зовнішніх додатках (Matlab, Matcad та ін.).

4.1. Програмні засоби моделювання електричних кіл постійного струму

У цьому параграфі розглядається застосування законів Кірхгофа для вирішення завдання розрахунку довільного електричного кола в класичній постановці: для заданої схеми з відомими параметрами (ЕРС і струми джерел, опору приймачів енергії) потрібно розрахувати струми.

Рекомендований порядок розрахунку електричного кола:

1. Виконується аналіз кола, тобто визначається кількість вузлів «$У$», гілок «$В$», кількість особливих гілок (гілки з нульовим опором «$В_0$», гілки з відомим струмом «$В_Т$»), вибираються довільні напрямки невідомих струмів гілок.

2. Записується система з $N_{\text{I}} = У–1$ лінійно незалежних рівнянь за I законом Кірхгофа для всіх крім одного будь-якого вузлів.

3. Недостаюча кількість лінійно незалежних рівнянь записується на основі II закону Кірхгофа для незалежних контурів, що не містять джерела струму: $N_{\text{II}} = B–(У–1)–B_T$.

4. Розв'язують систему $N = N_{\text{I}} + N_{\text{II}}$ рівнянь і визначають величини струмів.

5. Правильність рішення перевіряють складанням балансу потужностей: $\sum E \cdot I + \sum J_k \cdot U_k = \sum I^2 \cdot r$.

Напруга на клемах джерела струму визначається за II законом Кірхгофа, а відносна похибка по балансу потужностей не повинна перевищувати (3–5)%.

ПРИКЛАД 4.1. Методом рівнянь Кірхгофа розрахувати струми в схемі рис. 4.1, а, при наступних параметрах [33]:

$J_1 = 10$ *A*, $E_2 = 100$ *B*, $E_6 = 300$ *B*, $r_3 = r_4 = r_5 = r_6 = 20$ Ом.

На схемі вказані довільно обрані напрямки U_k, I_2, I_3, I_4, I_5, I_6.

Для цієї схеми кількість гілок з незалежними струмами $B = 5$, кількість вузлів $У = 4$.

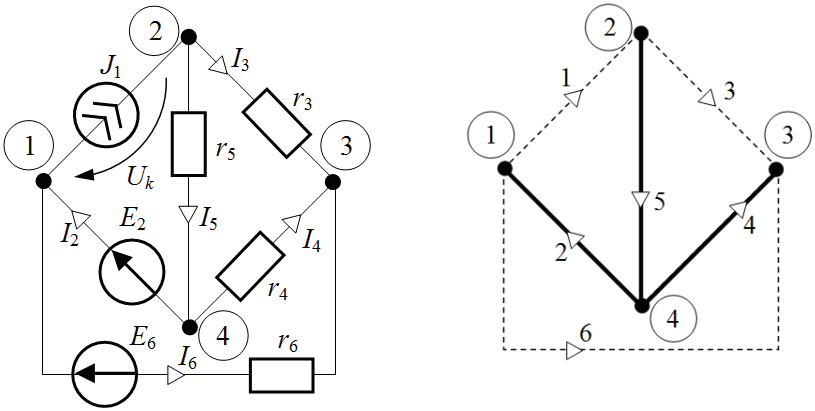


Рис. 4.1.

Рівняння, записані за I законом Кірхгофа для вузлів «1», «2», «3», відповідно, мають вигляд [33]:

$$J_1 – I_2 + I_6 = 0;\ I_3 + I_5 – J_1 = 0;\ –I_3 – I_4 – I_6 = 0. \qquad (4.1)$$

Для складання лінійно незалежних рівнянь за II законом Кірхгофа скористаємося напрямленим графом електричного кола рис. 4.1, б. Цей граф побудований з урахуванням однієї особливості вихідного електричного кола: схема має гілку з джерелом струму J_1, внутрішній опір якого $r_B = \infty$ і ЕРС $E_B = \infty$. З цієї причини для головного контуру неможливо записати

рівняння за II законом Кірхгофа виду $\sum I \cdot r = \sum E$, оскільки воно становиться невизначеним:

$$I_5 \cdot r_5 + J_1 \cdot r_B = E_2 + E_B.$$

При цьому необхідно пам'ятати, що нескінченості, що стоять різних частинах рівняння можна об'єднати в лівій частині наведеного рівняння, і їх різниця дасть кінцеве число U_k, що випливає з рівняння, записаного за II законом Кірхгофа для контуру, що включає напругу $U_k = E_B - J_1 \cdot r_B$ замість внутрішнього кола джерела струму: $I_5 \cdot r_5 - U_k = E_2$.

Таким чином, вказаний контур використовується для визначення напруги $U_k = -E_2 + I_5 \cdot r_5$ на клемах джерела струму, а для складання рівнянь для рорахунку невідомих струмів гілок використовуємо головні контури з невідомими струмами гілок зв'язку, обходячи їх в напрямку струмів гілок зв'язку.

Для контура з гілками 3, 4, 5 отримуємо:

$$I_3 \Delta r_3 - I_4 \Delta r_4 - I_5 \cdot r_5 = 0, \quad (4.2)$$

для контура з гілками 6, 4, 2

$$I_6 \cdot r_6 - I_4 \cdot r_4 = -E_6 + E_2. \quad (4.3)$$

Система розрахункових рівнянь (4.1), (4.2), (4.3) включає 5 рівнянь. Зменшимо число рівнянь в системі, використовуючи метод підстановки. Для цього з рівнянь (4.1) виразимо струми гілок дерева графа через струми гілок зв'язку: $I_2 = J_1 + I_6$; $I_5 = J_1 - I_3$; $I_4 = -I_3 - I_6$.

Отримані вирази підставимо в (1.2) і (1.3), та приведемо подібні. Отримаємо

$$I_3 \cdot (r_3 + r_4 + r_5) + I_6 \cdot r_4 - J_1 \cdot r_5 = 0, \quad (4.4)$$

$$I_6 \cdot (r_6 + r_4) + I_3 \cdot r_4 = -E_6 + E_2. \quad (4.5)$$

Підставимо числа в отриману систему і перенесемо відоме $J_1 \cdot r_5$ в праву частину (4.4):

$$\begin{cases} 60 \cdot I_3 + 20 \cdot I_6 = 200, \\ 20 \cdot I_6 + 40 \cdot I_3 = -200. \end{cases}$$

Розв'яжемо цю систему методом Крамера. Детермінант системи

$$\Delta = \begin{vmatrix} 60 & 20 \\ 20 & 40 \end{vmatrix} = 60 \cdot 40 - 20 \cdot 20 = 2000.$$

Допоміжні детермінанти

$$\Delta_3 = \begin{vmatrix} 200 & 20 \\ -200 & 40 \end{vmatrix} = 200 \begin{vmatrix} 1 & 20 \\ -1 & 40 \end{vmatrix} = 200 \cdot (40 + 20) = 12000$$

$$\Delta_6 = \begin{vmatrix} 60 & 200 \\ 20 & -200 \end{vmatrix} = 200 \cdot \begin{vmatrix} 60 & 1 \\ 20 & -1 \end{vmatrix} = 200 \cdot (-60 - 20) = -16000$$

Струми гілок зв'язку:

$$I_3 = \frac{\Delta_3}{\Delta} = \frac{12000}{2000} = 6A$$

$$I_6 = \frac{\Delta_6}{\Delta} = \frac{16000}{2000} = 8A$$

Струми гілок дерева: $I_2 = J_1 + I_6 = 10 - 8 = 2\ A$,

$I_4 = -I_3 - I_6 = -6 - (-8) = 2\ A$,

$I_5 = J_1 - I_3 = 10 - 6 = 4\ A$.

Розраховані значення будуть використані для перевірки правильності роботи програми. Приступимо до програмної реалізації алгоритму розрахунку струмів в схемі. Після створення нового проекту в редакторі середовища розробки Visual Studio .NET потрібно набрати наступну програму:

```
#include "stdafx.h"
#include <iostream>

using namespace std;
int main()
{
	double I1; // оголошення змінної, в якій зберігатиметься значення
// струму джерела, введене користувачем
	cout << "Enter value of I1:"; // виведення на екран текстової строки
// "Введіть значення I1:"
	cin >> I1; // зчитування в змінну I1 значення, введеного користувачем
	double E2, E6; // оголошення змінних, в яких зберігатиметься значення ЕРС,
// введені користувачем
	cout << "Enter value of E2:"; // виведення на екран текстової строки
// "Введіть значення E2:"
	cin >> E2; // зчитування в змінну E2 значення, введеного користувачем
	cout << "Enter value of E6:"; // виведення на екран текстової строки
// "Введіть значення E6:"
	cin >> E6; // зчитування в змінну E6 значення, введеного користувачем
	double r3, r4, r5, r6; // оголошення змінних для зберігання
// значень опорів резисторів,
// введених користувачем
	cout << "Enter value of r3:"; // виведення на екран текстової строки
// "Введіть значення r3:"
	cin >> r3; // зчитування в змінну r3 значення, введеного користувачем
	cout << "Enter value of r4:"; // виведення на екран текстової строки
// "Введіть значення r4:"
	cin >> r4; // зчитування в змінну r4 значення, введеного користувачем
	cout << "Enter value of r5:"; // виведення на екран текстової строки
// "Введіть значення r5:"
	cin >> r5; // зчитування в змінну r5 значення, введеного користувачем
	cout << "Enter value of r6:"; // виведення на екран текстової строки
// "Введіть значення r6:"
	cin >> r6; // зчитування в змінну r6 значення, введеного користувачем
	double r345; // оголошення змінної, в якій зберігатиметься значення
// опору контура з резисторами r3, r4 та r5
	r345 = r3 + r4 + r5; // запис в змінну r345 суми опорів r3, r4 та r5
	double r46; // оголошення змінної, в якій зберігатиметься значення
// опору контура з резисторами r4 та r6
```

```
        r46 = r4 + r6; // запис в змінну r46 суми опорів r4 та r6
        double Delta, Delta3, Delta6; // оголошення змінних для зберігання
// розрахованих значень детермінантів матриць
        // розрахунок детермінантів матриць
        Delta = r345 * r46 - r4 * r4;
        Delta3 = (I1*r5)*r46 - r4 * (E2 - E6);
        Delta6 = r345 * (E2 - E6) - (I1*r5)*r4;
        double I2, I3, I4, I5, I6; // оголошення змінних, в яких зберігатимуться
// розраховані значення струмів
        // розрахунок струмів
        I3 = Delta3 / Delta;
        I6 = Delta6 / Delta;
        I2 = I1 + I6;
        I4 = -I3 - I6;
        I5 = I1 - I3;
        // виведення отриманих результатів на екран монітору
        cout << "I2=" << I2 << " A\n";
        cout << "I3=" << I3 << " A\n";
        cout << "I4=" << I4 << " A\n";
        cout << "I5=" << I5 << " A\n";
        cout << "I6=" << I6 << " A\n";

        // формування паузи для того, щоб вікно програми не закривалося відразу після
// завершення, і користувач мав би змогу побачити на екрані
// отримані результати
        system("pause");
        return 0;
}
```

Після запуску програми на виконання та введення параметрів елементів схеми, у вікні відобразяться розраховані значення струмів, як показано на рис. 4.2. Як видно з результатів виконання програми, отримані значення повністю співпадають з розрахованими аналітично, що підтверджує коректність реалізації алгоритму розрахунку схеми.

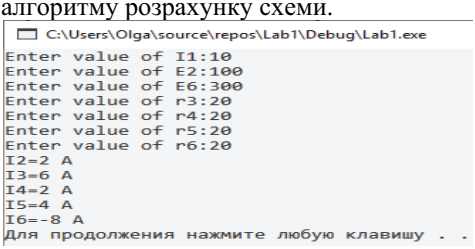


Рис. 4.2.

На схемі рис. 4.3 різниця потенціалів точок *a* і *b* називається напругою між цими точками: $U_{ab} = \varphi_a - \varphi_b$. В системі одиниць СІ напругу, як і ЕРС, виражають в Вольтах (В). Напруга є скалярна величина, якій приписується

певний напрям. Зазвичай під позитивним напрямком напруги розуміють напрямок від точки ланцюга з більшим потенціалом до точки з меншим потенціалом.

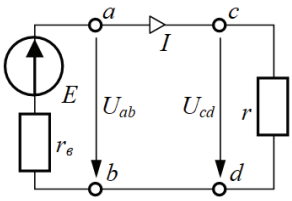


Рис. 4.3.

У відповідності до закону збереження енергії для рис. 4.3

$E = U_{ab} + I \cdot r_в$, звідки $U_{ab} = E - I \cdot r_в$.

Тобто напруга на зажимах джерела відрізняється від ЕРС на величину внутрішнього падіння напруги. При зміні опору приймача змінюється струм кола, відповідно, і напруга на зажимах джерела.

Параметричний аналіз полягає в тому, що деякий показник (струм, напруга), що характеризує режим роботи схеми, потрібно проаналізувати для різних параметрів схеми (одного або декількох), наприклад, при зміні опору навантаження, або ЕРС. Для реалізації параметричного аналізу використовується одне або кілька полів введення даних, що налаштовуються, за допомогою яких обчислюється значення такого показника. Зазвичай потрібно вказати початкове значення параметра, кінцеве значення та крок його зміни. виконується власне аналіз схеми шляхом виконання розрахунків для кожного із заданих значень, групування або сортування даних (наприклад, у вигляді таблиці).

В якості прикладу розглянемо параметричний аналіз схеми рис. 4.3 для різних значень опору навантаження. Потрібно визначити струм в колі та напругу між вузлами *c* і *d* схеми для різних значень опору.

На рис. 4.4 представлений приклад діалогового вікна програми, спроектований в середовищі Visual Studio .NET.

На діалоговому вікні розташовані елементи Edit Control для введення параметрів схеми. Всі поля введення підписано для того, щоб користувач розумів, який саме параметр та в яких одиницях вимірюванні задається. В правій частині діалогового вікна розташований елемент List Box (список), в якому будуть відображатися результати розрахунків. Кнопка Button з написом «Calculate» запускає процес параметричного аналізу схеми і відображення результатів.

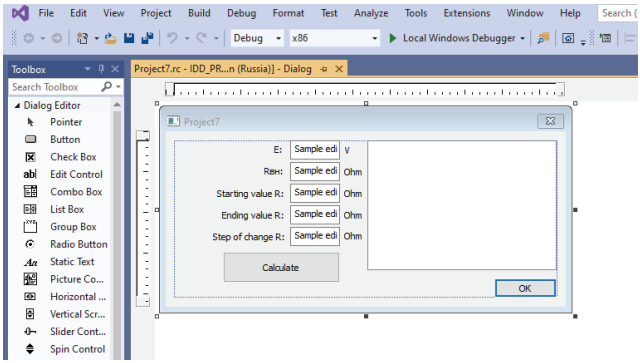


Рис. 4.4.

Для того, щоб мати доступ з програми до вмісту елементів керування, з кожним з них потрібно зв'язати змінну. Для цього слід виділити потрібний елемент та викликати контекстне меню шляхом натискання правої кнопки миші на ньому. З контекстного меню потрібно вибрати команду «Add Variable», як показано на рис. 4.5.

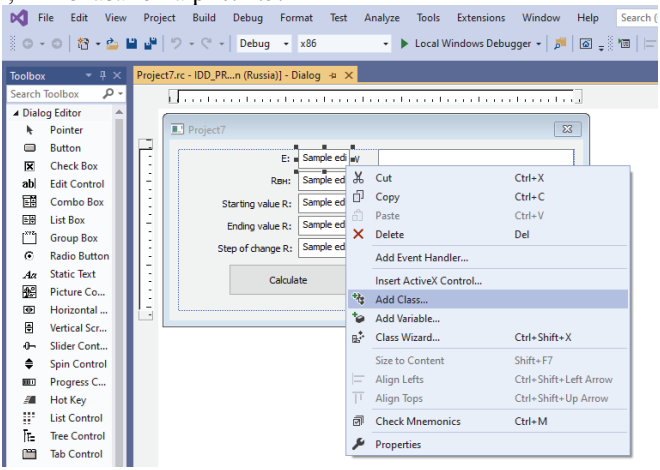


Рис. 4.5.

У діалоговому вікні, що з'явиться (рис. 4.6), потрібно вибрати категорію змінної з меню «Category». Для всіх елементів введення Edit Control слід вибирати категорію «Value». Далі, в полі «Name» вводиться назва змінної. Для ЕРС змінна називається m_E, для струму I назва змінної m_I, внутрішній опір джерела m_Rin. Початкове значення опору навантаження m_Rstart (рис. 4.7, а), кінцеве значення опору навантаження m_Rend (рис. 4.7, б), приростення опору (крок) m_dR.

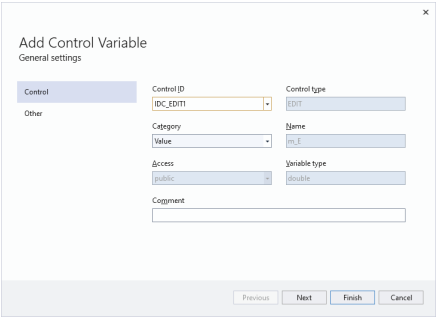
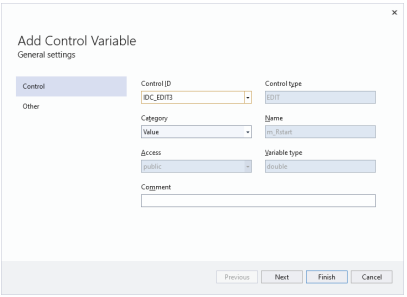
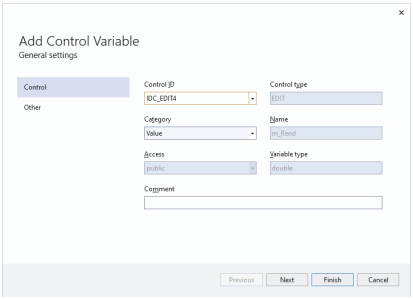


Рис. 4.6.



а



б

Рис. 4.7.

З елементом керування список ListBox потрібно зв’язати змінну іншої категорії – «Control», як показано на рис. 4.8. Назва цієї змінної m_list.

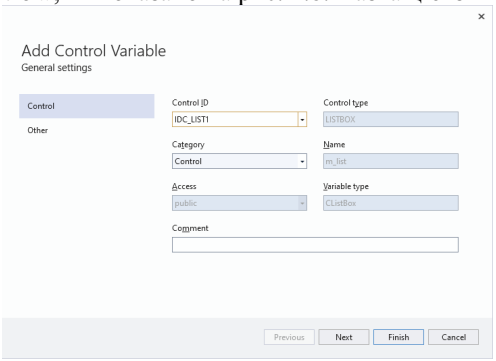


Рис. 4.8.

Після того, як для кожного елемента керування діалогового вікна створено змінну, потрібно додати обробник події натискання на кнопку

«Calculate». Для цього достатньо двічі натиснути ліву кнопку миші на цьому елементі. Відкривається текстовий редактор, в яму потрібно набрати наступний текст програми:

```
void CProject7Dlg::OnBnClickedButton1()
{
// зчитування в змінні значень з елементів керування,
// розташованих на діалоговому вікні
UpdateData(true);

// очищення списку, в який виводяться результати розрахунків
m_list.ResetContent();

// оголошення змінної для зберігання результатів розрахунку струму
        double I;

// оголошення змінної для зберігання розрахованого значення напруги на
// навантаженні
        double Ucd;

// створення об'єкту для роботи з текстовими рядками; цей об'єкт
// буде використовуватися при перетворенні чисел в строки
        CString str;

        // в циклі виконуються розрахунки струму в колі та напруги на
// навантаженні для  різних значень опору навантаження і незмінному
// внутрішньому опорі джерела
        // початкове значення опору навантаження, його кінцеве значення
        // та крок зміни задаються користувачем шляхом введення значень в
// елементи керування, які розташовані на діалоговому вікні
        for (double r = m_Rstart; r <= m_Rend; r = r + m_dR)
        {
         I = m_E / (m_Rin + r);          // розрахунок струму в колі
         Ucd = I * r;                    // розрахунок напруги на навантаженні

         // перетворення чисел в строки та форматування результатів розрахунку
         // для зручності сприйняття користувачем
         str.Format(_T("R=%.2f Ом; I=%.2f A; Ucd=%.2f B"), r, I, Ucd);
 // додавання строки в список та її відображення
         m_list.AddString(str);
        }
}
```

Після запуску програми на виконання з'явиться діалогове вікно, представлене на рис. 4.9. В елементи введення слід ввести значення параметрів схеми і діапазон зміни опору навантаження, після чого натиснути на кнопку «Calculate». В списку праворуч відобразяться результати розрахунків для різних значень опорів схеми.

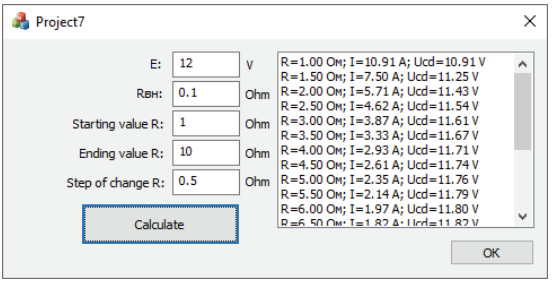
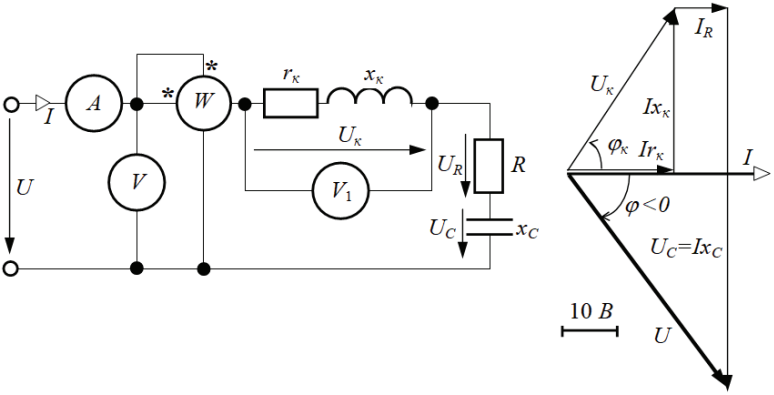


Рис. 4.9.

4.2. Розробка програм для параметричного аналізу лінійних кіл змінного струму

ПРИКЛАД 4.2. Знайти струми та напруги в електричному колі на рис. 4.10, якщо: активний опір котушки $r_к = 4$ Ом, індуктивний опір котушки $x_к = 6$ Ом, активний опір реостата $R = 2$ Ом, ємнісний опір конденсатора $x_C = 14$ Ом, напруга мережі змінного струму $U = 50$ *В*. При незмінній напрузі мережі і параметрах $r_к$, $x_к$, R в широких межах змінюється опір конденсатора $x_C(0\ |\infty)$ [33]. Побудувати резонансні криві $I(x_C)$, $U_L(x_C)$, $U_C(x_C)$.



а б

Рис. 4.10.

Струм в послідовному ланцюзі на рис. 4.10, а, розраховується наступним чином:

$$I = \frac{U}{\sqrt{(r_k + R)^2 + (x_k - x_c)^2}} = \frac{50}{\sqrt{6^2 + (6 - x_c)^2}}\ A$$

Напруга на індуктивності: $U_L = I \cdot x_L = I \cdot 6$ *В*,
напруга на конденсаторі:

$$U_c = I \cdot x_C = \frac{U \cdot x_C}{\sqrt{(r_k + R)^2 + (x_k - x_C)^2}} = \frac{50 \cdot x_C}{\sqrt{6^2 + (6 - x_C)^2}} \text{ В}$$

Результати розрахунку резонансних кривих представлені в табл. 4.1.

Таблиця 4.1.

x_C, *Ом*	0	2	4	6	8	10	12	14	20	30	∞
I, A	5,89	6,93	7,9	8,33	7,9	6,93	5,89	5	3,28	2,02	0
U_L, *B*	35,4	41,6	47,4	50	47,4	41,6	35,4	30	19,7	12,1	0
U_C, *B*	0	13,9	31,6	50	63,2	69,3	70,7	70	65,6	60,6	$U = 50$

В табл. 4.1 виділений стовпець, коли $x_к = x_C = 6$ Ом і в колі наступає резонанс напруг $U_L = U_C$. При цьому вхідний опір кола мінімальний:

$$Z_{вх\ min} = r_к + R = 6 \text{ Ом},$$

а струм максимальний:

$$I_{max} = U/Z_{вхmin} = 50/6 = 8{,}33 \text{ А}.$$

Крива напруги U_L повторює по формі криву струму, оскільки $x_к$=const, і тоді $U_{Lmax} = I_{max} \cdot x_к = 8{,}33 \cdot 6 = 50$ *B*.

Знайдемо максимальне значення U_{Cmax} в залежності від x_C, дослідивши криву $U_C(x_C)$ на максимум.

Координата x_C при $U_C = U_{Cmax}$ визначається рівнянням:

$$\frac{dU_C}{dx_C} = 0, \text{ або: } \sqrt{(r_k - R)^2 + (x_k - x_C)^2} + \frac{x_C \cdot (x_k - x_C)}{\sqrt{(r_k + R)^2 + (x_k - x_c)^2}} = 0,$$

$$\text{або: } (r_к + R)^2 + x_к^2 - 2\, x_к\, x_C + x_C^2 + x_к\, x_C - x_C^2 = 0.$$

Звідки

$$x_{C_{|U_C = U_{Cmax}}} = \frac{(r_k + R)^2 + x_k^2}{x_k} = \frac{6^2 + 6^2}{6} = 12 \text{ Ом}$$

Резонансні криві приведені на рис. 4.11 [33].

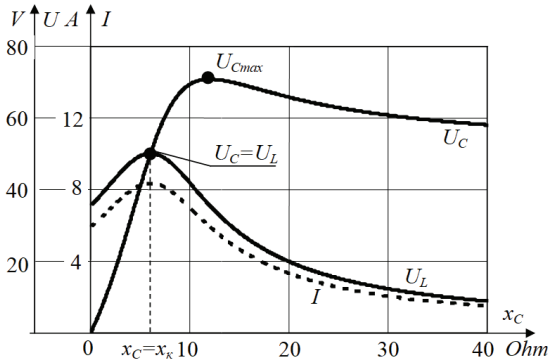


Рис. 4.11.

Для розробки програми, що виконує побудову резонансних кривих у відповідності до завдання, перш за все потрібно створити новий проект на основі діалогу у середовищі розробки Microsoft Visual Studio .NET, як було показано раніше. Проаналізувавши завдання, можна зрозуміти, що від користувача потребується введення діапазону зміни реактивного опору конденсатора та кроку зміни цього опору. Отже, на діалоговому вікні буде розташовано 3 елементи Edit Control для введення початкового значення x_C, кінцевого значення x_C та Δx_C. Також на діалоговому вікні потрібно розмістити елемент Static Text, на якому буде намальовано відповідні резонансні криві. Всі розрахунки та візуалізація результатів буде відбуватися в обробнику події натискання на кнопку Calculate. На рис. 4.12 представлено зовнішній вигляд макету діалогового вікна, створеного у редакторі ресурсів середовища розробки.

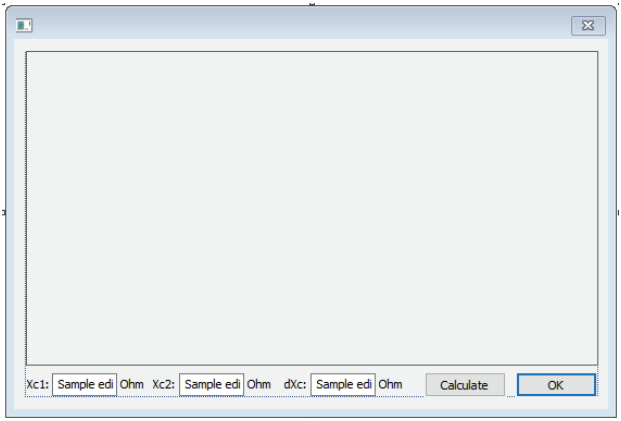


Рис. 4.12.

Після того, як макет діалогового вікна створено, необхідно з елементами керування зв’язати змінні для того, щоб мати можливість програмного доступу до їх вмісту. Для цього використовується Class Wizard, вікно якого можна відкрити натиснувши комбінацію клавіш Ctrl+Shift+X. Додавання змінних, зв’язаних з елементами керування. виконується на вкладці Member Variables, як показано на рис. 4.13.

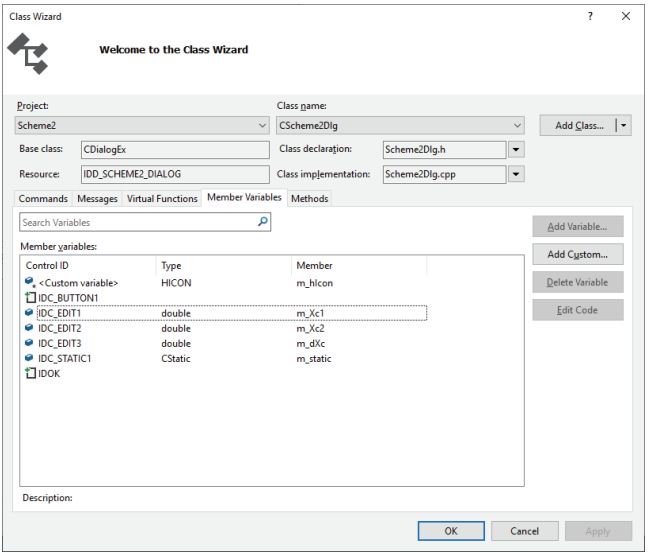


Рис. 4.13.

Для того, щоб зв'язати з елементом керування змінну, потрібно вибрати відповідний елемент зі списку та натиснути на кнопку Add Variable. Для елементів керування Edit Control всі змінні відносяться до категорії value та мають тип double (рис. 4.14). З елементом керування Static Text зв'язана змінна категорії Control, а тип змінної CStatic. Якщо у списку у цьому вікні відсутній елемент IDC_STATIC1, тоді у вікні властивостей цього елемента потрібно відредагувати його ID, змінивши на IDC_STATIC1.

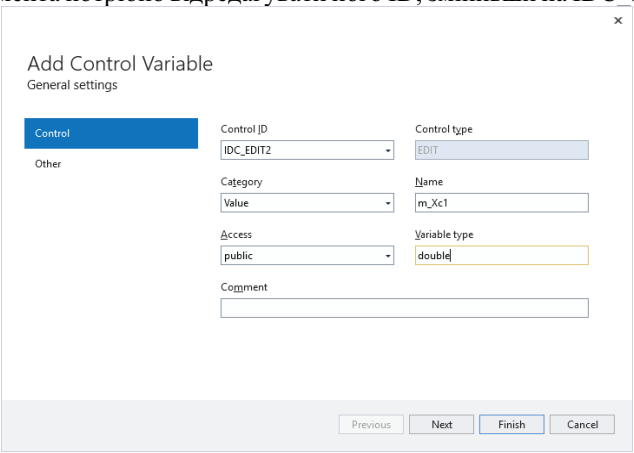


Рис. 4.14.

Після того, як з усіма елементами керування діалогового інтерфейсу зв'язані відповідні змінні, можна переходити до розробки програми. Для цього необхідно додати обробник натискання на кнопку Calculate, в якому буде розташовано весь програмний код. Це можна зробити подвійним натисканням лівої кнопки миші на цій кнопці. У текстовому редакторі, що відкриється після цього, потрібно набрати наступний код:

```
void CScheme2Dlg::OnBnClickedButton1()
{
	const double rk = 4; // реактивний опір котушки індуктивності
	const double xk = 6; // активний опір котушки індуктивності
	const double R = 2;     // опір резистора
	const double U = 50; // напруга живлення схеми

	// зчитування даних з елементів керування у відповідні змінні
	UpdateData(true);

	// створення контексту пристрою для роботи з графікою
	CClientDC dc(&m_static);
	// оголошення об'єкта для зберігання інформації про прямокутну область
	CRect rect;
	// отримання інформації про розміри елемента Static Text,
	// в якому відбуватиметься малювання графіків
	m_static.GetClientRect(&rect);
	// в змінній W зберігається значення ширини прямокутної області
	int W = rect.Width();
	// в змінній W зберігається значення висоти прямокутної області
	int H = rect.Height();

	// заповнення прямокутної області білим кольором
	dc.FillRect(&rect, WHITE_BRUSH);

	// малювання вертикальних ліній сітки
	for (int x = 0; x < W; x = x + W / 4) {
		dc.MoveTo(x, 0);
		dc.LineTo(x, H);
	}
	// малювання горизонтальних ліній сітки
	for (int y = 0; y < H; y = y + H / 4) {
		dc.MoveTo(0, y);
		dc.LineTo(W, y);
	}

	// в змінній Xc зберігається початкове значення опору конденсатора
	double Xc = m_Xc1;
	// розрахунок кількості кроків для виконання розрахунків
	int num_of_steps = (int)(m_Xc2 - m_Xc1) / m_dXc;
	// розрахунок значення прирощення координати x
	int dx = (int) W / num_of_steps;
	// початкове значення координати x
	int x = 0;
```

```
	// створення об’єкту пера червоного кольору, тонка пунктирна лінія
	CPen redPen;
	redPen.CreatePen(PS_DASH, 1, RGB(255,0,0));

	// створення об’єкту пера зеленого кольору, товста суцільна лінія
	CPen greenPen;
	greenPen.CreatePen(PS_SOLID, 2, RGB(0, 255, 0));

	// створення об’єкту пера синього кольору, товста суцільна лінія
	CPen bluePen;
	bluePen.CreatePen(PS_SOLID, 2, RGB(0, 0, 255));

	// оголошення змінної для зберігання розрахованого значення струму
	double I;
	// змінна, в якій зберігається значення коефіцієнту масштабування
	// оскільки на одному графіку виводяться різні величини (струм та напруги)
	// для більш наглядного відображення результатів потрібно масштабувати
	// розраховані значення
	double scaleI = 20;
	// розрахунок початкового значення струму при значенні опору конденсатора,
	// яке було введено в поле Xc1
	double prevI = U / sqrt(pow(rk + R, 2) + pow(xk - m_Xc1, 2))*scaleI;

	// оголошення змінної для зберігання розрахованого значення
	// напруги на котушці індуктивності
	double Ul;
	// коефіцієнт масштабування напруги, що використовується
	// при відображенні резонансних кривих напруги
	double scaleU = 4;
	// розрахунок початкового значення напруги на індуктивності
	// при значенні опору конденсатора, яке було введено в поле Xc1
	double prevUl = prevI*xk*scaleU/scaleI;

	// оголошення змінної для зберігання розрахованого значення
	// напруги на конденсаторі
	double Uc;
	// розрахунок початкового значення напруги на конденсаторі
	// при значенні опору конденсатора, яке було введено в поле Xc1
	double prevUc = prevI*m_Xc1*scaleU/scaleI;

	// зміна значення опору конденсатора на величину кроку прирощення,
	// яке було введено в елемент керування на діалозі
	Xc = Xc + m_dXc;

	// перевірка умови: початкове значення опору конденсатора
// повинно відрізнятися
	// від кінцевого значення, а крок зміни опору не повинен дорівнювати 0;
	// в іншому випадку видається повідомлення та подальше виконання
	// програми припиняється
	if ((m_Xc1 == m_Xc2) || (m_dXc == 0)) {
		MessageBox(_T("Invalid values!"));
```

```
            return;
        }

        // в циклі відбувається розрахунок значень струму та напруг в колі
        // при різних значеннях опору конденсатора
        while (Xc <= m_Xc2) {
            // вибір у контекст пристрою червоного пера
            dc.SelectObject(&redPen);
            // розрахунок значення струму в колі
            I = U / sqrt(pow(rk + R, 2) + pow(xk - Xc, 2))*scaleI;
            // малювання лінії, що з'єднує два сусідніх значення струму
            dc.MoveTo(x+dx, H-I);
            dc.LineTo(x, H-prevI);

            // вибір у контекст пристрою зеленого пера
            dc.SelectObject(&greenPen);
            // розрахунок значення напруги на конденсаторі
            Uc = I * Xc * scaleU / scaleI;
            // малювання лінії, що з'єднує два сусідніх значення напруги Uc
            dc.MoveTo(x + dx, H - Uc);
            dc.LineTo(x, H - prevUc);

            // вибір у контекст пристрою синього пера
            dc.SelectObject(&bluePen);
            // розрахунок значення напруги на котушці індуктивності
            Ul = I * xk * scaleU/scaleI;
            // малювання лінії, що з'єднує два сусідніх значення напруги Ul
            dc.MoveTo(x + dx, H - Ul);
            dc.LineTo(x, H - prevUl);

            // запис розрахованих значень струму та напруг у змінні
            // для того, щоб в наступній ітерації циклу побудувати наступний
            // фрагмент лінії та забезпечити її цілісність
            prevI = I;
            prevUc = Uc;
            prevUl = Ul;
            // зміна значення ємності конденсатора на величину кроку
            Xc = Xc + m_dXc;
            // зміна значення координати x для побудови графіка
            x = x + dx;
        }
}
```

Після компіляції проекту, задавши відповідні значення початкового опору конденсатора (0 Ом), кінцевого значення опору (30 Ом) та кроку його зміни (2 Ом) та натиснувши на кнопку Calculate, на діалоговому вікні відобразяться резонансні криві, як показано на рис. 4.15.

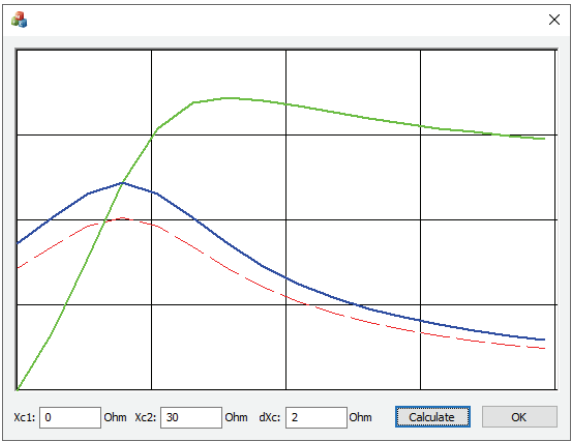


Рис. 4.15.

Порівнюючи отримані результати при виконанні програми з розрахунковими можна зробити висновок про їх повну ідентичність. Що свідчить про правильність програмної реалізації обчислень.

4.3. Програмні засоби для модельних досліджень резонансних явищ в електронних системах

ПРИКЛАД 4.3. На рис. 4.16 представлена схема, що містить реактивні елементи (котушка індуктивності та конденсатор). В цій схемі можливий режим резонансу напруг. Необхідно побудувати резонансні криві $I(x_C)$, якщо ємність конденсатора змінюється в широкому діапазоні, що задається користувачем.

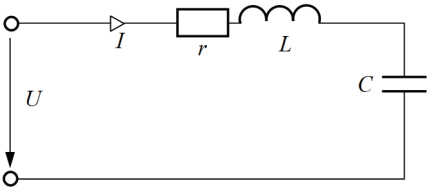


Рис. 4.16.

На рис. 4.17 представлено спроектований інтерфейс діалогового вікна програми. Більшу частину інтерфейсу займає елемент Static Text, який використовується для малювання на ньому резонансних кривих. Справа в тому, що для роботи з графікою в бібліотеці MFC є клас CClientDC. Для створення об'єкту цього класу в конструктор потрібно передати вказівник на деякий об'єкт, що наслідується від класу CWnd. Оскільки всі елементи інтерфейсу наслідуються від цього класу, тому в якості об'єкту, який передається в конструктор при створенні екземпляру класу CClientDC, можна

було б використати будь-який елемент керування. Однак елемент Static Text є найпростішим з усіх елементів, тому біло обрано саме його. Для того, щоб мати можливість створити змінну-член класу діалогу, яка буде зв'язана з елементом Static Text, необхідно у вікні властивостей цього елементу змінити ідентифікатор (поле ID) з IDC_STATIC, який задається за замовчуванням, на IDC_STATIC1. В іншому випадку зв'язати змінну з цим елементом і, відповідно, використовувати його в програмі, не вдасться.

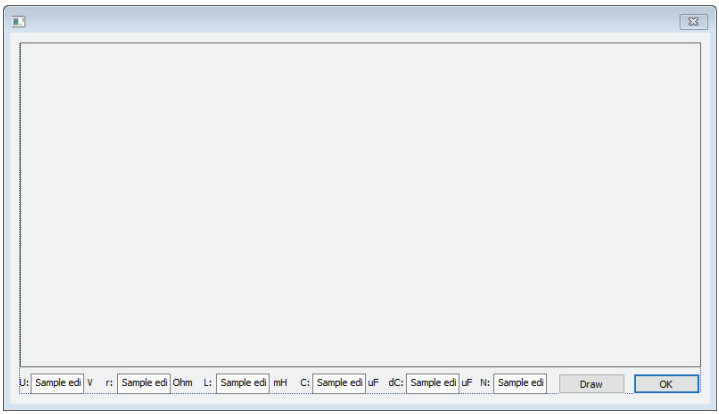


Рис. 4.17.

В нижній частині діалогового вікна розташовані елементи керування Edit Control, за допомогою яких задається вхідна напруга, значення активного опору, індуктивність котушки, значення початкової ємності конденсатора, крок зміни значення ємності та кількість резонансних кривих, що потрібно побудувати. З усіма елементами керування за допомогою Class Wizard, який можна відкрити шляхом натискання комбінації клавіш Ctrl+Shift+X, зв'язані змінні, що мають категорію Value та тип double. Змінна m_N, в якій зберігається введена користувачем кількість резонансних кривих, що треба побудувати, має тип int. Також на діалоговому вікні розташовано кнопку з написом «Draw», натискання на яку призведе до виконання необхідних розрахунків та побудови графіків.

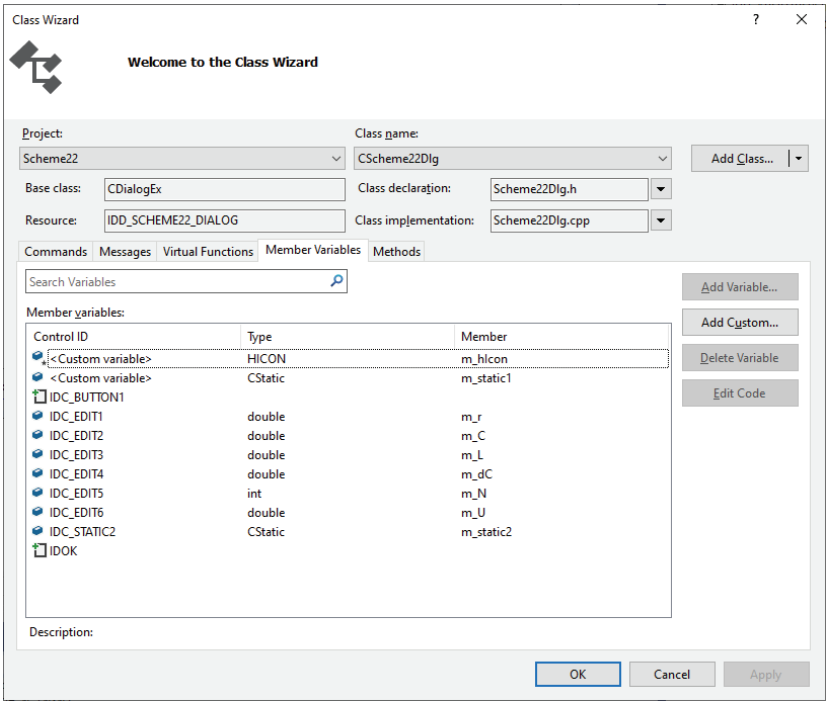


Рис. 4.18.

Для того, щоб в програмі була можливість використання математичних функцій, а також для робот з комплексними числами, потрібно підключити відповідні бібліотеки. Це робиться шляхом додавання в початок файлу вихідного коду класу діалогу, що має розширення .cpp, наступних рядків:

```
#define _USE_MATH_DEFINES
#include "math.h"       // підключення бібліотеки для роботи з математичними функціями

// підключення стандартної бібліотеки для роботи з комплексними числами
#include <complex>
using namespace std;
using namespace std::complex_literals;
```

Код, що відповідає за розрахунки та побудову резонансних кривих, буде розміщено в функції-обробнику події натискання на кнопку «Draw». Для того, щоб додати обробник натискання на кнопку, в редакторі діалогових вікон слід двічі натиснути ліву кнопку миші на потрібний елемент керування (кнопку з надписом «Draw»), після чого відкривається текстовий редактор, і курсор буде встановлено в те місце, куди потрібно ввести текст

програми. Реалізацію функції-обробника натискання на кнопку «Draw» представлено далі.

```
void CScheme22Dlg::DrawGraph()
{
	// виклик функції малювання масштабної сітки (лінійний масштаб)
	DrawGrid();

	// створення об’єкта контексту пристрою для роботи з графікою
	// параметр конструктора - вказівник на елемент керування,
	// на якому буде виводитися графіка - в даному випадку це Static Text
	CClientDC dc(&m_static2);
	// створення об’єкту для зберігання інформації про прямокутну область
	// (розміри області малювання графіка струму,
// тобто розміри елементу Static Text)
	CRect rect;
	// отримання інформації про розміри елементу Static Text
	m_static2.GetClientRect(&rect);
	// значення висоти та ширини елементу Static Text копіюються в окремі
	// змінні для зручності їх використання в подальшому
	int height = rect.Height();
	int width = rect.Width();

	// створення об’єкту для зберігання комплексного значення
	// віходного опору кола
	complex<double> Z;

	// створення об’єкту для зберігання значення комплексного значення струму
	complex<double> I;
	// максимальне значення частоти (5 кГц), до якої виконується побудова графіку
	int w_max = 5000;
	// змінні для зберігання розрахованого значення модуля струму
	double mod;
	// змінна, в якій зберігається значення масштабу, що дорівнює добутку
	// висоти області побудови на коефіцієнт, що дорівнює
// значенню максимального струму
	double scale = height * m_r / m_U;
	// початкове значення частоти 0 Гц
	double w = 0;
	// початкове значення координати x, з якої починається малювання графіку
	int x0 = 0;
	// початкове значення координати y
	int y0 = 0;

	// створення об’єкту пера червоного кольору товщиною 2 пікселі, суцільна лінія
	CPen redPen;
	redPen.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
	// вибір об’єкту пера в контекст пристрою
	dc.SelectObject(&redPen);
	// в циклі відбувається побудова графіку зміни струму
	// для заданого значення ємності конденсатора
```

```
// при зміні частоти від 1 Гц до 5 кГц;
	// в наступній ітерації відбувається збільшення значення ємності та побудова
	// іншого графіку струму для нових параметрів елементів схеми
for (int n = 0; n < m_N; n++)
	{
		x0 = 0;
		y0 = 0;
		// в циклі відбувається побудова графіку;
// ідея полягає в розрахунку координат
		// двох сусідніх точок, які з'єднуються прямою лінією
		// координата x не може перевищувати ширину області побудови
		for (int x = 1; x < width; x++)
		{
			// розрахунок значення кутової частоти
// (це значення не може перевищувати w_max)
			w = 2 * M_PI * x * w_max / width;
			// розрахунок значення комплексу вхідного опору
// з урахуванням того,
			// що значення індуктивності задане в мГн,
// а значення ємності в мкФ
			Z = m_r + (1i * w * m_L * 1e-3 + 1.0 / (1i * w * m_C * 1e-6));
			// розрахунок струму
			I = m_U / Z;
			// отримання значення модуля комплексу струму для поточного
			// значення частоти та його масштабування
			mod = abs(I) * scale;
			// малювання лінії, що з'єднує дві точки - поточну та попередню
			dc.MoveTo(x0, height - y0);
			dc.LineTo(x, height - mod);
			// копіювання координат поточної точки у змінні,
// що містять координати попередньої
			// точки для використання в наступній ітерації циклу
			x0 = x;
			y0 = mod;
		}
		m_C = m_C + m_dC;
	}
}

void CScheme22Dlg::DrawGrid()
{
	// створення об'єкта контексту пристрою для роботи з графікою
	// параметр конструктора - вказівник на елемент керування,
	// на якому буде виводитися масштабна сітка - в даному випадку це Static Text
	CClientDC dc(&m_static2);
	// створення об'єкту для зберігання інформації про прямокутну область
	CRect rect;
	// отримання інформації про розміри елементу Static Text
	m_static2.GetClientRect(&rect);
	// значення висоти та ширини елементу Static Text копіюються в окремі
	// змінні для зручності їх використання в подальшому
	int height = rect.Height();
```

```
        int width = rect.Width();
        // заповнення прямокутної області білим кольором (створення білого фону)
        dc.FillRect(&rect, WHITE_BRUSH);

        // кількість вертикальних та горизонтальних ліній сітки дорівнює 10
        int num_of_lines = 10;
        // розрахунок значення проміжку між горизонтальними лініями сітки
        int dy = (height / 2) / num_of_lines;

        // створення об’єкту пера зеленого кольору товщиною 2 пікселі, суцільна лінія
        CPen greenPen;
        greenPen.CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
        // вибір об’єкту пера в контекст пристрою
        dc.SelectObject(&greenPen);

        // малювання вертикальних ліній сітки
        for (int x = 0; x < width; x = x + width / num_of_lines)
        {
                dc.MoveTo(x, 0);
                dc.LineTo(x, height);
        }
        // малювання горизонтальних ліній сітки від центу області відображення вгору
        for (int y = height / 2; y < height; y = y + dy)
        {
                dc.MoveTo(0, y);
                dc.LineTo(width, y);
        }
        // малювання горизонтальних ліній сітки від центу області відображення вниз
        for (int y = height / 2; y > 0; y = y - dy)
        {
                dc.MoveTo(0, y);
                dc.LineTo(width, y);
        }

        // створення об’єкту пера чорного кольору товщиною 2 пікселі, суцільна лінія
        CPen blackPen;
        blackPen.CreatePen(PS_SOLID, 2, RGB(0, 0, 0));
        // вибір об’єкту пера в контекст пристрою
        dc.SelectObject(&blackPen);
        // малювання вісей координат
        dc.MoveTo(0, height);
        dc.LineTo(width, height);
        dc.MoveTo(0, height);
        dc.LineTo(0, 0);

        // встановлення прозорого фону для текстових надписів
        dc.SetBkMode(TRANSPARENT);
        // виведення текстового надпису "I(w)" у верхньому лівому куті
// області побудови АЧХ
        dc.TextOutW(7, 3, _T("I(w)"));
        // виведення текстового надпису "0" біля початку вісі абсцис
        dc.TextOutW(7, height - 20, _T("0"));
```

```
        // виведення надпису "2.5 kHz" в кінці вісі абсцис (вісь x)
        dc.TextOutW(width / 2 - 25, height - 20, _T("2.5 kHz"));
        // виведення надпису "5 kHz" в кінці вісі абсцис (вісь x)
        dc.TextOutW(width - 50, height - 20, _T("5 kHz"));
        // виведення значення максимального струму при резонансі біля вісі ординат
        CString str;
        str.Format(_T("%.1f A"), m_U / m_r);
        dc.TextOutW(35, 3, str);
}
```

Після компіляції проекту і запуску програми на виконання можна перевірити її роботу. На рис. 4.19 представлено 10 побудованих резонансних кривих для випадку,коли активний опір дорівнює 5 Ом, індуктивність 3 мГн, а ємність змінюється починаючи з 1 мкФ з кроком 1 мкФ (тобто 1 мкФ, 2 мкФ, 3 мкФ, 4 мкФ і т.д.). Напруга живлення складає 40 В.

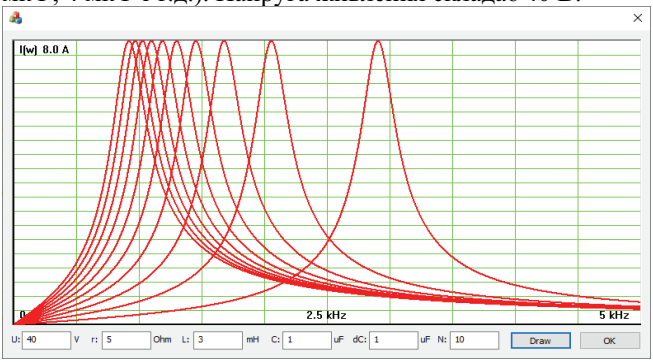


Рис. 4.19.

На рис. 4.20 представлено 5 побудованих резонансних кривих для випадку, коли активний опір дорівнює 5 Ом, індуктивність 10 мГн, а ємність змінюється починаючи з 1 мкФ з кроком 1 мкФ.

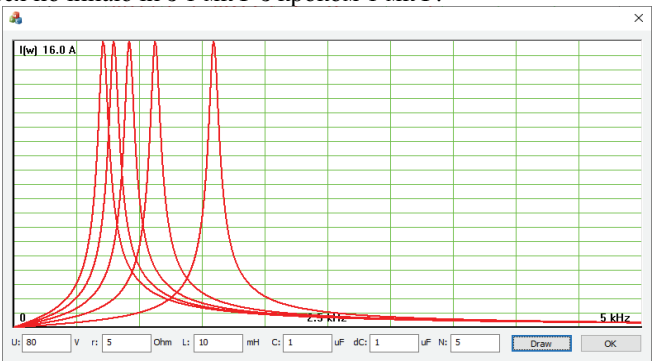


Рис. 4.20.

На рис. 4.21 представлено 7 побудованих резонансних кривих для випадку,коли активний опір дорівнює 10 Ом, індуктивність 0,5 мГн, а ємність змінюється починаючи з 5 мкФ з кроком 1 мкФ (тобто 5 мкФ, 6 мкФ, 7 мкФ, 8 мкФ і т.д.). Напруга живлення складає 80 В.

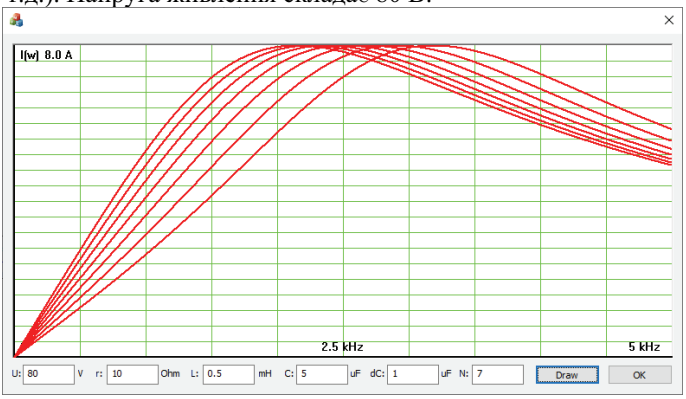


Рис. 4.21.

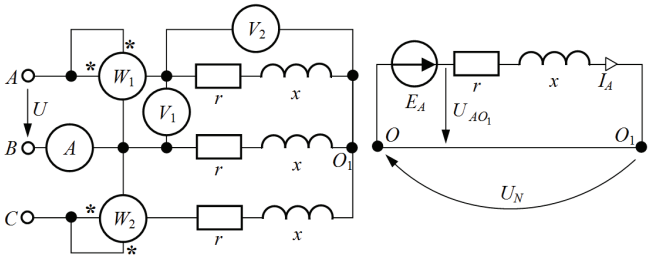
Таким чином, розроблена програма дозволяє дослідити вплив параметрів елементів схеми (в даному випадку ємності конденсатора), на значення частоти резонансу, та побудувати резонансні криві. Як видно з наведених рисунків, на всіх графіках є чітко виражений максимум на резонансній частоті. На частотах до резонансної вхідний опір схеми має ємнісний характер (реактивний опір конденсатора перевищує опір котушки індуктивності), на частоті резонансу дорівнює активному опору, а на частотах вище резонансної має індуктивний характер (реактивний опір котушки індуктивності перевищує опір конденсатора).

4.4. Розробка програмних засобів для моделювання трифазних кіл

Розрахунок симетричних трифазних кіл зводиться до розрахунку однієї фази за однолінійною еквівалентною схемою. У двох інших фазах будуть такі ж по величині струми і напруги, як в першій, але зміщені по фазі на кути 120° і –120°.

ПРИКЛАД 4.4. До симетричного трифазного кола з напругою U = 380 В підключений симетричний приймач, фази якого з'єднані в «зірку» (рис. 4.22, а), сопори фаз r = 12 Ом, x = 16 Ом [33].

Знайти покази приладів, побудувати векторну діаграму та осцилограми фазних струмів.



а б

Рис. 4.22.

Коли говорять про напругу трифазного кола, мають на увазі лінійну напругу мережі (за замовчуванням). Якщо задається фазна напруга, то це підкреслюється спеціально.

В умовах цієї задачі напруга мережі $U = 380$ *В*, причому за умовою симетрії: $U_{AB} = U_{BC} = U_{CA} = U = 380$ *В*.

Фазні напруги менші за лінійні в $\sqrt{3}$ разів:

$U_A = U_B = U_C = U/\sqrt{3} = 380/\sqrt{3} = 220$ *В*.

Припустимо, що фази генератора з'єднані в зірку, тоді фазні ЕРС генератора $E_A = E_B = E_C = U_Ф = 220$ *В*.

В симетричній системі Y-Y вузлова напруга $U_N = 0$ і розрахунок стану кола можна вести по схемі заміщення для однієї фази (рис. 4.22, б).

Приймемо $\underline{E}_A = 220$ *В*, струм в схемі заміщення:

$$\underline{I}_A = \frac{\underline{E}_A - \underline{U}_N}{\underline{Z}_A} = \frac{220}{12 + j16} = 11 \cdot e^{-j53{,}13^\circ} \text{ А},$$

причому фазна напруга приймача $\underline{U}_{AO_1} = \underline{E}_A - \underline{U}_N = \underline{E}_A = \underline{U}_A$.

Результати розрахунку режиму роботи інших фаз (*В* і *С*) запишемо на основі поняття о симетричній трифазній системі прямого порядку слідування фаз:

$$\underline{E}_B = \underline{U}_{BO_1} = \underline{U}_B = \underline{E}_A \cdot e^{-j120^\circ} = 220 \cdot e^{-j120^\circ} \text{ В};$$

$$\underline{E}_C = \underline{U}_{CO_1} = \underline{U}_C = \underline{E}_A \cdot e^{j120^\circ} = 220 \cdot e^{j120^\circ} \text{ В}.$$

Аналогічно для струмів:

$$\underline{I}_B = \underline{I}_A \cdot e^{-j120^\circ} = 11 \cdot e^{-j173{,}13^\circ} \text{ А};$$

$$\underline{I}_C = \underline{I}_A \cdot e^{j120^\circ} = 11 \cdot e^{j66{,}87^\circ} \text{ А}.$$

Векторна діаграма кола приведена на рис. 4.23, для якої потенціал нульової точки генератора, обмотки якого з'єднані в «зірку», $\underline{\varphi}_O = 0$ (рис. 4.22, б).

$\underline{\varphi}_A = \underline{E}_A = 220$ В; $\underline{\varphi}_B = 220 \cdot e^{-j120^\circ}$ В; $\underline{\varphi}_C = 220 \cdot e^{j120^\circ}$ В; $\underline{\varphi}_{O_1} = \underline{\varphi}_O + \underline{U}_N = 0$;

$$\underline{U}_{AO} = \underline{\varphi}_A - \underline{\varphi}_O = \underline{E}_A = \underline{U}_{AO_1} = \underline{\varphi}_A - \underline{\varphi}_{O_1} = \underline{U}_A = 220 \text{ В};$$

аналогічно $\underline{U}_B = \underline{\varphi}_B - \underline{\varphi}_O = 220 \cdot e^{-j120^\circ}$ В; $\underline{U}_C = \underline{\varphi}_C - \underline{\varphi}_O = 220 \cdot e^{j120^\circ}$ В;

а струми фаз орієнтовані відносно своїх фазних напруг (зсунуті на кут $\varphi = \text{arctg}\frac{x}{r} = \text{arctg}\frac{16}{12} = 53{,}13°$).

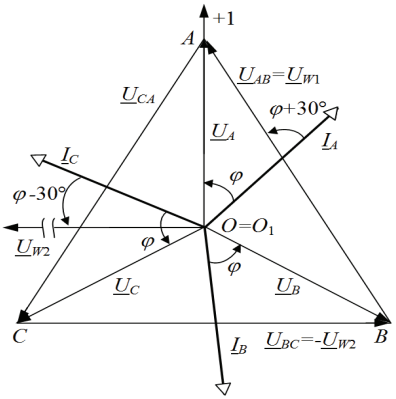


Рис. 4.23.

Лінійні напруги:

$$\underline{U}_{AB} = \underline{\varphi}_A - \underline{\varphi}_B = \underline{U}_A \cdot \sqrt{3} \cdot e^{j30°} = 380 \cdot e^{j30°} \text{ В};$$
$$\underline{U}_{BC} = \underline{U}_{AB} \cdot e^{-j120°} = 380 \cdot e^{-j90°} \text{ В};$$
$$\underline{U}_{CA} = \underline{U}_{AB} \cdot e^{j120°} = 380 \cdot e^{j150°} \text{ В}.$$

Напруги і струми котушок ватметрів:

$$\underline{U}_{W1} = \underline{U}_{AB} = 380 \cdot e^{j30°} \text{ В};$$
$$\underline{I}_{W1} = \underline{I}_A = 11 \cdot e^{-j53{,}13°} \text{ А};$$
$$\underline{U}_{W2} = -\underline{U}_{BC} = 380 \cdot e^{j90°} \text{ В};$$
$$\underline{I}_{W2} = \underline{I}_C = 11 \cdot e^{j66{,}87°} \text{ А}.$$

Покази приладів:

– амперметр A вимірює лінійний струм I_B і його покази $I_B = 11$ А;

– вольтметр V_1 вимірює лінійну напругу $U_{AB} = 380$ В;

– вольтметр V_2 вимірює фазну напругу приймача $U_A = 220$ В;

– покази ваттметрів:

$$P_{W1} = \text{Re}\left[\underline{U}_{W1} \cdot \underline{I}^*_{W1}\right] = \text{Re}\left[\underline{U}_{AB} \cdot \underline{I}^*_A\right] = U_{AB} \cdot I_A \cdot \cos(\varphi + 30°) =$$
$$= 380 \cdot 11 \cdot \cos(53{,}13° + 30°) = 500 \text{ Вт};$$
$$P_{W2} = \text{Re}\left[\underline{U}_{W2} \cdot \underline{I}^*_{W2}\right] = \text{Re}\left[\underline{U}_{CB} \cdot \underline{I}^*_C\right] = U_{CB} \cdot I_C \cdot \cos(\varphi - 30°) =$$
$$= 380 \cdot 11 \cdot \cos(53{,}13° + 30°) = 3846 \text{ Вт}.$$

Звертаємо увагу на суму показів двох ватметрів:

$$P_{W1} + P_{W2} = U \cdot I \cdot \cos(\varphi + 30°) + U \cdot I \cdot \cos(\varphi - 30°) =$$
$$= U \cdot I \cdot \cos 30° \cdot \cos\varphi - U \cdot I \cdot \sin 30° \cdot \sin\varphi + U \cdot I \cdot \cos 30° \cdot \cos\varphi + U \cdot I \cdot \sin 30° \cdot \sin\varphi =$$
$$= 2 \cdot U \cdot I \cdot \cos 30° \cdot \cos\varphi = \sqrt{3} \cdot U \cdot I \cdot \cos\varphi = P.$$

В цьому прикладі:

$$\sqrt{3}\cdot U\cdot I\cdot\cos\varphi=\sqrt{3}\cdot 380\cdot 11\cdot\cos(53{,}13°) = 4346 = 500 + 3846 \text{ Вт}.$$

Перед початком програмної реалізації методу розрахунку трифазного кола, в середовищі Visual Studio потрібно створити новий проект MFC App на основі діалогового вікна. Для того, щоб зробити інтерфейс діалогового вікна більш зрозумілим та надати користувачу інформацію про трифазне електричне коло, для якого ведеться розрахунок, на діалоговому вікні буде розташовано її зображення. Для того, щоб додати зображення схеми в проект у вікні ресурсів (рис. 4.24) потрібно натиснути праву кнопку миші на вузлі Scheme10.rc (за умови що при створенні нового проекту було задано його назву Scheme10), та викликати контекстне меню. В контекстному меню потрібно вибрати команду Add Resource, як показано на рис. 4.24.

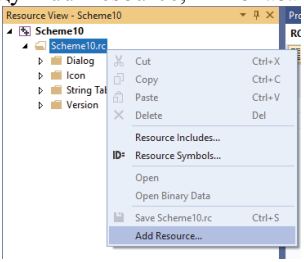


Рис. 4.24.

У діалоговому вікні додавання нового ресурсу, що з'явиться після цього (рис. 4.25), потрібно вибрати тип ресурсу Bitmap та натиснути на кнопку New. Якщо зображення схеми вже збережено у файлі з розширенням .bmp, то цей файл можна імпортувати в проект натиснути на кнопку Import.

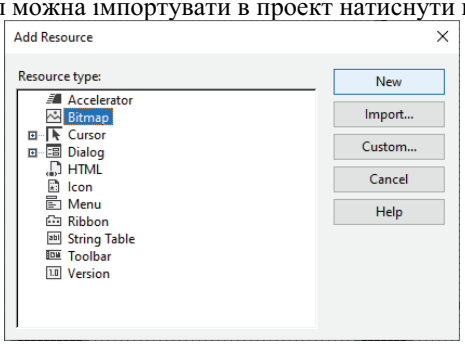


Рис. 4.25.

Відкриється вбудований в середовище розробки графічний редактор, який дуже схожий на стандартний додаток для роботи з растровою графікою Paint. В цьому редакторі слід намалювати схему, або вставити з буфера об-

міну, якщо схему було намальовано в іншому редакторі. На рис. 4.26 представлене вікно графічного редактора середовища розробки з намальованою схемою.

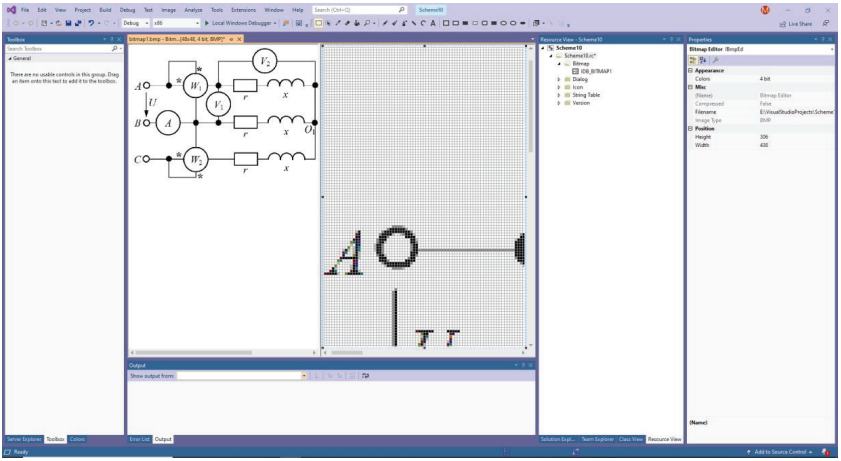


Рис. 4.26.

Тепер слід перейти до редагування діалогового вікна, для чого у вікні ресурсів слід двічі натиснути ліву кнопку миші на вузлі з ідентифікатором діалогового вікна, як показано на рис. 4.27.

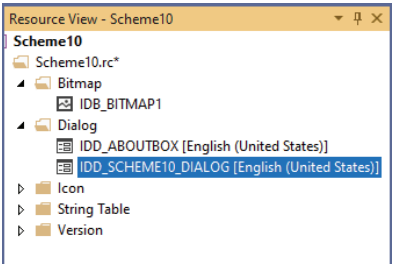


Рис. 4.27.

Власне на діалоговому вікні розташовуються елемент Picture Control для відображення схеми, елементи Static Text, в яких буде відображено векторну діаграму та осцилограми струмів, елементи Edit Control та елементи Static Text для відображення текстових надписів. Елемент Static Text, в якому буде відображатися векторна діаграма, слід зробити квадратної форми та встановити поле Border у вікні властивостей в «true». Тоді навколо елемента буде відображено рамку. Також необхідно змінити ідентифікатор цього елемента з IDC_STATIC, який задається за замовчуванням, на IDC_STATIC1, як показано на рис. 4.28.

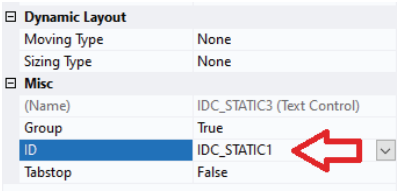


Рис. 4.28.

Тоді з'явиться можливість зв'язати з цим елементом керування змінну та отримати до нього доступ з програми. Для елемента Static Text, на якому буде намальовано осцилограми струмів, також потрібно змінити ідентифікатор ID з IDC_STATIC на IDC_STATIC2. На рис. 4.29 представлено макет діалогового вікна з розташованими на ньому елементами керування.

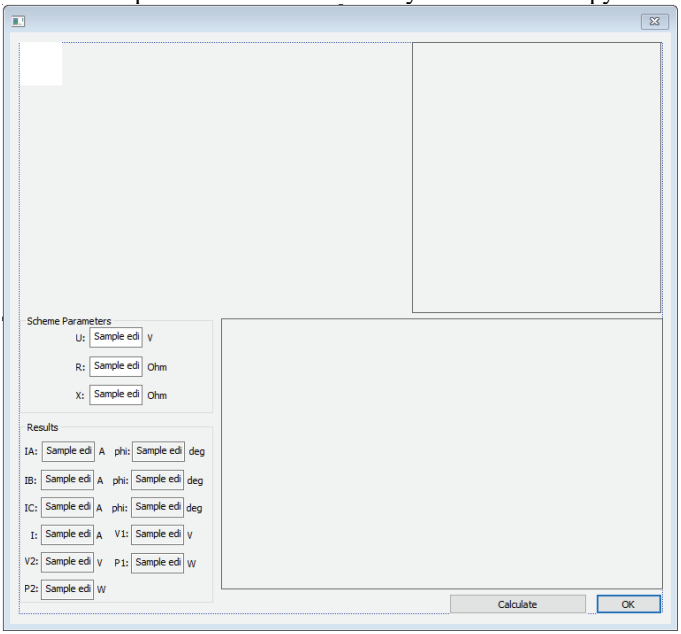


Рис. 4.29.

Для того, щоб в елементі Picture Control відображалася схема, у вікні властивостей цього елемента, яке представлено на рис. 4.30, потрібно задати для поля Type значення Bitmap, а у полі Image вказати ідентифікатор зображення (в даному випадку IDB_BITMAP1).

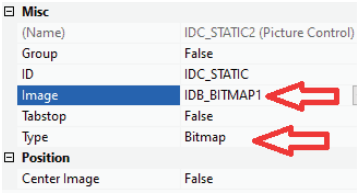


Рис. 4.30.

Після того, як це зроблено, з елементами керування, що розташовані на діалоговому вікні, потрібно зв'язати змінні. Для цього використовується Class Wizard, вікно якого можна відкрити шляхом натискання комбінації клавіш Ctrl+Shift+X. На вкладинці Member Variables для кожного елемента керування слід додати змінну. Всі змінні для елементів Edit Control, які призначені для введення користувачем числових даних, будуть відноситися до категорії Value та мати тип double. На рис. 4.31 представлене вікно додавання змінної для елементу Edit Control, в який користувач вводитиме значення напруги. При додаванні змінних для інших елементів керування це вікно матиме аналогічний вигляд, за винятком назв змінних, які задаються в полі Name.

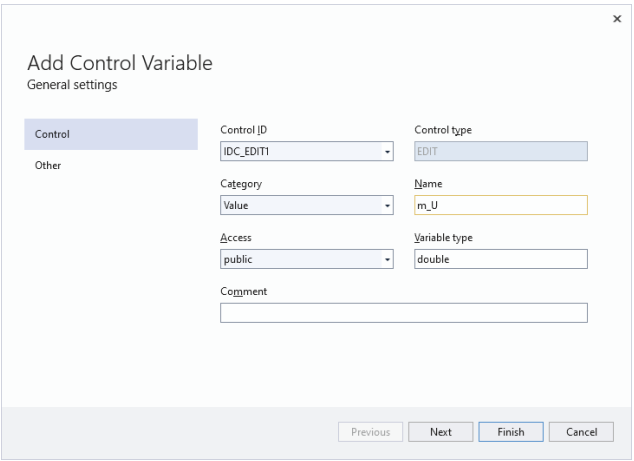


Рис. 4.31.

Для елементів керування Static Text, на яких відображатимуться векторна діаграма та осцилограма струмів, змінні повинні відноситися до категорії Control та мати тип CStatic. Після додавання всіх змінних вікно Class Wizard повинно мати вигляд, як показано на рис. 4.32.

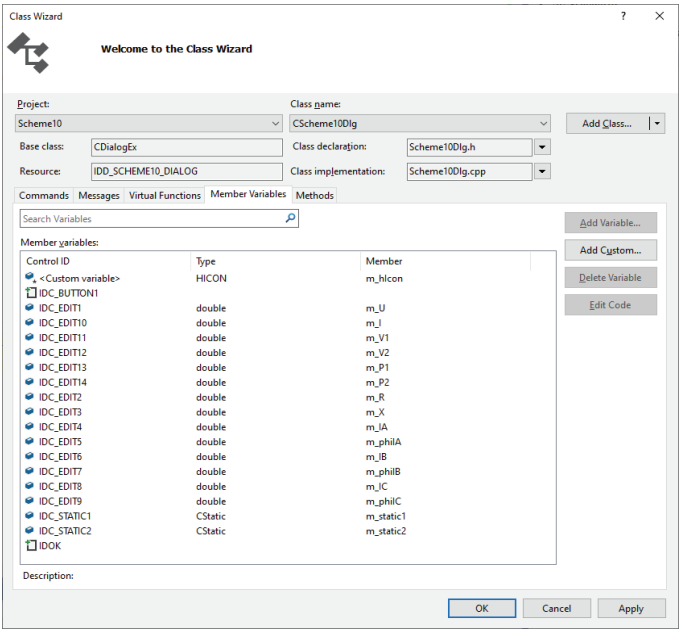


Рис. 4.32.

Для того, щоб структурувати програму та відокремити програмний код, що відповідає за розрахунки від коду, що відповідає за відображення результатів розрахунків в графічному вигляді, в клас діалогу потрібно додати функцію-член. Для цього у вікні Clas View слід викликати контекстне меню для класу діалогу та в підменю Add вибрати команду Add Function, як показано на рис. 4.33.

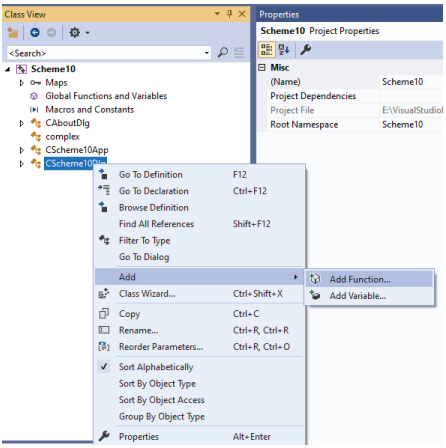


Рис. 4.33.

У діалоговому вікні, що з’явиться (рис. 4.34), потрібно задати назву нової функції та тип даних, що повертається з функції (void). В даному випадку функція називається DrawGraph.

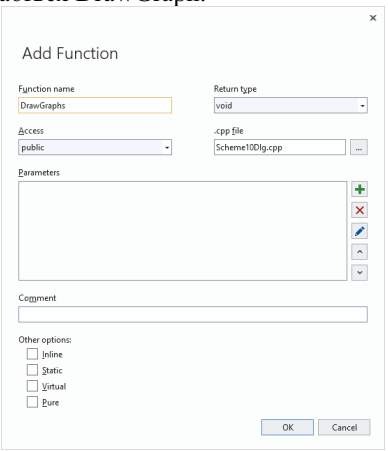


Рис. 4.34.

Тепер можна переходити до написання програмного коду. Всі розрахунки виконуватимуться після натискання на кнопку «Calculate», яка розташована на діалоговому вікні. Для того, щоб додати функцію-обробник натискання на цю кнопку достатньо у редакторі ресурсів двічі на неї натиснути лівою кнопкою миші. Програмний код функції-обробника натискання на кнопку з надписом «Calculate», а також коментарі, представлено нижче.

```
void CScheme10Dlg::OnBnClickedButton1()
{
	// зчитування введених користувачем даних з елементів керування
	// в змінні
	UpdateData(true);

	// створення об’єктів, в яких зберігаються значення фазних ЕРС
	// параметри, що передаються в конструктор - модуль та фаза,
	// тому третій параметр має значення "істина"
// (дані представлені в полярній формі)
	complex EA(m_U / sqrt(3), 0, true);
	complex EB(m_U / sqrt(3), -120.0 * M_PI / 180, true);
	complex EC(m_U / sqrt(3), 120.0 * M_PI / 180, true);

	// комплексний опір навантаження
	complex Z(m_R, m_X, false);

	// об’єкти для збереження значень комплексних струмів фаз
	complex IA, IB, IC;

	// розрахунок фазних струмів
```

```
        IA = EA / Z;
        IB = EB / Z;
        IC = EC / Z;

        // зчитування значень модуля та фази струмів у відповідні змінні
        // для їх подальшого відображення на діалоговому вікні
        IA.ModPhase(m_IA, m_phiIA);
        IB.ModPhase(m_IB, m_phiIB);
        IC.ModPhase(m_IC, m_phiIC);

        // покази амперметра
        m_I = m_IB;
        // покази першого вольтметра - лінійна напруга
        m_V1 = m_U;
        // покази другого вольтметра - напруга на навантаженні = фазна напруга
        m_V2 = m_U / sqrt(3);

        // комплекси лінійних напруг
        complex UAB, UCB;
        // розрахунок лінійної напруги між фазами A та B
        UAB = EA - EB;
        // розрахунок лінійної напруги між фазами C та B
        UCB = EC - EB;
        // об'єкти для розрахунку потужностей
        complex S1, S2;
        complex C(0, -1, false);
        // розрахунок значення повних потужностей
        S1 = UAB * (IA * C);
        S2 = UCB * (IC * C);

        // оголошення додаткових змінних для зберігання інформації
        // про дійсні складові потужності та кутів зсуву фаз між струм та напругою
        double P1, P2, phi1, phi2;
        S1.ModPhase(P1, phi1);
        S2.ModPhase(P2, phi2);
        // розрахунок значень активних потужностей
        m_P1 = P1 * cos(-m_phiIA + M_PI / 6);
        m_P2 = P2 * cos(-m_phiIA - M_PI / 6);

        // переведення значень фаз струмів з радіан в градуси
        m_phiIA = m_phiIA * 180 / M_PI;
        m_phiIB = m_phiIB * 180 / M_PI;
        m_phiIC = m_phiIC * 180 / M_PI;
        // виведення вмісту змінних у відповідні елементи керування
        UpdateData(false);

// виклик функції малювання векторної діаграми та осцилограми
        DrawGraphs();

}
```

Після компіляції проекту і запуску програми з’явиться діалогове вікно, представлене на рис. 4.35. Після введення в елементи керування у групі «Scheme Parameter» значення напруги та опорів елементів електричного кола, можна натиснути на кнопку «Calculate» та побачити результати розрахунків, що відображаються в елементах керування у групі «Results».

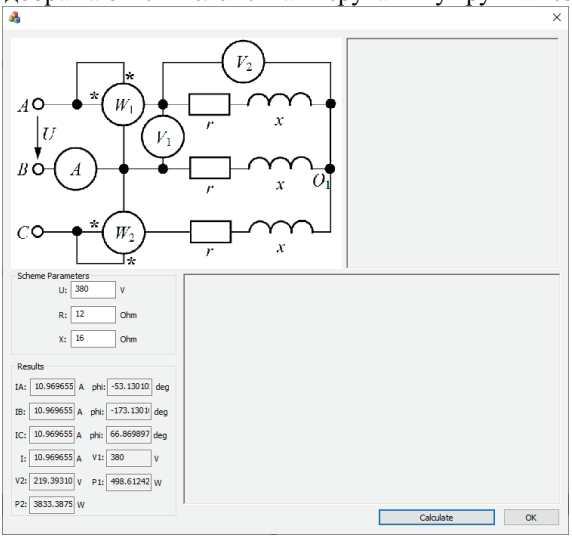


Рис. 4.35.

Векторна діаграма та осцилограми струмів ще не відображаються, оскільки функцію DrawGraphs, що відповідає за відображення результатів розрахунків в графічному вигляді, ще не реалізовано. Програмний код цієї функції наведено далі.

```
void CScheme10Dlg::DrawGraphs()
{
	// створення контексту пристрою для роботи з графікою
	// в конструктор об’єкту передається вказівник на елемент Static Text,
	// на якому відображатиметься векторна діаграма
	CClientDC dc(&m_static1);
	// створення об’єкту для зберігання інформації про прямокутну область
	CRect rect;
	// отримання інформації про розміри елементу Static Text
	m_static1.GetClientRect(&rect);
	// для зручності значення ширини та висоти елемента
	// розміщуються в окремих змінних
	int W = rect.Width();
	int H = rect.Height();
	// заповнення прямокутної області білим кольором
	dc.FillRect(&rect, WHITE_BRUSH);
```

```
// створення об’єкту пера (CPen)
CPen grayPen;
// оголошення вказівника на об’єкт класу CPen
CPen* oldPen;
// створення пера світло-сірого кольору товщиною 1 піксель
// для малювання координатних вісей
grayPen.CreatePen(PS_SOLID, 1, RGB(180, 180, 180));
// вибір об’єкту пера в контекст пристрою
oldPen =dc.SelectObject(&grayPen);
// малювання горизонтальної лінії (вісь абсцис)
dc.MoveTo(0, H / 2);
dc.LineTo(W, H / 2);
// малювання вертикальної лінії (вісь ординат)
dc.MoveTo(W / 2, 0);
dc.LineTo(W / 2, H);
// вибір в контекст пристрою попереднього пера
// за замовчуванням чорного кольору товщиною 1 піксель
// для малювання кругової шкали
dc.SelectObject(oldPen);
// радіус зовнішнього кола, куди будуть вписані риски
int R = W / 2 - 5;
// змінна для збереження значення радіусу внутрішнього кола
int r;
// координата x центру області побудови векторної діаграми
int x0 = W / 2;
// координата y центру області побудови векторної діаграми
int y0 = H / 2;
// оголошення додаткових змінних, що будуть використані
// при малюванні векторів
int x1, y1, x2, y2, x3, y3;

// в циклі виконується малювання рисок кругової шкали
for (int a = 0; a <= 360; a=a+5)
{
	// кожні 30 градусів малюється довга риска
	if ((a % 30) == 0)
	{
		r = R - 15;
	}
	else  // інші риски мають меншу довжину
	{
		r = R - 7;
	}
	// обчислення координат початку та кінця кожної риски
	x1 = x0 + R * cos(a * M_PI / 180);
	y1 = y0 - R * sin(a * M_PI / 180);
	x2 = x0 + r * cos(a * M_PI / 180);
	y2 = y0 - r * sin(a * M_PI / 180);
	// малювання риски
	dc.MoveTo(x1, y1);
	dc.LineTo(x2, y2);
}
```

```
// коригування фаз струмів для того, щоб кут 0 градусів
// відраховувався с додатного напрямку вісі ординат
double phiIA = m_phiIA + 90;
double phiIB = m_phiIB + 90;
double phiIC = m_phiIC + 90;

// малювання вектора струму фази А
// створення об’єкту пера червоного кольору товщиною 2 пікселі
CPen redPen;
redPen.CreatePen(PS_SOLID, 2, RGB(200, 0, 0));
// вибір об’єкту в контекст пристрою
oldPen = dc.SelectObject(&redPen);

// значення масштабу струму при відображенні осцилограми
int scaleI = 10;

// розрахунок координат кінця вектора струму
x1 = x0 + m_IA * cos(phiIA * M_PI / 180) * scaleI;
y1 = y0 - m_IA * sin(phiIA * M_PI / 180) * scaleI;
// розрахунок координат кінця першої риски стрілки
x2 = x0 + (0.9 * m_IA) * cos((phiIA - 2) * M_PI / 180) * scaleI;
y2 = y0 - (0.9 * m_IA) * sin((phiIA - 2) * M_PI / 180) * scaleI;
// розрахунок координат кінця другої риски стрілки
x3 = x0 + (0.9 * m_IA) * cos((phiIA + 2) * M_PI / 180) * scaleI;
y3 = y0 - (0.9 * m_IA) * sin((phiIA + 2) * M_PI / 180) * scaleI;
// малювання вектора струму
dc.MoveTo(x0, y0);
dc.LineTo(x1, y1);
// малювання стрілки на кінці вектора
dc.LineTo(x2, y2);
dc.MoveTo(x1, y1);
dc.LineTo(x3, y3);

// малювання вектора струму фази В
// створення об’єкту пера зеленого кольору товщиною 2 пікселі
CPen greenPen;
greenPen.CreatePen(PS_SOLID, 2, RGB(0, 200, 0));
// вибір об’єкту в контекст пристрою
dc.SelectObject(&greenPen);

// розрахунок координат кінця вектора струму
x1 = x0 + m_IB * cos(phiIB * M_PI / 180) * scaleI;
y1 = y0 - m_IB * sin(phiIB * M_PI / 180) * scaleI;
// розрахунок координат кінця першої риски стрілки
x2 = x0 + (0.9 * m_IB) * cos((phiIB - 2) * M_PI / 180) * scaleI;
y2 = y0 - (0.9 * m_IB) * sin((phiIB - 2) * M_PI / 180) * scaleI;
// розрахунок координат кінця другої риски стрілки
x3 = x0 + (0.9 * m_IB) * cos((phiIB + 2) * M_PI / 180) * scaleI;
y3 = y0 - (0.9 * m_IB) * sin((phiIB + 2) * M_PI / 180) * scaleI;
// малювання вектора струму
dc.MoveTo(x0, y0);
dc.LineTo(x1, y1);
```

```
// малювання стрілки на кінці вектора
dc.LineTo(x2, y2);
dc.MoveTo(x1, y1);
dc.LineTo(x3, y3);

// малювання вектора струму фази С
// створення об’єкту пера синього кольору товщиною 2 пікселі
CPen bluePen;
bluePen.CreatePen(PS_SOLID, 2, RGB(0, 0, 200));
// вибір об’єкту в контекст пристрою
dc.SelectObject(&bluePen);

// розрахунок координат кінця вектора струму
x1 = x0 + m_IC * cos(phiIC * M_PI / 180) * scaleI;
y1 = y0 - m_IC * sin(phiIC * M_PI / 180) * scaleI;
// розрахунок координат кінця першої риски стрілки
x2 = x0 + (0.9 * m_IC) * cos((phiIC - 2) * M_PI / 180) * scaleI;
y2 = y0 - (0.9 * m_IC) * sin((phiIC - 2) * M_PI / 180) * scaleI;
// розрахунок координат кінця другої риски стрілки
x3 = x0 + (0.9 * m_IC) * cos((phiIC + 2) * M_PI / 180) * scaleI;
y3 = y0 - (0.9 * m_IC) * sin((phiIC + 2) * M_PI / 180) * scaleI;
// малювання вектора струму
dc.MoveTo(x0, y0);
dc.LineTo(x1, y1);
// малювання стрілки на кінці вектора
dc.LineTo(x2, y2);
dc.MoveTo(x1, y1);
dc.LineTo(x3, y3);

// створення контексту пристрою для роботи з графікою
// в конструктор об’єкту передається вказівник на елемент Static Text,
// на якому відображатиметься осцилограма струмів
CClientDC dc2(&m_static2);
// отримання інформації про розміри елементу Static Text
m_static2.GetClientRect(&rect);
// для зручності значення ширини та висоти елемента
// розміщуються в окремих змінних
W = rect.Width();
H = rect.Height();
// заповнення прямокутної області білим кольором
dc2.FillRect(&rect, WHITE_BRUSH);

// малювання вертикальних ліній координатної сітки
for (int x = 0; x < W; x = x + 10)
{
        dc2.MoveTo(x, 0);
        dc2.LineTo(x, H);
}
// малювання горизонтальних ліній координатної сітки
for (int y = 0; y < H; y = y + 10)
{
        dc2.MoveTo(0, y);
```

```
                dc2.LineTo(W, y);
        }

        // вибір в контекст пристрою об’єкту пера червоного кольору
        dc2.SelectObject(redPen);
        // малювання осцилограми струму фази A
        for (int x = 1; x < W; x++)
        {
dc2.MoveTo(x, H / 2 - m_IA * sin(2 * M_PI * x / 200 + m_phiIA * M_PI / 180) * scaleI);
dc2.LineTo(x-1, H/2 - m_IA * sin(2* M_PI * (x - 1) / 200 + m_phiIA * M_PI / 180) * scaleI);
        }

        // вибір в контекст пристрою об’єкту пера зеленого кольору
        dc2.SelectObject(greenPen);
        // малювання осцилограми струму фази B
        for (int x = 1; x < W; x++)
        {
dc2.MoveTo(x, H / 2 - m_IB * sin(2 * M_PI * x / 200 + m_phiIB * M_PI / 180) * scaleI);
dc2.LineTo(x - 1, H / 2 - m_IB * sin(2 * M_PI * (x - 1) / 200 + m_phiIB * M_PI / 180) *
scaleI);
        }

        // вибір в контекст пристрою об’єкту пера синього кольору
        dc2.SelectObject(bluePen);
        // малювання осцилограми струму фази C
        for (int x = 1; x < W; x++)
        {
dc2.MoveTo(x, H / 2 - m_IC * sin(2 * M_PI * x / 200 + m_phiIC * M_PI / 180) * scaleI);
dc2.LineTo(x - 1, H / 2 - m_IC * sin(2 * M_PI * (x - 1) / 200 + m_phiIC * M_PI / 180) *
scaleI);
        }
}
```

Отже, тепер запустивши програму на виконання та після введення в елементи керування діалогового вікна параметрів схеми, можна побачити результати розрахунків також у графічному вигляді, як показано на рис. 4.36.

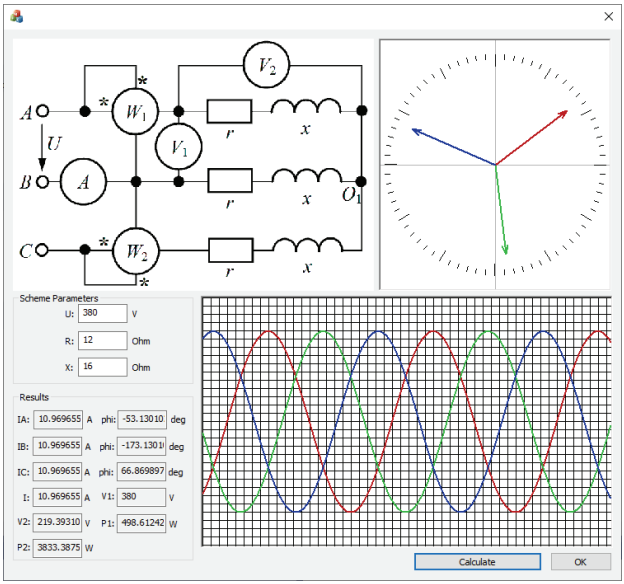


Рис. 4.36.

Як видно з рисунку, отримані дані та вигляд векторної діаграми співпадають з аналітичними розрахунками. Можна задати інші параметри схеми та отримати розраховані значення струмів, як показано на рис. 4.37.

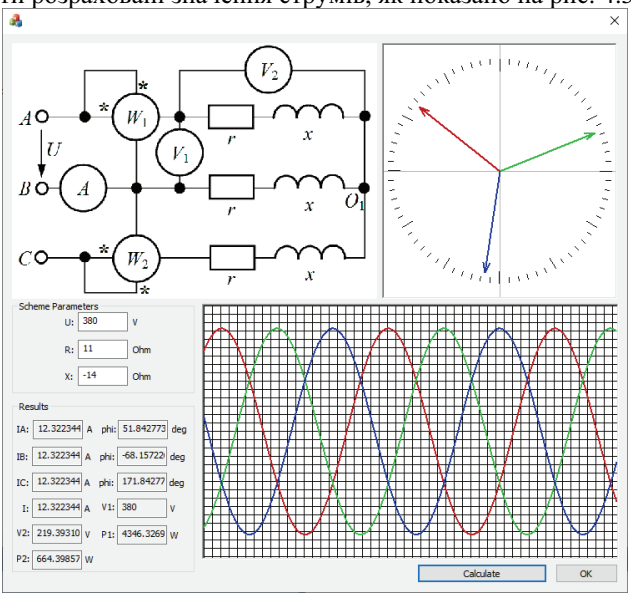


Рис. 4.37.

4.5. Програмні засоби для моделювання активних фільтрів

Активні RC-фільтри належать до широко розповсюдженого класу частотно-вибіркових ланцюгів та знаходять використання у системах передачі інформації, системах автоматичного керування і регулювання, вимірювальній техніці й різного виду функціональних перетворювачах. Активні RC-фільтри (АФ) містять пасивні вибіркові RC-ланцюги й активні пристрої (збуджувачі, гіратори, конвертори від'ємного опору), за допомогою яких отримують необхідну добротність ланок. Основним завданням при проектуванні АФ є отримання заданої форми амплітудно-частотної характеристики.

Під електричним фільтром розуміють чоторьохполюсник, модуль передатної функції якого залишається практично постійним у спеціальній області частот, яка називається смуга пропускання, і досить різко падає з віддаленням від меж цієї ділянки. Межі ділянки смуги пропускання називають граничними частотами. Ділянка частот з достатньо великим придушенням амплітуди сигналу називається смугою загородження. Між смугами пропускання й загородження знаходиться перехідна ділянка.

Синтез частотно-вибіркових ланцюгів пов'язаний з вирішенням двох завдань:

– завдання утворення функції апроксимації за вихідними даними;

– завдання реалізації знайденої апроксимуючої функції електричного ланцюга.

Схема фільтра нижних частот першого порядку показано на рис. 4.38.

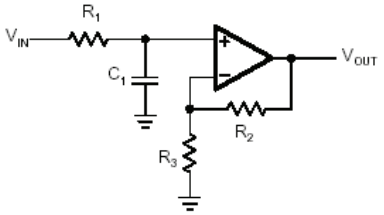


Рис. 4.38.

Передатна функція такого фільтру має вигляд:

$$A(s) = \frac{1 + \dfrac{R_2}{R_3}}{1 + \omega_C R_1 C_1 p}.$$

Коефіцієнт підсилення:

$$A_0 = 1 + \frac{R_2}{R_3}.$$

Коефіцієнт a_1:

$$a_1 = \omega_c R_1 C_1.$$

Для розрахунку значень параметрів елементів схеми необхідно обрати частоту зрізу фільтру (f_c), обрати значення ємності конденсатора C_1. Після цього виконується розрахунок значень опорів резисторів за формулами:

$$R_1 = \frac{a_1}{2\pi f_c C_1},$$

$$R_2 = R_3(A_0 - 1).$$

Значення коефіцієнта a_1 береться в залежності від вимог до характеристики фільтру. Для фільтрів першого порядку цей коефіцієнт дорівнює 1. Якщо необхідно забезпечити одиничний коефіцієнт підсилення фільтру, значення резистора R_2 обирають рівним 0, а резистор R_3 прибирають зі схеми (рис. 4.39).

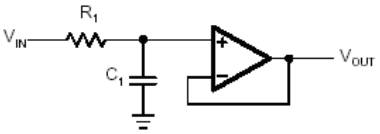


Рис. 4.39.

ПРИКЛАД 4.5. Розробити програму, що по заданим параметрам активного фільтру першого порядку з одиничним коефіцієнтом підсилення будує його амплітудно-частотну характеристику. Також необхідно виконати параметричний аналіз схеми та дослідити вплив зміни опору резистора в схемі при зміні температури на вигляд амплітудно-частотної характеристики.

Розробка програми розпочинається з проектування інтерфейсу користувача. Для цього після створення проекту MFC Application на базі діалогово вікна в середовищі розробки Visual Studio потрібно відкрити вікно редактора ресурсів та розташувати на діалозі потрібні елементи керування, як показано на рис. 4.40.

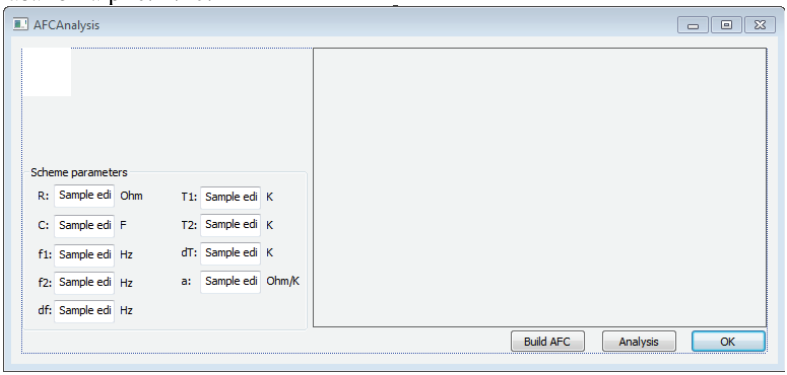


Рис. 4.40.

Основну частину діалогу займає елемент Static Text. На цьому елементі буде відображатися в графічному вигляді амплітудно-частотна характеристика активного фільтру. У вікні властивостей цього елемента керування поле Border потрібно встановити в true, а також змінити поле ID з IDC_STATIC на IDC_STATIC1 для того, щоб з цим елементом керування можна було б зв'язати змінну. В лівому верхньому куті діалогу розташовано елемент Picture Control, в якому відображається схема фільтра. Зображення схеми необхідно імпортувати в проект як растрове зображення. Також на діалозі розташовані елементи Edit Control, в які користувач зможе ввести параметри елементів схеми та інші дані, необхідні для виконання параметричного аналізу схеми. Кожен елемент Edit Control має відповідний підпис, що дає змогу користувачу зрозуміти, який саме параметр вводиться в елемент керування та в яких одиницях. Також на діалозі розташовано дві кнопки. Натискання на кнопку з надписом «Build AFC» призводить до побудови та відображення амплітудно-частотної характеристики фільтру при заданих значеннях опору резистора та ємності конденсатора. При цьому на екрані відображається АЧХ в діапазоні частот, що задані в параметрах f1 (початкова частота) та f2 (кінцева частота). Змінюючи ці значення можна виконувати «навігацію» по АЧХ і більш детально розглянути потрібну ділянку. Для того, щоб виконати аналіз впливу температури на вигляд АЧХ потрібно натиснути на кнопку з надписом «Analysis». Аналіз схеми та побудова сімейства характеристик виконується для різних значень температур (від T1 до T2) з кроком dT. При цьому для розрахунку значення опору резистора використовується температурний коефіцієнт опору (поле введення даних з позначкою «a»). На рис. 4.41 представлено діалогове вікно Class Wizard (вкладка Member Variables), за допомогою якого в клас діалога було додано змінні-члени, зв'язані з відповідними елементами керування. З назв змінних зрозуміло, з якими саме елементами керування вони зв'язані.

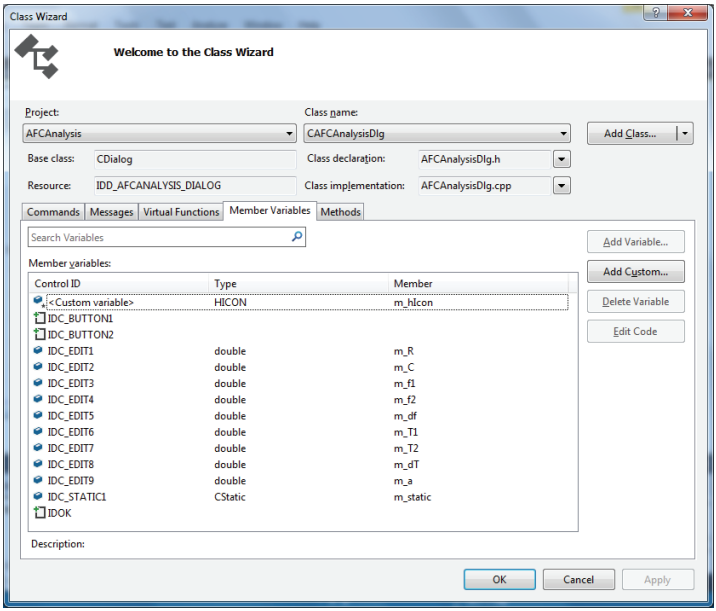


Рис. 4.41.

В програмі для роботи з комплексними числами використовується клас зі стандартної бібліотеки. Для того, щоб мати можливість його використовувати, у файлі реалізації класу діалогу, що має розширення .cpp, потрібно підключити відповідну бібліотеку шляхом додавання наступного рядка:

```
#include <complex>
```

Подвійне натискання на кнопку з надписом «Build AFC», що розташована на діалозі, призведе до створення функції-обробника події натискання на цю кнопку. Саме в цій функції виконується розрахунок та побудова амплітудно-частотної характеристики фільтра:

```
using namespace std::complex_literals;
void CAFCAnalysisDlg::OnBnClickedButton1()
{
	// зчитування значень з елементів керування у відповідні змінні
	UpdateData(true);
	// створення об’єкту контексту пристрою
	CClientDC dc(&m_static);
	// об’єкт для збереження інформації про прямокутну область
	CRect rect;
	// зчитування інформації про розміри елементу Static Text,
	// на якому відображатиметься АЧХ фільтра
	m_static.GetClientRect(&rect);
```

```
// в окремі змінні копіюються значення висоти та ширини прямокутної
// області побудови АЧХ
int height = rect.Height();
int width = rect.Width();
// створення об’єкту регіону для обмеження області виведення графіки
CRgn rgn;
// регіон має прямокутну форму
rgn.CreateRectRgn(0, 0, width, height);
// вибір створеного регіону в контекст пристрою
dc.SelectClipRgn(&rgn);
// заповнення області побудови АЧХ білим кольором
dc.FillRect(&rect, WHITE_BRUSH);
// оголошення змінної для зберігання значення масштабу
double scale = 200;
// створення об’єкту пера зеленого кольору для малювання ліній сітки
CPen greenPen(PS_SOLID, 1, RGB(0,180,0));
// вибір зеленого пера в контекст пристрою
dc.SelectObject(&greenPen);

// малювання вертикальних ліній сітки, крок 50 пікселів
for (int x = 0; x < width; x = x + 50)
{
	dc.MoveTo(x, 0);
	dc.LineTo(x, height);
}
// малювання горизонтальних ліній сітки, крок 25 пікселів
for (int y = height; y > 0; y = y - 25)
{
	dc.MoveTo(0, y);
	dc.LineTo(width, y);
}
// малювання текстових надписів біля ліній сітки
dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
// значення коефіцієнту передачі фільтра (текстові надписи біля ліній сітки)
for (int i = 1; i < 5; i++)
{
	CString s;
	s.Format(_T("%.2f"), 50 * i / scale);
	dc.TextOutW(5, height - 20 - i * 50, s);
}
// значення частоти(текстові надписи біля ліній сітки)
for (int i = 0; i < 8; i++)
{
	CString s;
	s.Format(_T("%d"), (int)m_f1 + i * 50);
	dc.TextOutW(5 + i * 50, height - 20, s);
}
// виведення текстового надпису у верхньому куті ліворуч
dc.TextOutW(5, 5, _T("H(f)"));
// виведення текстового надпису у нижньому куті праворуч
dc.TextOutW(width - 40, height - 20, _T("f, Hz"));
```

```
	// копіювання значень опору та ємності в окремі змінні
	double R = m_R;
	double C = m_C;
	// оголошення змінної для зберігання значення кутової частоти
	double w;
	// визначення необхідної кількості елементів масиву для
	// зберігання розрахованих значень передавальної функції фільтра
	// для кожної з частот
	int SIZE = (int)(m_f2 - m_f1) / m_df;
	// динамічне виділення пам'яті для масиву
	double *afc = new double[SIZE];
	// встановлення всіх значень масиву в 0
	memset(afc, 0, sizeof(afc));
	// оголошення додаткових змінних для реалізації алгоритму розрахунку
	int N = 0;
	double f=m_f1;
	// цикл виконується доки умова виконується
	while(f<m_f2)
	{
		w = 6.28 * f; // розрахунок кутової частоти
		// комплексна передавальна функція фільтра
		std::complex<double> H = 1. / (1. + 1i * w * R * C);
		afc[N] = abs(H); // модуль передавальної функції
		N++; // інкремент змінної
		f = f + m_df; // збільшення частоти на величину кроку
	}

	// створення об’єкту пера чорного кольору для побудови АЧХ
	CPen blackPen(PS_SOLID, 2, RGB(0, 0, 0));
	// вибір пера в контекст пристрою
	dc.SelectObject(&blackPen);
	// в циклі відбувається малювання АЧХ фільтра по розрахованим значенням
	for (int k = 1; k < SIZE; k++) {
		dc.MoveTo(k, height-afc[k]*scale);
		dc.LineTo(k-1, height - afc[k-1] * scale);
	}
	// очищення динамічно виділеної пам'яті
	delete[] afc;
}
```

Після компіляції проекту та запуску програми на виконання можна перевірити її роботу. На рис. 4.42 представлено діалогове вікно програми в режимі виконання. Для заданих значень опору резистора та ємності конденсатора виконано побудову АЧХ.

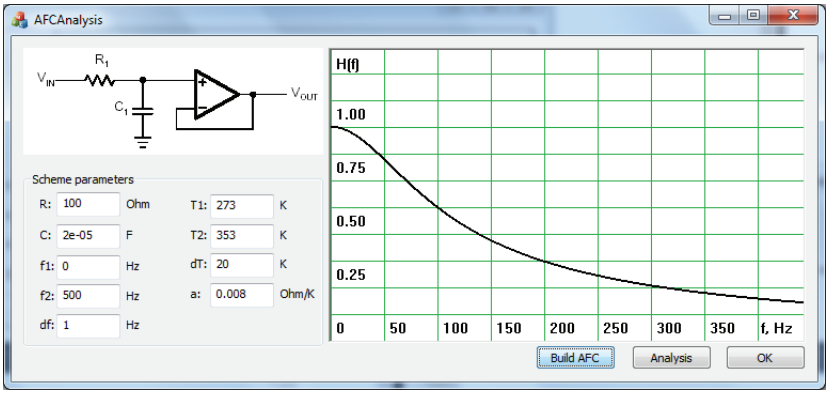


Рис. 4.42.

Реалізація функції-обробника натискання на кнопку з надписом «Analysis» має вигляд:

```
void CAFCAnalysisDlg::OnBnClickedButton2()
{
	// зчитування значень з елементів керування у відповідні змінні
	UpdateData(true);
	// створення об’єкту контексту пристрою
	CClientDC dc(&m_static);
	// об’єкт для збереження інформації про прямокутну область
	CRect rect;
	// зчитування інформації про розміри елементу Static Text,
	// на якому відображатиметься АЧХ фільтра
	m_static.GetClientRect(&rect);
	// в окремі змінні копіюються значення висоти та ширини прямокутної
	// області побудови АЧХ
	int height = rect.Height();
	int width = rect.Width();
	// створення об’єкту регіону для обмеження області виведення графіки
	CRgn rgn;
	// регіон має прямокутну форму
	rgn.CreateRectRgn(0, 0, width, height);
	// вибір створеного регіону в контекст пристрою
	dc.SelectClipRgn(&rgn);
	// заповнення області побудови АЧХ білим кольором
	dc.FillRect(&rect, WHITE_BRUSH);
	// оголошення змінної для зберігання значення масштабу
	double scale = 200;
	// створення об’єкту пера зеленого кольору для малювання ліній сітки
	CPen greenPen(PS_SOLID, 1, RGB(0, 180, 0));
	// вибір зеленого пера в контекст пристрою
	dc.SelectObject(&greenPen);

	// малювання вертикальних ліній сітки, крок 50 пікселів
```

```
for (int x = 0; x < width; x = x + 50)
{
	dc.MoveTo(x, 0);
	dc.LineTo(x, height);
}
// малювання горизонтальних ліній сітки, крок 25 пікселів
for (int y = height; y > 0; y = y - 25)
{
	dc.MoveTo(0, y);
	dc.LineTo(width, y);
}
// малювання текстових надписів біля ліній сітки
dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
 // значення коефіцієнту передачі (текстові надписи біля ліній сітки)
for (int i = 1; i < 5; i++)
{
	CString s;
	s.Format(_T("%.2f"), 50 * i / scale);
	dc.TextOutW(5, height - 20 - i * 50, s);
}
// значення частоти (текстові надписи біля ліній сітки)
for (int i = 0; i < 8; i++)
{
	CString s;
	s.Format(_T("%d"), (int)m_f1 + i * 50);
	dc.TextOutW(5 + i * 50, height - 20, s);
}
// виведення текстового надпису у верхньому куті ліворуч
dc.TextOutW(5, 5, _T("H(f)"));
// виведення текстового надпису у нижньому куті праворуч
dc.TextOutW(width - 40, height - 20, _T("f, Hz"));

// розрахунок кількості ітерацій циклу
int c = 255 / ((m_T2 - m_T1) / m_dT);
// додаткова змінна для керування кольором відображення АЧХ
int color = 0;
// початкова температура
double T = m_T1;
// цикл виконується доки умова дійсна
while (T <= m_T2)
{
	// розрахунок значення опору при заданій температурі
	double R = m_R * (1 + m_a * (T - m_T1));
	// копіювання значення ємності в окрему змінну
	double C = m_C;
	// додаткова змінна для зберігання значення кутової частоти
	double w;
	// розрахунок кількості необхідних елементів масиву для
	// зберігання розрахованих значень
	int SIZE = (int)(m_f2 - m_f1) / m_df + 10;
	// динамічне виділення пам'яті для масиву
	double *afc = new double[SIZE];
```

```
            // очищення масиву (значення всіх елементів масиву дорівнює 0)
            memset(afc, 0, sizeof(afc));
            // оголошення додаткових змінних для реалізації алгоритму
            int N = 0;
            double f = m_f1;
            // цикл виконується доки умова дійсна
            while (f < m_f2)
            {
                  w = 6.28 * f; // розрахунок значення кутової частоти
                  // передавальна функція фільтра першого порядку
                  std::complex<double> H = 1. / (1. + 1i * w * R * C);
                  // модуль частотної характеристики
                  afc[N] = abs(H);
                  // інкремент змінної
                  N++;
                  // збільшення частоти на величину кроку
                  f = f + m_df;
            }
            // створення пера для малювання АЧХ
            // колір пера залежить від температури і з її збільшенням
            // поступово змінюється від синього до червоного
            CPen pen(PS_SOLID, 2, RGB(color, 0, 255-color));
            // вибір об’єкту пера в контекст пристрою
            dc.SelectObject(&pen);
            // в циклі відбувається малювання АЧХ
            for (int k = 1; k < SIZE; k++) {
                  dc.MoveTo(k, height - afc[k] * scale);
                  dc.LineTo(k - 1, height - afc[k - 1] * scale);
            }
            // збільшення значення температури на величину кроку
            T = T + m_dT;
            // додавання значення кроку зміни кольору
            color += c;
            // звільнення динамічно виділеної пам'яті
            delete[] afc;
      }
}
```

Запустивши програму на виконання, в поле T1 потрібно ввести значення початкової температури, в поле T2 – значення кінцевої температури, а в поле dT – крок зміни температури. Також потрібно ввести значення температурного коефіцієнту опору (поле з надписом «a»). Після натискання на кнопку «Analysis» буде побудовано сімейство амплітудо-частотних характеристик для різних значень опорів резистора, що залежить від температури, як показано на рис. 4.43.

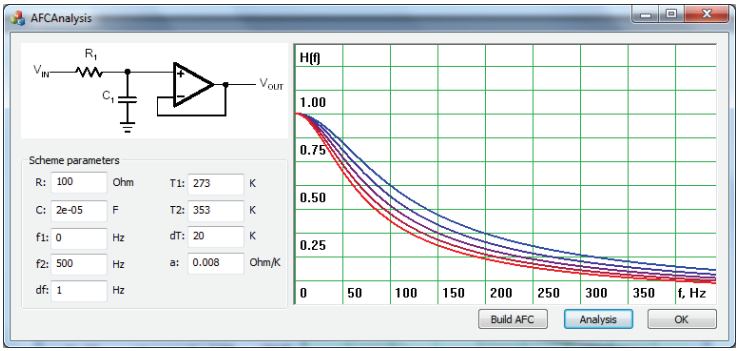


Рис. 4.43.

Як видно з рисунку, при збільшенні температури опір резистора також збільшується, що призводить до зсуву частоти зрізу фільтра в область низьких частот.

Для підвищення крутизни спаду АЧХ фільтру використовуються фільтри більш високих порядків, які отримуються шляхом послідовного з'єднання фільтрів 1-го й 2-го порядків.

На рис. 4.44 представлено схему активного фільтру низьких частот 2 порядку.

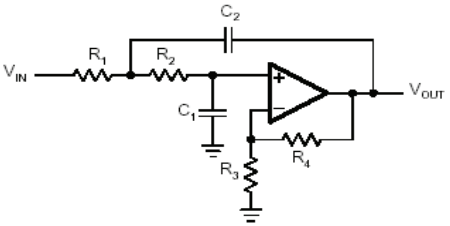


Рис. 4.44.

Коефіцієнт підсилення для заданої схеми дорівнює:

$$A_0 = 1 + \frac{R_4}{R_3}.$$

Якщо необхідно одиничний коефіцієнт підсилення, схема набуває вигляду, як показано на рис. 4.45.

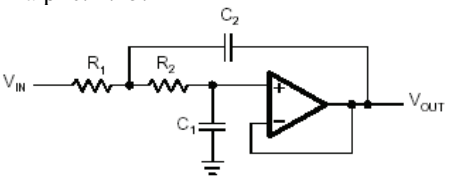


Рис. 4.45.

Передатна функція такого фільтру має вигляд:

$$A(s) = \frac{1}{1 + \omega_c C_1 (R_1 + R_2) p + \omega_c^2 R_1 R_2 C_1 C_2 p^2}.$$

Коефіцієнти a_1й b_1 визначаються виразами:

$$A_0 = 1,$$
$$a_1 = \omega_c C_1 (R_1 + R_2),$$
$$b_1 = \omega_c^2 R_1 R_2 C_1 C_2.$$

При проектуванні фільтру обирають значення ємностей конденсаторів C_1 й C_2, при цьому повинна виконуватися нерівність:

$$C_2 \geq C_1 \frac{4b_1}{a_1^2}.$$

Після цього розраховуються опори резисторів за формулою:

$$R_{1,2} = \frac{a_1 C_2 \mp \sqrt{a_1^2 C_2^2 - 4 b_1 C_1 C_2}}{4\pi f_c C_1 C_2}.$$

На рис. 4.46 представлено зовнішній вигляд діалогово вікна програми, призначеної для аналізу частотних характеристик активних фільтрів низьких частот другого порядку. На діалозі розміщено елемент Picture Control у верхньому куті ліворуч, в якому відображатиметься принципова схема фільтру. В порівнянні з попередньою програмою на діалозі розміщено додаткові елементи Edit Control для введення значень опору другого резистора та ємності ще одного конденсатора. Для виконання параметричного аналізу схеми та дослідження впливу зміни температури на характеристики фільтра внаслідок зміни опорів резисторів використовуються температурні коефіцієнти опору a_1 та a_2 (для першого та другого резистора відповідно). Загалом, інтерфейс користувача цієї програми дуже схожий на інтерфейс, розглянутий в попередньому прикладі.

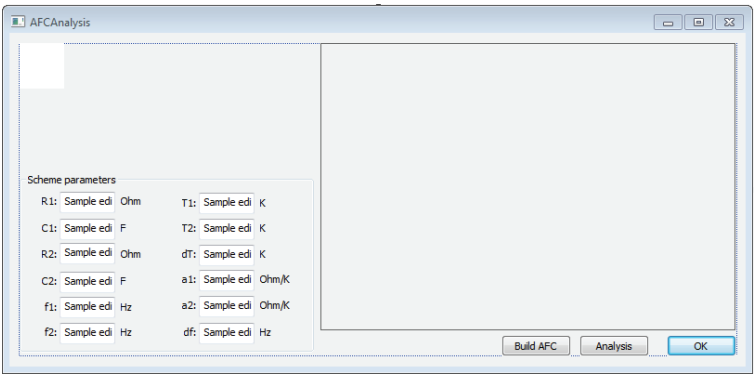


Рис. 4.46.

На рис. 4.47 представлено діалогове вікно Class Wizard (вкладка Member Variables), за допомогою якого в клас діалога було додано змінні-члени, зв'язані з відповідними елементами керування. З назв змінних зрозуміло, з якими саме елементами керування вони зв'язані.

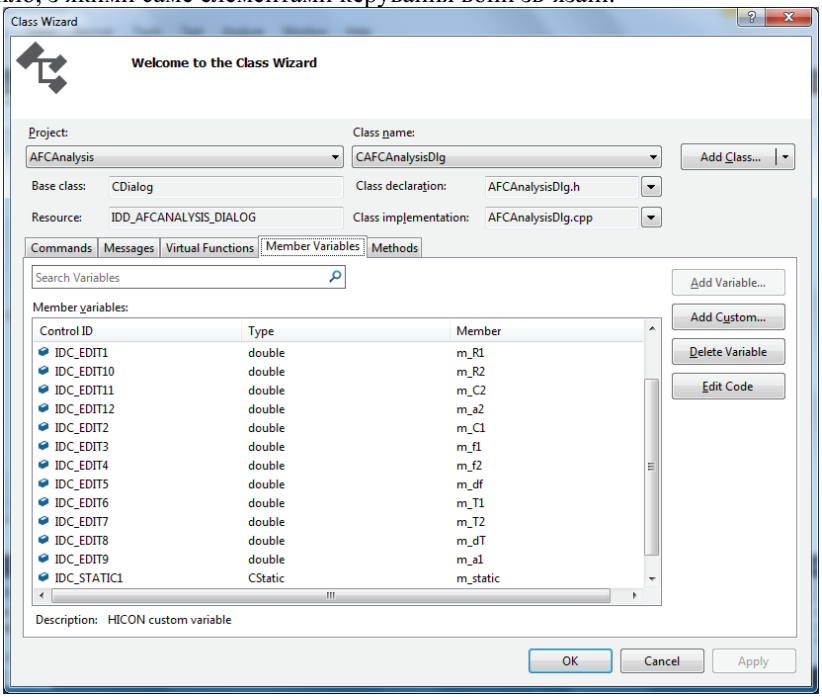


Рис. 4.47.

В програмі для роботи з комплексними числами використовується клас зі стандартної бібліотеки. Для того, щоб мати можливість його використовувати, у файлі реалізації класу діалогу, що має розширення .cpp, потрібно підключити відповідну бібліотеку шляхом додавання наступного рядка:

```
#include <complex>
```

Подвійне натискання на кнопку з надписом «Build AFC», що розташована на діалозі, призведе до створення функції-обробника події натискання на цю кнопку. Саме в цій функції виконується розрахунок та побудова амплітудно-частотної характеристики фільтра:

```
using namespace std::complex_literals;
void CAFCAnalysisDlg::OnBnClickedButton1()
{
// зчитування значень з елементів керування у відповідні змінні
```

```
UpdateData(true);
// створення об’єкту контексту пристрою
CClientDC dc(&m_static);
// об’єкт для збереження інформації про прямокутну область
CRect rect;
// зчитування інформації про розміри елементу Static Text,
// на якому відображатиметься АЧХ фільтра
m_static.GetClientRect(&rect);
// в окремі змінні копіюються значення висоти та ширини прямокутної
// області побудови АЧХ
int height = rect.Height();
int width = rect.Width();
// створення об’єкту регіону для обмеження області виведення графіки
CRgn rgn;
// регіон має прямокутну форму
rgn.CreateRectRgn(0, 0, width, height);
// вибір створеного регіону в контест пристрою
dc.SelectClipRgn(&rgn);
// заповнення області побудови АЧХ білим кольором
dc.FillRect(&rect, WHITE_BRUSH);
// оголошення змінної для зберігання значення масштабу
double scale = 200;
// створення об’єкту пера зеленого кольору для малювання ліній сітки
CPen greenPen(PS_SOLID, 1, RGB(0,180,0));
// вибір зеленого пера в контекст пристрою
dc.SelectObject(&greenPen);

// малювання вертикальних ліній сітки, крок 50 пікселів
for (int x = 0; x < width; x = x + 50)
{
        dc.MoveTo(x, 0);
        dc.LineTo(x, height);
}
// малювання горизонтальних ліній сітки, крок 25 пікселів
for (int y = height; y > 0; y = y - 25)
{
        dc.MoveTo(0, y);
        dc.LineTo(width, y);
}
// малювання текстових надписів біля ліній сітки
dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
        // значення коефіцієнту передачі фільтра (текстові надписи біля ліній сітки)
for (int i = 1; i < 5; i++)
{
        CString s;
        s.Format(_T("%.2f"), 50 * i / scale);
        dc.TextOutW(5, height - 20 - i * 50, s);
}
// значення частоти (текстові надписи біля ліній сітки)
for (int i = 0; i < 8; i++)
{
        CString s;
```

```
        s.Format(_T("%d"), (int)m_f1 + i * 50);
        dc.TextOutW(5 + i * 50, height - 20, s);
}
// виведення текстового надпису у верхньому куті ліворуч
dc.TextOutW(5, 5, _T("H(f)"));
// виведення текстового надпису у нижньому куті праворуч
dc.TextOutW(width - 40, height - 20, _T("f, Hz"));

// оголошення змінної для зберігання значення кутової частоти
double w;
// визначення необхідної кількості елементів масиву для
// зберігання розрахованих значень передавальної функції фільтра
// для кожної з частот
int SIZE = (int)(m_f2 - m_f1) / m_df;
// динамічне виділення пам'яті для масиву
double *afc = new double[SIZE];
// встановлення всіх значень масиву в 0
memset(afc, 0, sizeof(afc));
// оголошення додаткових змінних для реалізації алгоритму розрахунку
int N = 0;
double f=m_f1;
// цикл виконується доки умова виконується
while(f<m_f2)
{
        w = 6.28 * f; // розрахунок кутової частоти
        // комплексна передавальна функція фільтра
        std::complex<double> H = 1. / (1. + 1i * w * (m_R1 + m_R2) * m_C1 + 1i * 1i * w *
w * m_R1 * m_R2 * m_C1 * m_C2);
        afc[N] = abs(H); // модуль передавальної функції
        N++; // інкремент змінної
        f = f + m_df; // збільшення частоти на величину кроку
}

// створення об’єкту пера чорного кольору для побудови АЧХ
CPen blackPen(PS_SOLID, 2, RGB(0, 0, 0));
// вибір пера в контекст пристрою
dc.SelectObject(&blackPen);
// в циклі відбувається малювання АЧХ фільтра по розрахованим значенням
for (int k = 1; k < SIZE; k++) {
        dc.MoveTo(k, height-afc[k]*scale);
        dc.LineTo(k-1, height - afc[k-1] * scale);
}
// очищення динамічно виділеної пам'яті
delete[] afc;
}
```

Після компіляції проекту та запуску програми на виконання можна перевірити її роботу. На рис. 4.48 представлено діалогове вікно програми в режимі виконання. Для заданих значень опору резистора та ємності конденсатора виконано побудову АЧХ. Як видно з рисунку, за даних параметрів схеми фільтр представляє собою фільтр Чебишева.

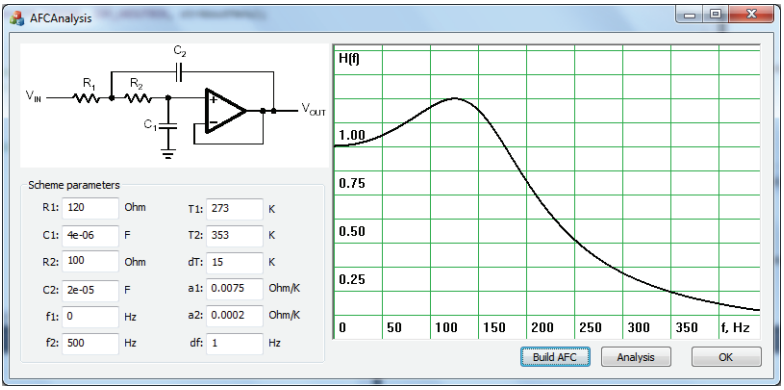


Рис. 4.48.

Реалізація функції-обробника натискання на кнопку з надписом «Analysis» має вигляд:

```
void CAFCAnalysisDlg::OnBnClickedButton2()
{
	// зчитування значень з елементів керування у відповідні змінні
	UpdateData(true);
	// створення об’єкту контексту пристрою
	CClientDC dc(&m_static);
	// об’єкт для збереження інформації про прямокутну область
	CRect rect;
	// зчитування інформації про розміри елементу Static Text,
	// на якому відображатиметься АЧХ фільтра
	m_static.GetClientRect(&rect);
	// в окремі змінні копіюються значення висоти та ширини прямокутної
	// області побудови АЧХ
	int height = rect.Height();
	int width = rect.Width();
	// створення об’єкту регіону для обмеження області виведення графіки
	CRgn rgn;
	// регіон має прямокутну форму
	rgn.CreateRectRgn(0, 0, width, height);
	// вибір створеного регіону в контекст пристрою
	dc.SelectClipRgn(&rgn);
	// заповнення області побудови АЧХ білим кольором
	dc.FillRect(&rect, WHITE_BRUSH);
	// оголошення змінної для зберігання значення масштабу
	double scale = 200;
	// створення об’єкту пера зеленого кольору для малювання ліній сітки
	CPen greenPen(PS_SOLID, 1, RGB(0, 180, 0));
	// вибір зеленого пера в контекст пристрою
	dc.SelectObject(&greenPen);

	// малювання вертикальних ліній сітки, крок 50 пікселів
```

```
for (int x = 0; x < width; x = x + 50)
{
	dc.MoveTo(x, 0);
	dc.LineTo(x, height);
}
// малювання горизонтальних ліній сітки, крок 25 пікселів
for (int y = height; y > 0; y = y - 25)
{
	dc.MoveTo(0, y);
	dc.LineTo(width, y);
}
// малювання текстових надписів біля ліній сітки
dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
 // значення коефіцієнту передачі (текстові надписи біля ліній сітки)
for (int i = 1; i < 5; i++)
{
	CString s;
	s.Format(_T("%.2f"), 50 * i / scale);
	dc.TextOutW(5, height - 20 - i * 50, s);
}
// значення частоти (текстові надписи біля ліній сітки)
for (int i = 0; i < 8; i++)
{
	CString s;
	s.Format(_T("%d"), (int)m_f1 + i * 50);
	dc.TextOutW(5 + i * 50, height - 20, s);
}
// виведення текстового надпису у верхньому куті ліворуч
dc.TextOutW(5, 5, _T("H(f)"));
// виведення текстового надпису у нижньому куті праворуч
dc.TextOutW(width - 40, height - 20, _T("f, Hz"));

// розрахунок кількості ітерацій циклу
int c = 255 / ((m_T2 - m_T1) / m_dT);
// додаткова змінна для керування кольором відображенння АЧХ
int color = 0;
// початкова температура
double T = m_T1;
// цикл виконується поки умова дійсна
while (T <= m_T2)
{
	// розрахунок значення опору при заданій температурі
	double R1 = m_R1 * (1 + m_a1 * (T - m_T1));
	double R2 = m_R2 * (1 + m_a2 * (T - m_T1));

	// додаткова змінна для зберігання значення кутової частоти
	double w;
	// розрахунок кількості необхідних елементів масиву для
	// зберігання розрахованих значень
	int SIZE = (int)(m_f2 - m_f1) / m_df;
	// динамічне виділення пам'яті для масиву
	double *afc = new double[SIZE];
```

```
            // очищення масиву (значення всіх елементів масиву дорівнює 0)
            memset(afc, 0, sizeof(afc));
            // оголошення додаткових змінних для реалізації алгоритму
            int N = 0;
            double f = m_f1;
            // цикл виконується поки умова дійсна
            while (f < m_f2)
            {
                w = 6.28 * f; // розрахунок значення кутової частоти
                // передавальна функція фільтра другого порядку
                std::complex<double> H = 1. / (1. + 1i * w * (R1 + R2) * m_C1 + 1i *
                1i * w * w * R1 * R2 * m_C1 * m_C2);
                // модуль частотної характеристики
                afc[N] = abs(H);
                // інкремент змінної
                N++;
                // збільшення частоти на величину кроку
                f = f + m_df;
            }
            // створення пера для малювання АЧХ
            // колір пера залежить від температури і з її збільшенням
            // поступово змінюється від синього до червоного
            CPen pen(PS_SOLID, 2, RGB(color, 0, 255-color));
            // вибір об'єкту пера в контекст пристрою
            dc.SelectObject(&pen);
            // в циклі відбувається малювання АЧХ
            for (int k = 1; k < SIZE; k++) {
                dc.MoveTo(k, height - afc[k] * scale);
                dc.LineTo(k - 1, height - afc[k - 1] * scale);
            }
            // збільшення значення температури на величину кроку
            T = T + m_dT;
            // додавання значення кроку зміни кольору
            color += c;
            // звільнення динамічно виділеної пам'яті
            delete[] afc;
        }
}
```

Запустивши програму на виконання, в поле T1 потрібно ввести значення початкової температури, в поле T2 – значення кінцевої температури, а в поле dT – крок зміни температури. Також потрібно ввести значення температурних коефіцієнтів опору (поля з написом «a1» та «a2»). Після натискання на кнопку «Analysis» буде побудовано сімейство амплітудно-частотних характеристик для різних значень опорів резисторів, що залежать від температури, як показано на рис. 4.49.

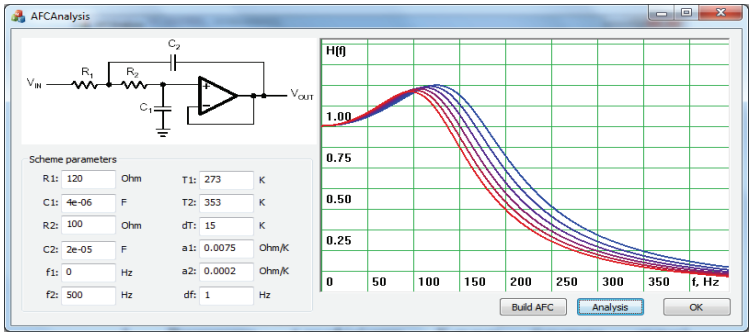


Рис. 4.49.

В даному прикладі значення температурних коефіцієнтів опору різні, і при зміні разом зі зсувом АЧХ в напрямку низьких частот також зменшується величина нерівномірності в смузі пропускання фільтру. Не виключена ситуація, коли при певних значеннях опорів резисторів та температурних коефіцієнтів опорів зміна температури оточуючого середовища призведе до перетворення фільтру Чебишева у фільтр Батерворта.

4.6. Використання чисельних методів при моделюванні перехідних процесів в електронних системах

Якщо електричне коло досить довго зберігало незмінний вид, то в ньому створюється так званий сталий (вимушений) режим. Останньому відповідають певні закони зміни енергії електричних полів конденсаторів і магнітних полів індуктивностей ланцюга. У разі зміни схеми при перемиканнях, які будемо називати комутаціями, енергія полів повинна змінитися, а для цього потрібен деякий час. Процес, що виникає в електричному колі при переході від одного сталого режиму до іншого, називається перехідним [34].

Протягом перехідного процесу (ПП) електричне коло може бути описане системою динамічних рівнянь, яка може бути зведена відносно однієї електричної величини (струму або напруги) до диференціального рівняння n-го порядку, причому його порядок визначається кількістю накопичувачів енергії (до них відносяться індуктивності і ємності). Диференціальне рівняння, що виникає, є лінійним, неоднорідним, з постійними коефіцієнтами [34].

Загальний розв'язок отриманого неоднорідного лінійного диференціального рівняння є сумою двох величин: частинний розв'язок неоднорідного рівняння, що виражає вимушений режим, що задається джерелами, і розв'язок відповідного однорідного диференціального рівняння, що виражає вільний режим.

У відповідності до цього для будь-якого струму або напруги можна записати: $i = i_{пр} + i_{св}$, $u = u_{пр} + u_{св}$,

де $i_{пр}$, $u_{пр}$ – вимушені складові струму та напруги;

$i_{св}$, $u_{св}$ – вільні складові струму та напруги.

Метод знаходження електричних величин у вигляді суми двох розглянутих складових називається класичним.

Вимушені складові розраховуються будь-якими раніше вивченими методами, а вигляд вільних складових залежить від числа і виду коренів характеристичного рівняння. Існує кілька способів складання характеристичного рівняння.

1 спосіб. За наявним диференціальним рівнянням:

$$K_n \frac{d^n i}{dt^n} + K_{n-1} \frac{d^{n-1} i}{dt^{n-1}} + \ldots + K_1 \frac{di}{dt} + K_0 \cdot i = f(t).$$

n-а похідна замінюється на p^n; . . . ; перша похідна на p; сама величина – 1; правая частина – 0, тобто:

$$K_n \cdot p^n + K_{n-1} \cdot p^{n-1} + \ldots + K_1 \cdot p + K_0 = 0.$$

2 спосіб. Шляхом запису вхідного опору в операторній формі:

1. Джерела замінюються їх внутрішніми опорами, а ключ показується в післякомутаційному стані.

2. Ланцюг розмикається в будь-якому місці. Рекомендується розривати в гілці з конденсатором, а при його відсутності – в гілці з індуктивністю.

3. Відносно отриманих зажимів записується вхідний опір в комплексній формі $\underline{Z}(j\omega)$ (індуктивний опір – $j\omega L$, а ємність – $1/(j\omega C)$.

4. Виконується заміна $j\omega = p$. Отримуємо вхідний опір $Z(p)$ в операторній формі.

5. Отриманий опір прирівнюємо до нуля, тобто $Z(p) = 0$. Це і є характеристичний опір.

3 спосіб. Використовуючи систему динамічних рівнянь ланцюга:

1. Складається система динамічних рівнянь за законами Кірхгофа для післякомутаційного стану ланцюга.

2. Отримана система з диференціальних рівнянь перетворюються в алгебраїчні в операторній формі.

3. Визначник системи прирівнюється до нуля і виходить характеристичне рівняння.

Якщо корінь характеристичного рівняння один (обов'язково від'ємний), вільна складова має вигляд: $i_{св}(t) = A \cdot e^{pt}$,

де A – постійна інтегрування;

Якщо коренів два, обидві дійсні, від'ємні, різні, причому $|p_1| < |p_2|$, тоді

$$i_{св}(t) = A_1 \cdot e^{p_1 t} + A_2 \cdot e^{p_2 t}.$$

Якщо коренів два – дійсні, від'ємні, рівні ($p_1 = p_2 = p$), тоді

$$i_{св}(t) = A_1 \cdot e^{pt} + A_2 \cdot t \cdot e^{pt},$$

де A_1 і A_2 – дві постійні інтегрування;

Якщо коренів два – комплексні, сполучені, тобто $p_{1,2} = -b \pm jz$, тоді

$$i_{св}(t) = A \cdot e^{-bt} \cdot \sin(zt + \theta),$$

де A і θ – постійні інтегрування.

Кількість коренів характеристичного рівняння визначає число постійних інтегрування і дорівнює кількості накопичувачів енергії в ланцюгу після комутації.

Постійні інтегрування знаходяться з початкових умов (значення електричних величин та їх похідних в початковий момент після комутації), які діляться на незалежні і залежні. До незалежних відносяться значення в момент комутації потокозчеплення і струму індуктивності, заряду і напруги конденсатора. Решта початкових умов вважаються залежними.

Зазначені вище положення про те, що запас енергії магнітного або електричного поля може змінюватися тільки плавно, без стрибків, становлять принцип безперервності в часі потокозчеплення індуктивності і електричного заряду ємності, і називаються законами комутації [34].

Перший закон комутації: в індуктивному елементі струм і магнітний потік безпосередньо після комутації зберігають значення, які вони мали безпосередньо перед комутацією, і далі починають змінюватися саме з цих значень: $\Psi(0_+) = \Psi(0_-)$, $i_L(0_+) = i_L(0_-)$,

де $t = 0_+$ – момент відразу після комутації,

$t = 0_-$ – момент безпосередньо перед комутацією.

Другий закон комутації: на ємнісному елементі напруга і заряд зберігають в момент комутації ті значення, які вони мали безпосередньо перед комутацією, і в подальшому змінюються, починаючи з цих значень: $q(0_+) = q(0_-)$, $u_C(0_+) = u_C(0_-)$.

При нульових початкових умовах ($i_L(0_-) = 0$, $u_C(0_-) = 0$) індуктивність в початковий момент після комутації рівносильний розриву ланцюга, а ємність – короткому замиканню. У випадку ненульових початкових умов ($i_L(0_-) \neq 0$, $u_C(0_-) \neq 0$) індуктивність в момент $t = 0_+$ рівносильна джерелу струму, а ємність – джерелу ЕРС.

Залежно від порядку диференціальних рівнянь розрізняють кола першого, другого і більш високого порядку.

Рекомендується наступний порядок розрахунку ПП:

1. Аналіз ланцюга до комутації та визначення незалежних початкових умов.

2. Запис шуканих електричних величин (струмів і напруг) у вигляді суми двох складових – вимушеної і вільної.

3. Розрахунок вимушених складових.

4. Форма вільних складових залежить від числа і виду коренів характеристичного рівняння. Тому складається і вирішується характеристичне рівняння.

5. Запис вільних складових з урахуванням виду коренів.

6. Визначення необхідних початкових умов.

7. Знаходження постійних інтегрування виходячи з початкових умов.

8. Запис шуканих величин в остаточній формі.

4.6.1. Програмна реалізація явного і неявного методів чисельного інтегрування Ейлера

Якщо процеси в електричному колі описуються диференціальним рівнянням високого порядку, одержати аналітичний вираз для коренів характеристичного рівняння досить важко. У таких випадках для знаходження коренів застосовуються універсальні обчислювальні алгоритми – чисельні методи розв'язання рівнянь. При використанні такого методу розрахунку чисельними методами розв'язується диференціальне рівняння електричного кола. При використанні чисельних методів не одержуються аналітичні вирази для напруг $u(t)$ та струмів $i(t)$, однак можна визначити їх чисельні значення в будь-який заданий момент часу t_k: $u(t_k)$ та $i(t_k)$. Головним недоліком чисельних методів є те, що результати аналізу стосуються конкретного електричного кола у конкретному режимі його роботи. При будь-яких змінах в електричному колі, чисельний аналіз необхідно виконувати спочатку.

ПРИКЛАД 4.6. Виконати розрахунок перехідного процесу в схемі на рис. 4.50, застосувавши явний і неявний метод чисельного інтегрування Ейлера. Параметри елементів схеми наступні: E=50 В, L=0,02 Гн, R_1=10 Ом, R_2=20 Ом.

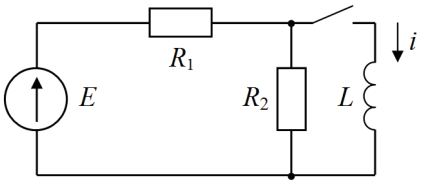


Рис. 4.50.

Складаємо диференціальне рівняння застосувавши метод еквівалентного генератора:

$$E_e = \frac{ER_2}{R_1 + R_2},$$

$$R_e = \frac{R_1 R_2}{R_1 + R_2},$$

$$u_{Le} = L_e \frac{di}{dt} = E - ir.$$

Початкові умови $i(0_-) = i(0_+) = 0$. Крок інтегрування h=dt.

Розрахуємо перехідний процес, застосувавши явний і неявний метод чисельного інтегрування Ейлера. Складаємо різницеве рівняння:

$$\frac{di}{dt} = \frac{i_k - i_{k-1}}{h}.$$

Неявний метод Ейлера:

$$\frac{i_k - i_{k-1}}{h} = \frac{1}{L}(E_e - i_{k-1}R_e);$$

$$i_{ke} = \frac{h}{L}(E_e - i_{k-1}R) + i_{k-1} \text{ при } k = 1,\ i_0 = 0.$$

Явний метод Ейлера:

$$\frac{i_k - i_{k-1}}{h} = \frac{1}{L}(E_e - i_k R_e),$$

$$i_k = \left(\frac{hE_e}{L} + i_{k-1}\right) / \left(1 + \frac{hR_e}{L}\right) \text{ при } k = 1,\ i_0 = 0.$$

На рис. 4.51 представлено вікно середовища розробки Visual Studio .NET з відкритим редактором ресурсів. На рисунку представлено макет діалогового вікна, на якому розташовано елемент керування Static Text, в якому відображатиметься осцилограма струму котушки під час перехідного процесу, а також кнопку з надписом «Calculate», натискання на яку призведе до виконання алгоритму розрахунку перехідного процесу чисельним методом та відображення відповідних результатів.

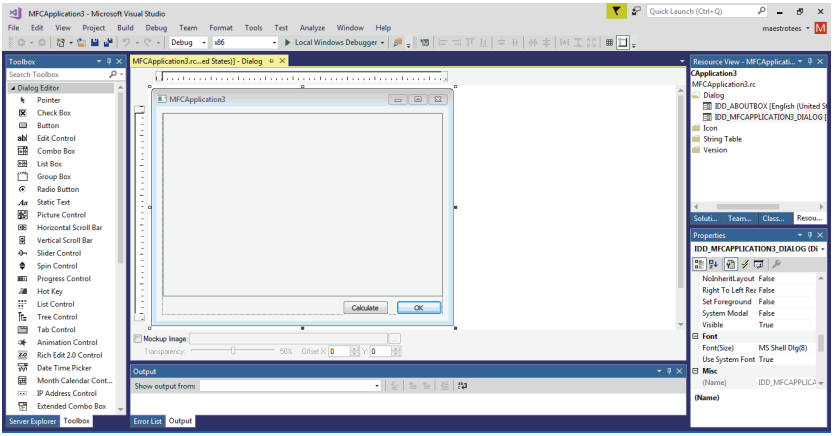


Рис. 4.51.

Для того, щоб мати програмний доступ до елементів керування на діалоговому вікні, з ними зв'язані змінні-члени класу. На рис. 4.52 представлено діалогове вікно Class Wizard, в якому на вкладці Member Variables видно назву та тип змінної, що зв'язана з відповідним елементом керування.

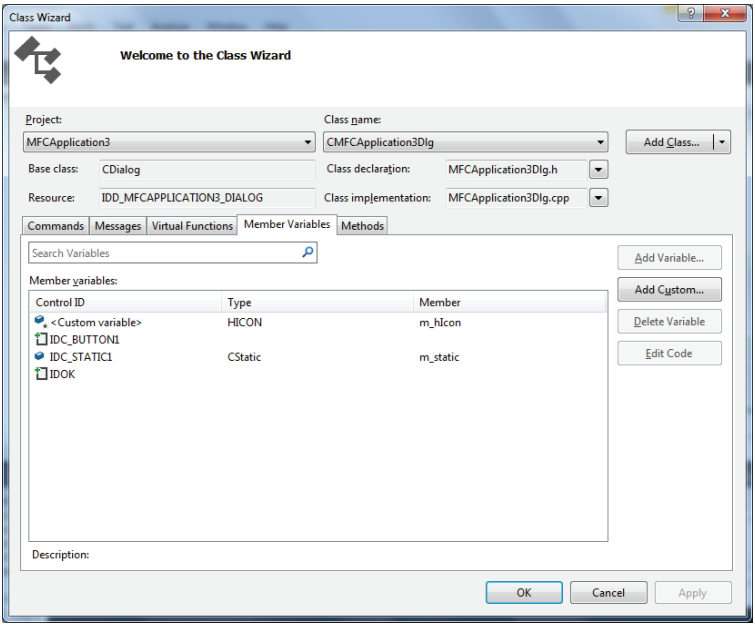


Рис. 4.52.

Функція-обробник натискання на кнопку «Calculate» має вигляд:

```
void CMFCApplication3Dlg::OnBnClickedButton1()
{
	UpdateData(true); // зчитування даних з елементів керування у відповідні змінні

	CRect rect; // об’єкт для збереження інформації про прямокутну область
	// отримання даних про розміри елементу Static Text
m_static.GetClientRect(&rect);
// значення ширини та висоти елементу керування копіюються
	int W = rect.Width();
	int H = rect.Height(); // в окремі змінні для зручності використання в подальшому
	CClientDC dc(&m_static); // створення об’єкту контексту пристрою
// заповнення прямокутної області білим кольором
	dc.FillRect(&rect, WHITE_BRUSH);

	double scale_I = 50; // змінна для зберігання значення масштабу струму
	double scale_U = 5; // змінна для зберігання значення масштабу напруги

	// параметри елементів сїеми
	double R1 = 10, R2 = 40, E = 50, L = 0.02;
	// розрахунок еквівалентного опору кола
	double Re = (R1*R2) / (R1 + R2);
	// розрахунок значення еквівалентної ЕРС
	double Ee = (E*R2) / (R1 + R2);
```

```
        // масив для зберігання розрахованих значень струму неявним методом Ейлера
        double I[500] = { 0 };
        // масив для зберігання розрахованих значень струму явним методом Ейлера
        double I1[500] = { 0 };
        // масив для зберігання розрахованих значень напруги на котушці
        double U[500] = { 0 };

        I[0] = 0; // значення струму в котушці в момент комутації (початкові умови)
        I1[0] = 0;
        U[0] = 0; // значення напруги на котушці в момент комутації
        double h = 0.00005; // крок інтегрування
        // малювання вертикальних ліній сітки, крок 50 пікселів
        for (int x = 0; x < W; x = x + 50)
        {
                dc.MoveTo(x, 0);
                dc.LineTo(x, H);
        }
        // малювання горизонтальних ліній сітки, крок 50 пікселів
        for (int y = H; y > 0; y = y - 50)
        {
                dc.MoveTo(0, y);
                dc.LineTo(W, y);
        }
        // малювання текстових надписів біля ліній сітки
        dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
// значення струму (текстові надписи біля ліній сітки ліворуч)
        for (int i = 0; i < 6; i++)
        {
                CString s;
                s.Format(_T("%.0f"), 50 * i / scale_I);
                dc.TextOutW(5, H - 15 - i * 50, s);
        }

        // значення напруги (текстові надписи біля ліній сітки праворуч)
        for (int i = 0; i < 6; i++)
        {
                CString s;
                s.Format(_T("%.0f"), 50 * i / scale_U);
                dc.TextOutW(W - 25, H - 15 - i * 50, s);
        }

        // значення часу (текстові надписи біля ліній сітки)
        for (int i = 1; i < 8; i++)
        {
                CString s;
                s.Format(_T("%.0f"), 50 * i * h * 1000);
                dc.TextOutW(5 + i * 50, H - 15, s);
        }
        // виведення текстового надпису у верхньому куті ліворуч
        dc.TextOutW(5, 5, _T("I, A"));
        // виведення текстового надпису у верхньому куті праворуч
        dc.TextOutW(W - 35, 5, _T("U, V"));
```

```
        // виведення текстового надпису у нижньому куті праворуч
        dc.TextOutW(W - 70, H - 15, _T("t, ms"));
        // в циклі виконується розрахунок струму та напруги за
// допомогою чисельних методів
        for (int k = 1; k < 500; k++)
        {
                // неявний метод Ейлера
                I[k] = (h / L)*(Ee - I[k - 1] * Re) + I[k - 1];
                // явний метод Ейлера
                I1[k] = (h*Ee / L + I1[k - 1]) / (1 + h * Re / L);
        // напруга на котушці (похідна струму помножена на значення індуктивності)
                U[k] = L * (I[k] - I[k - 1]) / h;
        }

        // створення об’єктів пера різних кольорів
        CPen redPen(PS_SOLID, 2, RGB(255, 0, 0));          // перо червоного кольору
        CPen magentaPen(PS_SOLID, 1, RGB(255, 0, 255)); // перо фіолетового кольору
        CPen bluePen(PS_SOLID, 2, RGB(0, 0, 255)); // перо синього кольору

        dc.SelectObject(&redPen); // вибір червоного пера в контекст пристрою
        // в циклі відбувається малювання графіку струму в котушці
        // під час перехідного процесу розрахованого неявним методом Ейлера
        for (int k = 1; k < W; k++)
        {
                dc.MoveTo(k, H - scale_I * I[k]);
                dc.LineTo(k - 1, H - scale_I * I[k - 1]);
        }

        dc.SelectObject(&magentaPen); // вибір фіолетового пера в контекст пристрою
        // в циклі відбувається малювання графіку струму в котушці
        // під час перехідного процесу розрахованого явним методом Ейлера
        for (int k = 1; k < W; k++)
        {
                dc.MoveTo(k, H - scale_I * I1[k]);
                dc.LineTo(k - 1, H - scale_I * I1[k - 1]);
        }

        dc.SelectObject(&bluePen); // вибір синього пера в контекст пристрою
        // в циклі відбувається малювання графіку напруги на котушці
        for (int k = 1; k < W; k++)
        {
                dc.MoveTo(k, H - scale_U * U[k]);
                dc.LineTo(k - 1, H - scale_U * U[k - 1]);
        }
}
```

Після компіляції проекту та запуску програми на виконання можна перевірити її роботу. Після натискання на кнопку Calculate на діалозі (рис. 4.53) відобразяться графіки струму (червона лінія) та напруги (синя лінія) на котушці під час перехідного процесу.

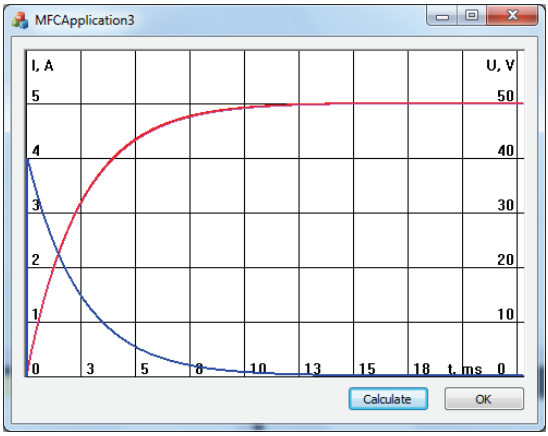


Рис. 4.53.

4.6.2. Програмна реалізація методу Рунге-Кутта

ПРИКЛАД 4.7. Розглянемо послідовне *RLC*-коло (рис. 4.54). Будемо вважати, що ємнісний елемент заряджений до напруги $u_C(0)=U_0$, а початковий струм індуктивного елементу $i_L(0)=0$. Оскільки незалежні джерела в колі відсутні, струми і напруги в такому колі є реакцією при нульовому вхідному сигналі (реакцією при нульовому вході).

У відповідності до другого закону Кірхгофа:

$$L\frac{di}{dt}+Ri+\frac{1}{C}\int idt=0.$$

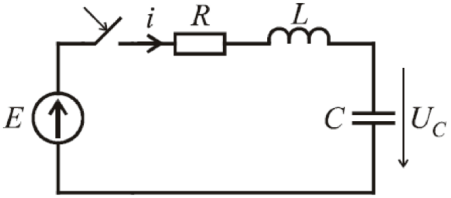


Рис. 4.54.

Приймемо, що незалежні початкові умови нульові, тобто $u_C(0)=0$, $i_L(0)=0$.

Продиференціювавши ліву і праву частини, отримуємо однорідне диференціальне рівняння другого порядку:

$$L\frac{d^2i}{dt}+R\frac{di}{dt}+\frac{1}{C}i=0.$$

Постійна загасання $\alpha=R/2L$. Частота власних коливань $\omega_0=1/LC$.

Чисельне рішення лінійного однорідного диференціального рівняння 2 порядку методом Рунге-Кутта.

Розглядається лінійне однорідне диференціальне рівняння другого порядку виду:

$$a\frac{d^2y}{dt}+b\frac{dy}{dt}+cy=0.$$

Для того, щоб застосувати до нього чисельний метод Рунге-Кутта, слід звести це рівняння до системи 2-х диференціальних рівнянь 1-го порядку:

Спростимо: $b=b/a$, $c=c/a$.

Ввівши нову функцію $x(t)$, отримуємо:

$$\frac{dy}{dt}=x,$$

$$\frac{dx}{dt}=-bx-cy.$$

Далі застосовується метод Рунге-Кутта. Розрахунок ведеться за формулами:

$k1[i] = hf'(x[i], f[i])$;
$k2[i] = hf'(x[i]+h/2, f[i]+k1[i]/2)$;
$k3[i] = hf'(x[i]+h/2, f[i]+k2[i]/2)$;
$k4[i] = hf'(x[i]+h/2, f[i]+k3[i])$;

$deltaf[i]=1/6(k1[i]+2\cdot k2[i]+2\cdot k3[i]+k4[i])$;
$f[i+1]=f[i]+deltaf[i]$.

У випадку цієї системи формули виглядають наступним чином:

Для функції x:
$kx1[i]=h\cdot(-c\cdot y[i]-b\cdot x[i])$;
$kx2[i]=h\cdot(-c\cdot y[i]-b\cdot(x[i]+kx1[i]/2))$;
$kx3[i]=h\cdot(-c\cdot y[i]-b\cdot(x[i]+kx2[i]/2))$;
$kx4[i]=h\cdot(-c\cdot y[i]-b\cdot x[i]+kx3[i]))$;
$deltax[i]=1/6\cdot(kx1[i]+2\cdot kx2[i]+2\cdot kx3[i]+kx4[i])$;

$x[i+1]=x[i]+deltax[i]$.

Для функції y:
$ky1[i]=h\cdot x[i]$;
$ky2[i]=h\cdot(x[i]+ky1[i]/2)$;
$ky3[i]=h\cdot(x[i]+ky2[i]/2)$;
$ky4[i]=h\cdot(x[i]+ky3[i])$;
$deltay[i]=1/6\cdot(ky1[i]+2\cdot ky2[i]+2\cdot ky3[i]+ky4[i])$;

$y[i+1]=y[i]+deltay[i]$.

На рис. 4.55 представлено вигляд діалогово вікна, спроектованого у редакторі ресурсів середовища розробки Visual Studio.

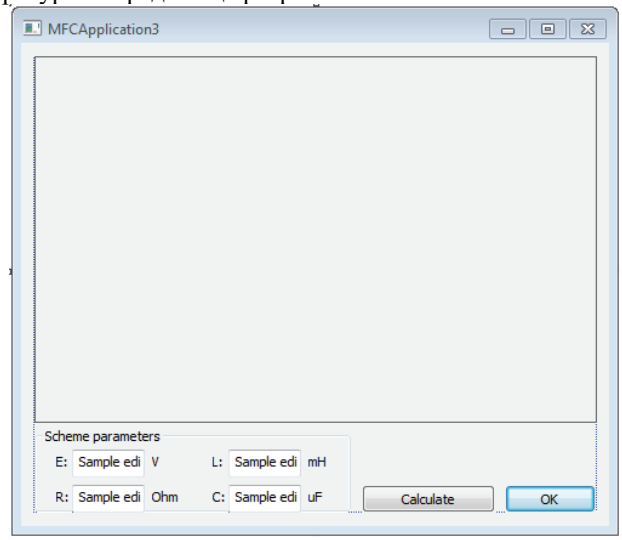


Рис. 4.55.

В елементи керування, що розташовані на діалоговому вікні, користувачу при проботі з програмою потрібно буде ввести значення ЕРС, опору резистора, індуктивність котушки та ємність конденсатора. Після натискання на кнопку Calculate буде виконано розрахунок перехідного процесу та в елементі Static Text, який займає більшу частину діалогово вікна та має рамку, буде відображено графік струму в колі.

Для додавання змінної-члену класу діалога, що зв’язана з елементом керування, потрібно натиснути праву кнопку миші на потрібному елементі та вибрати з контестного меню команду Add Variable. У діалоговому вікні, що відкриється після цього (рис. 4.56) потрібно задати назву змінної, вибрати її категорію та вказати тип.

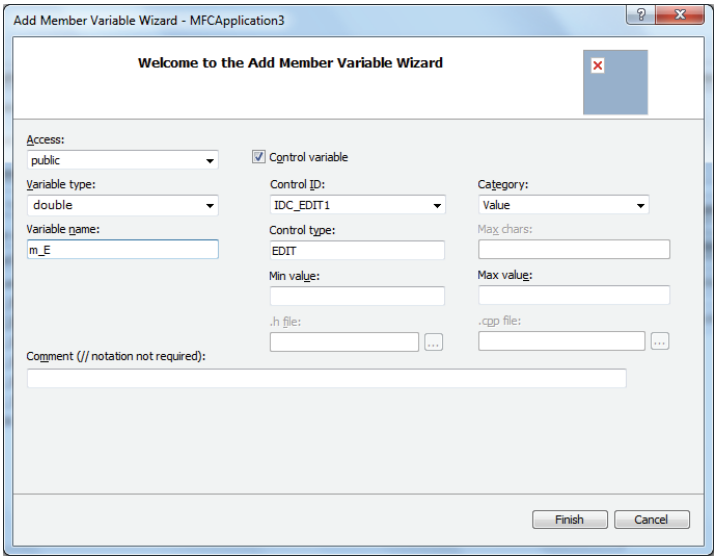


Рис. 4.56.

З усіма елементами керування, що розташовані на діалоговому вікні, зв'язано ідповідні змінні за допомогою Class Wizard (Ctrl+Shift+X), вікно якого представлено на рис. 4.57.

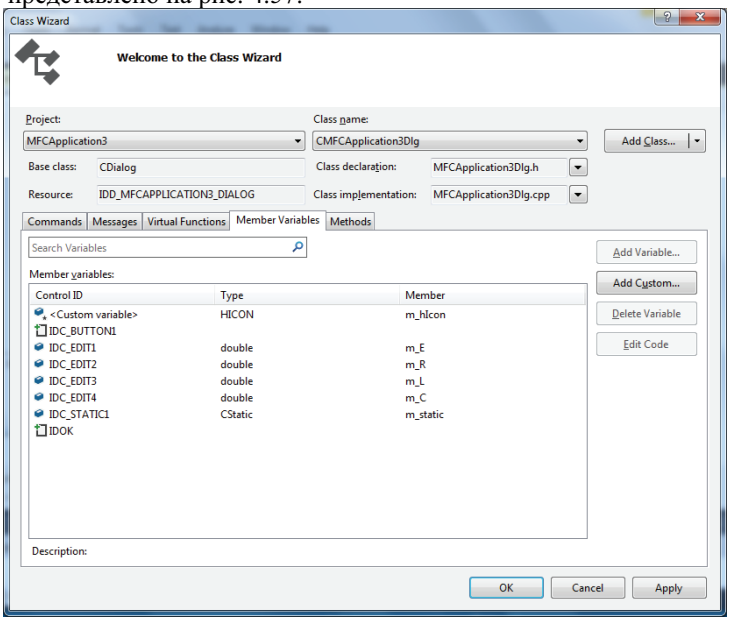


Рис. 4.57.

Програмна реалізація розрахунку перехідного процесу з використанням чисельного методу Рунге-Кутта має вигляд:

```
void CMFCApplication3Dlg::OnBnClickedButton1()
{
	UpdateData(true); // зчитування даних з елементів керування у відповідні змінні
	CRect rect; // об’єкт для зберігання інформації про прямокутну область
	m_static.GetClientRect(&rect); // отримання розмірів елементу Static Text
	int W = rect.Width(); // значення ширини елементу керування
// копіюється в окрему змінну
	int H = rect.Height(); // значення висоти елементу керування
// копіюється в окрему змінну
	CClientDC dc(&m_static); // створення об’єкту контексту пристрою
// заповнення прямокутної області білим кольором
	dc.FillRect(&rect, WHITE_BRUSH);

	double scale_I = 50; // значення масштабу для відображення струму
	// значення параметрів елементів схеми
	double R = m_R, L = m_L / 1000, C = m_C / 1000000, E = m_E;

	double a, b, c;
	a = L;
	b = R;
	c = 1 / C;
	// спрощуємо рівняння
	b = b / a;
	c = c / a;
	double I0;
	double II0;
	double X[500] = { 0 };
	double Y[500] = { 0 };
	// оголошення додаткових змінних для використання при реалізації
// розрахунків методом Рунге-Кутта
	double kx1, kx2, kx3, kx4;
	double ky1, ky2, ky3, ky4;
	double delta_x, delta_y;

	I0 = 0;  // значення струму в момент комутації (початкові умови)
	II0 = E / L; // значення першої похідної струму в момент комутації

	double h = 0.00005; // крок інтегрування

// задання початкових умов
	X[0] = II0; // значення першої похідної струму в момент комутації
	Y[0] = I0; // значення струму в момент комутації
// реалізація чисельного методу Рунге-Кутта
	for (int k = 0; k < 499; k++)
	{
		kx1 = h * (-c * Y[k] - b * X[k]);
		kx2 = h * (-c * Y[k] - b * (X[k] + kx1 / 2));
		kx3 = h * (-c * Y[k] - b * (X[k] + kx2 / 2));
```

```
		kx4 = h * (-c * Y[k] - b * (X[k] + kx3));
		delta_x = (double) 1 / 6 * (kx1 + 2 * kx2 + 2 * kx3 + kx4);
		X[k + 1] = X[k] + delta_x;

		ky1  = h * X[k];
		ky2 = h * (X[k] + ky1 / 2);
		ky3 = h * (X[k] + ky2 / 2);
		ky4 = h * (X[k] + ky3);
		delta_y = (double) 1 / 6 * (ky1 + 2 * ky2 + 2 * ky3 + ky4);
		Y[k + 1] = Y[k] + delta_y;
	}

	// малювання вертикальних ліній сітки, крок 50 пікселів
	for (int x = 0; x < W; x = x + 50)
	{
		dc.MoveTo(x, 0);
		dc.LineTo(x, H);
	}
	// малювання горизонтальних ліній сітки, крок 50 пікселів
	for (int y = H; y > 0; y = y - 50)
	{
		dc.MoveTo(0, y);
		dc.LineTo(W, y);
	}
	// малювання текстових надписів біля ліній сітки
	dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
	 // значення струму (текстові надписи біля ліній сітки ліворуч)
	for (int i = 1; i < 6; i++)
	{
		CString s;
		s.Format(_T("%.0f"), 50 * (i - 3) / scale_I);
		dc.TextOutW(5, H - 15 - i * 50, s);
	}
	// значення часу (текстові надписи біля ліній сітки)
	for (int i = 0; i < 8; i++)
	{
		CString s;
		s.Format(_T("%.0f"), 50 * i * h * 1000);
		dc.TextOutW(5 + i * 50, H - 15, s);
	}
	// виведення текстового надпису у верхньому куті ліворуч
	dc.TextOutW(5, 5, _T("I, A"));
	// виведення текстового надпису у нижньому куті праворуч
	dc.TextOutW(W - 40, H - 15, _T("t, ms"));

	// створення об’єкту пера червоного кольору
	CPen redPen(PS_SOLID, 2, RGB(255, 0, 0));
	// вибір об’єкту пера в контекст пристрою
	dc.SelectObject(&redPen);
// малювання графіку струму під час перехідного процесу
	for (int k = 1; k < W; k++)
	{
```

```
            dc.MoveTo(k, H / 2 - scale_I * Y[k]);
            dc.LineTo(k - 1, H / 2 - scale_I * Y[k - 1]);
        }
}
```

Результати виконання програми представлені на рис. 4.58 та рис. 4.59.

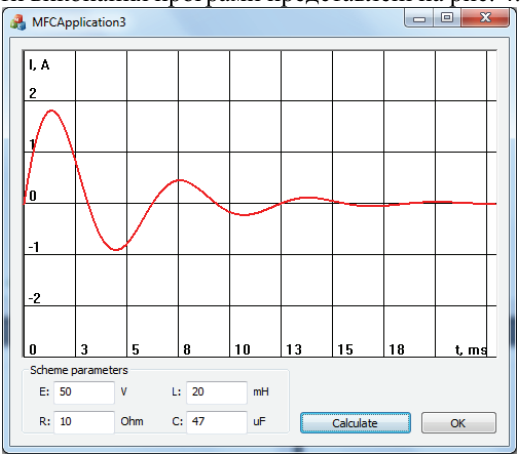


Рис. 4.58.

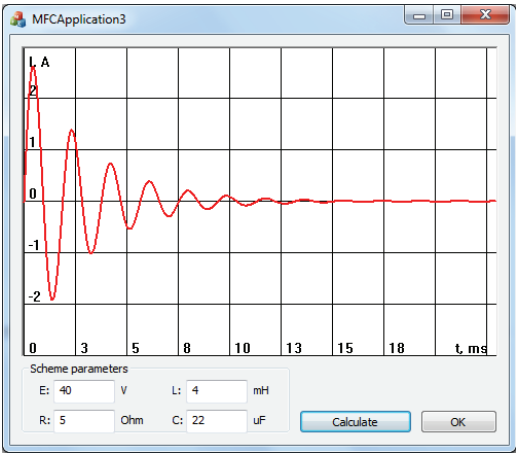


Рис. 4.59.

Застосувавши класичний метод розрахунку перехідного процесу можна отримати такі самі результати. Читачу пропонується самостійно виконати такий розрахунок.

4.7. Програмні засоби моделювання електронних систем з напівпровідниковими елементами

На рис. 4.60 наведено схему включення біполярного транзистора з загальним емітером. Для оцінки роботи біполярного транзистора в широкому діапазоні імпульсних і постійних струмів, потужностей і напруг, а також для розрахунку ланцюга зміщення, стабілізації режиму використовуються сімейства вхідних і вихідних вольтамперних характеристик (ВАХ). На рис. 4.61 представлено сімейства вихідних ВАХ широко розповсюджених в минулому транзисторів КТ315.

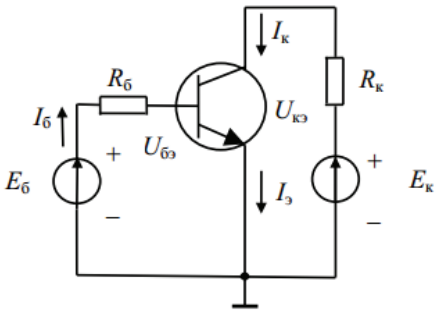


Рис. 4.60.

Необхідно розробити програму, що дозволяє виконати розрахунок струму та колектора та напруги колектор-емітер. Вхідна ВАХ транзистора задається у вигляді опорних точок.

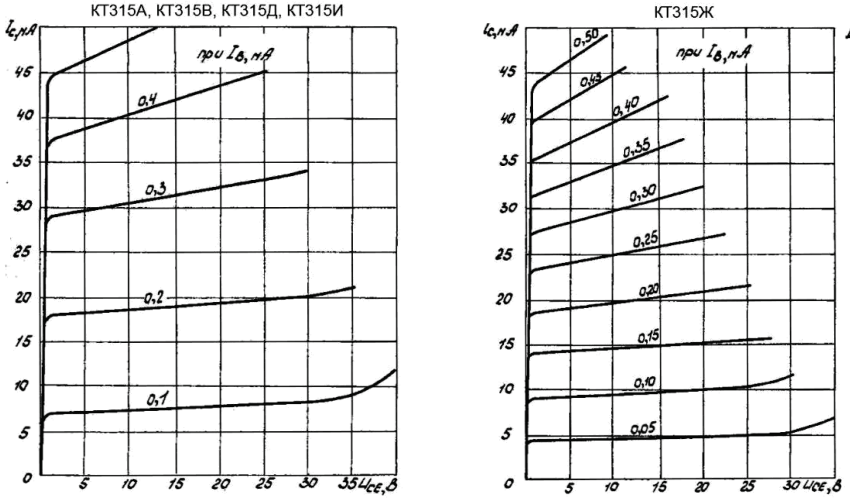


Рис. 4.61.

Першим етапом розробки програмного забезпечення є проектування інтерфейсу користувача. На рис. 4.62 представлено один з можливих варіантів реалізації інтерфейсу програми. В лівій частині діалогово вікна розміщено елемент Static Text, в якому відображатиметься вхідна ВАХ транзистора та навантажувальна характеристика. В правій частині діалогово вікна також розміщено елемент Static Text для відображення сімейства вихідних ВАХ транзистора. Ідентифікатори обох елементів керування змінено з IDC_STATIC на IDC_STATIC1 та IDC_STATIC2 відповідно. В нижній частині діалогу розміщено елементи керування Edit Control, в які користувач вводитиме параметри елементів схеми, а також для відображення розрахованого значення струму колектора та напруги колектор-емітер.

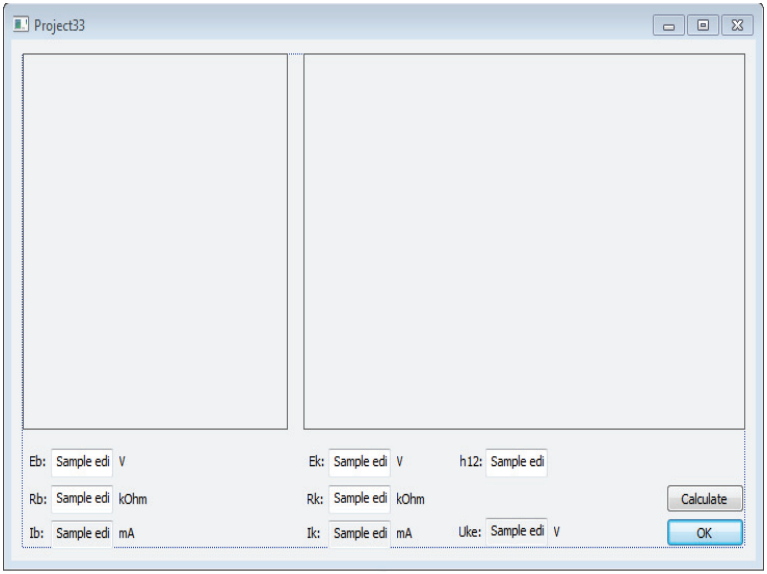


Рис. 4.62.

Для того, щоб мати можливість зчитувати введені користувачем значення з елементів керування та виводити в них результати розрахунків (тобто мати доступ до елементів керування діалогово вікна з програми), з кожним елементом керування потрібно зв’язати змінну за допомогою Class Wizard. На рис. 4.63 представлене вікно Class Wizard після додавання всіх змінних-членів класу діалогу. На рисунку видно, який саме тип мають змінні та з яким елементом керування вони зв’язані.

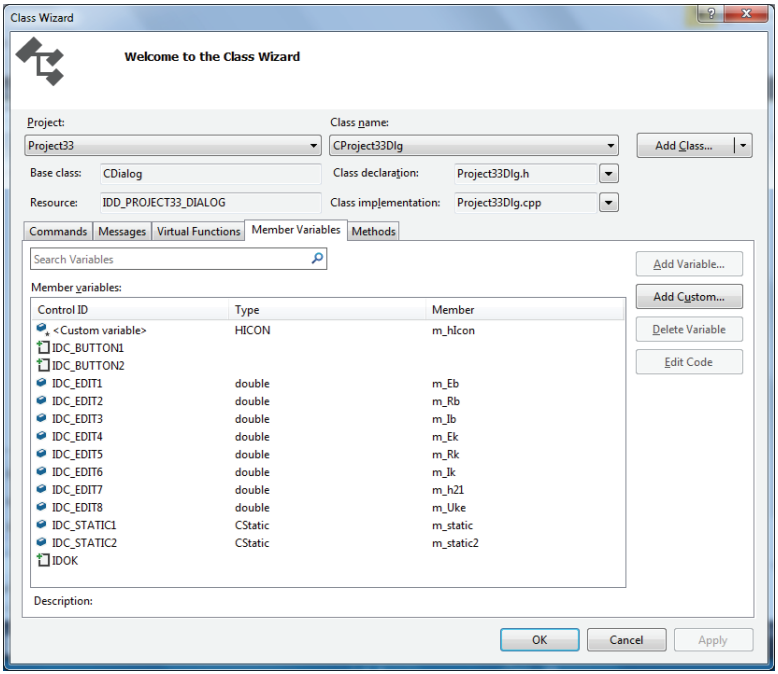


Рис. 4.63.

Як і в попередніх прикладах, на діалозі розміщено кнопку з написом Calculate, після натискання на яку будуть виконуватися всі розрахунки. Весь програмний код розміщується в функції-обробнику події натискання на цю кнопку та має наступний вигляд:

```
void CProject33Dlg::OnBnClickedButton1()
{
	// зчитування введених користувачем значень в елементи керування
	// на діалозі у відповідні змінні, що з ними зв’язані
	UpdateData(true);

	// створення об’єкту контексту пристрою
	CClientDC dc(&m_static);
	// створення об’єкту для зберігання інформації про прямокутну область
	CRect rect;
	// зчитування інформації про розміри прямокутної області, на якій
	// будуть виводитися графіки (елемент Static Text)
	m_static.GetClientRect(&rect);
	// для зручності в окремі змінні копіюються значення
	// ширини та висоти прямокутної області побудови графіків
	int W = rect.Width();
	int H = rect.Height();
	// перший елемент масиву - напруга в вольтах,
```

```
	// другий елемент масиву - струм в міліамперах
	double f[11][2] = { { 0, 0 },{ 0.5, 0.1 },{ 0.75, 0.2 },{ 0.9, 0.3 },
	{ 1,0.4 },{ 1.05, 0.5 },{ 1.1, 0.6 },{ 1.15, 0.7 },{ 1.18, 0.8 },{ 1.2, 0.9 },{ 1.22, 1 } };

	double f_interpolated[122][2] = { { 0, 0 } };
	// оголошення додаткових змінних для використання в алгоритмі інтерполяції
	double Ux = 0, x0, x1;
	int i = 0;
	int k = 0;
	// присвоєння початкових значень змінних
	x0 = f[k][0];
	x1 = f[k + 1][0];
	// в циклі виконується розрахунок значень струмів для діапазону напруг
	// від 0 В до 1,22 В з кроком 0,01 В
	while (i<122)
	{
		Ux = (double)i / 100;
		// якщо умова виконується, змінюються значення змінних x0 та x1
		// це означає, що відбувся перехід до наступного лінійного відрізку
		// вхідної ВАХ транзистора
		if (Ux > f[k + 1][0])
		{
			k++;
			x0 = f[k][0];
			x1 = f[k + 1][0];
		}
		// значення напруги зберігається в першому елементі масиву
		f_interpolated[i][0] = Ux;
		// значення струму розраховується за формулою лінійної інтерполяції і
		// зберігається в другому елементі масиву
		f_interpolated[i][1] = f[k][1] + ((f[k + 1][1] - f[k][1]) / (x1 - x0))*(Ux - x0);
		// збільшення значення напруги на 1 (на 0,01 В)
		Ux = Ux + 1;
		// збільшення значення змінної, що використовується
// для індексації масиву,
		// на 1
		i++;
	}

	// заповнення елементу керування білим кольором (створення білого фону)
	dc.FillRect(&rect, WHITE_BRUSH);

	// оголошення додаткових змінних для зберігання масштабних коефіцієнтів
	double scale_U = 100;
	double scale_I = 200;

	// малювання вертикальних ліній сітки, крок 50 пікселів
	for (int x = 0; x < W; x = x + 50)
	{
		dc.MoveTo(x, 0);
		dc.LineTo(x, H);
	}
```

```
	// малювання горизонтальних ліній сітки, крок 50 пікселів
	for (int y = H; y > 0; y = y - 50)
	{
		dc.MoveTo(0, y);
		dc.LineTo(W, y);
	}
	// малювання текстових надписів біля ліній сітки
	dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
	// значення струму (текстові надписи біля ліній сітки)
	for (int i = 0; i < 6; i++)
	{
		CString s;
		s.Format(_T("%.2f"), 50 * i / scale_I);
		dc.TextOutW(5, H - 15 - i * 50, s);
	}
	// значення напруги (текстові написи біля ліній сітки)
	for (int i = 1; i < 4; i++)
	{
		CString s;
		s.Format(_T("%.1f"), 50 * i / scale_U);
		dc.TextOutW(5 + i * 50, H - 15, s);
	}
	// виведення текстового надпису у верхньому куті ліворуч
	dc.TextOutW(5, 5, _T("Ib, mA"));
	// виведення текстового надпису у нижньому куті праворуч
	dc.TextOutW(W - 50, H - 15, _T("Ube, V"));

	// створення об’єктів пера різних кольорів для малювання ВАХ та ліній
	CPen magentaPen(PS_SOLID, 2, RGB(150, 0, 150));	// фіолетове перо
	CPen redPen(PS_SOLID, 2, RGB(150, 0, 0));		// червоне перо

	// сіре перо, товщина 1 піксель, пунктирна лінія
	CPen grayPen(PS_DASH, 1, RGB(128, 128, 128));

	// вибір в контекст пристрою об’єкту пера фіолетового кольору
	dc.SelectObject(&magentaPen);
	// малювання вхідної ВАХ транзистора
	for (int i = 1; i < 122; i++)
	{
	dc.MoveTo(f_interpolated[i][0] * scale_U, H - (f_interpolated[i][1] * scale_I));
	dc.LineTo(f_interpolated[i - 1][0] * scale_U, H - (f_interpolated[i-1][1] * scale_I));
	}
	// вибір в контекст пристрою об’єкту пера червоного кольору
	dc.SelectObject(&redPen);
	// малювання навантажувальної характеристики
	dc.MoveTo(0, H - scale_I * m_Eb / m_Rb);
	dc.LineTo(m_Eb*scale_U, H);
	// пошук точки перетину вхідної ВАХ транзистора
// та навантажувальної характеристики
	for (int i = 0; i < 122; i++)
	{
		// напруга змінюється з кроком 0,01 В
```

```
                Ux = (double)i / 100;
                // якщо точку перетину знайдено
                if (f_interpolated[i][1] > (m_Eb - Ux) / m_Rb)
                {
                        // копіювання знайденого значення струму бази в змінну
                        m_Ib = f_interpolated[i-1][1];
                        // вибір пера сірого кольору
                        dc.SelectObject(grayPen);
                        // малювання вертикальної пунктирної лінії,
// що вказує на точку перетину
                        dc.MoveTo(Ux * scale_U, H - m_Ib * scale_I);
                        dc.LineTo(Ux * scale_U, H);
                        // малювання горизонтальної пунктирної лінії,
// що показує точку перетину
                        dc.MoveTo(0, H - m_Ib * scale_I);
                        dc.LineTo(f_interpolated[i - 1][0] * scale_U, H - m_Ib * scale_I);
                        // зупинка та вихід з циклу
                        break;
                }
        }

        // виклик функції розрахунку вихідних параметрів
        // (струму колектора та напруги колектор-емітер)
        CalculateParameters();
        // виведення розрахованих значень в елементи керування на діалозі
        UpdateData(false);
}
```

В наведеному програмному коді відбувається виклик функції-члену (методу) класу діалога CalculateParameters(). Для того, щоб додати до класу діалога таку функцію у вікні Class View необхідно викликати контекстне меню для вузла класу діалога проекту (в даному прикладі клас діалога має назву CProject33Dlg) шляхом натискання на відповідному вузлі правої кнопки миші. З меню, що відкриється (рис. 4.64) потрібно вибрати команду Add Function.

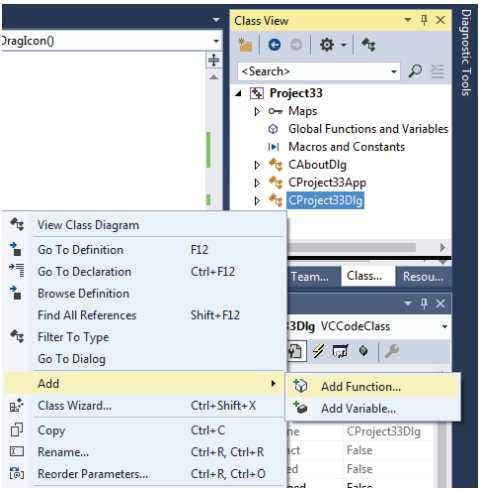


Рис. 4.64.

У вікні, що відкриється після цього (рис. 4.65), необхідно задати назву нової функції у полі Function name, вказати в полі Return type тип значення, що повертається функцією (в даному випадку void) і натиснут ина кнопку Finish.

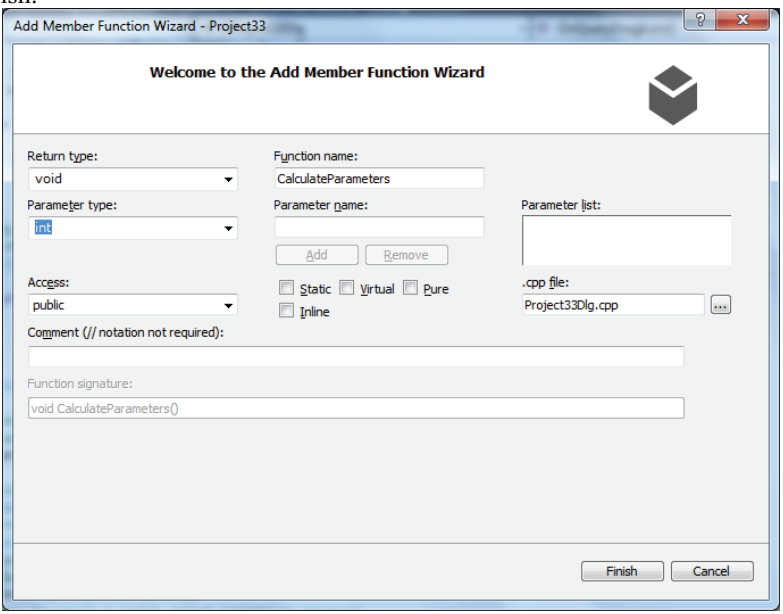


Рис. 4.65.

Реалізація функції представлена далі.

```
void CProject33Dlg::CalculateParameters()
{
	// створення об’єкту контексту пристрою
	CClientDC dc(&m_static2);
	// створення об’єкту для зберігання інформації про прямокутну область
	CRect rect;
	// зчитування інформації про розміри прямокутної області, на якій
	// будуть виводитися графіки (елемент Static Text)
	m_static2.GetClientRect(&rect);
	// для зручності в окремі змінні копіюються значення
	// ширини та висоти прямокутної області побудови графіків
	int W = rect.Width();
	int H = rect.Height();

	// заповнення елементу керування білим кольором (створення білого фону)
	dc.FillRect(&rect, WHITE_BRUSH);

	// оголошення додаткових змінних для зберігання масштабних коефіцієнтів
	double scale_U = 10;
	double scale_I = 5;

	// малювання вертикальних ліній сітки, крок 50 пікселів
	for (int x = 0; x < W; x = x + 50)
	{
		dc.MoveTo(x, 0);
		dc.LineTo(x, H);
	}
	// малювання горизонтальних ліній сітки, крок 50 пікселів
	for (int y = H; y > 0; y = y - 50)
	{
		dc.MoveTo(0, y);
		dc.LineTo(W, y);
	}
	// малювання текстових надписів біля ліній сітки
	dc.SetBkMode(TRANSPARENT); // фон текстових надписів прозорий
	// значення струму (текстові надписи біля ліній сітки)
	for (int i = 0; i < 6; i++)
	{
		CString s;
		s.Format(_T("%.0f"), 50 * i / scale_I);
		dc.TextOutW(5, H - 15 - i * 50, s);
	}
	// значення напруги (текстові надписи біля ліній сітки)
	for (int i = 1; i < 7; i++)
	{
		CString s;
		s.Format(_T("%.0f"), 50 * i / scale_U);
		dc.TextOutW(5 + i * 50, H - 15, s);
	}
	// виведення текстового надпису у верхньому куті ліворуч
```

```
dc.TextOutW(5, 5, _T("Ik, mA"));
// виведення текстового надпису у нижньому куті праворуч
dc.TextOutW(W - 50, H - 15, _T("Uke, V"));

// створення об’єктів пера різних кольорів для малювання ВАХ та ліній
CPen greenPen(PS_SOLID, 2, RGB(0, 200, 0));              // зелене перо
CPen magentaPen(PS_SOLID, 2, RGB(150, 0, 150));          // фіолетове перо
CPen bluePen(PS_SOLID, 2, RGB(0, 0, 200));               // синє перо
// сіре перо, товщина 1 піксель, пунктирна лінія
CPen grayPen(PS_DASH, 1, RGB(128, 128, 128));
// оголошення додаткових змінних для використання в алгоритмі
// максимальне значення струму колектора
const int  Ik_max = 50;
// додаткові коефіцієнти, що визначають нахил вихідних ВАХ транзистора
double c1 = 2000;
// та "ширину" області насичення
double c2 = 2;
// створення двовимірного масиву, кожний елемент якого містить
// значення струму колектора в залежності від напруги колектор-емітер
// та значення струму бази
double f[400][Ik_max];
for (int u = 0; u < 400; u++)
{
        for (int i = 0; i < Ik_max; i++)
        {
        // коефіцієнт c1 використовується для задання нахилу ВАХ транзистора
                double k;
                k = (double) 1.0 + (double)u / c1;
                f[u][i] = (double)k * i;
        }
}
// вибір в контекст пристрою пера зеленого кольору
dc.SelectObject(&greenPen);
// малювання лінії, що показує зону насичення транзистора
dc.MoveTo(0, H);
dc.LineTo(Ik_max / c2, H - 50 * scale_I);

// малювання сімейства вихідних ВАХ транзистора
for (int i = 3; i < Ik_max; i = i + 5)
{
        for (int u = 1; u < 400; u = u + 1)
        {
                // врахування області насичення при малюванні ВАХ
                if (u > i / c2)
                {
                        dc.MoveTo(u, H - f[u][i] * scale_I);
                        dc.LineTo(u - 1, H - f[u - 1][i] * scale_I);
                }
        }
}

// вибір в контекст пристрою пера фіолетового кольору
```

```
dc.SelectObject(&magentaPen);
// розрахунок струму колектора за відомою формулою
m_Ik = m_Ib * m_h21;
// в циклі відбувається малювання ВАХ, що відповідає розрахованому
// струму бази
for (int u = 1; u < 400; u = u + 1)
{
	if (u > f[u][(int)m_Ik] / c2)
	{
		dc.MoveTo(u, H - f[u][(int)m_Ik] * scale_I);
		dc.LineTo(u - 1, H - f[u - 1][(int)m_Ik] * scale_I);
	}
}

// оголошення масиву для зберігання розрахованих значень
// навантажувальної характеристики
double f1[400][2];

// розрахунок значень навантажувальної характеристики
for (int u = 0; u < m_Ek * 10; u++)
{
	// крок зміни напруги складає 0,1 В
	double Uk = (double)u / 10;
	f1[u][0] = Uk;
	// розрахунок значення струму
	f1[u][1] = (m_Ek - Uk) / m_Rk;
}

// вибір в контекст пристрою об’єкту пера синього кольору
dc.SelectObject(&bluePen);
// малювання навантажувальної характеристики
for (int u = 1; u < m_Ek * 10; u++)
{
	dc.MoveTo(f1[u][0] * scale_U, H - (f1[u][1] * scale_I));
	dc.LineTo(f1[u - 1][0] * scale_U, H - (f1[u - 1][1] * scale_I));
}
// оголошення додаткової змінної
double Ux;
// в циклі відбувається пошук точки перетину вихідної ВАХ транзистора
// та навантажувальної характеристики
for (int u = 0; u < 400; u++)
{
	// крок напруги складає 0,1 В
	Ux = (double)u / 10;
	// якщо точку перетину знайдено, ця умова виконується
	if (f[u][(int)m_Ik] >f1[u][1])
	{
		// запис в змінну знайденого струму колектора
		m_Ik = f[u][(int)m_Ik];
		// вибір в контекст пристрою пера сірого кольору
		dc.SelectObject(grayPen);
	// малювання вертикальної пунктирної лінії, що показує точку перетину
```

```
                        dc.MoveTo(Ux * scale_U, H - m_Ik * scale_I);
                        dc.LineTo(Ux * scale_U, H);
                        // малювання горизонтальної пунктирної лінії,
// що показує точку перетину
                        dc.MoveTo(0, H - m_Ik * scale_I);
                        dc.LineTo(Ux * scale_U, H - m_Ik * scale_I);
                        // запис в змінну значення напруги колектор-емітер
                        m_Uke = Ux;
                        // зупинка виконання циклу та вихід з нього
                        break;
                }
        }
}
```

Після компіляції проекту та запуску програми можна перевірити її роботу. Коефіцієнти, що використовуються в програмі, і які задають нахил ліній ВАХ, в даному випадку мають такі значення: C1=2000, c2=2. На рис. 4.66 представлено результати розрахунків.

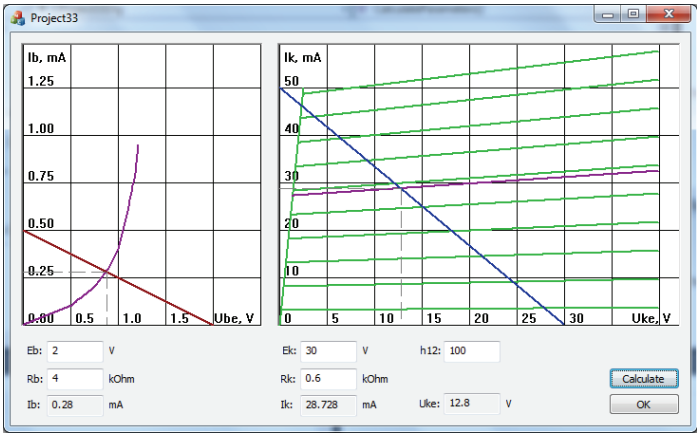


Рис. 4.66.

Фіолетовим кольором відображаються ВАХ транзистора. Червоний колір використано для зображення навантажувальної характеристики вхідного кола, а синім – навантажувальної характеристики вихідного кола. Сімейств вихідних ВАХ зображено зеленим кольором. Пунктирні лінії показують точки перетину навантажувальних характеристик з ВАХ транзистора. Координати точок перетину визначають струм бази, струм колектора та напругу колектор-емітер.

На рис. 4.67 також продемонстровано результат роботи програми. Однак в даному випадку коефіцієнти, що визначають форму вихідних ВАХ транзистора, мають такі значення: C1=5000, c2=4. Ці значення було підіб-

рано експериментально таким чином, щоб сімейство вихідних ВАХ транзистора співпадало з характеристиками реального транзистора, які наведено в довідниках.

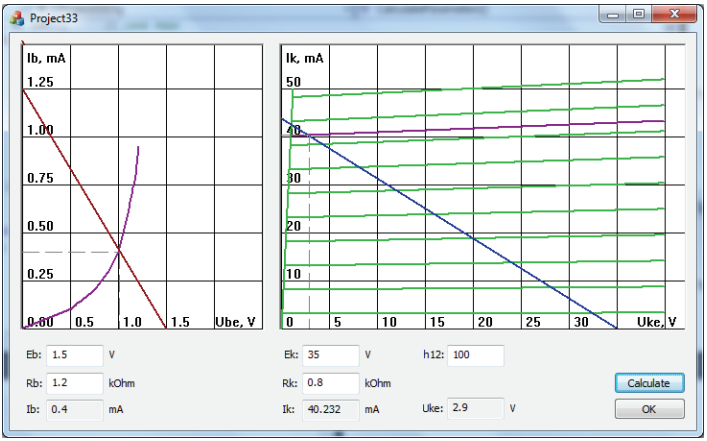


Рис. 4.67.

Практичне значення розглянутих в цьому розділі прикладів полягає в тому, що запропонований підхід до моделювання та аналізу електронних кіл з використанням інформаційних технологій, з одного боку, дозволяє більш глибоко розуміти процеси перетворення сигналів, що відбуваються в електронних системах, а з іншого – сприяє системному формуванню професійної підготовки фахівців, при якій вони будуть мати у своєму розпорядженні систему знань та вмінь в актуальних для них областях електроніки та інформаційних технологій, які в сучасному світі нерозривно пов'язані. Крім того, розроблене програмне забезпечення представляє собою інструментальні засоби, які можуть бути корисні інженеру з електроніки для виконання розрахунків, моделювання та аналізу на етапі проектування і виробництва електронних та електротехнічних систем різного призначення.

РОЗДІЛ 5. ІМІТАЦІЙНЕ ТА КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ ПРОЦЕСІВ В СУДНОВИХ ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМАХ

5.1. Моделювання процесів комутації навантаження в СЕЕС

Однією з причин появи провалів та сплесків напруги є зміна навантаження суднової електростанції, особливо при пуску потужних асинхронних двигунів. Принциповим аспектом роботи суднової електроенергетичної системи є випадковий характер вхідного потоку заявок на комутацію навантаження. В даному випадку під заявкою розуміється сигнал на вмикання або вмикання споживача електроенергії, наприклад асинхронного двигуна.

Отже, адекватною моделлю такого процесу буде система масового обслуговування, яка характеризується стохастичним процесом $x(t)$ з дискретною множиною значень, що утворюють безперервний ланцюг Маркова:

$$\Pr[X(t) = j \mid X(\tau)\, for (\tau_1 \le \tau \le \tau_2 \le t)] = \Pr[X(t) = j \mid X(\tau_2)].$$

Оскільки в даному випадку розглядаються події вмикання та вимикання споживачів електроенергії, можна припустити, що їх тривалість роботи підпорядковується показниковому закону розподілення. Таке розподілення тривалості заявок на комутацію навантаження має марківську властивість – відсутність післядії (пам'яті). Відповідно, вірогідність завершення обслуговування виклику на малому інтервалі часу має вигяд:

$$P(t, t + \Delta t) = \mu \Delta t + f(\Delta t),$$

і не залежить від положення цього інтервалу на вісі часу.

Оскільки всі процеси в системі, що досліджуються, мають марківські властивості, то всі потоки заяв, що переводять систему з одного стану в інший, є пуасонівськими (в даному випадку під різними станами розуміється режим роботи електросистеми з різною потужністю, що генерується силовими агрегатами, і, відповідно, споживається навантаженням.

Для спрощення аналізу системи будемо розглядати весь потік заявок на комутацію навантаження як сигнали, що поступають на деякий комутатор в судновій електроенергетичній системі (СЕЕС) (рис. 5.1).

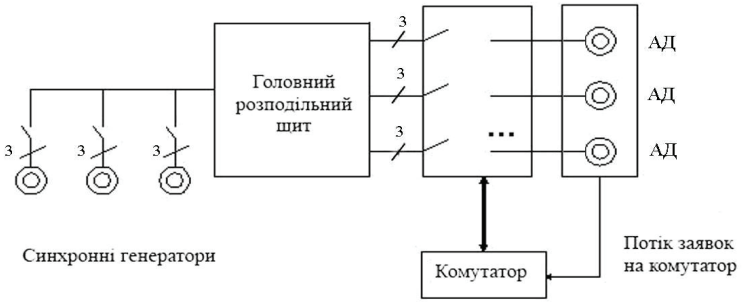


Рис. 5.1.

Загальний випадок опису комутатора як системи масового обслуговування, що має марківські властивості, має важливе практичне значення – випадок чистої системи масового обслуговування з очікуванням та пуасонівським навантаженням. Така система має буфер, що дозволяє зберігати чергу нескінченної довжини, стан якого можна ототожнити з кількістю запитів, шо є в системі в кожний момент часу.

Добовий графік навантаження АЕЕС розбивається на часові інтервали Δt тривалістю 10 хвилин. Час обробки заявки диспетчером (тривалість перехідного процесу після комутації навантаження) складає кілька одиниць секунд, і значно менший, ніж тривалість роботи споживача.

В рамках теорії булевих функцій кожний елемент системи може мати лише два стани – ввімкнений чи вимкнений (включений або виключений). Відзначимо, що булева функція – це функція, аргументи якої, як і сама функція, приймають значення з множини, яка складається з двох елементів [0, 1].

При побудові булевих моделей приймається припущення, згідно якому миттєві стани елементів (в даному випадку стан навантаження – включене чи виключене) однозначно задають стан системи в той самий момент часу. Кожному *i*-му елементу присвоюється булева змінна:

$$x_{_{И}} = \begin{cases} 1, якщо\ i-й\ елемент\ \ включений, \\ 0, якщо\ i-й\ елемент\ виключений. \end{cases}$$

Функціонування всієї електроенергетичної системи інтерпретується як випадковий процес, який в кожний момент часу для кожного з навантаження приймає значення 1 або 0. Поведінка *i*-го елементу (i = 1, 2, ..., N) описується випадковим процесом $y_i(t)$.

5.2. Використання ланцюгів Маркова для моделювання та аналізу режимів роботи СЕЕС

Для кількісного аналізу системи з використанням графу, використаний марківський процес [35; 36].

Марківським процесом називається випадковий процес із скінченною кількістю станів, який розвивається в дискретному часі та має марківські властивості: поведінка процесу, починаючи з будь-якого моменту часу, визначається лише поточним станом і не залежить від всієї передісторії процесу переходу з одного стану в інші. Марківський ланцюг з n різними станами характеризується матрицею значень так званих перехідних вірогідностей:

$$P = \begin{vmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{vmatrix}.$$

Елементи матриці переходів p_{ij} відповідають вірогідності того, що марківський ланцюг, який знаходиться в стані i, на наступному кроці перейде в стан j. Вірогідність p_{ij} представляє вірогідність того, що на наступному кроці процесу стан i збережеться.

Перехід з одного стану системи в інший описується випадковою величиною, що має своє розподілення $F_k(t)$ для кожного k-го стану. Для марківського процесу всі розподілення $F_k(t)$ є експоненційними.

Марківський ланцюг зручно описувати орієнтованим графом, «ваги» ребер якого відповідають інтенсивності переходу з одного стану в інший. Знаючи перехідні вірогідності p_{ij} та параметр λ_i розподілення $K_i(t)$ часу перебування процесу в i-м стані, можна знайти «ваги» ребер по формулі:

$$\lambda_{ij} = p_{ij}\,\lambda_i\,.$$

Метод графу станів, з енергетичної точки зору, є найбільш універсальним методом побудови структурних схем для дослідження діаграми навантаження СЕЕС. Суттєвою перевагою цього методу є можливість моделювання процесів комутації навантаження та визначення статистичних характеристик таких процесів з метою оперативного короткострокового планування навантаження СЕЕС та резервування потужності.

Розглянемо систему, що складається з одного генератора та одного споживача електроенергії – асинхронного двигуна (АД). Цей приклад дозволяє розглянути методологічні підходи до аналізу функціонування енергосистеми та побудови в подальшому діаграми навантаження. Отже, асинхроний двигун може знаходитися в двох станах: S_1 – виключений, S_2 – включений. На рис. 5.2 представлений граф станів енергосистеми.

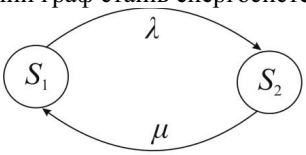


Рис. 5.2. Граф станів енергосистеми

Перехід зі стану S_1 у стан S_2 відбувається за принципом однорідного марківського процесу з інтенсивністю λ. Перехід із стану 2 в стан 1 відбувається з інтенсивністю μ .

Ймовірність того, що в момент часу $\tau + \Delta\tau$ навантаження знаходиться в стані S_1 (виключене), оцінюється наступним алгоритмом. В стані S_1 енергоблок по закінченню часу $\tau + \Delta\tau$ може знаходитись по двох причинах:

– в момент часу τ він знаходився в стані S_1 і не перейшов до стану S_2 за проміжок $\Delta\tau$;

– в момент часу τ енергоблок знаходився у стані S_2 і за період часу $\Delta\tau$ перейшов у стан S_1 .

Вірогідність знаходження енергоблоку в стані S_1 визначається добутком $P_1(\tau)$ на вірогідність $(1–\lambda\cdot\Delta\tau)$ того, що енергоблок не перейшов в стан S_2 за проміжок часу $\Delta\tau$. Аналогічно визначається вірогідність знаходження

енергоблоку в стані S_2: $P_2(\tau)\mu\Delta T$. Використовуючи правило додавання вірогідностей знайдемо:

$$P_1(\tau+\Delta\tau)=P_1(\tau)(1-\lambda\Delta\tau)+P_2(\tau)\mu\Delta\tau.$$

Перетворимо це рівняння до вигляду:

$$\frac{P_1(\tau+\Delta\tau)-P_1(\tau)}{\Delta\tau}=-\lambda P_1(\tau)+\mu P_2(\tau).$$

Подібні рівняння можна скласти для кожного стану графа. При цьому граф станів буде описуватися системою диференційних рівнянь і їх кількість дорівнює кількості станів на графі. Отже, для графа, що представлений на рис. 5.2 можна записати:

$$\begin{cases}\dfrac{dP_1}{d\tau}=-\lambda P_1+\mu P_2,\\ P_1+P_2=1.\end{cases}$$

Граф станів формується на основі послідовних процедури перетворення функціональної схеми енергоблоку в структурну схему, на базі якої базується граф станів.

Розглянемо енергосистему, що складається з двох генераторів та визначеної кількості споживачів. При наявності достатньої кількості резерву потужності (в цьому режимі включені не всі споживачі), на навантаження працює лише один генератор, а другий знаходиться в резерві. При нестачі потужності, що генерується, другий генератор вводиться на паралельну роботу з першим генератором і бере на себе частину навантаження. Спрощена схема та граф станів такого енергоблоку представлений на рис. 5.3.

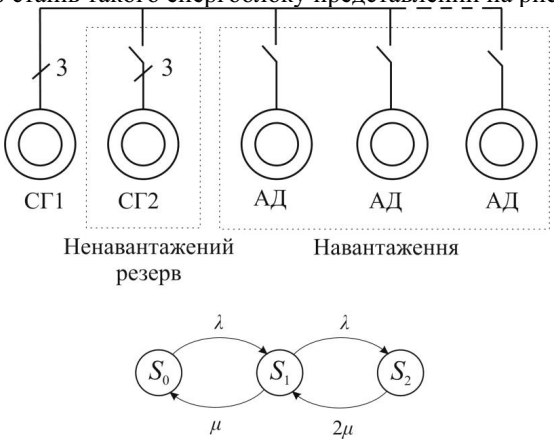


Рис. 5.3.

Така схема роботи може використовуватись у тих випадках, коли тривалість вмикання в роботу додаткового генератора невелика, і його можна тримати у виключеному стані. Для стаціонарних потоків заявок на комутацію навантаження можна записати:

$$\begin{cases} -\lambda P_0 + \mu P_1 = 0, \\ -(\lambda + \mu) P_1 + \lambda P_0 + 2\mu P_0 = 0, \\ P_0 + P_1 + P_2 = 1. \end{cases}$$

Рішенням системи рівнянь є:

$$P_0 = \frac{2}{\left(1 + \frac{\lambda}{\mu}\right)^2 + 1}; \; P_1 = \frac{\lambda P_0}{\mu}; \; P_2 = P_0 + P_1.$$

Для режиму, коли обидва генератори працюють, вірогідність станів визначається виразом:

$$P_0 = \frac{1}{1 + \left(\frac{\lambda}{\mu}\right)^2}; \; P_1 = \frac{2\lambda P_0}{\mu}.$$

Для енергосистеми, що складається з трьох генераторів, два з яких знаходяться у роботі, а один – в резерві (ненавантажений), вірогідності станів можна визначити наступним чином:

$$P_0 = \frac{3}{2\left(1 + \frac{\lambda}{\mu}\right)^3 + 1}; \; P_1 = \frac{2\lambda P_0}{\mu}; \; P_2 = \frac{\lambda P_1}{\mu}.$$

Для побудови моделі енергосистеми, що складається з трьох синхронних генераторів, використовується апарат марковських процесів. Енергосистема в будь-який момент часу може знаходитися в одному з 5 станів: S_0 – вимкнені всі генератори, S_1 – працює перший генератор, другий – в гарячому резерві; S_2 – працює перший та другий генератори, третій знаходиться в резерві; S_3 – всі три генератори працюють. На рис. 5.4 представлена часова діаграма, яка показує стан кожного з генераторів протягом доби. Як зазначалося раніше, генератор, як елемент системи, може бути описаний булевою функцією, яка має два значення – 0 (виключений) та 1 (включений). Потужність кожного з генераторів складає 300 кВт.

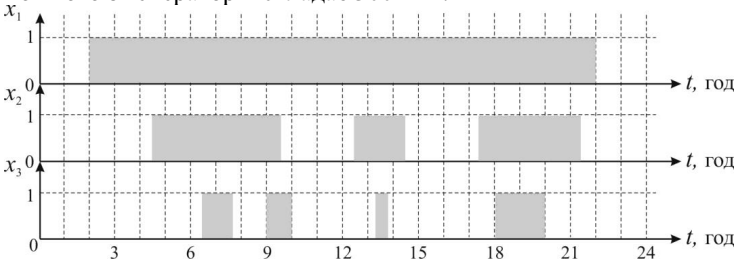


Рис. 5.4.

Для електроенергетичної системи, яка розглядається, виконаємо імітаційне моделювання, яке полягає в логіко-аналітичній імітації функціонування системи.

Задачею імітаційного моделювання в даному випадку є побудова алгоритму поведінки підсистем та окремих елементів системи генераторів у часі. Цей алгоритм може бути реалізований у вигляді програми.

Потоком випадкових подій у даному випадку є заявки на підключення (відключення) генератора на паралельну роботу з іншим генератором. Статистична обробка цієї інформації дозволяє отримати статистичні оцінки показників ефективності. Спеціальним чином генерується вхідний потік заявок, причому момент появи кожної заявки є випадковою подією. Блок-схема алгоритму програми імітації роботи системи представлена на рис. 5.5.

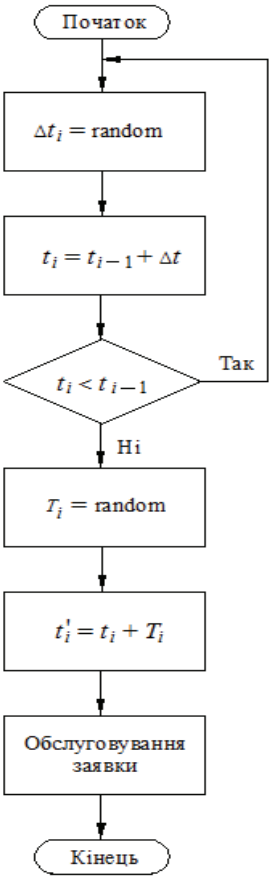


Рис. 5.5.

На рисунку позначено: t_i – момент появи заявки; t'_i – момент завершення обслуговування; T_i – тривалість обслуговування:

$$T_i = t_i - t'_i.$$

СЕЕС розглядається на двох рівнях деталізації. Зміна кількості генераторів, що працюють паралельно на загальне навантаження розглядається як процес більш високого рівня, а комутація навантаження (кожного з двигунів) – як процес більш низького рівня. Навантаження (асинхронний двигун) може знаходитись у двох станах – включений або виключений. Перехідні процеси, що супроводжуються пуском асинхронних двигунів на даних рівнях деталізації не розглядаються. На рис. 5.6 показані різні рівні деталізації. Використані наступні позначення: S_0 – жоден з генераторів не працює, S_1 – працює один генратор, S_2 – працює 2 генератори, S_3 – працює 3 генератори, S_{1j} – робота визначеної групи асинхронних двигунів, ($1 \le j \le n$), де n – кількість двигунів в енергосистемі, S_{13i} – стани ($1 \le i \le 5$), в які переходить асинхронний двигун при запуску або зупинці.

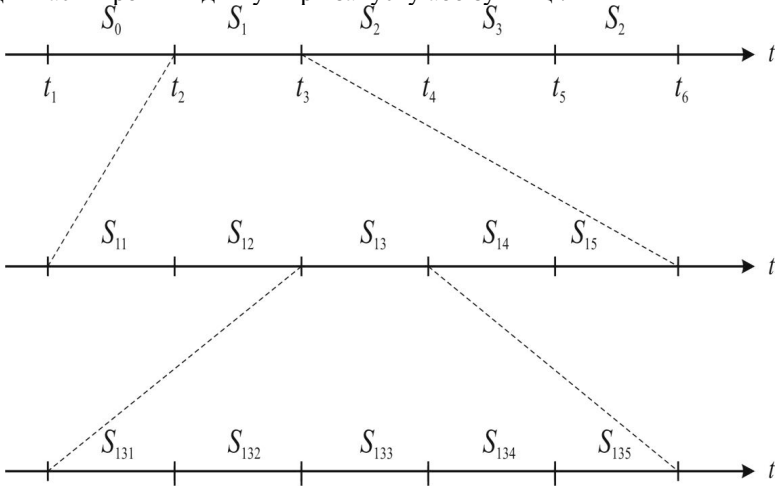


Рис. 5.6.

Тривалість знаходження системи у визначеному стані на порядок перевищує час, протягом якого система перейшла з одного стану в інший. Тому будемо вважати переходи системи з одного стану в інший такими, що відбуваються миттєво.

Суднова електроенергетична система в кожний момент часу може знаходитися одному з декількох санів, $S_0, S_1, ..., S_n$, що визначаються кількістю паралельно працюючих генераторів. Основна властивість ланцюга Маркова полягає в тому, що стан, в якому система опиниться в наступний момент часу, залежить тільки від поточного стану і не залежить від попередніх [35].

Кожному моменту часу відповідає вектор станів $[S(t)]$, елементами якого є вірогідності знаходження системи в кожному із можливих станів.

$$\left[S(t_k)\right]=\left[S_1(t_k),S_2(t_k),...,S_n(t_n)\right],$$

$$\sum_{i=1}^{n}S_i(t_k)=1,$$

де n – кількість генераторів в СЕЕС.

Виконаємо імітаційне моделювання поведінки дискретного ланцюга Маркова. Початковий стан системи заданий. Строка матриці вірогідностей переходів (рис. 5.7) для цього стану встановлює ймовірності переходів з поточного стану в будь який інший. Новий стан отримаємо як дискретне випадкове значення.

$$|p|=\begin{array}{c} \begin{array}{cccc} S_0 & S_1 & S_2 & S_3 \end{array} \\ \left|\begin{array}{cccc} 0,1 & 0,9 & 0 & 0 \\ 0,05 & 0,7 & 0,25 & 0 \\ 0 & 0,6 & 0,3 & 0,1 \\ 0 & 0 & 0,8 & 0,2 \end{array}\right| \end{array}$$

Рис. 5.7.

Розглянемо строку матриці переходів, якій відповідає стан СЕЕС з одним працюючим генератором. Ймовірність того, що система залишиться в цьому ж стані дорівнює 0,7. Вірогідність підключення другого генератора на паралельну роботу дорівнює 0,25. Ймовірність зупинки генератора дорівнює 0,05. Ряд ймовірностей представлений на рис. 5.7. Генерується випадкове значення ξ. Припустимо, що воно дорівнює 0,473. Це означає, що система із стану S_0 перейде в стан S_1, тобто був включений один генератор. Генерується наступне випадкове значення, наприклад 0,831. Це означає, що система перейде в стан S_2, тобто буде включений другий генератор на паралельну роботу з першим. Нове випадкове значення дорівнює 0,936, і система залишається в стані S_2 (з двома паралельно працюючими генераторами). Цей алгоритм продовжується доти, доки не буде отримані стани, в яких може знаходитися СЕЕС протягом доби. Необхідно виконати 6·24=144 ітерацій за умови, що крок дискретного часу складає 10 хвилин. На рис. 5.4 представлений результат імітації режимів роботи СЕЕС.

В кожному із режимів роботи енергосистеми сумарна потужність, що споживається навантаженням, може знаходитися у визначених діапазонах. Рис. 5.8 демонструє вищезазначене.

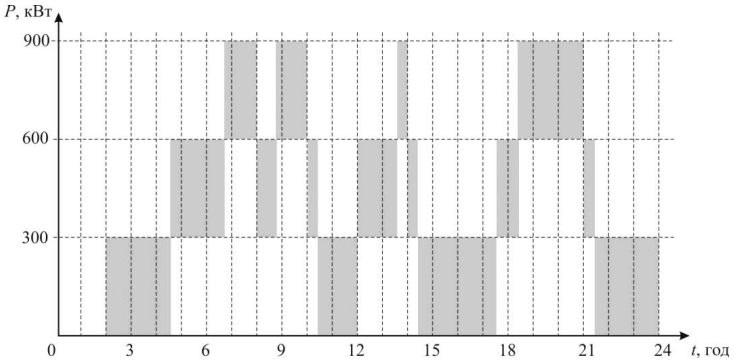


Рис. 5.8.

В кожній із зазначених областей значення споживаної потужності визначається кількістю навантаження, що підключене до шини ГРЩ, і є випадковою величиною y, яка є функцією $y = f(\alpha, \beta, \ldots, \omega)$ випадкових величин $\alpha, \beta, \ldots, \omega$, з відомими функціями розподілення, які визначаються шляхом аналізу діаграми навантаження в попередні моменти часу.

Коефіцієнт заповнення графіка навантаження показує, у скільки разів вироблена (спожита) кількість електроенергії за аналізований період (добу) менше тієї кількості енергії, яка було б вироблена (спожита) за той же час, якби навантаження установки весь час було максимальним. Очевидно, що чим рівномірніший графік, тим ближче значення ККД до максимально можливого. Для характеристики графіка навантаження установки можна скористатися також умовною тривалістю використання максимального навантаження:

$$T_{\max} = \frac{W_n}{P_{\max}} = \frac{P_{\text{сер}} T}{P_{\max}} = K_{\text{зап}} T.$$

Ця величина показує, скільки годин за аналізований період T (доба) установка повинна була б працювати з незмінним максимальним навантаженням, щоб виробити дійсну кількість електроенергії W_n за цей період часу.

Ступінь нерівномірності графіка роботи установки оцінюється коефіцієнтом заповнення:

$$k_{\text{зап}} = \frac{W_n}{P_{\max} T} = \frac{P_{\text{сер}}}{P_{\max}}.$$

Таким чином, процес імітації роботи енергосистеми і зміни її станів виконується з використанням керуючих послідовностей, що визначаються по функціям розподілення вірогідностей вихідних випробувань. Загальний алгоритм проведення моделювання представлений на рис. 5.9.

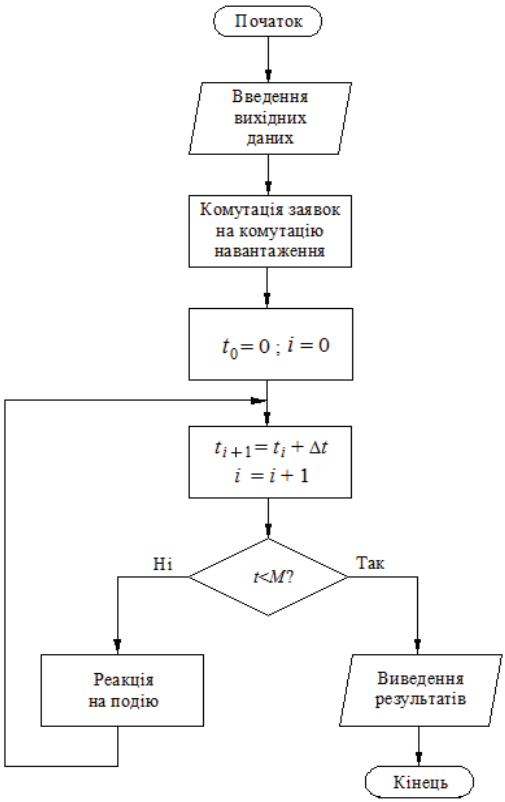


Рис. 5.9.

Модель системи представляє собою сукупність моделей елементів, і для задачі, що розглядається, під моделлю елемента системи розуміється набір правил поведінки елемента по відношенню до вхідних викликів та зміна станів елемента. Метою імітаційного моделювання є:

– відтворення поведінки навантаження в процесі роботи електроенергетичної системи;

– накопичення статистичних даних про роботу кожного асинхронного двигуна;

– подальша статистична обробка цих даних для отримання статистичних оцінок кількісних характеристик.

На практиці, при роботі реальної системи, існують випадкові фактори, які призводять до зміни станів системи. Загалом, ці фактори визначаються як технологічним процесом, електроживлення якого забезпечує СЕЕС, часом доби, структурою електроенергетичної системи, і навіть погодою (при

великій температурі працюють кондиціонери та вентилятори, при низькій температурі – обігрівачі, потужність яких співрозмірна з потужністю генераторів). Отже, виникає задача моделювання випадкових величин із різними законами розподілення.

В якості базових використовуються випадкові величини з рівномірним законом розподілення, які з однаковою вірогідністю можуть приймати будь-які значення в даному інтервалі [35].

Генератор випадкових величин видає незалежні, рівномірно розподілені в заданому діапазоні випадкові величини. Оскільки розглядається задача короткострокового прогнозування навантаження СЕЕС (протягом доби), то кількість дискретних інтервалів часу протягом доби складає $24 \oplus 6 = 144$.

Шаг дискретного часу складає 10 хвилин, отже генерується масив випадкових чисел, що розподілені в інтервалі часу від 0 до 24 годин, тобто число з діапазону [0..144]. Це значення визначає момент часу, коли буде відбуватися комутація навантаження. Крім того, генерується друге випадкове значення, яке визначає тривалість роботи навантаження. Якщо цю процедуру повторити декілька разів, то для кожного окремого навантаження отримуємо добовий графік роботи.

За принципом суперпозиції сумарне навантаження можна розрахувати шляхом додавання потужності, що споживається повним навантаженням. Це і буде вихідними даними для побудови добового графіку навантаження СЕЕС.

На рис. 5.10 представлений інтерфейс програми, яка дозволяє автоматизувати процес отримання діаграми роботи кожного із споживачів електроенергії.

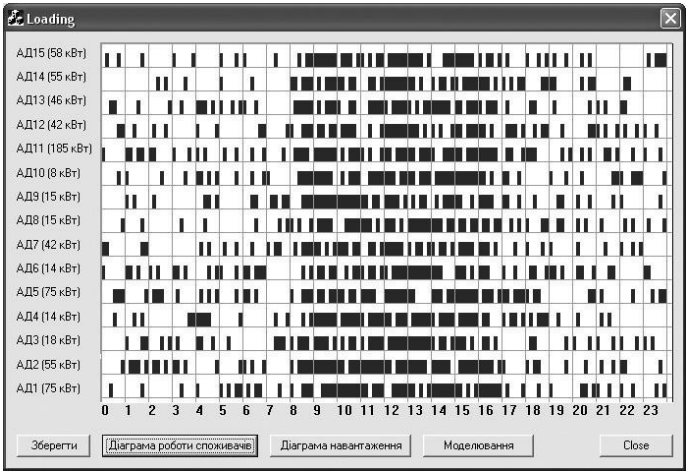


Рис. 5.10.

Для генерації базових випадкових величин використовуються різні способи: апаратний, табличний, алгоритмічний [35]. В даній роботі генерування випадкових величин відбувається з використанням табличного способу, оскільки такий підхід дозволяє гнучко змінювати закон розподілення та вводити обмеження на одночасну роботу окремих споживачів. Побудована таким чином модель в найбільшій мірі відповідатиме реальній системі.

На рис. 5.11 представлений приклад випадкових чисел з рівномірним розподіленням, що представляють собою заявки на вимикання або вмикання асинхронного двигуна.

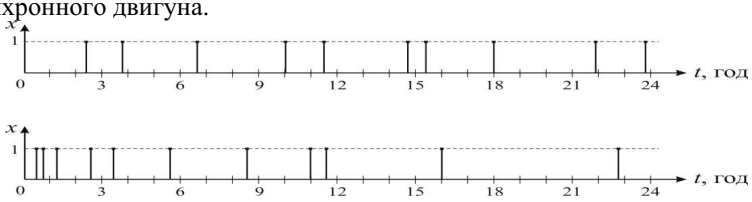


Рис. 5.11.

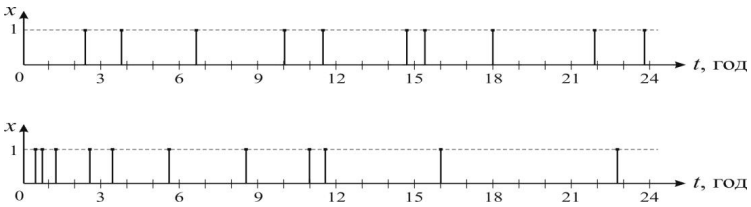


Рис. 5.12.

Для експоненціального розподілення функція розподілення відповідностей має вигляд:

$$F(k) = 1 - e^{-\lambda k}.$$

На рис. 5.12 представлений приклад випадкових величин з експоненціальним розподіленням. Отже, для моделювання всієї енергосистеми для кожного з навантажень необхідно задати характер розподілення, що визначається безпосередньо функціями, які виконує окремий споживач електроенергії (насос, вентилятор, освітлення, електропривод та ін.).

В судновій електроенергетичній системі можна виділити групи споживачів, які можуть бути описані функціями з рівномірним, експоненціальним, нормальним або іншим видом розподілення. Всі ці фактори потрібно враховувати при побудові діаграми навантаження. Для нормального розподілення функція розподілення вірогідностей має вигляд:

$$F(k) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{t} e^{\frac{-(k-m)^2}{2\sigma^2}},$$

де σ – середньоквадратичне відхилення, σ^2 – дисперсія, m – математичне очікування.

Процедура отримання чисел, розподілених по одному із зазначених законів полягає в наступному. Проводиться визначена кількість випробувань. В процесі кожного випробування значення кожної випадкової величини ξ порівнюється із p. Якщо $\xi \le p$, то до лічильника подій додається 1. Після завершення n випробувань, значення лічильника можна вважати випадковою величиною, розподіленою по закону Пуасона. Для отримання потоку Ерланга достатньо виконати «просіювання» простішого потоку [15].

Задачу імітаційного моделювання роботи СЕЕС з метою побудови діаграми навантаження розглянемо на прикладі секції головного розподільного щита електроенергетичної системи риболовного траулера «Знахідка», споживачі якої відображені у табл. 5.1.

Таблиця 5.1.

N	Найменування електроприймача	Тип двигуна	$P_н$ (кВт)	$I_н$ (А)	$U_н$ (В)	cos∏ $_н$	$R_н$ (Ом)
1	2	3	4	5	6	7	8
1	Насос рульової машини	МАФ83-94	75	147,6	380	0,86	0,02
2	Масляний насос	4А180М40	55	100,5	380	0,90	0,07
3	Компресор пуск. повітря	4А180М6	18,5	37	380	0,87	0,34
4	Вентиляція МКО	МАФ83-64	14,5	27,9	380	0,9	0,28
5	Електропривід ЯШП	4А250М40	75	136,5	380	0,9	0,04
6	Насос технічних потреб	АМ72-6	14	31,7	380	0,78	0,12
7	Загальносуднова вентиляція	АН91-4	42	92,8	380	0,79	0,11
8	Сушилка	4А180М8	15	55,6	380	0,82	0,43
9	Мельнична установка	4Р180М8	15	55,6	380	0,82	0,43
10	Сепаратор відокремлення жиру	АМ70-6	8	20,5	380	0,73	0,62
11	Гідронасос промисл. либідки	АН112	185	368	380	0,85	0,02
12	Пожежний насос	АН82-2	42	85,6	380	0,83	0,08
14	Виробнича вентиляція	МАФ83-85	46	91,6	380	0,90	0,05
14	Масляний насос	4А180М40	55	100,5	380	0,90	0,07
15	Насос охолодження забортної води	МАФ83-94	58	101,0	380	0,88	0,04

Кожен споживач характеризується своєю діаграмою роботи, яка є незалежною від режимів роботи інших споживачів. Розглянуті споживачі характеризуються індивідуальними режимами роботи та процес їх комутація носить випадковий характер. На рис. 5.13 представлені діаграми роботи кожного із споживачів електроенергії, а на рис. 5.14 потік заявок на комутацію навантаження.

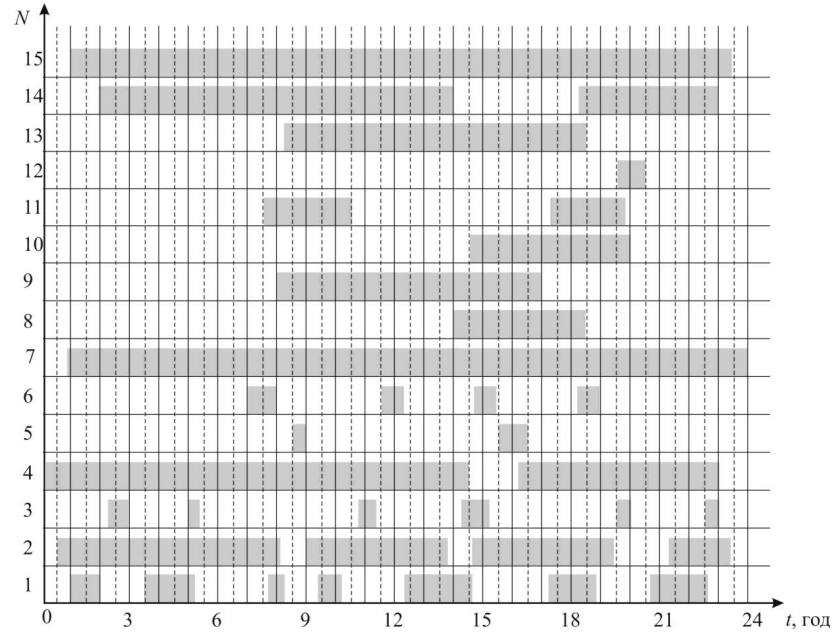


Рис. 5.13.

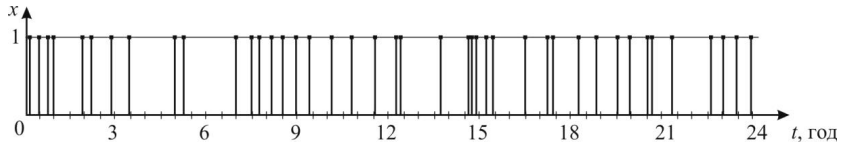


Рис. 5.14.

Від одночасної роботи великої кількості приймачів електроенергії в експлуатаційних режимах з різним характером навантаження залежить сумарне навантаження СЕЕС. Визначення сумарних різкозмінних навантажень слід проводити на підставі індивідуальних графіків навантажень окремих споживачів з урахуванням їх розбіжності. У добовому графіку навантаження СЕЕС є піки, обумовлені включенням і відключенням навантажень в експлуатаційних режимах роботи електростанції, виявлення яких необхідне для запобігання несприятливих умов роботи електроприймачів.

Блок-схема алгоритму, який дозволяє отримати одне з вищезазначених розподілень, представлена на рис. 5.15.

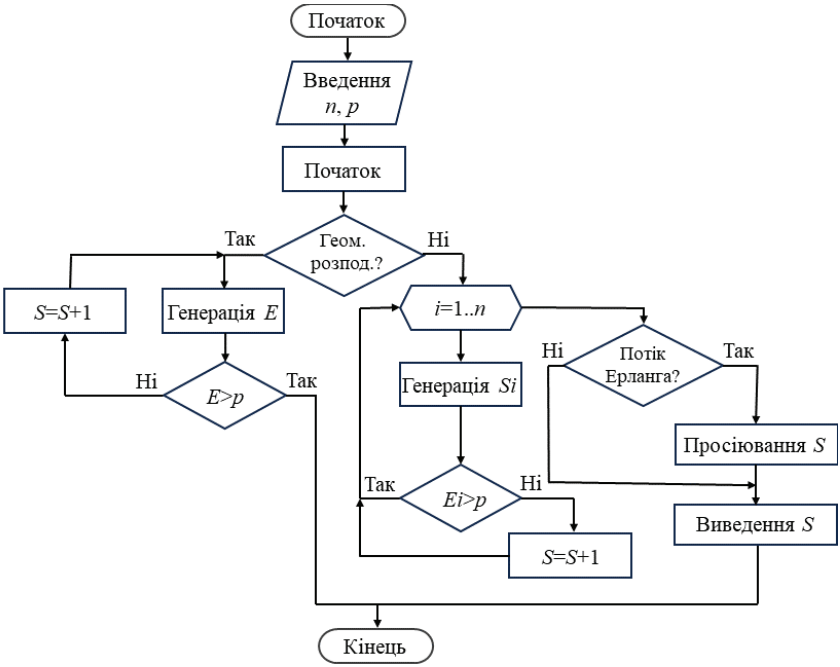


Рис. 5.15.

В розглянутому випадку зміна стану системи відбувається шляхом включення або відключення лише одного генератора, тобто, наприклад, якщо працює два генератори, то система може перейти або в стан з одним працюючим генератором, або в стан з трьома паралельно працюючими генераторами, однак аналіз програми навантаження показує, що при працюючому одному генераторі, при запуску споживачів можлива ситуація, коли сумарна потужність споживачів, що підключається, буде більшою за сумарну потужність двох генераторів. Тому система зі стану з одним працюючим генератором в K-момент може перейти в стан з трьома працюючими генераторами в $(K + 1)$-момент і навпаки, аналогічно ситуация можлива і для стану, коли жоден з генераторів не працює, однак для забезпечення споживачів електроенергією необхідна робота відразу двох генераторів (перехід з Q_0 в Q_2).

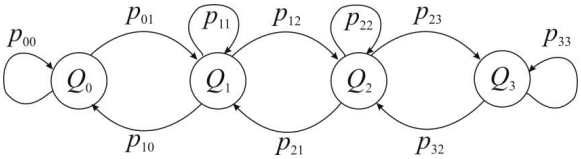


Рис. 5.16.

На рис. 5.16 позначено: Q_0 – жоден з генераторів не працює, сумарна потужність споживачів дорівнює 0; Q_1 – працює один генератор, сумарна потужність споживачів не більше 250 кВт; Q_2 – працює два генератори, сумарна потужність споживачів не більше 500 кВт; Q_3 – працює три генератори, сумарна потужність споживачів не більше 750 кВт; $\{Q_0, Q_1, Q_2, Q_3\}$ – можливі стани системи.

В загальному вигляді матриця перехідних вірогідностей має вигляд:

$$\left| P_{ij} \right| = \begin{vmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{21} & P_{31} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{vmatrix},$$

$$\sum_{j=0}^{n} p_{ij} = 1, \; i = \overline{1,n}.$$

Крім того, ланцюг Маркова характеризується вектором вірогідностей станів:

$$S(t) = \{S_0(t), S_1(t), S_2(t), S_3(t)\},$$

який показує вірогідність $S_i(t)$ знаходження системи в стані Q_i :

$$\sum_{i=1}^{n} S_i(t) = 1.$$

Для суднової електроенергетичної системи, що складається з трьох генераторів, матриця перехідних вірогідностей має вигляд:

$$\left[P_{ij} \right] = \begin{vmatrix} 0,05 & 0,95 & 0 & 0 \\ 0,1 & 0,7 & 0,2 & 0 \\ 0 & 0,6 & 0,3 & 0,1 \\ 0 & 0 & 0,9 & 0,1 \end{vmatrix}.$$

Рівняння для визначення стаціонарних ймовірностей має вигляд:

$$\begin{cases} S_0 = p_{00}S_0 + p_{10}S_1; \\ S_1 = p_{01}S_0 + p_{11}S_1 + p_{21}S_2; \\ S_2 = p_{12}S_1 + p_{22}S_2 + p_{32}S_3; \\ S_3 = p_{23}S_2 + p_{33}S_3. \end{cases}$$

Для установленного режиму складемо систему рівнянь:

$$\begin{cases} S_0 = 0,05S_0 + 0,1S_1; \\ S_1 = 0,95S_0 + 0,7S_1 + 0,6S_2; \\ S_2 = 0,2S_1 + 0,3S_2 + 0,9S_3; \\ S_3 = 0,1S_2 + 0,1S_3. \end{cases}$$

Рішення цієї системи рівнянь дозволяє визначити вірогідності знаходження СЕЕС в кожному зі станів. На рис. 5.17 представлений граф станів системи для випадку, коли одночасно (у випадку дефіциту потужності для живлення споживачів) можуть бути включені в работу не один, а два генератори.

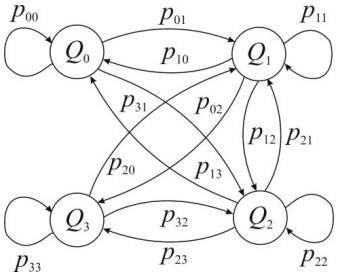


Рис. 5.17.

Для такого випадку система рівнянь стаціонарних вірогідностей матиме вигляд:

$$\begin{cases} S_0 = p_{00}S_0 + p_{10}S_1 + p_{20}S_3; \\ S_1 = p_{01}S_0 + p_{11}S_1 + p_{21}S_2 + p_{31}S_3; \\ S_2 = p_{02}S_0 + p_{12}S_1 + p_{22}S_2 + p_{32}S_3; \\ S_3 = p_{13}S_1 + p_{23}S_2 + p_{33}S_3. \end{cases}$$

Матриця перехідних вірогідностей матиме вигляд:

$$\left[P_{ij}\right] = \begin{vmatrix} 0,05 & 0,9 & 0,05 & 0 \\ 0,1 & 0,6 & 0,2 & 0 \\ 0,05 & 0,6 & 0,25 & 0,1 \\ 0 & 0,2 & 0,7 & 0,1 \end{vmatrix}.$$

Методика статистичного дослідження сплесків потужності в автономних електростанціях, яка буде розглянута в наступному параграфі, включає автоматизацію обробки результатів спостереження за споживанням електроенергії шляхом розробки програмних модулів автоматичного виводу графіків та параметрів режиму роботи електростанції.

5.3. Модельні дослідження впливу характеристик інформаційного каналу на якість керування СЕЕС

Часова затримка сигналу на шляху його проходження від відправника до одержувача складається з декількох частин: час на обробку в кожному вузлі, час на передачу й час на поширення сигналу. Кожний із цих етапів може бути різним по тривалості, що визначається технічним станом мережі, різними неконтрольованими випадковими причинами. Статистичні методи аналізу дозволяють побудувати емпіричним шляхом криву розподілу даної випадкової величини. Після того, як вона буде побудована, можна знайти наближену часову затримку як площу під кривою розподілу в даному діапазоні. Задачу про відшукання ймовірності влучення затримки в даний діапазон можна розв'язати за допомогою визначеного інтеграла:

$$P(\alpha < \tau < \beta) = \int_{\alpha}^{\beta} f(\tau) d\tau,$$

де τ – часова затримка, $f(\tau)$ – щільність імовірності випадкової величини τ.

Об'єктом керування є обмотка збудження синхронного генератора та дизельний агрегат з регулятором обертів. На рис. 5.18 представлена структурна схема розподіленої системи керування з дистанційним регулятором.

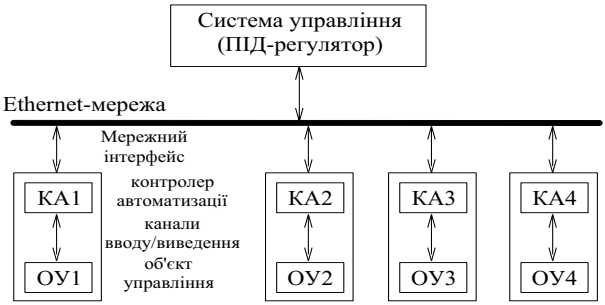


Рис. 5.18.

Лінія затримки вносить запізнювання сигналу на величину T, яка залежить від інтервалу дискретизації ν й інтенсивності запитів λ:

$$T = f(\nu, \lambda).$$

Моделювання роботи систем керування виконується шляхом заміни безперервного регулятора на цифровий, який програмно реалізований у вигляді цифрового автомату в мікроконтролері, та варіюванням часових інтервалів між зміною станів автомату. Моделі системи збудження синхронного генератора та дизельного агрегату представлено на рис. 5.19.

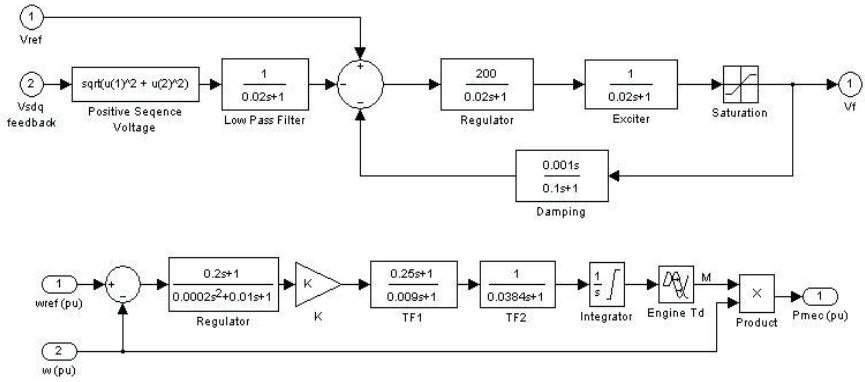


Рис. 5.19.

При заміні безперервних регуляторів цифровими моделі, представлені на рис. 5.19, будуть гібридними, які характеризуватимуться змінним значенням запізнювання в сигналах керування. Передатна функція системи збудження синхронного генератора має вигляд:

$$W_{EX}(s) = \frac{K_p}{T_d s + 1} = \frac{10}{0,01s + 1},$$

де K_p – коефіцієнт підсилення, T_d – постійна часу інтегрування.

Реалізація цифрової системи керування збудженням генератора у вигляді діаграми станів цифрового автомату представлена на рис. 5.20, а.

Передатна функція системи керування дизельним агрегатом має вигляд:

$$W_{DG}(s) = \frac{K(T_0 s + 1)}{T_1 s^2 + T_2 s + 1} = \frac{40(0,2s + 1)}{0,0002s^2 + 0,01s + 1},$$

де K – коефіцієнт підсилення, T_1, T_2, T_3 – постійні часу регулятора.

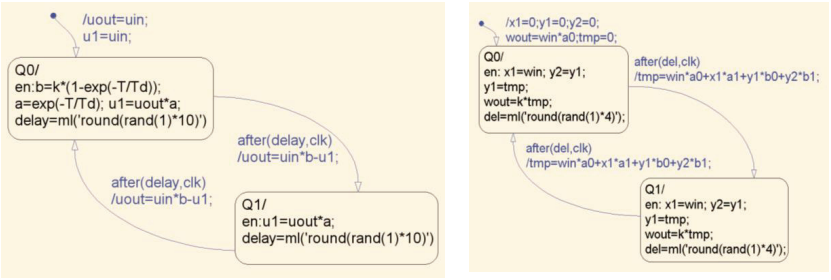
Після виконання z-перетворення, передатна функція цифрового регулятора при періоді дискретизації 0,01 сек. має вигляд:

$$W_{DG}(z) = 40 \cdot \frac{7,437 - 7,06z^{-1}}{1 - 1,229z^{-1} + 0,606z^{-2}}.$$

При зменшенні періоду дискретизації в 10 разів, передатна функція регулятора обертів дизеля приймає вигляд:

$$W_{DG}(z) = 40 \cdot \frac{0,9771 - 0,9722z^{-1}}{1 - 1,946z^{-1} + 0,9512z^{-2}}.$$

На рис. 5.20, б, представлений граф переходів цифрового автомату, що реалізує наведений закон керування та моделюється часова затримка сигналу керування (запізнювання).



а б

Рис. 5.20.

На рис. 5.21 представлена модель для дослідження цифрового регулятора системи збудження СГ.

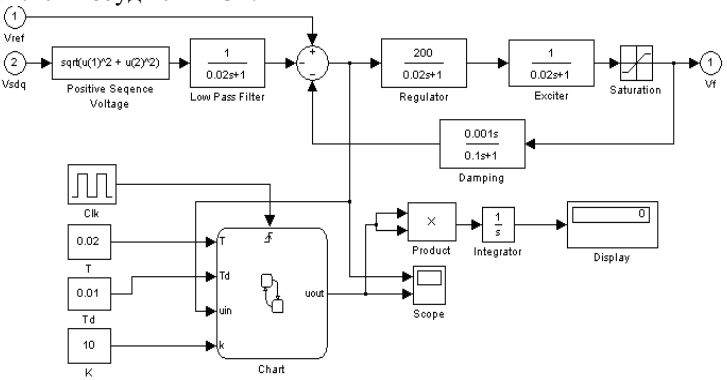


Рис. 5.21.

На рис. 5.22 представлена модель для дослідження роботи цифрової системи керування дизельним агрегатом.

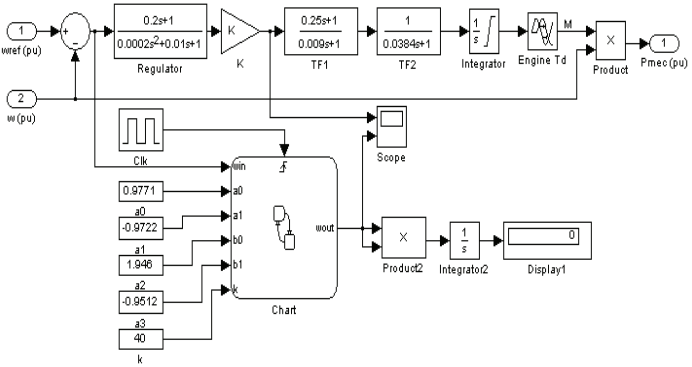


Рис. 5.22.

На рис. 5.23 представлені осцилограми виходу безперервного і дискретного регулятора оборотами дизеля. З рисунку видно дискретний характер вихідної дії регулятора.

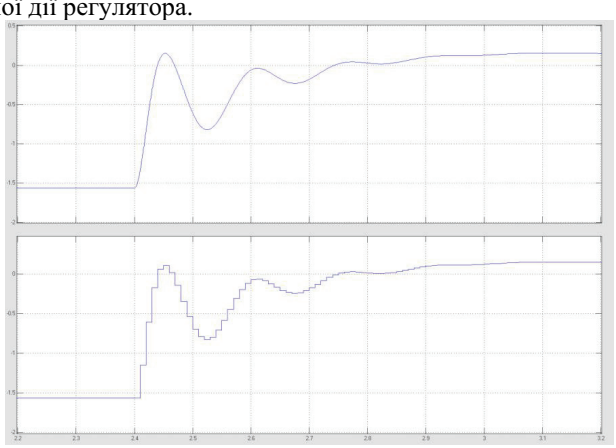


Рис. 5.23.

При зменшенні періоду дискретизації в 10 разів, передавальна функція регулятора обертів дизеля набуде вигляду:

$$W_{DG}(z) = 40 \cdot \frac{0,9771 - 0,9722z^{-1}}{1 - 1,946z^{-1} + 0,9512z^{-2}}.$$

Реалізація наведеної передавальної функції у вигляді цифрового автомата представлена на рис. 5.24.

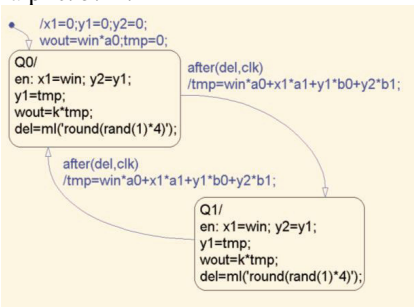


Рис. 5.24.

Середнє значення запізнювання τ, обчислюється за формулою:

$$\overline{\tau} = \frac{1}{n}\sum_{i=1}^{k} \tau_i n_i.$$

У якості характеристик непостійності цього параметру приймається стандартне відхилення S і коефіцієнт варіації V_R, що обчислюються за формулами:

$$S = \sqrt{\frac{1}{n}\sum_{i=1}^{k}(\tau_i - \overline{\tau})^2 n_i},$$

$$V_R = \left|\frac{S}{\overline{\tau}}\right| \cdot 100\%.$$

Якщо коефіцієнт варіації менше 30%, то це означає, що вибірка досить компактна, тобто часова затримка змінюється в невеликому діапазоні.

В роботі для оцінки якості керування використаний інтегральний показчик критерію якості керування A:

$$A = \int_0^{\infty} (h(t) - h_{cm}(t))^2 dt.$$

Значення інтегрального показника критерію якості керування збудженням генератора при різних значеннях флуктуацій періоду дискретизації представлені в табл. 5.2.

Таблиця 5.2.

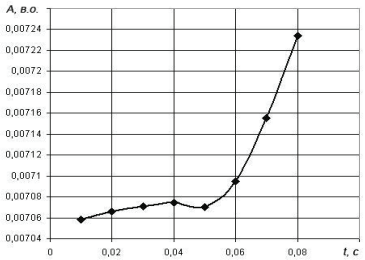
$\Delta t_д$, сек	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08
I, в.о.	0,007058	0,007066	0,007071	0,0070744	0,00707	0,007095	0,007155	0,007234

Значення інтегрального показника критерію якості керування оборотами дизеля при різних значеннях флуктуацій періоду дискретизації представлені в табл. 5.3.

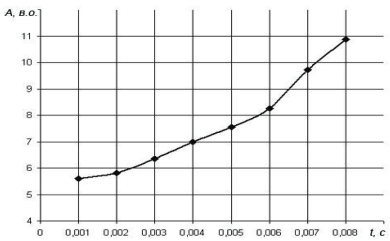
Таблиця 5.3.

$\Delta t_д$, сек	0,001	0,002	0,003	0,004	0,005	0,006	0,007	0,008
I, в.о.	5,559	5,817	6,346	6,994	7,549	8,252	9,708	10,871

Залежність інтегрального показчика критерію якості керування від затримки передачі інформаційно-керуючих пакетів на віддалений регулятор системи збудження синхронного генератора представлена на рис. 5.25, а. Аналогічна залежність для віддаленого регулятора обертів дизельного агрегату представлена на рис. 5.25, б.



а



б

Рис. 5.25.

Як видно з графіків, при збільшенні величини запізнювання якість керування погіршується. Таким чином, для керування об'єктами автоматизації в режимі реального часу необхідно мінімізувати затримки передачі пакетів по мережі.

На рис. 5.26 представлені осцилограми сигналів на виході цифрового регулятора при різних значеннях часової затримки: 1 – часова затримка відсутня; 2 – затримка 40 мс.; 3 – затримка 80 мс.

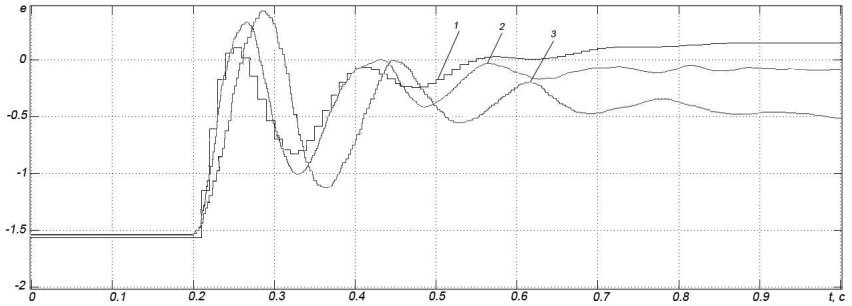


Рис. 5.26.

Структурна схема контуру дистанційного ПІД-регулятора представлена на рис. 5.27.

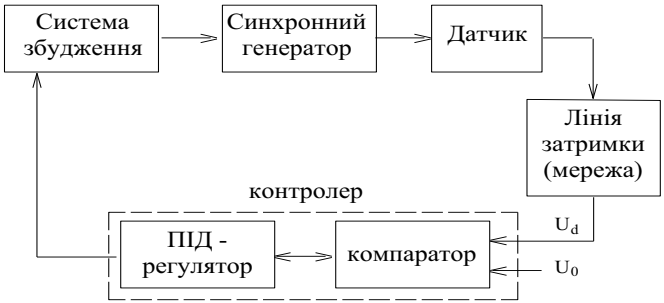


Рис. 5.27.

Лінія затримки вносить запізнення сигналу на величину, яка залежить від інтервалу дискретизації й інтенсивності запитів:

$$T = f(\nu, \lambda).$$

Розроблена модель включає мінімальний набір об'єктів, зв'язаних між собою, і які допускають зовнішнє керування (рис. 5.28).

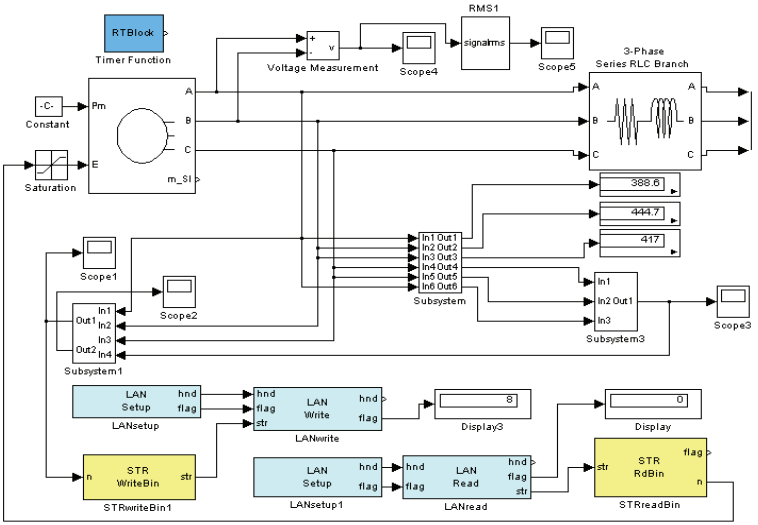
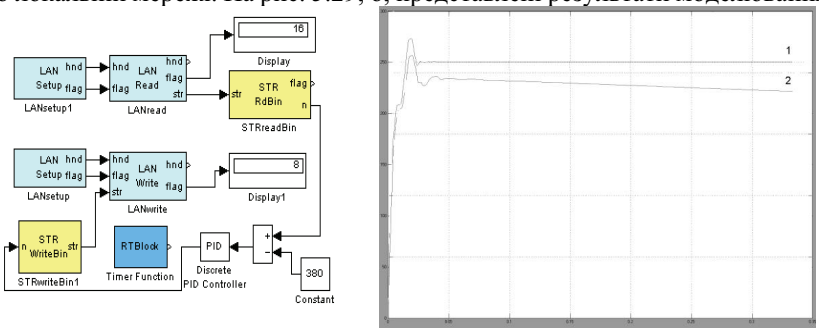


Рис. 5.28.

Генератор представлений моделлю синхронної машини з демпферною обмоткою. Параметри машини задаються в системі абсолютних одиниць. Потужність генератора 400 кВт, лінійна напруга на статорі 380 В. Генератор має 2 пари полюсів (номінальна частота обертання 1500 об/хв). Для моделювання процесу використовуються два комп'ютери з'єднані через локальну мережу Ethernet.

На рис. 5.29, а, представлена модель дистанційного регулятора, яка містить безпосередньо сам регулятор, а також блоки для організації зв'язку по локальній мережі. На рис. 5.29, б, представлені результати моделювання.



а б

Рис. 5.29.

В результаті моделювання були отримані дві криві перехідного процесу. Крива (2) є графіком перехідного процесу при моделюванні дистанційного ПІД-регулятора.

5.4. Оцінка впливу процесу комутації навантаження на завантаженість каналу зв'язку

При комплексній автоматизації СЕЕС включення двигуна відбувається відразу після надходження команди, але лише після перевірки наявності резерву потужності. За відсутності резерву потужності для пуску потужного двигуна система управління автоматично виконує всі функції, пов'язані з введенням в паралельну роботу додаткового генератора. Після завершення всіх операцій із зміни режиму роботи СЕЕС виконується запуск споживача електроенергії. Для реалізації такого механізму необхідно, щоб усі заявки на комутацію споживачів надходили до системи управління СЕЕС, яка, в свою чергу, визначає можливість комутації навантаження. Всі вузли розподіленої системи управління СЕЕС об'єднані у мережу (рис. 5.30). Тому запит на комутацію навантаження являє собою інформаційний пакет, який передається від системи управління навантаженням (Н1-НN) в комплексну систему управління (КСУ), а також пакет у відповідь з командою дозволу (заборони) пуску споживача електроенергії.

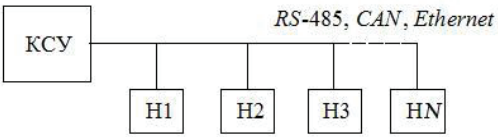


Рис. 5.30.

Надходження заявок на комутацію навантаження можна розглядати як випадковий процес, що відбувається у мережі. Для Марківського процесу справедлива така система рівнянь [16; 35]:

$$P_j(t+\Delta t)=\sum_{i=1}^{n}P_i(t)P_{ij}(\Delta t),\ \ j=1,2,\ldots,n, \tag{5.1}$$

де P_{ij} – умовна ймовірність того, що система за час Δt перейшла із стану S_i в стан S_j. Ця ймовірність залежить тільки від того, в якому стані була система і в який стан вона може перейти через час Δt, і не залежить від того, як система опинилась в цьому стані. Оскільки

$$\sum_{i=1}^{n}P_{ij}(\Delta t)=1$$

то

$$P_{ij}(\Delta t)=1-\sum_{i=1}^{n}P_{ij}(\Delta t),\ i\neq j. \tag{5.2}$$

Підставив рівняння (5.2) в систему (5.1) отримаємо:

$$P_j(t+\Delta t) = P_j(t)\left[1 - \sum_{i=1,\, i\neq j}^{n} P_{ij}(\Delta t)\right] + \sum_{i=1,\, i\neq j}^{n} P_j(t) P_{ij}(\Delta t), \tag{5.3}$$

де $j = 0, 1, 2 \ldots n$.

Інтенсивність заявок на комутацію навантажень (отже, інтенсивність інформаційно-керуючих пакетів), які викликають зміну стану системи (перехід зі стану S_i у стан S_j) визначається виразом:

$$\lambda_{ij}(t) = \lim_{\Delta t \to 0} \frac{P_{ij}(t, t+\Delta t)}{\Delta t}. \tag{5.4}$$

Для однорідних Марківських процесів інтенсивності переходів не залежать від часу, тобто $\lambda_{ij}(t) = \lambda_{ij}$.

З цього рівняння можна записати:

$$P_{ij}(\Delta t) = \lambda_{ij}\Delta t + \sigma(\Delta t),$$

де $\sigma(\Delta t)$ враховує кінцевість Δt.

З рівняння (5.3) маємо:

$$\frac{P_j(t+\Delta t) - P_j(t)}{\Delta t} = -\left(\sum_{i=1, i\neq j}^{n} \lambda_{ij}\right) P_j(t) + \sum_{i=j, i\neq j}^{n} P_j(t)\lambda_{ij} + \frac{\sigma(t)}{\Delta t}.$$

При $\Delta t \to 0$ отримуємо:

$$P_j(t) = \left(\sum_{i=1,\, i\neq j}^{n} \lambda_{ij}\right) P_i(t) + \sum_{i=1,\, i\neq j}^{n} P_j(t)\lambda_{ij},\ j = 1, 2, \ldots, n.$$

Це рівняння відображає динаміку мережі. Розв'язок системи рівнянь дозволить визначити характеристики Марківського випадкового процесу.

Досліджуємо вплив процесу комутації навантажень у СЕЕС на інформаційний обмін у мережі у разі, якщо використовується мережа Ethernet. Сигнал на комутацію навантаження надходить у деякий мікропроцесорний пристрій, який формує інформаційний пакет і передає його комплексній системі управління СЕЕС.

Для передачі інформаційного пакета пристрою необхідно отримати доступ до каналу зв'язку. Якщо пристрій отримав доступ до каналу зв'язку, то інші вузли мережі будуть очікувати на його звільнення. Розглядається випадок інформаційного обміну між двома вузлами – мікропроцесорною системою управління станом навантаження та комплексною системою управління СЕЕС. Кожен із вузлів може бути в одному з трьох станів: робота, інформаційний обмін та очікування обміну даними. Тоді для такої системи можна виділити наступні стани:

S_{00} – обидва вузли працюють;

S_{10} – перший вузол у стані обміну, другий – у стані роботи;

S_{01} – перший вузол у стані роботи, другий – у стані обміну;

S_{21} – перший у стані очікування обміну, другий – у стані обміну;

S_{12} – перший у стані обміну, другий – у стані очікування обміну.

Граф, що відображає динаміку мережі, представлений на рис. 5.31.

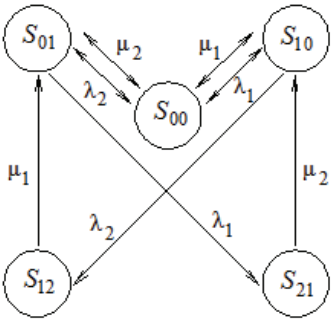


Рис. 5.31.

Тривалість автономної роботи кожного із вузлів мережі є випадковою величиною. Припустимо, що ці величини розподілені за показовим законом:

$$P\left(\tau_1^{(i)} \le t\right) = 1 - e^{-\lambda_i t},$$

$$P\left(\tau_2^{(i)} \le t\right) = 1 - e^{-\mu_i t},$$

де $\tau_1^{(i)}$ – тривалість автономної роботи i-го вузла мережі,

$\tau_2^{(i)}$ – тривалість інформаційного обміну між вузлами мережі.

Інтенсивність переходів системи між станами визначена згідно з виразом (5.4). Система рівнянь для графа на рис. 5.31 має вигляд:

$$\begin{cases} -(\lambda_1+\lambda_2)p_{00} + \mu_1 p_{10} + \mu_2 p_{10} = 0; \\ -(\lambda_2+\mu_1)p_{10} + \lambda_1 p_{00} + \mu_2 p_{20} = 0; \\ -(\lambda_1+\mu_2)p_{01} + \lambda_2 p_{00} + \mu_2 p_{12} = 0; \\ \lambda_2 p_{10} - \mu_1 p_{12} = 0; \\ \lambda_1 p_{01} - \mu_2 p_{21} = 0. \end{cases}$$

При розв'язку цієї системи рівнянь необхідно врахувати, що

$$\sum_{i=1}^{n} P_j(t) = 1.$$

Введемо змінні

$$a_1 = \lambda_1 / \mu_1, a_2 = \lambda_2 / \mu_2.$$

Розв'язок системи рівнянь має вигляд:

$$p_{00} = \frac{1 + a_1\frac{\mu_1}{\mu_2} + a_2\frac{\mu_2}{\mu_1}}{\left(1 + a_1\frac{\mu_1}{\mu_2} + a_2\frac{\mu_2}{\mu_1}\right)\left(1 + a_1 + a_2 + a_1 a_2\right) + a_1 a_2\left(1 + a_1 + a_2\right)},$$

$$p_{12} = \frac{a_1 a_2 \frac{\mu_2}{\mu_1}\left(1 + a_1 \frac{\mu_1}{\mu_2} + a_2\right)}{\left(1 + a_1 \frac{\mu_1}{\mu_2} + a_2 \frac{\mu_2}{\mu_1}\right)\left(1 + a_1 + a_2 + a_1 a_2\right) + a_1 a_2 \left(1 + a_1 + a_2\right)},$$

$$p_{21} = \frac{a_1 a_2 \frac{\mu_1}{\mu_2}\left(1 + a_2 \frac{\mu_2}{\mu_1} + a_1\right)}{\left(1 + a_1 \frac{\mu_1}{\mu_2} + a_2 \frac{\mu_2}{\mu_1}\right)\left(1 + a_1 + a_2 + a_1 a_2\right) + a_1 a_2 \left(1 + a_1 + a_2\right)}.$$

Коефіцієнт завантаження каналу зв'язку визначається виразом:

$$\eta = 1 - p_{00},$$

де p_{00} – ймовірність того, що всі вузли мережі перебувають у стані роботи. Дані вирази дозволяють побудувати залежності $\eta = f(\lambda_1, \lambda_2, \mu_1, \mu_2)$, та оцінити вплив процесів комутації навантажень у СЕЕС на завантаженість каналу зв'язку. Ця інформація може бути використана для розрахунку часу затримки передачі інформаційних пакетів мережі розподіленої системи управління.

5.5. Оцінка ефективності використання паралельних обчислень у завданнях управління

Система управління складається з кількох взаємопов'язаних між собою модулів, забезпечуючи розподіл на рівні обчислень. Для кожного з генераторів передбачений окремий обчислювальний модуль на основі 8-бітного мікроконтролера, який приймає та обробляє дані, та головний обчислювальний модуль на основі 32-бітного мікроконтролера, що приймає розрахунки від інших модулів (рис. 5.32).

Для реалізації розподілених обчислень був розроблений програмний алгоритм, що дозволяє головному обчислювальному модулю мати в своєму розпорядженні інформацію про хід виконання операцій кожного з модулів мережі в певний момент часу. Головний модуль можна розглядати як систему масового обслуговування (СМО), яка приймає заявки виконання деяких завдань, і направляє в обчислювальні модулі. Кожен із процесорів може окремо виконувати алгоритм обробки заявки, або всі три процесори виконують алгоритм паралельної обробки. Щоб порівняти ефективність використання паралельних обчислень, необхідно запровадити комплексний показник ефективності. Пропонується використовувати наступний показник ефективності:

$$E = 30\lambda^{\text{О}} - 15\tau_{\text{об}} - 10\lambda^{\text{П}},$$

де $\lambda^{\text{о}}$ – інтенсивність оброблених заявок, $\tau_{\text{об}}$ – середній час обслуговування заявки, $\lambda^{\text{п}}$ – інтенсивність втрачених заявок (через зайнятість всіх процесорів).

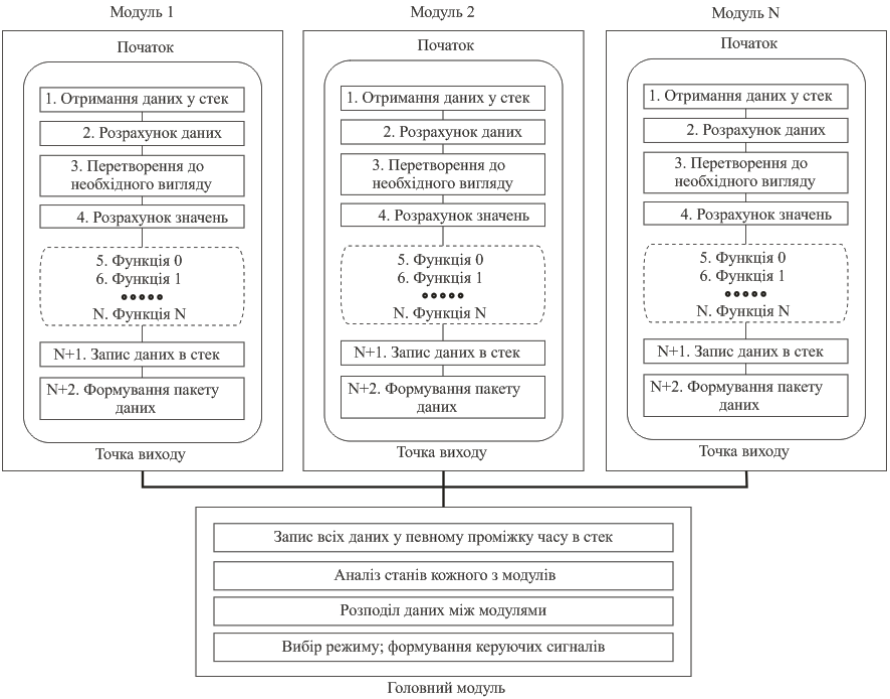


Рис. 5.32.

На рис. 5.33 представлено моделі систем масового обслуговування: а – без паралельної обработки; б – при паралельній обробці.

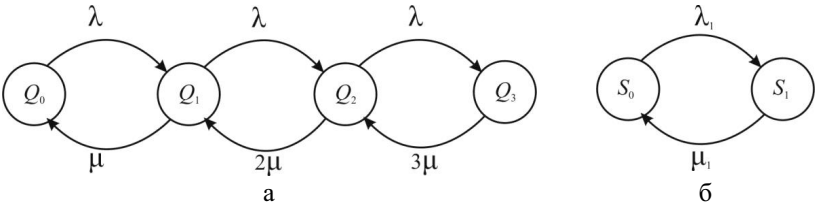


Рис. 5.33.

У стані Q_0 всі обчислювальні модулі є вільними і в системі відсутні заявки на обслуговування. У стані Q_1 обробку заявки виконує перший процесор, у стані Q_2 обслуговуванням заявки зайняті два процесори, у стані Q_3 – всі процесори зайняті обробкою. Ймовірності знаходження системи у кожному зі станів позначимо $p_0 - p_3$.

Розглядається триканальна система масового обслуговування ($m = 3$). Інтенсивність потоку завдань $\lambda = 25$ с$^{-1}$, середня трудомісткість завдання

$\tau = 4 \cdot 10^5$ операцій, середня швидкодія процесора $B = 8 \cdot 10^6$ операцій. Тоді інтенсивність обслуговування заявок $\mu = B/\tau = 20$ с$^{-1}$. Для моделі СМО на рис. 5.33, а, система рівнянь має вигляд:

$$\begin{cases} -\lambda p_0 + \mu p_1 = 0; \\ -(\lambda + \mu) p_1 + \lambda p_0 + 2\mu p_0 = 0; \\ \lambda p_1 - (\lambda + \mu) p_2 + 3\mu p_3 = 0; \\ p_0 + p_1 + p_2 + p_3 = 0. \end{cases}$$

Ймовірність втрати заявок $p_{отк}$ дорівнює ймовірності p_3, оскільки в стані Q_3 всі процесори зайняті обробкою, і заявка, що надійшла, отримає відмову. Інтенсивність втрачених заявок визначається виразом:

$$\lambda^{П} = \lambda p_{отк},$$

ймовірність обслуговування заявок:

$$p_{об} = 1 - p_{отк},$$

середній час обслуговування:

$$\tau_{об} = \frac{p_{об}}{\mu},$$

інтенсивність оброблених:

$$\lambda^{О} = \lambda p_{об}$$

На рис. 5.33, б, представлена модель СМО для варіанту паралельної обробки. Система рівнянь має вигляд:

$$\begin{cases} -\lambda_1 p_0 + \mu_1 p_1 = 0; \\ p_0 + p_1 = 1. \end{cases}$$

Ймовірність втрати заявок $p_{отк}$ дорівнює ймовірності p_1, ймовірність обслуговування $p_{об} = p_0$. Інтенсивність потоку обслуговування μ при мультипроцесорній обробці заявки не дорівнюватиме сумі інтенсивностей обробки заявок кожним допоміжним процесором, оскільки головному процесору потрібен час на розподіл завдання на підзавдання та передачі інформації кожному з допоміжних процесорів. З цих причин час обробки зменшується у 2,5 рази (а не у 3 рази). Розв'язок системи рівнянь та розрахунок комплексного показника ефективності показав, що паралельна обробка заявок є ефективнішою.

5.6. Моделювання динамічних та аварійних режимів в СЕЕС

Основними споживачами електроенергії на промислових підприємствах є асинхронні електродвигуни. Оскільки пусковий струм асинхронного двигуна (АД) у кілька разів перевищує його номінальний струм, як зазначено в [37], то при пуску виникають значні провали напруги мережі. Такі провали напруги (ПН) впливають на підключені до мережі споживачі елек-

троенергії, які можуть мати різні рівні стійкості до ПН та якості електроенергії, а також можуть призвести до виникнення аварійної ситуації та знеструмлення системи, що продемонстровано у роботах [38; 39]. Перевищення рівня стійкості електроприймача до ПН зазвичай призводить до порушення умов його нормального функціонування або відмови. З метою підвищення достовірності розрахункових значень струмів короткого замикання та пускових струмів потужних навантажень з огляду на те, що підприємство-виробник електророзподільних пристроїв (ЕРП) несе відповідальність за якість функціонування електростанції, необхідно виконати додаткові перевірки на моделі суднової електроенергетичної системи (СЕЕС). Комутаційна апаратура додатково перевіряється як за струмом навантаження, так і за величиною струму короткого замикання (КЗ) за допомогою спеціальних станцій. Це пов'язано з тим, що електромагнітні контактори мають низьку точність спрацьовування. Електронні контактори забезпечують високу точність, але їхня вартість значно вища.

Дослідженню процесів в асинхронних короткозамкнених електродвигунах промислового виконання присвячено роботу [40]. Аналіз процесів в електродвигунах суднового виконання виконаний в [41; 42]. У зазначених статтях представлені теоретичні та експериментальні дослідження як прямого пуску з різними за величиною та характером моментами опору на валу двигуна, так і регульованого пуску. Однак у цих роботах прийнято, що джерело живлення, яким є трифазний генератор, має нескінченну потужність, тобто не враховувався вплив на мережу струмів при прямому запуску двигуна. Оскільки за своїм характером пусковий струм асинхронного короткозамкнутого електродвигуна індуктивний (коефіцієнт потужності при пуску дорівнює 0,15–0,20), що продемонстровано в [43], то він створює на синхронний генератор таку розмагнічуючу дію, яка не скомпенсує в ряді випадків намагнічування синхронного генератора (мається на увазі пуск потужного асинхронно короткозамкнутого електродвигуна). У зв'язку з тим, що СЕЕС має обмежену потужність, то прямий пуск двигуна від окремого синхронного генератора може викликати неприпустимі провали напруги та частоти в мережі, час відновлення яких може виявитися більшим за допустиме, що обґрунтовано в [44]. Це додатково підтверджує необхідність виконання на етапі проектування СЕЕС модельних досліджень її роботи в аварійних та динамічних режимах для оцінки впливу таких процесів на величину напруги у різних вузлах мережі.

Для створення моделі СЕЕС повинен використовуватися програмний комплекс розрахунків перехідних електромагнітних і електромеханічних процесів у системах електропостачання. Математична модель суднової електроенергетичної системи повинна містити всі джерела живлення, що задаються своїми параметрами, лінії передачі електроенергії, трансформатори, реактори, навантаження, засоби захисту як автоматичних вимикачів зі своїми параметрами спрацьовування. Моделі СЕЕС можуть бути використані

також для вирішення завдань оптимізації розташування ЕРП у щитах управління, а також при оцінці можливих шляхів зниження величини струмів короткого замикання за встановлені заздалегідь межі.

Далі розглядається вирішення актуальної науково-технічної проблеми розробки моделі суднової електроенергетичної системи для дослідження динамічних та аварійних режимів її роботи, оцінки величини зміни напруги у різних точках мережі та здійснення найбільш обґрунтованого вибору заходів щодо оптимізації якості електроенергії, яка може бути використана при проектуванні перспективних СЕЕС із захистом від провалів напруги.

Параметри провалів напруги та формули для їх визначення наведено на рис. 5.34, з яких тривалість провалу напруги віднесена до основних показників якості електроенергії, а глибина та частота появи провалу – до допоміжних [45].

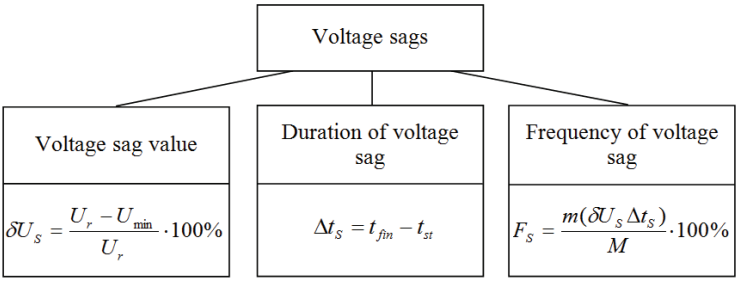


Рис. 5.34.

На рисунку позначено: U_r – номінальне значення напруги, $U_{\min}$ – мінімальне значення напруги при пуску асинхронного двигуна, t_{st} – початковий момент часу комутації навантаження, t_{fin} – значення часу, при якому напруга відновлюється до номінальної, M – кількість комутацій навантаження, що викликає провали напруги.

Розв'язання цього наукового завдання передбачає постановку двох взаємозалежних завдань:

1. Отримання ймовірнісно-статистичних оцінок провалів напруги (головним чином, тривалостей провалів напруги), що виникають в СЕЕС випадковим чином, що приймаються як вихідні дані для розробки заходів щодо зниження провалів напруги в перспективних СЕЕС;

2. Проектування СЕЕС на основі розробки методів та засобів подолання появи провалів напруги, що забезпечують підвищення безперебійності живлення відповідальних електроприймачів та врахованих на етапі обґрунтування технічних пропозицій щодо створення СЕЕС.

Відповідно до ISO 50001 «Energy Management Model» провал напруги – це раптове зниження напруги в точці електричної мережі нижче $0{,}9U_r$, за яким наступає відновлення напруги до початкового або близького до нього

рівня через проміжок часу від десяти мілісекунд до декількох десятків секунд [46]. Таким чином, провал напруги – це зниження напруги в точці електричної мережі, за яким слідує відновлення напруги до початкового або близького до нього рівня через проміжок часу від кількох періодів до кількох десятків секунд. Провал напруги характеризується трьома параметрами: глибиною δU_S, тривалістю Δt_S та частістю F_S.

Нерегулярні сплески – провали напруги мережі живлення, доцільно описати на основі теорії викидів випадкових процесів. Це зумовлено тим, що «чутливість» електронної апаратури до зовнішніх впливів проявляється в основному, при перевищенні заданого порогового рівня U_T. Усередині цього рівня електронна апаратура має демонструвати достатню «грубість».

Вихідними даними при практичному вирішенні завдання є середня частота викидів f_{AB} за рівень U_T протягом визначеного інтервалу часу, средня кількість викидів N_{AB} за час T_B, середня тривалість викидів τ_{AB}, ймовірність перевищення випадковим процесом рівня U_T ($P(u_A>U_T)$), ймовірність викиду в інтервалі Δt.

Однак визначення параметрів викидів можливе лише для процесів, для яких кореляційні функції, що їх описують, є диференційованими, наприклад, функцією виду:

$$R(\tau) = \sigma_U^2 \exp(-\alpha_U |\tau|)(1 + \alpha_U |\tau|). \tag{5.5}$$

Таким чином, шуканими є параметри кореляційної функції (5.1), яка описує випадковий процес, і рівень U_T.

Середню частоту викидів випадкового процесу можна визначити за формулою [47]:

$$f_{AB} = \frac{\alpha_U}{\sqrt{2\pi}} \varphi(U_T), \tag{5.6}$$

де

$$\varphi(U_T) = \exp\left(-\frac{U_T^2}{2}\right).$$

Таким чином, задаючи допустиму грубість систем автоматики СЕЕС, а також знаючи експериментально зняті параметри коефіцієнта кореляційного зв'язку α_U, можемо визначити середню частоту викидів за заданий пороговий рівень U_T.

Середня тривалість викидів буде пов'язана із середньою частотою викидів співвідношенням:

$$\tau_A = \frac{P(u > U_T)}{f_{AB}},$$

де ймовірність перевищення випадковим процесом рівня U_T визначається по відомій формулі:

$$P(u > U_T) = 1 - \int_{-\infty}^{U_T} p(u)du = 1 - \Phi(U_T),$$

де $p(u)$ – густина ймовірності розподілу напруг СЕЕС.

Середня кількість викидів за інтервал T_B знаходиться за формулою:

$$N_B = \frac{T_B}{f_{AB}}.$$

Таким чином, отримані результати дозволяють на етапі проектування СЕЕС спрогнозувати всі необхідні параметри перешкод, які можуть впливати на електронну апаратуру. Також це допоможе обґрунтувати доцільність вживання більш жорстких заходів щодо підвищення стабільності параметрів суднової мережі, і вирішувати необхідні оптимізаційні завдання щодо регуляторів апаратури, що відповідає за якість напруги мережі живлення.

На рис. 5.35 представлена схема для розрахунку струмів короткого замикання. Схема орієнтована на типовий широкий ряд електростанцій, що мають три дизель-генераторні агрегати, а діапазон потужностей генеруючих агрегатів знаходиться в діапазоні від сотень кВт до декількох тисяч [48; 49]. Такі суднові електростанції широко використовуються як резервні електростанції на підприємствах з безперервним циклом роботи, а також знаходять застосування на морських судах. При розрахунку системи необхідно визначити значення струмів КЗ на таких елементах: шинах генератора, розподільних або комутаційних щитах, що підключені до генераторних шин головного розподільного щита, які отримують живлення через трансформатор.

В аварійних режимах роботи електростанції розрахунки струмів короткого замикання системи повинні охоплювати струми КЗ, вироблені всіма двигунами і генераторами, підключеними до системи, включаючи струми на шинах генераторів, вторинних комутаційних щитах і розподільних щитах.

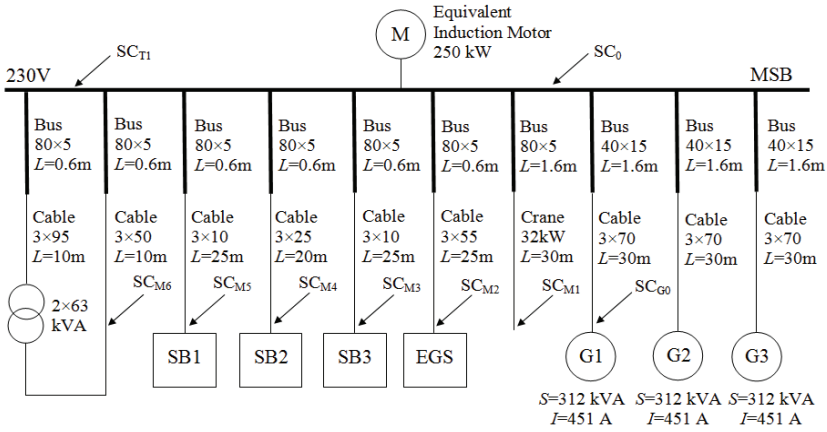


Рис. 5.35.

На рис. 5.35 використані наступні позначення: SBx – розподільний щит, Gx –генератор, SC_x – коротке замикання, MSB – ГРЩ, M – двигун, EGS – розподільний щит аварійного дизель-генератора.

Значення мінімальної напруги у відносних одиницях при роботі електростанції в динамічних режимах (пуск потужного асинхронного двигуна) визначається за формулою [50]:

$$U_{\min} = U_{st} + \left(U_{st} - U_{est}\right)e^{-t\min/\tau'_d} + k_2 k\left[t_{\min} - \tau'_d\left(1 - e^{-t\min/\tau'_d}\right)\right], \tag{5.7}$$

де $t_{\min}$ – час досягнення мінімального значення напруги для генератора із самозбудженням:

$$t_{\min} = \tau'_d \ln\left(\frac{U_{st} - U_{est}}{k_2 k \tau'_d}\right), \tag{5.8}$$

початкову напругу у виразі (5.9) можна визначити за формулою:

$$U_{st} = E'_d \frac{x_d}{x_g + x'_d}, \tag{5.9}$$

$$k_2 = \frac{x_g}{x_g + x'_d}, \tag{5.10}$$

де r_a, x_d, x_q – відносні активний та реактивний опори по повздовжній та поперечній вісям.

Напруга, що встановилася в (5.7) визначається виразом:

$$U_{est} = E_d \frac{x_g}{x_g + x_d}. \tag{5.11}$$

Постійна часу обмотки збудження (5.7) і (5.8) при замкнутій обмотці статора з опором x_g:

$$\tau'_d = \tau'_{d0} \frac{x_g + x'_d}{x_g + x_d}. \tag{5.12}$$

Реактивний опір асинхронного двигуна у формулах (5.9) – (5.12):

$$x_g = \frac{S_{rG}}{k_s S_{rM}}\left(\frac{U_{rM}}{U_{rG}}\right)^2, \tag{5.13}$$

де S_{rG}, U_{rG} – номінальні потужність та напруга генератора, S_{rM}, U_{rM} – номінальні потужність та напруга двигуна, k_s – кратність пускового струму асинхронного двигуна по відношенню до номінального струму.

На рис. 5.36 представлена модель СЕЕС, створена в пакеті MatLab та орієнтована для моделювання динамічних та аварійних режимів її роботи, тобто оцінки значень пускових струмів АД та струмів короткого замикання.

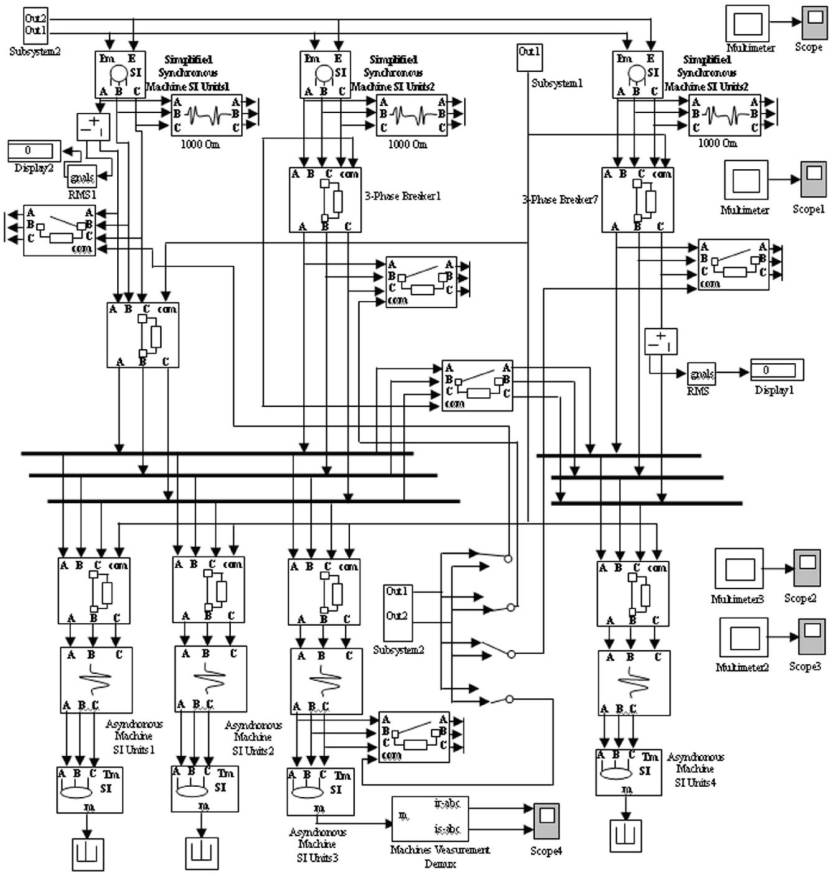
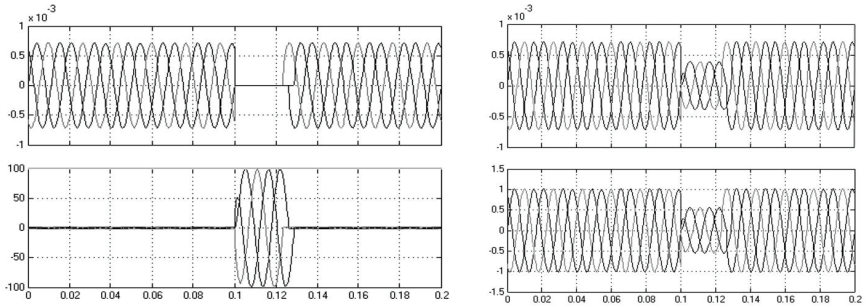


Рис. 5.36.

Електростанція містить три 3-х фазні джерела змінної напруги (3-Phase Source) з діючим значенням лінійної напруги 380 В. Джерела працюють на активно-індуктивне трифазне навантаження. Порти моделей A, B і C є виводами статорних обмоток машин. Блоки RMS та RMS1 показують діючі значення струмів фази. На вихідних портах m_SI формуються векторні сигнали, що складаються з 12 елементів: струмів (isa, isb, isc), напруг (va, vb, vc) та ЕРС (ea, eb, ec) обмотки статора, кутового положення (thetam) та швидкості ротора (Vm), а також електромагнітної потужності (Pe). Для зручності вилучення змінних стану асинхронних машин із вихідних векторів вимірюваних параметрів у бібліотеці SimPower Systems передбачено блок Machines Measurement Demux. Сигнали, що відповідають механічній потужності на валу машин, подаються на вхідні порти Pm, а на вихідні порти E подаються сигнали, що задають значення лінійних ЕРС обмоток статорів.

Існуючі методики розрахунку застосовуються для режиму трифазного симетричного короткого замикання, тобто коли трифазні провідники замкнуті один на одного, і коли коротке замикання виникає на всіх трьох полюсах одночасно. Для визначення струмів короткого замикання використовуються блоки 3-Phase Fault, у яких в певний момент часу відбувається міжфазне коротке замикання. Управління блоком 3-Phase Fault здійснюється через порт com.

Коротке замикання можна моделювати безпосередньо в джерелі напруги, лінії передачі напруги, і безпосередньо на навантаженні. Вибір місця короткого замикання здійснюється за допомогою ручних перемикачів Manual Switch, які з'єднані із блоком Subsystem (сигнал 1 – верхній вивід або 0 – нижній вивід). За допомогою ключів керуючі сигнали подаються на той чи інший блок 3-Phase Fault.



а б

Рис. 5.37.

В результаті моделювання роботи СЕЕС було отримано осцилограми напруги та струму короткого замикання, представлені на рис. 5.37, а. Для порівняння, на рис. 5.37, б, представлені осцилограми струмів і напруг на навантаженні Load1, де коротке замикання було відсутнє.

Для проектування захисту від ПН та запобігання їх наслідкам необхідно мати прогноз ПН для конкретної СЕЕС. Прогнози можна робити за наявною статистикою, яка отримується на основі багаторічних реєстрованих спостережень або розрахунків для конкретної мережі. Для реєстрації глибини і тривалості ПН необхідно організувати тривалі вимірювання спеціальними приладами, які дадуть тим більш достовірні прогнози, чим ширший масштаб охопленої вимірами мережі. Для розрахунку характеристик ПН необхідно враховувати реальну структуру та конфігурацію СЕЕС та режими роботи електроустаткування, підключеного до мережі. Модель досліджуваної СЕЕС повинна відображати її настільки докладно, щоб виконані розрахунки дали можливість визначати не тільки напруги в вузлах, що цікавлять, а й відхилення цих параметрів від нормальних установлених значень. До кожного елемента АЕЕС вибирається режим роботи системи, у якому даний

елемент перебуває у найважчих умовах під час пуску потужного асинхронного двигуна. Враховуються всі генератори, які можуть працювати на загальне навантаження одночасно (тобто паралельно), електромашинні перетворювачі, всі синхронні та асинхронні електродвигуни, що працюють у відповідних режимах. Опори елементів розрахункової схеми можуть бути виражені як в абсолютних (іменованих), так і у відносних одиницях.

Опір генераторів, двигунів (у ряді випадків й інших елементів схеми) задаються розробниками у відносних одиницях (в.о.) за номінальних умов даного елемента. Значення цих параметрів в іменованих одиницях можуть бути отримані за формулою:

$$A = A_* Z_N,$$

де A – параметр в іменованих одиницях; A_* – той самий параметр, але у в.о.; Z_N – номінальний опір, мОм, визначається за формулою:

$$Z_N = \frac{U_N^2}{S_N} = \frac{U_N}{\sqrt{3} I_N},$$

де U_N – номінальна напруга, В; S_N – повна номінальна потужність, кВА; I_N – номінальний струм, А.

Розрахунки перехідних процесів виконуються у відносних одиницях. З цією метою параметри схеми, що розглядається, наводяться до єдиних базисних умов. За базисні величини рекомендується приймати такі:

– базисну потужність, рівну сумі потужностей генераторів схеми, що розглядається (частини схеми):

$$S_b = S_{G1} + S_{G2} + ... + S_{Gn},$$

– базисна напруга, що дорівнює номінальній напрузі генераторів:

$$U_b = U_N,$$

$$I_b = \frac{S_b}{\sqrt{3} U_b},$$

$$Z_b = \frac{U_b^2}{S_b} = \frac{U_b}{\sqrt{3} U_b}.$$

Вибрані базисні одиниці служать для вирівнювання у в.о. всіх фізичних величин, як повних, так і складових (активних, реактивних та інших). За вибраних базисних умов відносні ЕРС, напруга, струм, потужності та опори визначаються таким чином:

$$E_{*b} = \frac{E}{U_b}, U_{*b} = \frac{U}{U_b}, I_{*b} = \frac{I}{I_b},$$

$$S_{*b} = \frac{S}{S_b}, P_{*b} = \frac{P}{S_b}, Z_{*b} = \frac{Z}{Z_b},$$

$$r_{*b} = \frac{r}{r_b},\ x_{*b} = \frac{x}{x_b},$$

де символ зірочка «*» вказує, що відповідна величина виражена у в.о., а індекс «*b*» – що вона приведена до базисних умов; *E*, *U*, *I*, *S*, *P*, *Z*, *r*, *x*, – відповідно ЕРС, напруга, струм, повна і активна потужності, повний, активний та індуктивний опори в тих же іменованих одиницях, що і відповідні базисні величини.

Переведення величин з відносних одиниць до іменованих виконується за формулами:

$$U = U_{*b}U_b,\ E = E_{*b}U_b,$$
$$S = S_{*b}S_b,\ P = P_{*b}S_b,$$
$$I = I_{*b}I_b,\ Z = Z_{*b}Z_b,$$
$$r = r_{*b}r_b,\ x = x_{*b}x_b.$$

Для обраного розрахункового режиму виходячи з принципової схеми СЕЕС складається однолінійна розрахункова схема. До розрахункової схеми включаються:

– джерела електроенергії, що працюють у розглянутому режимі;

– елементи установки, що зв'язують джерела електроенергії між собою та з аварійною ділянкою (силові трансформатори, реактори, комутаційно-захисна апаратура, ділянки кабельних трас);

– синхронні двигуни;

– асинхронне навантаження у вигляді одного або кількох еквівалентних двигунів або окремих двигунів великої потужності.

Потужність та номінальний струм еквівалентного двигуна РЕМ та INEM визначаються за таблицею навантажень як сума номінальних потужностей та номінальних струмів асинхронних двигунів, що працюють у розглянутому режимі. Еквівалентний двигун приєднується безпосередньо до шин головного розподільного щита.

З розрахункової схеми складається схема заміщення. Для цього елементи схеми (генератори, трансформатори, двигуни, реактори, ділянки кабельних трас та протяжних шин) заміщаються активними та індуктивними опорами. Магнітно зв'язані ділянки системи замінюються еквівалентними електрично зв'язаними ланцюгами. Синхронні генератори, синхронні та асинхронні двигуни у схемі заміщення враховуються своїми ЕРС, активними та індуктивними опорами. Якщо схема повністю симетрична (однакові джерела, однакові опори тощо), чи якась її частина має симетрію щодо деякої проміжної точки, її можна скласти по вісі симетрії, тобто з'єднати точки рівного потенціалу. І тут потужність джерел чи двигунів дорівнює сумі їх потужностей, а опори складаються паралельно.

Досліджувана електростанція містить три синхронні генератори потужністю 250 кВт кожен (рис. 5.38).

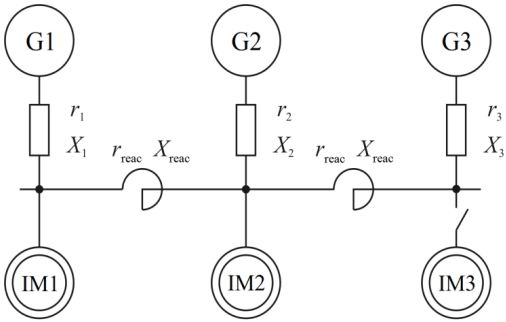


Рис. 5.38.

На рис. 5.39 представлена схема заміщення досліджуваної електроенергетичної системи.

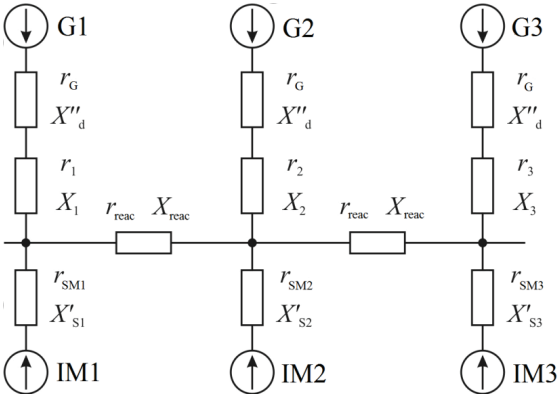


Рис. 5.39.

Перетворення схеми заміщення може виконуватися як в іменованих, так і у відносних (базисних) одиницях. У випадках, коли слід очікувати, що розрахункові значення пускових струмів будуть у межах допустимих, можна нехтувати відносно невеликими опорами автоматичних вимикачів, генераторних кабелів довжиною до 10–15 м, перехідними опорами контактів тощо.

Для визначення кількості сплесків і провалів напруги, які можуть виникати з різних причин, використано метод збору статистичних даних, що ґрунтується на обробці спостережень за подіями, що призводять до провалів напруги. Отримані таким шляхом дані необхідні для визначення емпіричного (статистичного) розподілу випадкової величини (сплесків і провалів).

На основі індивідуальних графіків навантаження побудований добовий графік навантажень секції ГРЩ (рис. 5.40). Орієнтовний час комутації

навантажень визначався виходячи з режимів їх роботи, що визначаються технологічними процесами.

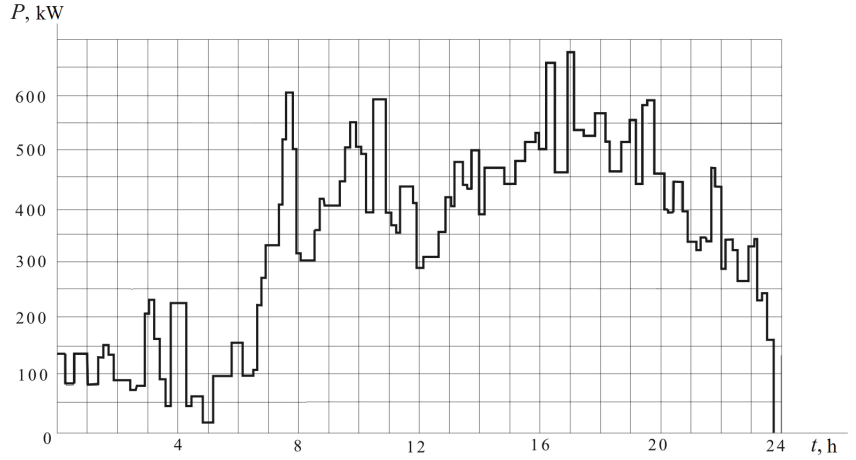


Рис. 5.40.

Для розрахунку числових параметрів випадкових процесів використовувалися формули:

$$M_X = \frac{1}{N_S} \sum_{k=1}^{N_B} x_k,$$

$$\sigma_X^2 = \frac{1}{N_S} \sum_{k=1}^{N_S} x_k^2 - M_X^2,$$

де M_x – математичне очікування, σ_X^2 – дисперсія, N_S – кількість вибірок x_k.

Середнє значення провалу напруги становить 13,8 В, мінімальне значення дорівнює 7 В, а максимальне 27 В. Середня тривалість провалу напруги становить 0,42 с. Середнє значення сплеску напруги становить 12,9 В, мінімальне значення сплеску дорівнює 5 В, максимальне значення сплеску становить 25 В. Середня тривалість провалу напруги становить 0,4 с. Частість появи провалів та сплесків напруги приблизно рівні та становлять 50 %. Генератори мають такі параметри: $S = 250$ кВА, $U = 400$ В, $x'_d = 0{,}20$; $x_d = 1{,}1$, $k = 20$, $t_1 = 0{,}05$ с. Такие значения соответствуют генератору GMS-13-31-12.

Розглянуті споживачі характеризуються індивідуальними режимами роботи і процес їх комутації має випадковий характер. Вибравши одиничний випадок включення споживача, визначаються втрати напруги, що передаються від ГРЩ до кінцевого споживача. На рис. 5.41 представлений графік перехідного процесу, отриманий при запуску асинхронного двигуна потужністю 185 кВт.

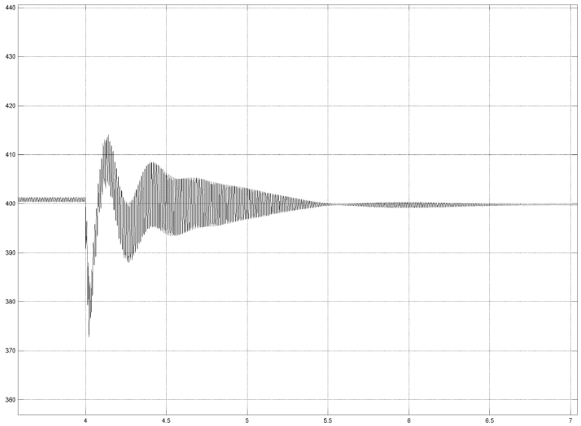


Рис. 5.41.

Час відновлення напруги визначається за такою формулою [16]:

$$t_{est} \le 1.5 \frac{\left[\left(r_a + 0.4\right)^2 + \left(x_d + 0.92\right)\left(x_q + 0.92\right)\left(r_a^2 + x'_d x_q\right)\right]}{\left[\left(r_a + 0.4\right)^2 + \left(x'_d + 0.92\right)\left(x_a + 0.92\right)\left(r_a^2 + x_d x_q\right)\right]},$$

де r_a, x_d, x_q – відносні активний та реактивний опори по поздовжній та поперечній вісях; 0,4 і 0,92 – відносні значення активного та індуктивного опору навантаження в режимі при cosφ=0.4. Максимальний провал напруги (у %):

$$U_{\max} = (1 - U_{\min}) \cdot 100\%.$$

Оскільки комутаційні процеси підключення – відключення навантажень носять випадковий характері і призводять до випадкових коливань амплітуд напруг і відхилень частоти мережі, останні певною мірою можуть бути оцінкою процесів комутації навантажень в судновій мережі. Для розробки засобів захисту від провалів напруги та запобігання їх наслідкам необхідно мати прогноз ПН для конкретної СЕЕС. Прогнози можна робити за наявною статистикою чи розрахунками для конкретної мережі. Цілком виключити провали напруги в системах зовнішнього електропостачання неможливо. Мінімізація збитків від провалів напруги досягається комплексом заходів, застосуванням спеціальних швидкодіючих захистів та автоматики, впровадженням самозапуску відповідальних електродвигунів. Зокрема, у [51; 52] авторами також проведено модельні дослідження систем частотного керування асинхронними двигунами та розглянуто аналітичний метод аналізу процесів керування двигунами та генераторами у складі СЕЕС. Також розглянуто формальний метод розв'язання задачі структурної оптимізації моделі системи управління АД, який може бути застосований до розглянутої в цьому параграфі моделі СЕЕС.

В результаті проведеного дослідження виконано розробку моделі суднової електроенергетичної системи, що має у складі 3 синхронних генератора та активно-індуктивне навантаження. Розроблена модель СЕЕС дозволяє вирішувати завдання визначення струмів короткого замикання та пускових струмів потужних споживачів електроенергії, наприклад, асинхронних двигунів. Проведено моделювання режиму короткого замикання в мережі та процесу прямого запуску потужного асинхронного двигуна. Виконано дослідження впливу цих процесів на напругу у мережі СЕЕС. Показано, що для розрахунку характеристик провалів напруги необхідно враховувати реальну структуру та конфігурацію енергосистеми, а також режим електрообладнання. Тому запропонована модель досліджуваної електростанції відображає її досить докладно для того, щоб виконані розрахунки дали можливість визначати не тільки напруги, струми, потужності в вузлах, що цікавлять, а й відхилення цих параметрів від нормальних значень. Проведена порівняльна оцінка теоретичних методів розрахунку та модельних досліджень показала високий ступінь точності результатів моделювання. Встановлено, що максимальний розкид розрахункових значень не перевищує 10%, що цілком достатньо для вирішення таких завдань. При розрахунку пікових значень пускових струмів похибки також перебували в межах 10–15%. Однак для практики розрахунку пускових струмів АД важливе значення мають струмові характеристики автоматичних вимикачів. Щодо цього модельні дослідження та отримання кривих перехідних процесів пуску потужних споживачів дають можливість більш точно вибирати тип автоматичного вимикача та його налаштування. Мінімізація збитків від провалів напруги досягається комплексом заходів, застосуванням спеціальних швидкодіючих систем захисту та автоматики, впровадженням ефективних структур та методів запуску відповідальних електродвигунів. Найбільш ефективним способом, що вирішує проблеми провалів напруги, є упереджуюче керування.

5.7. Комп'ютерна модель СЕЕС для дослідження процесу синхронізації генераторів

На рис. 5.42 представлена модель СЕЕС, що складається з 2-х синхронних генераторів потужністю 300 кВт з їх приводними дизелями, навантаженням, комутаційної апаратури і компонентів системи керування у вигляді скінченних автоматів: Chart – система розподілу активної потужності між генераторами, Chart1 – система синхронізації.

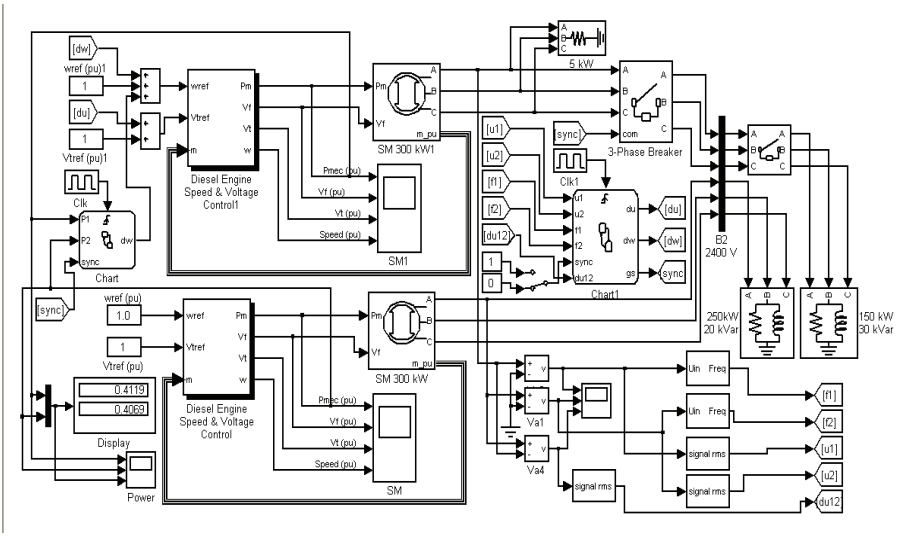


Рис. 5.42.

Система синхронізації генераторів формує імпульсні сигнали для корекції збудження генератора і обертів його приводного дизеля. Для вирівнювання напруги генератора, що підключається, з напругою працюючого, формуються імпульси на збільшення (x_1) або зменшення (x_2) збудження. Якщо $x = 1$ – контакти реле замкнуті.

$$x_1 = \frac{1}{2}(1 + \text{sign}(U_1 - U_2)),$$

$$x_2 = \frac{1}{2}(1 + \text{sign}(U_2 - U_1)),$$

$$\text{sign}(0) = -1.$$

Для корекції напруги генератора, що підключається, в ланцюзі збудження крім напруги холостого ходу з'явиться прирощення напруги від регулятора:

$$U_f(p) = U_{f_0} + \Delta U_f(p),$$

$$\Delta U_f(p) = K_U(1 - U_d),$$

де U_d – номінальна напруга синхронного генератора у в.о.;

K_U – коефіцієнт регулювання по відхиленню, у наведеній моделі дорівнює 0,001.

На рис. 5.43 представлений граф переходів системи синхронізації генераторів.

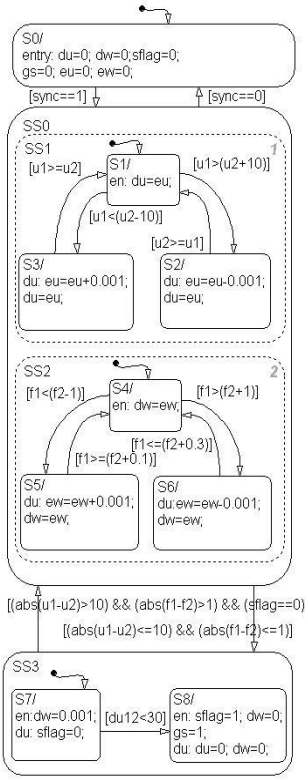


Рис. 5.43.

Для визначення різниці фаз між напругами генераторів використовується діюче значення напруги биття. Оскільки алгоритм роботи системи синхронізації побудований таким чином, що визначення різниці фаз виконується після того, як частоти і діючі значення напруг обох генераторів будуть лежати в допустимих межах (ΔU <10В, Δf <1Гц), то за напругою биття

$$U_6(t) = U_d(\sin w_1 t + \sin w_2 t)$$

можна визначити значення різниці фаз між напругами працюючого і генератора і генератора, що підключається. Частота напруги $U_6(t)$ дорівнює:

$$\Omega = (w_1 - w_2)/2.$$

Тоді

$$U_6(t) = 2U_d \cos \Omega t \cdot \sin wt,$$

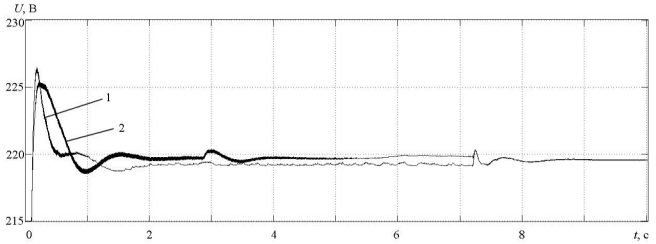
де

$$w = (w_1 + w_2)/2.$$

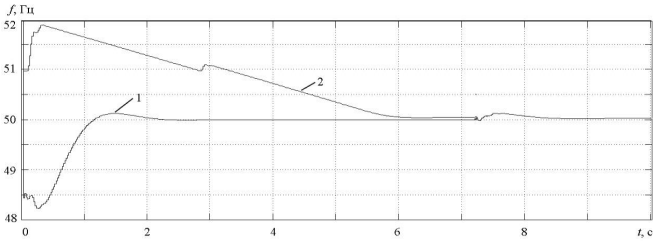
У моделі сигнал на включення автомата формується при U_6<30В, що відповідає припустимій різниці фаз між напругами генераторів.

У стані S0 система очікує зовнішнього сигналу sync для виконання автоматичної точної синхронізації. Після надходження команди початку процесу синхронізації система переходить в стан SS0, в яке вкладені два автомати, що працюють паралельно. У стані SS1 виконується коригування збудження підключається генератора, а в стані SS2 виконується коригування обертів приводного дизеля.

На рис. 5.44 представлені осцилограми зміни напруг і частот працюючого генератора (крива 1) і генератора, що підключається (крива 2).



а



б

Рис. 5.44.

Після синхронізації генераторів в роботу вступає система розподілу активної потужності (РАП), граф переходів якої представлений на рис. 5.45, а на рис. 5.46 представлений результат моделювання процесу розподілу активної потужності між генераторами.

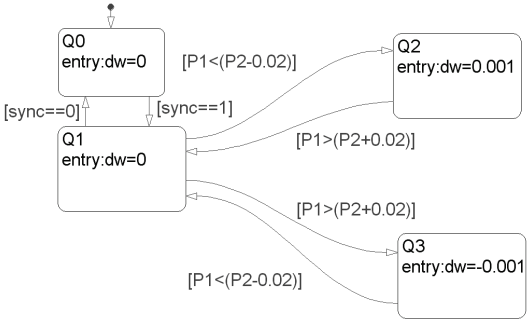


Рис. 5.45.

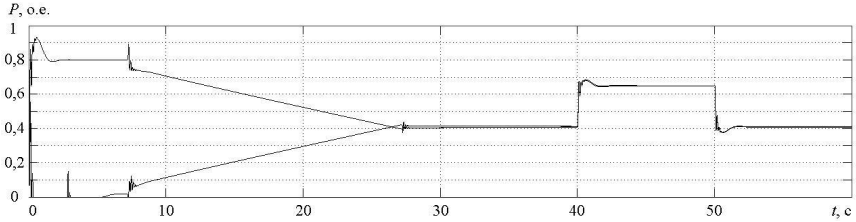


Рис. 5.46.

На рис. 5.47 представлена структура моделі мікропроцесорної системи керування, яка виконує керування електроенергетичною системою.

Рис. 5.47.

Перший мікропроцесор реалізує алгоритми керування дизель-генератором (керування обертами дизеля і збудженням генератора). Другий мікропроцесор виконує керування напівпровідниковим перетворювачем електроенергії. Із системи Matlab контрольовані параметри передаються в систему моделювання Proteus по послідовному порту (моделі засобів автоматизації наведено в додатку Б). Зв'язок між мікропроцесорами організований з використанням інтерфейсу SPI. Це дозволяє промоделювати складні зворотні зв'язки з урахуванням впливу на якість керування транспортних затримок, які виникають при передачі даних по мікропроцесорній мережі.

5.8. Модельні дослідження регуляторів для стабілізації коефіцієнту потужності в СЕЕС

Зростаюча складність сучасних автономних електроенергетичних систем (СЕЕС) і зростання вимог до якості електричної енергії в цих системах висуває задачу ретельного дослідження СЕЕС на різних етапах проектування.

Закономірною тенденцією є неперервне зростання потужності автономних електроенергетичних систем. Зростання потужності СЕЕС, в свою чергу, ускладнює процес забезпечення заданої якості електроенергії.

Споживачі електроенергії розраховані працювати на номінальні параметри, які забезпечують їх високі показники ККД, показників надійності, а також тривалий час роботи. Сучасний розвиток комп'ютерних систем зв'язку, обробки інформації, а також автоматичних систем керування технологічними процесами і виробничими комплексами, пред'являє підвищені вимоги не тільки до надійності електропостачання, але й до якості електроенергії.

Характерною особливістю встановленого режиму роботи електроенергетичної системи являється одночасність процесів генерування і споживання однієї кількості потужності. Отже, у встановленому режимі роботи СЕЕС в кожний момент часу зберігається баланс як активної, так і реактивної потужностей.

Генерація реактивної потужності здійснюється не тільки генераторами електростанції, але й високовольтними повітряними та кабельними лініями електропередачі (за рахунок їх ємнісної провідності), а також спеціально встановленими джерелами реактивної потужності, які також називаються компенсуючими пристроями.

Баланс реактивної потужності розраховується, як правило, для режиму найбільшого навантаження. Реактивна потужність, яка виробляється генераторами електростанції, визначається їх загрузкою активною потужністю та коефіцієнтом потужності cosφ, номінальне значення якого складає 0,8…0,9. Генератори являються основними джерелами реактивної потужності та виробляють близько 60% реактивної потужності в СЕЕС.

Оскільки існує залежність реактивної потужності, яка споживається, від напруги, очевидно, що існує і зворотна залежність між цими величинами. Зміна навантаження реактивної потужності, яке поступає у вузол, викличе зміну навантаження в цьому вузлі. Отже, потрібний рівень напруги в окремих вузлах електричної мережі може бути забезпечено тільки при окремому розподілу реактивних потужностей. Будь-яке відхилення від цього розподілу реактивних потужностей викличе відхилення напруги у вузах мережі від потрібного рівня.

При рішенні задач покращення якості електроенергії суднової електроенергетичної системи широко використовується моделювання, яке здійснюється на основі обчислювальних моделей. В теперішній час немає можливості створення універсальних моделей автономних електроенергетичних систем, які адекватно відображали їх властивості в різних режимах функціонування. Тому розглядаються спеціалізовані математичні моделі, які відповідають окремим процесам в СЕЕС.

Робота присвячена розробці системи покращення якості електроенергії шляхом керування коефіцієнтом потужності при роботі синхронного генератора паралельно з мережею. Коефіцієнт потужності змінюється шляхом

регулювання збудження синхронного генератора. Досліджується якість керування коефіцієнтом потужності з використанням трипозиційного регулятора, ПІД-регулятора та Fuzzy-регулятора.

Трипозиційні регулятори забезпечують гарну якість регулювання для інерційних об'єктів з малим запізнюванням [5]. Структурна схема трипозиційної системи регулювання збудженням синхронного генератора наведена на рис. 5.48.

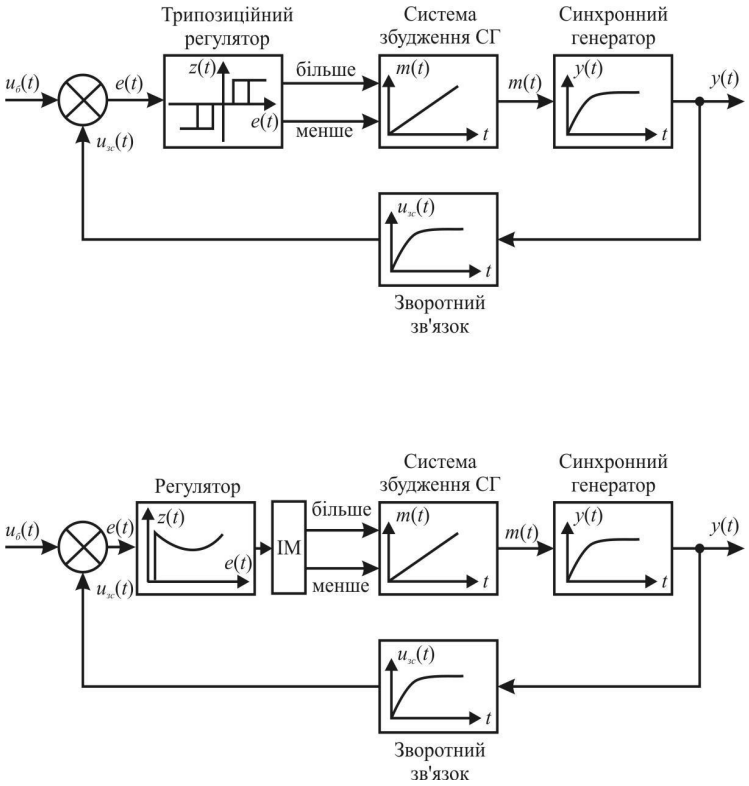


Рис. 5.48.

Для запобігання «тремтіння» керуючого вихідного пристрою (реле) і виконавчого механізму поблизу точки його включення передбачається гістерезис *H*. Також зона гістерезису призначена для виключення одночасного включення вихідних пристроїв на одночасне збільшення і зменшення збудження. Matlab-модель СЕЕС з трипозиційним регулятором cosφ представлена на рис. 5.49.

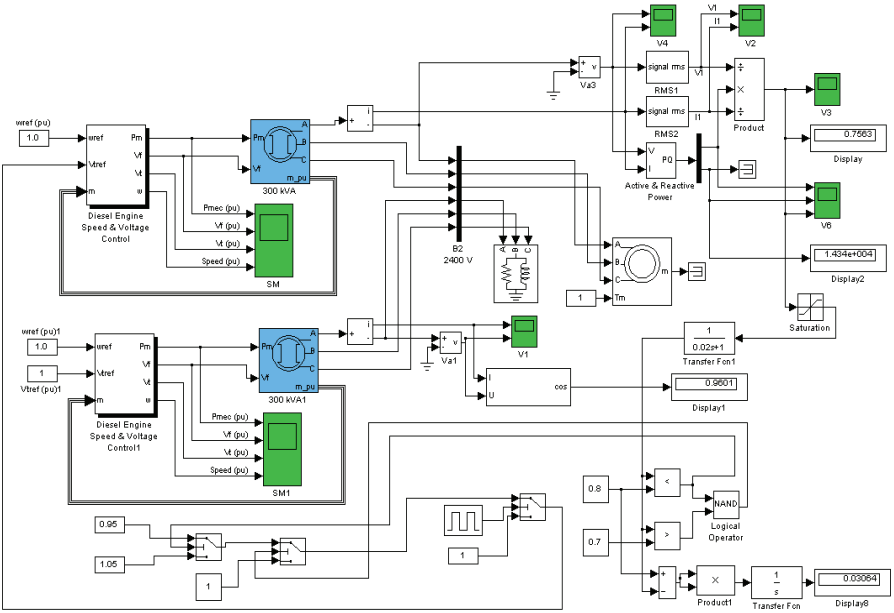
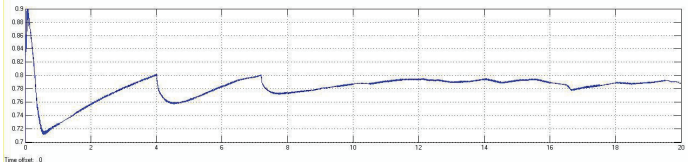
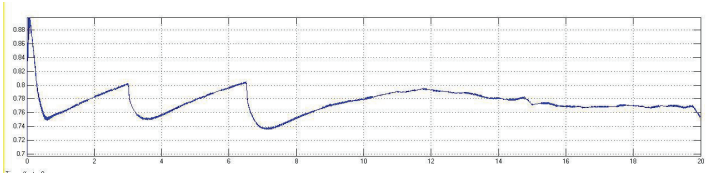


Рис. 5.49.

На рис. 5.50 представлені графіки зміни cosφ при паралельній роботі синхронного генератора потужністю 300 кВт з мережею при різних періодах імпульсів керування: а – при Т=1 с; б – при Т=0,5 с



а



б

Рис. 5.50.

Для одержання найкращих показників якості керування необхідно виконати оптимізацію регулятора. Оптимізації підлягають такі параметри регулятора, як період імпульсів керування та їх сквапність. При оптимізації параметрів регулятора мінімізується інтегральний критерій якості. В табл. 5.4. представлені значення інтегрального критерію *I* якості керування при різних значеннях періоду імпульсів керування, а також при різних значеннях завантаження генератора (25%, 50% та 75% від номінальної потужності).

Таблиця 5.4.

T, с	$I_{25\%}$	$I_{50\%}$	$I_{75\%}$
0,3	0,0585	0,0417	0,0512
0,6	0,05791	0,03534	0,05035
0,9	0,03327	0,02319	0,02627
1,2	0,04324	0,02799	0,02373
1,5	0,0463	0,0323	0,0293
1,8	0,0487	0,0342	0,0366
2,1	0,05229	0,0379	0,04442
2,4	0,05852	0,04375	0,0471
2,7	0,0596	0,0467	0,0495
3	0,06295	0,05279	0,054

На рис. 5.51 представлені графіки залежностей інтегрального критерію якості від періоду імпульсів керування.

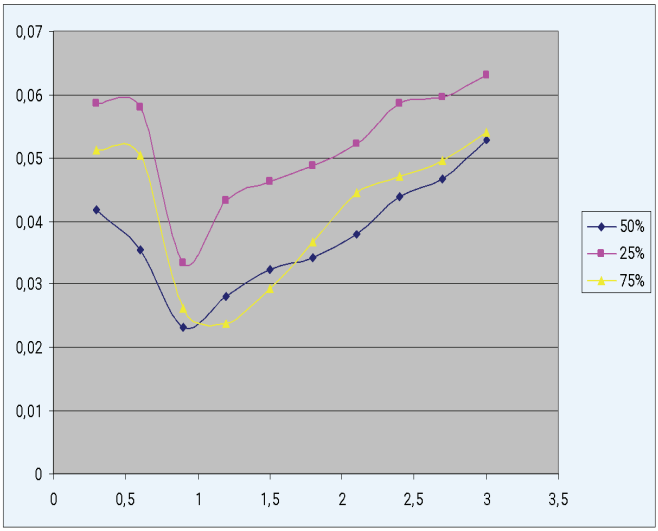


Рис. 5.51.

Процес трипозиційного регулювання є автоколивальним [5] – регульована величина як у перехідному, так і у сталому режимі періодично змінюється відносно заданого значення, тобто регульована величина, в даному випадку cosφ, піддана незатухаючим коливанням. Незважаючи на це, шляхом установки ширини зони нечутливості регулятора можливо забезпечити стабілізацію коефіцієнту потужності в заданих межах.

Структурна схема ПІД-регулятора з імпульсним виходом (ІМ) приведена на рис. 5.52. Вихідні керуючі сигнали регулятора – сигнали Більше і Менше (реле) впливають на систему збудження синхронного генератора.

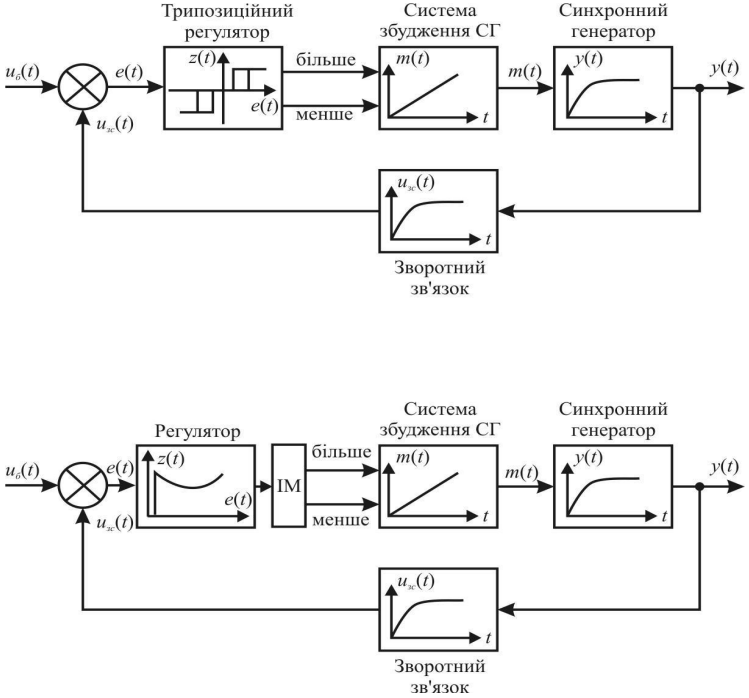


Рис. 5.52.

Досліджується робота системи стабілізації коефіцієнту потужності з ПІ-регулятором. Matlab-модель представлена на рис. 5.53.

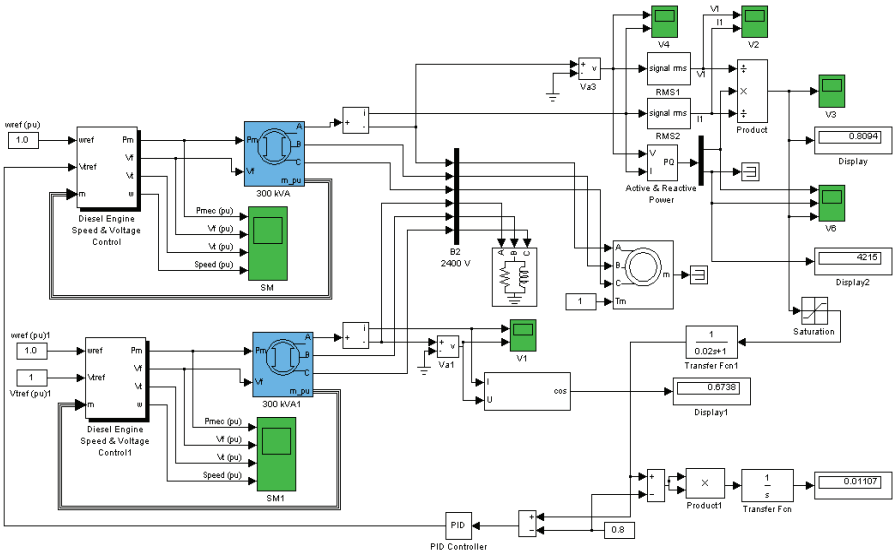


Рис. 5.53.

Результати моделювання (P, Q, cosφ) представлені на рис. 5.54.

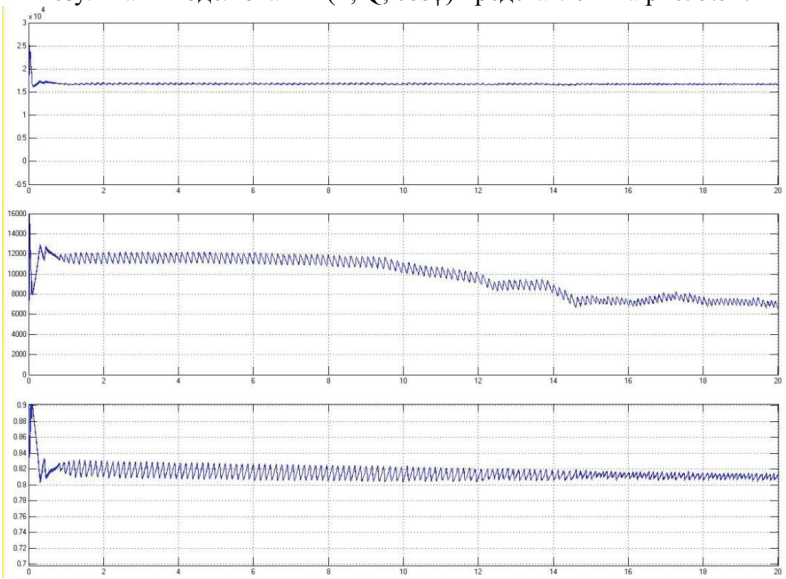


Рис. 5.54.

ПІД-регулятор достатньо просто налаштовується для роботи з конкретним об'єктом і забезпечує задовільну стабілізацію регулюємого параметра при незначних його змінах від заданої величини. Але при різких змінах режиму роботи об'єкта, яким керують, чи переході на інший режим роботи, якість перехідного процесу в системі з ПІД-регулятором може стати незадовільною.

Функціональна схема системи автоматичного керування на базі нечіткої логіки (системи керування з нечітким регулятором (НР)) представлена на рис. 5.55.

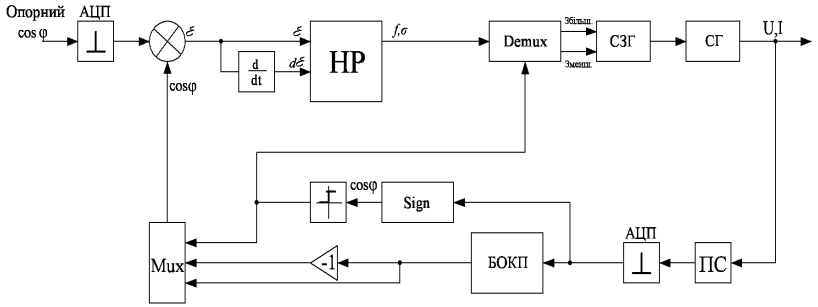


Рис. 5.55.

В якості вхідних змінних використовується поточне значення похибки cosφ ε, яка дорівнює різниці між опорним cosφ та поточним cosφ, а також швидкість зміни похибки ε (dε).

Для кожної змінної задані лінгвістичні терми, відповідні деяким діапазонам чітких значень. Щоб визначити граничні значення похибки ε розглянемо ймовірні значення опорного cosφ та поточного cosφ. Опорний cosφ задається, тому оптимальне значення, яке можна задати cosφ=0,3..1. Поточний cosφ може приймати значення від 0 до 1, але навантаження може мати різний характер: індуктивний та ємнісний. Для правильної роботи треба розрізняти значення характеру навантаження, тому приймаємо значення cosφ при ємнісному характеру умовно негативними, тоді поточне значення cosφ можуть приймати значення від –1 до 1. Виходячи з того, що ε=cosφ(оп.) –cosφ(пот.), похибка ε має такі границі [–0,7..2]. Для вхідної змінної «error» (ε) використовується шість термів: NH – від'ємне велике, NL – від'ємне мале, Z – близьке до нуля, PL – позитивне мале, PM – позитивне середнє, PH – позитивне велике.

Для вхідної змінної «derror» (dε) визначаємо такі границі [–0,2..0,2] так як швидкість зміни похибки може зростати та спадати, тоді приймаємо збільшення швидкості за позитивне а зменшення за негативне. Визначаємо такі терми: QD – швидко спадає, SD – повільно спадає, C – постійне, SI – повільно зростає, QI – швидко зростає.

В якості вихідних лінгвістичних змінних приймаємо частоту та сквапність вихідного сигналу. Лінгвістична змінна «частота» має границі [0…4] Гц та п'ять термів: Z – нульове значення, L – низьке значення, M1 – перше середнє значення, M2 – друге середнє значення, H – високе значення частоти. Лінгвістична змінна «скважність» має границі [0…100]% та чотири терми: Z – нульове значення, L – низьке значення, M – середнє значення, H – високе значення скважності.

Аналітичний вираз, що описує трикутні функції приналежності, має вигляд:

$$\mu(x) = \begin{cases} 0, x \le a; \\ \dfrac{x-a}{b-a}, a \le x \le b; \\ \dfrac{c-x}{c-b}, b \le x \le c; \\ 0, c \le x, \end{cases}$$

де *a, b, c* – параметри функцій належності.

База правил для «частоти» та «скважності» має такий вигляд:
ЯКЩО ε = PH **та** dε = QD **ТО** «частота» = L, «скважність» = M
ЯКЩО ε = PH **та** dε = SD **ТО** «частота» = L, «скважність» = H
ЯКЩО ε = PH **та** dε = C **ТО** «частота» = M1, «скважність» = M
ЯКЩО ε = PH **та** dε = SI **ТО** «частота» = M1, «скважність» = H
ЯКЩО ε = PH **та** dε = QI **ТО** «частота» = M2, «скважність» = M
ЯКЩО ε = PM **та** dε = QD **ТО** «частота» = M1, «скважність» = H
ЯКЩО ε = PM **та** dε = SD **ТО** «частота» = M2, «скважність» = M
ЯКЩО ε = PM **та** dε = C **ТО** «частота» = M2, «скважність» = H
ЯКЩО ε = PM **та** dε = SI **ТО** «частота» = H, «скважність» = M
ЯКЩО ε = PM **та** dε = QI **ТО** «частота» = H, «скважність» = H
ЯКЩО ε = PL **та** dε = QD **ТО** «частота» = L, «скважність» = L
ЯКЩО ε = PL **та** dε = SD **ТО** «частота» = L, «скважність» = M
ЯКЩО ε = PL **та** dε = C **ТО** «частота» = L, «скважність» = H
ЯКЩО ε = PL **та** dε = SI **ТО** «частота» = M1, «скважність» = M
ЯКЩО ε = PL **та** dε = QI **ТО** «частота» = M1, «скважність» = H
ЯКЩО ε = Z **та** dε = QD **ТО** «частота» = L, «скважність» = M
ЯКЩО ε = Z **та** dε = SD **ТО** «частота» = L, «скважність» = L
ЯКЩО ε = Z **та** dε = C **ТО** «частота» = Z, «скважність» = Z
ЯКЩО ε = Z **та** dε = SI **ТО** «частота» = L, «скважність» = L
ЯКЩО ε = Z **та** dε = QI **ТО** «частота» = L, «скважність» = M
ЯКЩО ε = NL **та** dε = QD **ТО** «частота» = M1, «скважність» = H
ЯКЩО ε = NL **та** dε = SD **ТО** «частота» = M1, «скважність» = M
ЯКЩО ε = NL **та** dε = C **ТО** «частота» = L, «скважність» = H
ЯКЩО ε = NL **та** dε = SI **ТО** «частота» = L, «скважність» = M
ЯКЩО ε = NL **та** dε = QI **ТО** «частота» = L, «скважність» = L
ЯКЩО ε = NH **та** dε = QD **ТО** «частота» = H, «скважність» = H

ЯКЩО ε = NH **та** dε = SD **ТО** «частота» = H, «скважність» = M
ЯКЩО ε = NH **та** dε = C **ТО** «частота» = M2, «скважність» = H
ЯКЩО ε = NH **та** dε = SI **ТО** «частота» = M2, «скважність» = M
ЯКЩО ε = NH **та** dε = QI **ТО** «частота» = M1, «скважність» = H

В результаті були отримані дві поверхні залежності «частоти» та «скважності» від «похибки» та «швидкості зміни похибки». На рис. 5.56 представлена поверхня залежність «частоти» від ε та dε.

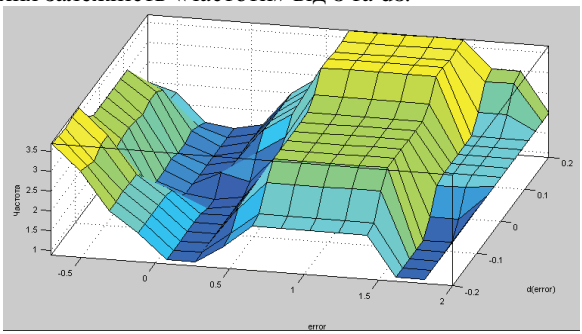


Рис. 5.56.

З рисунку бачимо, що поверхня дзеркальна відносно початку координат. Розглянемо поверхню залежності детальніше. Коли «похибка» лежить в районі нуля, «частота» лежить в районі нуля. При частоті зменшенні «похибки» в від'ємну область частота збільшується, але по-різному при різних значеннях «швидкості зміни похибки». В цій області при зміні dε від 0,2 до –0,2, «частота» збільшується. Область максимуму «частоти» знаходиться в області максимальних від'ємних значень ε та максимальних від'ємних значень dε. В області позитивних значень приблизно від 0,5 до 1,5 лежить область максимуму, в якій частота повільно спадає при зміні dε від 0,2 до –0,2. В області значень ε від 1,5 до 2 спостерігається область мінімумів, тому що «помилка» близька до того, щоб змінити свій характер (з ємкісного на індуктивний).

Розглянемо другу поверхню залежності «скважності» від ε та dε.

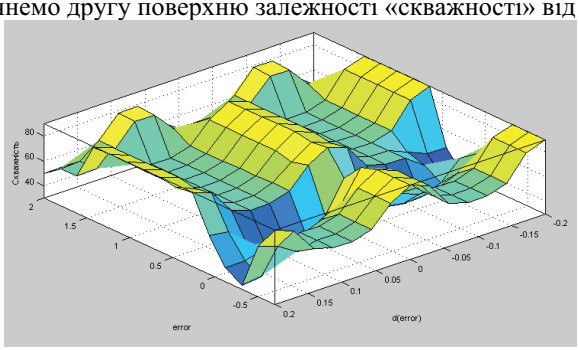


Рис. 5.57.

Бачимо, що поверхня залежності дзеркальна відносно початку координат. Область мінімумів спостерігається в районі ε=0. В районі dε=0 спостерігається область максимумів тому, що в цих умовах подаються імпульси низької частоти, щоб забезпечити кращу якість керування.

Matlab-модель представлена на рис. 5.58.

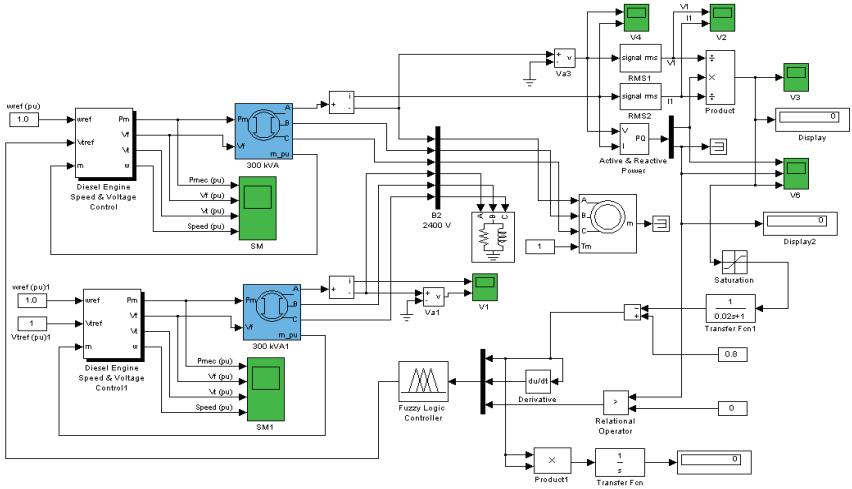


Рис. 5.58.

На рис. 5.59 представлені графіки зміни коефіцієнту потужності з різними типами регуляторів в системі стабілізації.

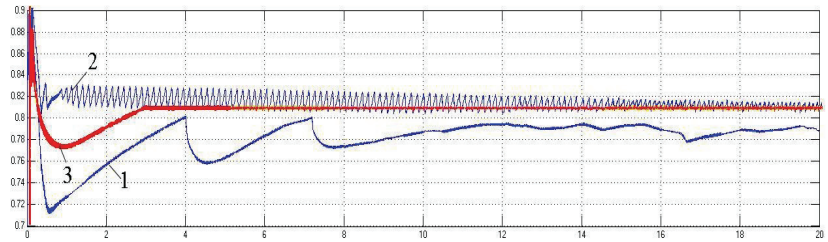


Рис. 5.59.

На рисунку позначено: 1 – трьохпозиційний регулятор, 2 – ПІД-регулятор, 3 – Fuzzy-регулятор.

Відносно невеликі постійні часу електричний ланцюгів в автономних електростанціях створюють передумови до збільшення швидкостей протікання електромагнітних перехідних процесів у генераторних установках. Значно менші інерційності дизель-генераторів призводять до великих швидкостей механічних перехідних процесів в енергетичних установках. Лінеа-

ризовані системи адекватні тільки в невеликій області відхилення від встановленого стану. В пікових та екстремальних ситуаціях, коли дизель-генератори працюють в режимах великих відхилень, значно проявляються їх нелінійні властивості і не врахування цих властивостей сприяє виникненню та розвитку системних аварій. Це значить, що для ефективного керування, яке забезпечує зберігання асимптотичної (динамічної) стійкості енергосистеми, необхідно розглядати нелінійні моделі енергооб'єктів і проводить синтез та проектування систем керування методами, які в найбільш повному ступені дозволяють врахувати явлення взаємопов'язаності та не лінійності процесів в енергооб'єктах.

Найефективнішим типом регулятора э регулятор на основі нечіткої логіки та ПІД-регулятор з оптимальними значеннями параметрів, що забезпечує найбільш швидкий перехідний процес з найменшим відхиленням cosφ від заданого значення.

5.9. Моделювання роботи споживачів електроенергії і силових агрегатів

Моделюванню електричних навантажень суднових електростанцій присвячені роботи [15; 16; 23], однак в них результатом моделювання є сумарний графік споживаної потужності СЕЕС, який використовується для визначення складу й потужностей генераторних агрегатів проектованої електростанції. Миттєве значення навантаження електростанції є випадковою величиною і визначається композицією миттєвих значень навантажень окремих споживачів. Графік споживання електроенергії кожного пристрою формується трьома складовими: рівнем потужності p, тривалістю роботи t і тривалістю паузи τ, які можуть бути постійними або статистично розподіленими в часі [16]. У роботах [15; 16; 23] прийнятий наступний алгоритм розрахунку навантажень, оформлений у вигляді пакета прикладних програм: моделювання значень рівня потужності, тривалостей роботи і пауз для кожного споживача; накладення складових на графік навантаження споживача; формування графіка сумарного навантаження на основі навантажень всіх промодельованих споживачів. Наведений алгоритм не дозволяє врахувати кореляційні зв'язки між режимами роботи споживачів. У цьому параграфі розглядаються особливості використання автоматного підходу для отримання графіків навантажень споживачів і роботи генераторних агрегатів, що дозволить формалізувати задачу моделювання та врахувати взаємозв'язки між споживачами.

Для кожного типу суден можна виділити характерні режими, що визначають діаграми роботи споживачів електроенергії. Крім того, в більшості режимів виділяються циклічні часові послідовності, тривалості етапів яких − випадкові величини, статистично розподілені в часі. Для більшості типів суден можна виділити групи споживачів, характеристики законів розподілу часів роботи, паузи і рівня потужності яких залежать від етапу циклу.

Наприклад, для промислових суден характерними режимами роботи є стоянка в порту, маневрування, перехід на промисел, хід з промислу, промисловий і аварійний режими. Для кожного режиму певного типу промислових суден можна виділити етапи добового циклу. В якості прикладу розглянуті судна ярусного лову в промисловому режимі, для якого характерними є етапи постановки, дрейфу і вибірки ярусу. Тривалості кожного з етапів – випадкові величини, розподілені за рівномірним законом. З етапами циклу лову пов'язані режими роботи споживачів електроенергії, що забезпечують видобуток риби (лебідки, промислові механізми, пошукове обладнання), її переробку (механізми рибцеху та морозильні камери) та зберігання (механізми рефрижераторної установки), − у кожному з них характеристики законів розподілу випадкових величин (рівня потужності, часів роботи і паузи) різні. Оскільки окремий споживач в кожен момент часу може знаходитися в одному з двох станів (включений або виключений), пропонується розглядати його як кінцевий автомат. Умовою переходу між станами є завершення випадкового проміжку часу, згенерованого на основі отриманих емпірично виду закону розподілу і його характеристик. Також у вигляді кінцевого автомата можна розглядати добовий цикл роботи судна: кожен етап відповідає окремому стану автомата, при цьому тривалість знаходження автомата в стані − випадкова величина, розподілена рівномірно в інтервалі, межі якого залежать від номера етапа. Як інструмент моделювання роботи споживачів використовується програма Stateflow середовища Matlab. Створена в ній мережа автоматів, описана далі, використовується як складова частина моделі електростанції і викликається на кожному кроці моделювання (одному кроку відповідає 5 хв. реального часу). На рис. 5.60 наведена Stateflow-діаграма кінцевого автомата S, що імітує зміну етапів технологічного циклу судна: стоянку, постановку, дрейф і вибірку ярусу (стани C0-C4 відповідно).

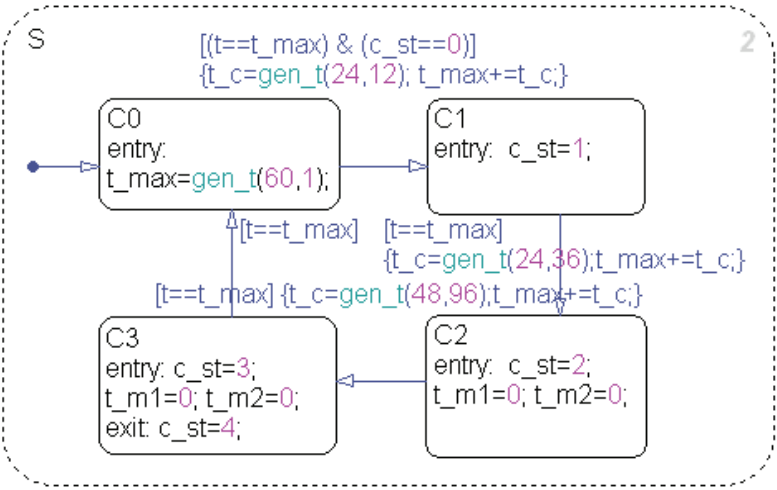
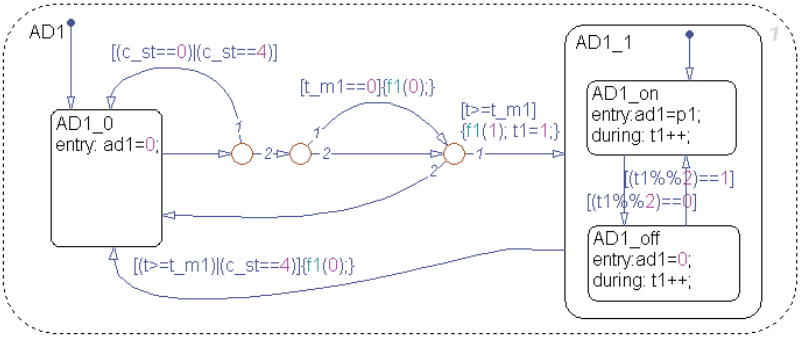


Рис. 5.60.

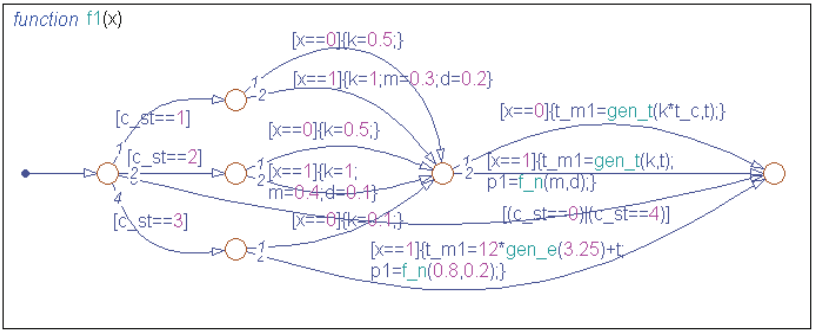
Зміна етапів добового циклу судна і робота кожного зі споживачів електроенергії є паралельними процесами, тому при створенні використана паралельна декомпозиція суперстанів, кожен з яких реалізує відповідний автомат.

На рис. 5.61, а наведена граф-схема автомата AD1, який імітує роботу залежного від етапу добового циклу судна споживача AD1. Стану AD1_0 відповідає вимкнений стан об'єкта, стану AD1_1 – увімкнений. Наведений споживач працює в повторно-короткочасному режимі (такий режим характерний для вантажопідйомних механізмів) з періодом 10 хв. і тривалістю включення двигуна 5 хв., тому стан AD1_1 розділений на два вкладених стани – включений і виключений під час роботи (AD1_on і AD1_off відповідно). Режим роботи споживача залежить від етапу технологічного циклу судна. На етапі стоянки (стан C0 автомата S) автомат AD1 знаходиться в стані AD1_0 (споживач виключений). При переході автомата S в стан C1, що відповідає переходу судна в режим постановки ярусу, генерується випадкове число, яке відповідає часу очікування включення в роботу споживачем. При завершенні згенерованого проміжку часу відбувається перехід автомата AD1 в стан AD1_1, що супроводжується генерацією тривалості перебування в ньому і рівня потужності, споживаної механізмом, у відносних одиницях. У стані AD1_1 відбувається циклічний перехід зі стану AD1_on в стан AD1_off і назад з періодом 2 такту, що імітує повторно-короткочасну роботу механізму. Збіг часу перебування автомата в стані AD1_1 зі згенерованою тривалістю ініціює перехід автомата в стан AD1_0, після чого описані дії повторюються. У режимах судна «дрейф» і «вибірка ярусу» (стани C2 і C3 автомата S) моделювання роботи споживача AD1 відбувається аналогічно; відрізняються лише види законів розподілу рівня потужності, інтервалів роботи і пауз, і їх характеристики. Для мінімізації громіздкості моделі генерація випадкових величин реалізована у вигляді графічної функції f1 (рис. 5.61, б) з параметрами − напрям переходу (з AD1_0 в AD1_1 або зворотній) і номер етапу циклу судна; повертані значення − час зворотного переходу в поточний стан і рівень потужності, споживаної механізмом.

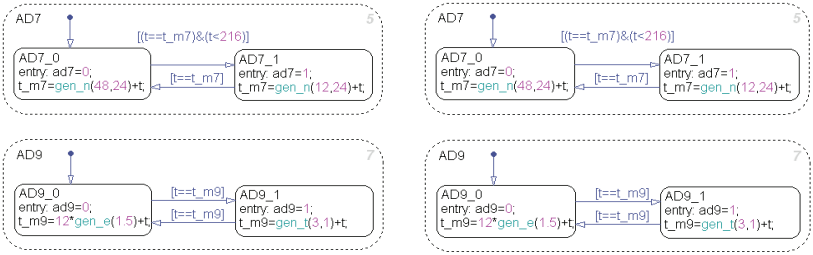
До другої групи споживачів відносяться споживачі, режим роботи яких не залежить від технологічного етапу циклу судна: механізми систем, що забезпечують роботу головного двигуна і рух судна, загальносуднових систем і систем зберігання вантажу (рис. 5.61, г). Режим роботи споживачів даної групи може залежати від часу доби (рис. 5.61, в). Аналогічно автоматам AD1, AD7 і AD9 проектуються автомати, які імітують роботу найбільш потужних і працюючих періодично або епізодично суднових споживачів електроенергії; споживана іншими пристроями потужність враховується у вигляді постійної складової. Сукупність паралельно об'єднаних автоматів утворюють Stateflow-модель роботи суднових споживачів електроенергії (рис. 5.62 та рис. 5.63).


AD1
[(c_st==0)|(c_st==4)]
AD1_0
entry: ad1=0;
[t_m1==0]{f1(0);}
[t>=t_m1]
{f1(1); t1=1;}
[(t>=t_m1)|(c_st==4)]{f1(0);}
AD1_1
AD1_on
entry:ad1=p1;
during: t1++;
[(t1%%2)==1]
[(t1%%2)==0]
AD1_off
entry:ad1=0;
during: t1++;

а


function f1(x)
[x==0]{k=0.5;}
[x==1]{k=1;m=0.3;d=0.2}
[c_st==1]
[x==0]{k=0.5;}
[x==0]{t_m1=gen_t(k*t_c,t);}
[c_st==2]
[x==1]{k=1;
m=0.4;d=0.1}
[x==1]{t_m1=gen_t(k,t);
p1=f_n(m,d);}
[c_st==3]
[x==0]{k=0.1;}
[(c_st==0)|(c_st==4)]
[x==1]{t_m1=12*gen_e(3.25)+t;
p1=f_n(0.8,0.2);}

б


AD7
[(t==t_m7)&(t<216)]
AD7_0
entry: ad7=0;
t_m7=gen_n(48,24)+t;
[t==t_m7]
AD7_1
entry: ad7=1;
t_m7=gen_n(12,24)+t;
AD9
AD9_0
entry: ad9=0;
t_m9=12*gen_e(1.5)+t;
[t==t_m9]
AD9_1
entry: ad9=1;
t_m9=gen_t(3,1)+t;

в г

Рис. 5.61.

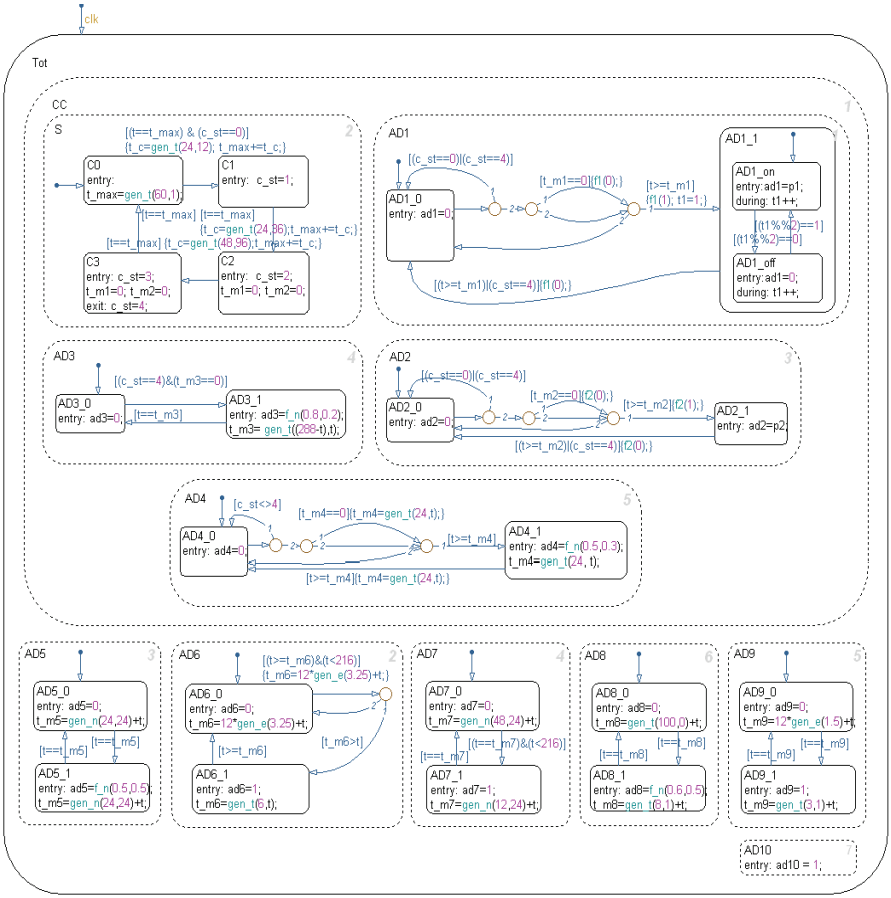


Рис. 5.62.

Вхідною подією для моделі є фронт тактового сигналу, який ініціює перевірку умов переходів та зміну станів автоматів. Вихідні сигнали − значення рівнів потужностей, споживаних пристроями, в частках від номінальних потужностей (у роботі розглядається 9 споживачів зі змінним навантаженням і один, узагальнений, з постійним).

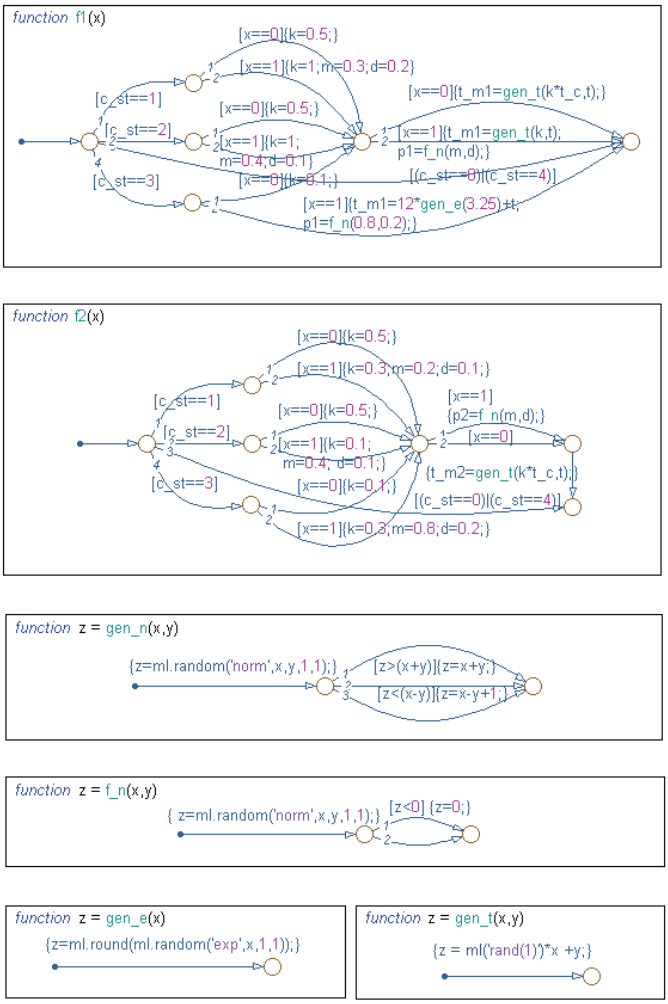
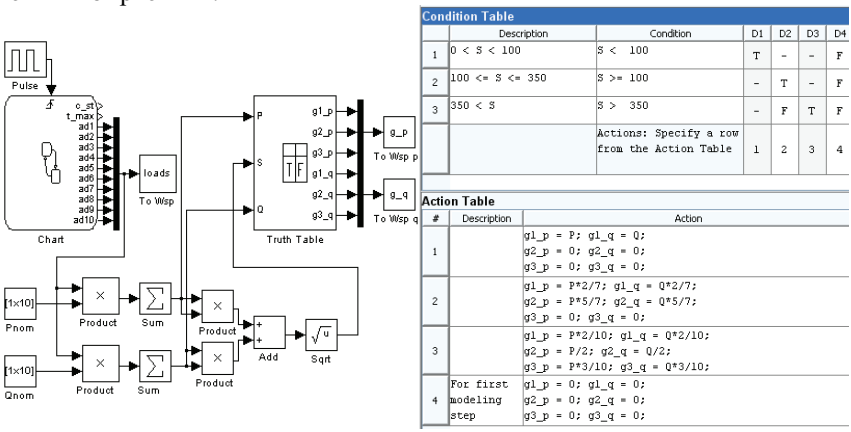


Рис. 5.63.

Для отримання діаграм роботи генераторних агрегатів Stateflow-модель використовується у складі Simulink-моделі, наведеної на рис. 5.64, а.

Вихідні сигнали моделі роботи споживачів електроенергії об’єднуються в вектор, і далі для переходу до абсолютних одиниць потужностей виконується його поелементне множення на вектора, що складаються з значень номінальних активних (Pnom) і реактивних (Qnom) потужностей споживачів.

На основі результуючих векторів обчислюються сумарні споживані потужності. Розрахунок потужностей, що віддаються кожним генераторним агрегатом (ГА) на загальне навантаження, виконується за допомогою блоку вибору варіанта (Таблиця Істинності в середовищі Simulink; рис. 5.64, б). У роботі розглядається електроенергетична установка, в якій в якості джерел електроенергії використовуються три ГА з різними номінальними потужностями (100 кВт, 250 кВт і 150 кВт). На вхід Таблиці Істинності надходять сумарні значення споживаних потужностей; рішення про включення на паралельну роботу ГА приймається на основі аналізу значення повної потужності. Якщо останнє менш 100 кВА (0,8 номінальної потужності першого ГА), працює тільки один ГА; якщо перебуває в межах 100-350 кВА, працює два ГА і якщо понад 350 кВА − всі три ГА. При паралельній роботі декількох ГА моделюється розподіл активного і реактивного навантажень пропорційно номінальним потужностям ГА. Вихідними параметрами Таблиці Істинності є значення активних і реактивних потужностей, що генеруються кожним з трьох ГА.



Condition Table

	Description	Condition	D1	D2	D3	D4
1	0 < S < 100	S < 100	T	-	-	F
2	100 <= S <= 350	S >= 100	-	T	-	F
3	350 < S	S > 350	-	F	T	F
		Actions: Specify a row from the Action Table	1	2	3	4

Action Table

#	Description	Action

1		g1_p = P; g1_q = Q;
		g2_p = 0; g2_q = 0;
		g3_p = 0; g3_q = 0;

2		g1_p = P*2/7; g1_q = Q*2/7;
		g2_p = P*5/7; g2_q = Q*5/7;
		g3_p = 0; g3_q = 0;

3		g1_p = P*2/10; g1_q = Q*2/10;
		g2_p = P/2; g2_q = Q/2;
		g3_p = P*3/10; g3_q = Q*3/10;

4	For first modeling step	g1_p = 0; g1_q = 0;
		g2_p = 0; g2_q = 0;
		g3_p = 0; g3_q = 0;

а б

Рис. 5.64.

Для запису отриманих в результаті моделювання діаграм роботи споживачів і генераторів в базу даних матриці значень рівнів споживаних і величин генерованих потужностей передаються в робочий простір Matlab. У роботі використовується СУБД PostgreSQL, для зв'язку з якою застосовується драйвер PostgreSQL JDBC. На основі даних з робочого простору Matlab формуються SQL-запити на заповнення таблиць (таблиці діаграм навантажень і таблиці режимів роботи ГА); підключення до бази даних і виконання запитів реалізовано на мові Java за допомогою класів, що надаються драйвером PostgreSQL JDBC.

РОЗДІЛ 6. ВИКОРИСТАННЯ UML ПРИ ВИРІШЕННІ ЗАВДАНЬ МОДЕЛЮВАННЯ АВТОМАТИЗОВАНИХ СИСТЕМ КЕРУВАННЯ

6.1. Ієрархічна модель автоматизованої системи керування судновою електроенергетичною системою

Системи керування СЕЕС будуються по ієрархічному принципу. У випадку відмови елементів верхнього рівня забезпечується працездатність СЕЕС за допомогою елементів нижніх рівнів системи.

На нижньому (нульовому) рівні ієрархії перебувають керовані елементи СЕЕС із датчиками й виконавчими органами. Цей рівень призначений для керування виконавчими пристроями. Тут же реалізовані різні види блокувань. При виникненні несправностей інформація передається на вищестоящий рівень.

Перший рівень ієрархії представляє собою сукупність локальних засобів автоматизації, що забезпечують стабілізацію напруги й частоти генераторів, їхній захист від перевантаження й к.з., від зворотної потужності. Вони дозволяють формувати керуючі впливи з метою збереження працездатності СЕЕС при відмові її елементів.

Другий рівень ієрархії представляє собою рівень локальних систем керування, включаючи системи керування генераторними агрегатами, на якому виконуються наступні функції: керування приводними двигунами генераторів за сигналом від автоматичного пристрою або оператора, контроль і захист генераторів, синхронізація з наступним розподілом навантаження, регулювання частоти. Також на цьому рівні перебувають пристрої місцевого керування генераторними агрегатами.

Третій рівень ієрархії представляє собою рівень систем керування функціональними комплексами. На цьому рівні виконуються наступні функції: контроль навантаження працюючих генераторів, контроль резерву потужності для забезпечення пуску потужних споживачів енергії, вибір послідовності пуску резервних генераторних агрегатів, контроль технічного стану СЕЕС, захист від струмів короткого замикання.

Четвертий рівень ієрархії – рівень комплексної системи керування технічними засобами, на якому забезпечується керування електростанцією у цілому оператором. На цьому рівні виконується завдання режимів використання СЕЕС залежно від режиму експлуатації, від якого залежить склад елементів СЕЕС.

Час, необхідний для виконання операцій керування, на різних рівнях ієрархії неоднаковий. Найбільш швидко виконуються операції 1-го рівня за допомогою систем автоматичного регулювання, контролю й захисту окремих елементів. Операції 2-го рівня вимагають більше тривалого часу, що визначається видом процесу керування приводними двигунами, порушенням генераторів. На цьому рівні є ряд програмних автоматів, які видають сигнал у систему керування СЕЕС про їхню готовність. Операції 3-го рівня вимагають швидкості керуючого пристрою. При виконанні операцій зміни

структури СЕЕС потрібен проміжок часу, необхідний для виконання операцій 2-м рівнем. Час для виконання операцій 4-го рівня визначається припустимими швидкостями зміни режимів експлуатації енергетичної системи.

Побудова складних систем керування повинна виконуватися із застосуванням досить простих структур, які також підтримують прості в реалізації програмно-апаратні засоби. Це стає можливим на основі побудови багаторівневої апаратної системи, що має у своєму складі обчислювальні прилади й модулі, спроектовані з використанням контролерів.

Мережа передачі даних може представляти собою локальну або загальну обчислювальну мережу, організовану по магістральному, радіальному, комбінованому або іншому принципу. Інтерфейс між вузлами системи на різних ієрархічних рівнях може бути різним. На рис. 6.1 представлена структура електроенергетичної системи, що складається із трьох дизель-генераторів.

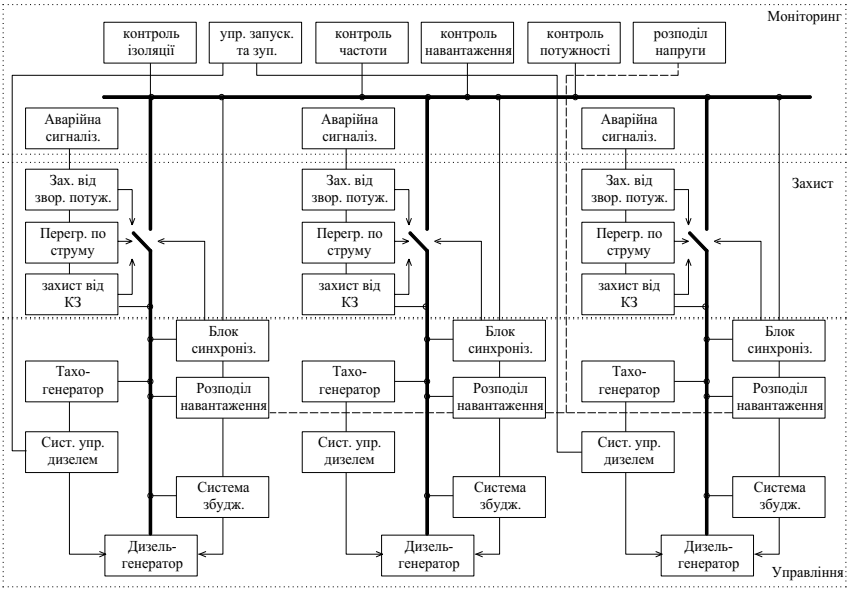


Рис. 6.1.

Система керування представляє собою мережу, вузли якої зв’язані каналами передачі даних. Вибір топології з’єднань вузлів мережі визначається структурою СЕЕС, вимогами надійності, припустимим часом очікування пакета даних від одного рівня до іншого. У загальному випадку, конфігурація взаємозв’язків мережі може залишатися незмінною в процесі експлуатації (статична), або ж оперативно змінюватися за допомогою програмних засобів (динамічна).

Використання динамічної топології дозволяє створити резервні шляхи, завантаженість яких низька, і використати ці шляхи для передачі команд керування. У мережах з комутацією з'єднань формується канал від вузла-джерела до вузла-приймача, що зберігається поки весь пакет, що доставляється, не досягне пункту призначення. Час існування такого каналу визначається довжиною пакета. З іншого боку, у мережі може використовуватися стратегія комутації пакетів. У цьому випадку передбачається, що повідомлення самостійно знаходить свій шлях до місця призначення, який щоразу може бути іншим. Пакет послідовно проходить через вузли мережі. Кожний вузол запам'ятовує пакет у своєму буфері, аналізує його й ухвалює рішення щодо подальших дій. Залежно від завантаженості мережі приймається рішення про можливості негайного пересилання пакета до наступного вузла й про подальший маршрут проходження пакета на шляху до мети. Якщо всі можливі канали зайняті, то в буфері вузла формується черга пакетів, що зменшується в міру звільнення ліній зв'язку між вузлами. Таке явище неприйнятно для системи керування, оскільки час очікування передачі пакета визначається завантаженістю каналів передачі, що у свою чергу визначається режимом роботи СЕЕС. Крім того, якщо черга пакетів перевищує розміри буфера, відбувається втрата даних, що також неприйнятно в умовах аварійної ситуації. Рішенням проблеми може служити збільшення кількості каналів зв'язку. При цьому пропорційно зменшується навантаження на кожний з них. Крім того, можна створити окремий канал зв'язку, що не використовується при нормальному режимі роботи СЕЕС. При виникненні аварійної ситуації резервний канал зв'язку дозволить з максимальною швидкістю передати відповідні команди вузлам системи. Кожний з вузлів мережі може бути як термінальним, так і комутатором (але не одночасно). Між термінальними вузлами повідомлення відправляються напряму. Збільшення числа комутаторів приводить до збільшення затримки. Оскільки система керування виконує також задачі прогнозування, навіть при відсутності аварійного режиму, при якому може бути використаний резервний канал, затримка передачі команди повинна бути мінімальною.

Основні функції системи керування полягають у наступному:

– автоматичний запуск і зупинка дизель-генераторних агрегатів залежно від режиму роботи електростанції;

– контроль за станом дизельних агрегатів;

– автоматична точна синхронізація генераторів і вивід з паралельної роботи;

– розподіл активної й реактивної потужностей;

– автоматичний пуск/зупинка дизель-генераторів залежно від споживаної потужності й наявності необхідного резерву;

– контроль за збільшенням споживаної потужності;

– захист від коротких замикань;

– забезпечення резерву потужності для пуску потужних споживачів;

– контроль частоти напруги;

– робота з певної програми залежно від режиму роботи;

– відбір енергії від валогенераторів.

Функція запуску й зупинки дизель-генераторів може виконуватися автоматично або вручну. Автоматична синхронізація виконується спеціальною системою синхронізації, що контролює параметри працюючого генератора й генератора, який підключають, і формує сигнал на замикання автоматичного вимикача. Час від моменту появи сигналу на запуск дизель-генератора до моменту підключення його на паралельну роботу звичайно не перевищує 45 секунд.

Система контролю за резервом потужності забезпечує запуск і зупинку дизель-генераторів залежно від навантаження на електростанцію. У випадку, якщо відсутній резерв потужності для запуску потужного споживача, а також якщо навантаження на генератори перевищує їхнє номінальне значення або струм генератора вище заданої межі протягом тривалого проміжку часу, виконується запуск дизель-генератора.

Якщо ж енергії, яка генерується, цілком достатньо для включення потужного навантаження, навантаження генераторів нижче деякого значення, то виконується автоматичний вивід генератора з паралельної роботи й зупинка дизеля.

У випадку зникнення напруги на шинах ГРЩ виконується автоматичний пуск всіх доступних дизель-генераторних агрегатів. До шин ГРЩ підключається генератор, що першим запустився, і на шинах якого з'явилася напруга. Потім по черзі підключаються на паралельну роботу інші генератори в міру їхньої готовності.

Перспективними напрямками в розвитку інформаційного забезпечення автономних електроенергетичних систем можна вважати:

– побудову систем моніторингу за станом об'єктів суднової електроенергетики з метою вибору оптимальних режимів роботи;

– використання систем контролю й забезпечення живучості суден;

– використання систем з оптичними каналами зв'язку, що забезпечують високу пропускну здатність;

– використання систем бездротової передачі даних;

– використання стандартних протоколів передачі даних і відкритих архітектур.

Система керування СЕЕС розглядається як сукупність мікропроцесорних систем, розосереджених по виробничих приміщеннях і об'єднаних по магістральній, кільцевій або іншій ознаці, яка забезпечує централізацію контролю (збір, обробку, подання інформації персоналу на постах керування) про технічний стан устаткування і якість протікання технологічних процесів, керування якими повинне забезпечуватися автоматично й дистанційно. Таким чином, по топології й організації керування система є розподіленою.

Структура ліній комутації між вузлами обчислювальної системи (топологія мережі передачі даних) визначається, як правило, з урахуванням

можливостей ефективної технічної реалізації. Важливу роль також при виборі структури мережі грає й аналіз інтенсивності інформаційних потоків при рішенні завдань керування.

До числа типових топологій зазвичай відносять наступні схеми комунікації [8; 15]:

– повний граф (completely–connected graph або clique) – система, у якій між будь–якою парою процесорів існує пряма лінія зв'язку. Така топологія забезпечує мінімальні витрати при передачі даних, однак складно реалізується при великій кількості процесорів;

– лінійка (linear array або farm) – система, у якій всі процесори перенумеровані по порядку й кожний процесор, крім першого і останнього, має лінії зв'язку тільки із двома сусідніми (з попереднім і наступним) процесорами. Така схема, з одного боку, просто реалізується, з іншого боку, відповідає структурі передачі даних при рішенні багатьох обчислювальних задач (наприклад, при організації конвеєрних обчислень);

– кільце (ring) – дана топологія отримується із лінійки процесорів з'єднанням першого й останнього процесорів лінійки;

– зірка (star) – система, у якій всі процесори мають лінії зв'язку з деяким керуючим процесором. Дана топологія є ефективною, наприклад, при організації централізованих схем паралельних обчислень;

– гратки (mesh) – система, у якій граф ліній зв'язку утворює прямокутну сітку (звичайно двох–або тривимірну). Подібна топологія може бути досить просто реалізована й, крім того, ефективно використана при паралельному виконанні багатьох чисельних алгоритмів (наприклад, при реалізації методів аналізу математичних моделей, описуваних диференціальними рівняннями в частинних похідних);

– гіперкуб (hypercube) – дана топологія являє собою окремий випадок структури гратки, коли по кожній розмірності сітки є тільки два процесори (тобто гіперкуб містить $2N$ процесорів при розмірності N). Такий варіант організації мережі передачі даних досить широко розповсюджений на практиці.

В якості основних характеристик топології мережі передачі даних найбільш широко використовується наступний ряд показників:

діаметр – показник, що визначається як максимальна відстань між двома процесорами мережі (під відстанню звичайно розуміється величина найкоротшого шляху між процесорами). Ця величина може характеризувати максимально необхідний час для передачі даних між вузлами, оскільки час передачі зазвичай прямо пропорційний довжині шляху;

з'вязність (connectivity) – показник, що характеризує наявність різних маршрутів передачі даних між процесорами мережі. Конкретний вид даного показника може бути визначений, наприклад, як мінімальна кількість дуг, яку треба видалити для поділу мережі передачі даних на дві незв'язні області;

ширина бінарного розподілу (bisection width) – показник, обумовлений як мінімальна кількість дуг, яку треба видалити для поділу мережі передачі даних на дві незв'язні області однакового розміру;

вартість – показник, що може бути визначений, наприклад, як загальна кількість ліній передачі даних системі.

Для порівняння в табл. 6.1 приводяться значення перерахованих показників для різних топологій мережі передачі даних.

Таблиця 6.1.

Топологія	Діаметр	Ширина	Зв'язність бісекції	Вартість
Повний граф	1	$P^2/4$	p–1	$p(p–1)/2$
Зірка	2	1	1	p–1
Повне двійкове дерево	2 $\log((p+1)/2)$	1	1	p–1
Лінійка	p–1	1	1	p–1
Кільце	$[p/2]$	2	2	p
Гратка N=2	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
Гратка–тор N=2	$2[\sqrt{p}/2]$	$2\sqrt{p}/2$	4	$2p$
Гіперкуб	$\log p$	$p/2$	$\log p$	$(p \log p)/2$

В таблиці позначено p – кількість вузлів.

Оскільки система керування СЕЕС будується по ієрархічному принципу, одним з варіантів топології може бути використана деревоподібна топологія [53], що представлена на рис. 6.2.

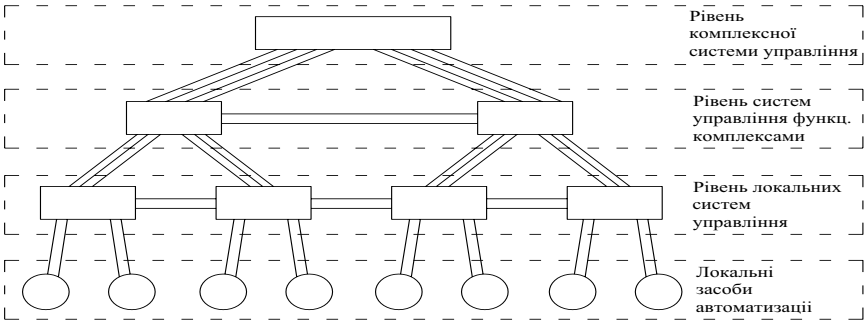


Рис. 6.2.

Стандартне двійкове дерево має той недолік, що в міру просування від нижніх рівнів до верхнього значно збільшується навантаження на інформаційні канали [53], що призводить до збільшення часу очікування. Повідомлення повинне проходити через один або кілька проміжних ланок. На більш високих рівнях мережі ймовірність затору через недостатню пропускну здатність лінії вища. Тому доцільніше використати «товсте дерево». Ідея товстого дерева полягає в збільшенні пропускної здатності комутаційних ліній

на прикореневих рівнях мережі. Із цією метою на верхніх рівнях мережі батьківські й дочірні вузли зв'язують не одним, а декількома каналами, причому, чим рівень вище, тим більше каналів. Розмір мережі визначається кількістю вузлів мережі й залежить від рівня автоматизації енергосистеми. Діаметр мережі – мінімальний шлях, по якому проходить повідомлення між двома найбільш віддаленими один від одного вузлами, визначається виразом:

$$D_1=2(h–1),$$

де h – висота дерева.

На рис. 6.2 представлена топологія мережі «товсте дерево», для якого діаметр дорівнює:

$$D_2=2(h–1)–1.$$

Але це окремий випадок. Це обумовлено введенням додаткових зв'язків між вузлами одного рівня. Однак, при збільшенні дерева завширшки, тобто збільшення кількості вузлів на кожному з рівнів, діаметр мережі визначається по виразу D_1. Порядок вузла – сума числа вхідних і вихідних каналів вузла, у розглянутій топології змінюється від 1 до 5 при переході від нижнього рівня до верхніх рівнів. Підвищення порядку вузлів дозволяє забезпечити менший діаметр мережі. Однак, оскільки порядок вузлів міняється, це ускладнює реалізацію системи по модульному принципу. Введення додаткових з'єднань між вузлами одного рівня дозволило збільшити зв'язність мережі. Цей параметр можна визначити як число вузлів або ліній зв'язку, які повинні вийти з ладу, щоб мережа розпалася на дві непересічні мережі. Так, для топології «класичного товстого дерева» цей параметр дорівнює одному, якщо розглядати нижні рівні, і двом при розгляді верхніх рівнів. При введенні додаткових зв'язків зв'язність мережі збільшується до трьох. Затримка передачі повідомлення залежить від маршруту, по якому воно буде передаватися.

На рис. 6.2 це відображено у вигляді множинних ліній між вузлами верхніх рівнів. Нижній рівень містить сукупність локальних засобів автоматизації, які управляють об'єктом. При відмові якого-небудь із вузлів цього рівня, виконується автоматичне включення резерву.

6.2. Статична UML – модель системи керування

В існуючий час росте інтерес до нових технологій проектування та до архітектур розподілених систем керування (СК) для автоматизації виробництва. Важливим моментом у процесі проектування СК є реалізація гнучкості й можливості швидкої реконфігурації. При цьому засоби проектування й моделювання систем керування повинні мати можливість швидкої модифікації проекту для адаптації до конкретного об'єкта керування [54].

Для правильного рішення питання керування технічним об'єктом важливе знання процесів, які відбуваються в об'єкті, і методології вивчення процесу керування, основою якого є системний підхід [11]. Для кожного

елемента вказується безліч можливих режимів роботи. Сукупність режимів всіх елементів визначає режими роботи об'єкта.

Уніфікація й стандартизація інтерфейсів дозволили перейти до побудови систем, заснованих на просторовому й функціональному розподілі складових частин, а також розподілі їх по керуванню. Заміна одного мікропроцесора (МП) декількома дає можливість розосередити їх по всьому судну. На кожний із МП можна покласти певні функції, що дозволить програмісту модифікувати робочу програму, що обслуговує окрему підсистему, без будь-якого втручання в робочі програми інших підсистем.

Використання уніфікованої мови моделювання UML при розробці розподілених систем керування, якою і є АСК СЕЕС, дозволяє значно полегшити й прискорити процес проектування шляхом реалізації загальних функціональних блоків з їх подальшим повторним використанням і можливістю вдосконалення.

При розробці розподілених систем керування повинні братися до уваги такі положення як створення програмного коду керуючого додатка, розбивка коду на функціональні модулі з подальшим розташуванням у керуючих пристроях і вибір топології системи керування й реалізація зв'язку між керуючими блоками [15; 16; 23].

Для кожної генераторної секції розробляються схеми керування з урахуванням прийнятого обсягу автоматизації кожного генераторного агрегату. На рис. 6.3 представлена UML – діаграма однієї генераторної секції й зв'язку між окремими функціональними пристроями.

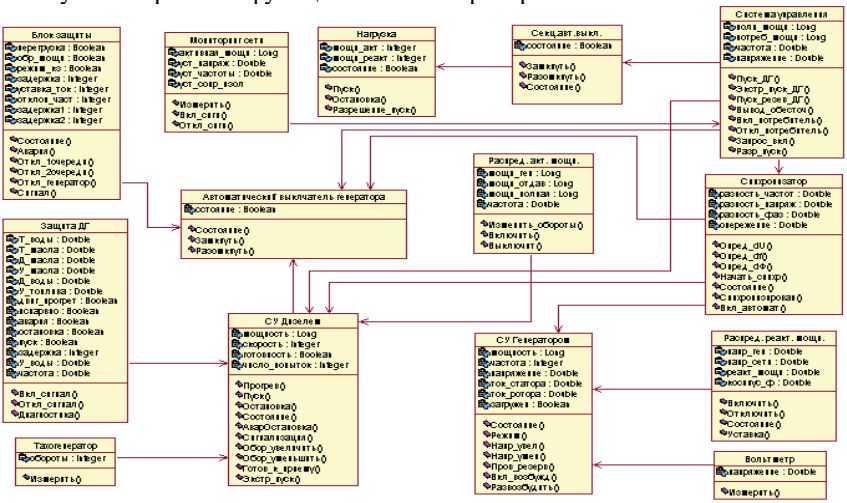


Рис. 6.3.

Модель на рис. 6.3 містить систему керування дизель-генератором (ДГ), блок захисту ДГ, пристрою розподілу активної/реактивної потужності,

синхронізатор, блоки захисту й моніторингу мережі, а також деякі вимірювальні прилади. Кожен із пристроїв представлений у вигляді класу, що має набір функцій, за допомогою яких описуються алгоритми роботи пристроїв і набір атрибутів, що визначають поточний стан кожного із пристроїв. Залежно від розв'язуваних конкретною системою керування задач, набір функцій кожного із пристроїв може мінятися.

Система керування дизельним агрегатом виконує наступні функції: підтримка ДГ у прогрітому стані (Прогрів()); пуск ДГ із холодного або прогрітого стану й вивід його на номінальну частоту обертання з видачею сигналу в систему керування про готовність до прийому навантаження (Екстр_пуск(), Пуск(), Готовий_до_прийому); захист ДГ при відхиленні параметрів (тиску масла й води, температури масла й води, частоти обертання) з видачею сигналу розшифровки причини аварії (Діагностика(), Вкл_сигнал(), Сигналізація()); видачу попереджувальних сигналів про стан параметрів ДГ (Стан()); зупинку ДГ по спеціальній програмі або екстрену (аварійну) зупинку (Зупинка(), Авар зупинка()). Кожний клас на діаграмі містить атрибути, з якими оперує система керування. Для дизеля це такі параметри як потужність, номінальна частота, стан (готовність до підключення навантаження), число спроб запуску. При роботі системи керування значення цих параметрів постійно обновляються, і залежно від їхніх значень відбувається зміна режимів роботи ДГ (стабілізація частоти обертання, перевірка наявності резерву потужності, діагностика).

Після того, як розроблена статична модель системи керування, виконується опис алгоритмів роботи системи керування СЕЕС і окремих пристроїв автоматизації при різних режимах роботи. Повнота й точність опису алгоритмів дозволить виконати функціональну розбивку завдань по керуванню електроенергетичною системою й визначитися з кількістю мікропроцесорів, необхідних для рішення завдання керування із заданою якістю й надійністю. У свою чергу це дозволить описати типи зв'язків між окремими керуючими модулями, визначити обсяг переданої інформації.

До складу програмного забезпечення повинні входити засоби для автоматизації розробки програм і мікропрограм, і їхню перевірку. Цим вимогам задовольняє пакет Rational Rose [55].

Систему алгоритмів керування СЕЕС будують за ієрархічним принципом, що забезпечує зручність розробки, рішення завдань програмування й налагодження її елементів. У ній можуть бути виділені структурна, функціональна й організаційна ієрархії.

На рис. 6.4 представлена діаграма послідовності, що описує послідовність включення потужного споживача електроенергії. З появою заявки на включення якого-небудь потужного споживача формується сигнал, що імітує потужність, що потрібна і системою керування визначається достатність наявного резерву потужності, що генерується. Це завдання система керування виконує шляхом звертання до системи керування дизель-генератором

(команда Пров_резерв()). У систему керування повертається значення доступної потужності (атрибут «завантаження»). Якщо це значення перевищує потужність споживача, що підключається, то видається сигнал на включення автоматичного вимикача. Якщо ж резерв потужності відсутній, система керування передає команду пуску резервного дизель-генератора. Коли оберти дизеля досягнуть номінального значення, подається сигнал на включення збудження генератора. Далі відбувається опитування системи керування дизелем на предмет готовності до прийняття навантаження. Виконується синхронізація генератора з мережею й після того, як синхронізація виконана, подається команда на дозвіл включення споживача. Діаграма послідовності показує взаємодію між окремими блоками системи керування.

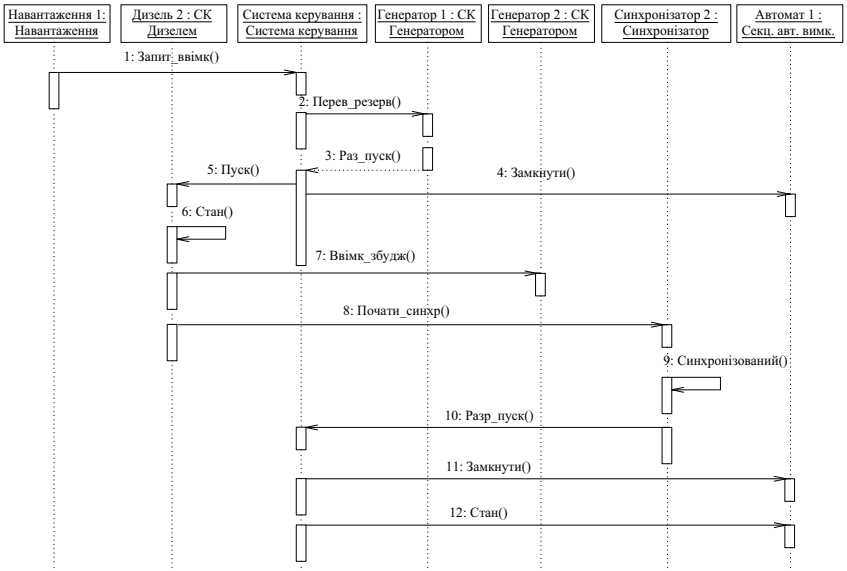


Рис. 6.4.

Організаційна ієрархія алгоритмів керування СЕЕС припускає наявність у системі елементів з різними пріоритетами дії. Причому є елементи рівних пріоритетів і елементи, що приймають рішення, які розташовуються ієрархічно. Дана ієрархія дозволяє здійснити класифікацію елементів системи алгоритмів по пріоритетах, забезпечити їхню взаємодію й реалізувати режим мультипрограмування. Реалізація режиму поділу часу й виконання програм у реальному масштабі часу забезпечуються відповідним вибором тимчасових характеристик функціональних елементів системи алгоритмів на підставі аналізу функціональних особливостей алгоритмів і програм, вивчення перехідних процесів у СЕЕС.

6.3. Розробка UML – моделі інформаційних потоків

В основі побудови динамічної моделі лежать раніше розроблені прецеденти (події). У СЕЕС до таких прецедентів відносяться:

– точна автоматична синхронізація генераторів;

– автоматичний розподіл активного навантаження між паралельно працюючими генераторами;

– автоматичне розвантаження генераторів, включення резервного генератора;

– включення потужних споживачів електроенергії;

– екстрений пуск дизель-генераторів і виведення судна зі знеструмленого стану;

– контроль параметрів працюючого дизель-генератора, захисту по частоті обертання;

– контроль і діагностування електричної мережі;

– звукова й світлова сигналізація й ін.

Для кожного прецеденту треба визначити об'єкти, що беруть участь у ньому, і те, як вони взаємодіють у цьому прецеденті. Ці взаємодії відображаються на діаграмі кооперації.

Розглянемо діаграму взаємодії при екстреному пуску резервних ДГ і виведенні судна зі знеструмленого стану ДГ (рис. 6.5). Екстрений пуск резервних необхідний для забезпечення електропостачання судна в аварійних ситуаціях. Команда на екстрений пуск формується при наявності однієї з наступних умов: напруга на шинах ГРЩ дорівнює нулю або є заявка на включення потужного споживача електроенергії, що викликає дефіцит генеруємої потужності. Сигнал екстреного пуску блокує затримку його пуску й датчик попереднього прокачування масла (цим скорочується час пуску ДГ).

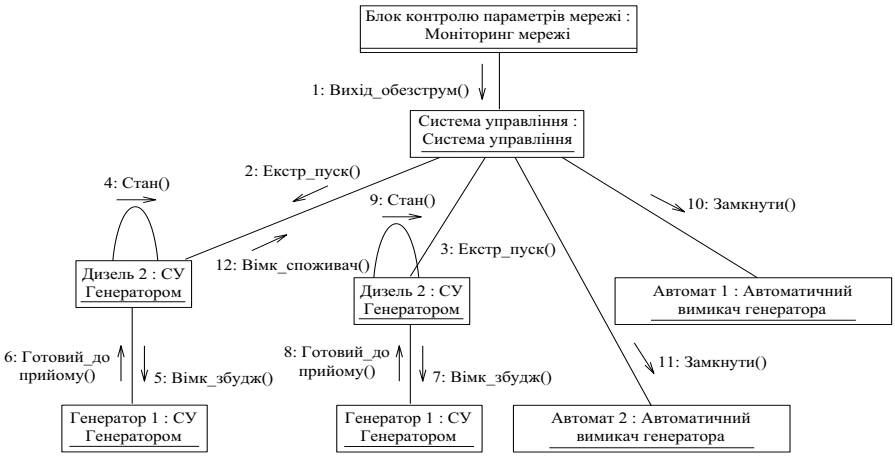


Рис. 6.5.

Знеструмлення СЕЕС оцінюється її системою керування по зникненню напруги на шинах ГРЩ або відключенню всіх автоматичних вимикачів генераторів. Ці функції виконує блок контролю параметрів мережі. При знеструмленні система керування електроенергетичною установкою забезпечує екстрений пуск мінімум двох резервних ДГ. Перший ДГ, що досяг номінальної частоти обертання, після подачі порушення на генератор, одержує команду включення на шини ГРЩ. Якщо він за якимись причинами не включиться, він буде блокований і наступний ДГ одержить аналогічну команду. Пуск декількох ДГ передбачається для запобігання невдалого пуску одного з них в умовах знеструмлення судна й надійного забезпечення швидко зростаючого навантаження після знеструмлення.

Сигнал на включення автоматичного вимикача резервного ДГ формується при виконанні наступних умов: відсутнє блокування ДГ (немає аварії, частота перевищує мінімально припустиму межу, немає команди на включення автоматичних вимикачів інших генераторів). При цьому можливо підключення тільки один резервного ДГ, що першим вийшов на задану частоту обертання. Сигнал на включення автоматичного вимикача подається протягом певного часу, після закінчення якого його включення блокується. Якщо напруги на шинах ГРЩ немає, то надається можливість для підключення наступного із тих, що запустилися ДГ.

При навантаженні працюючого ГА близького до 80% його номінальній потужності, можливі включення споживачів електроенергії, потужність яких перевищує 20% резерву потужності на шинах ГРЩ. Підключення таких споживачів може викликати перевантаження працюючого ГА, його відключення й знеструмлення судна. Надійне включення в мережу потужних споживачів електроенергії забезпечується системою керування СЕЕУ. Оцінюється потужність навантаження, яке вмикається, і якщо потужність електростанції недостатня, пуск цього навантаження блокується доти, поки резервний ДГ не буде пущений і синхронізований із шинами ГРЩ. Попередня оцінка резерву генеруючої потужності дозволить виконати запуск дизель-генератора й зробити його синхронізацію з мережею для виключення можливості знеструмлення судна. На рис. 6.6 представлена діаграма, що описує взаємодію об’єктів при відсутності резерву потужності.

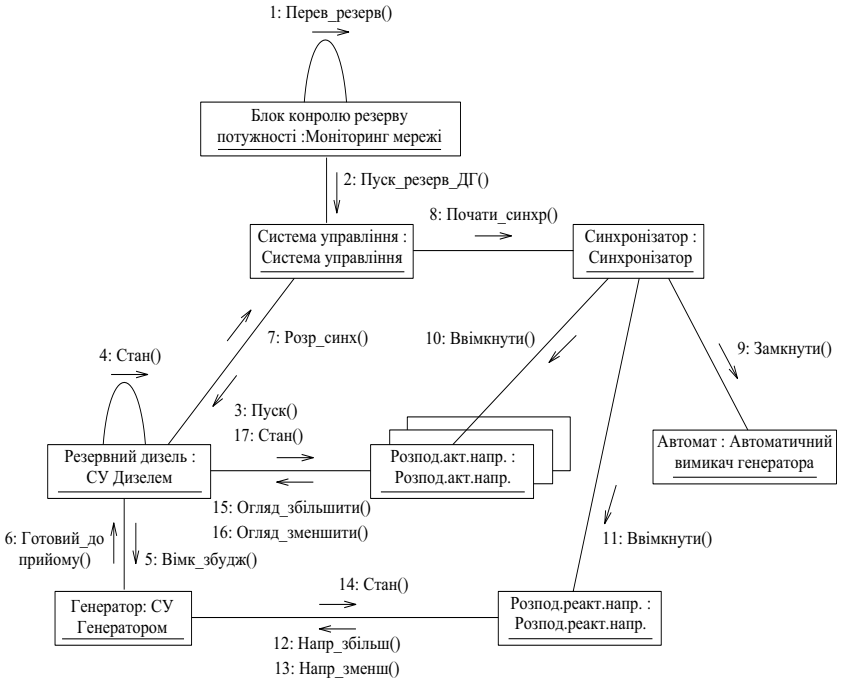


Рис. 6.6.

Систему керування можна розділити на кілька підсистем. Динамічні взаємодії між підсистемами можна зображати на діаграмі кооперації підсистем, що являє собою високорівневу консолідовану діаграму кооперації.

Консолідована діаграма взаємодії, яка показана на рис. 6.7, синтезується шляхом накладення одна на одну діаграм кооперацій для окремих прецедентів. У кожному випадку в консолідовану діаграму включаються нові об’єкти й повідомлення, тому вона постійно ускладнюється. Деякі об’єкти або повідомлення можуть бути присутніми декілька разів, але зображуються на консолідованій діаграмі тільки один раз. Розробка консолідованої діаграми кооперацій для кожної підсистеми й високорівневої діаграми кооперації підсистем дозволяє оцінити ступінь зв’язку між об’єктами у всій ієрархії системи керування. На консолідованій діаграмі взаємодії, що покликана дати повну картину всіх комунікацій, повинні бути представлені всі повідомлення. У такому випадку відображуваної інформації на діаграмі виявляється більше. Якщо один об’єкт посилає іншому кілька окремих повідомлень, то вони показуються не окремо, а в складі агрегованого повідомлення.

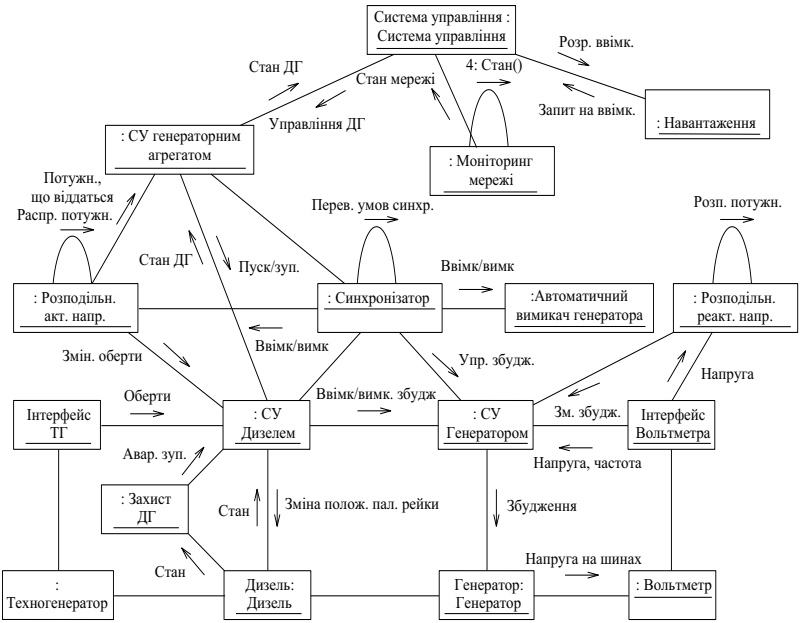


Рис. 6.7.

Мета динамічного аналізу полягає у визначенні взаємодії між керуючим об'єктом, об'єктами, які посилають повідомлення керуючому об'єкту, а також іншими об'єктами, які також беруть участь у прецеденті. Взаємодії можуть відбуватися між об'єктами як одного рівня ієрархії, так і різних рівнів.

Динамічний аналіз дозволяє розібратися в тому, як об'єкти взаємодіють між собою в різних прецедентах. Динамічний аналіз виконується для кожного прецеденту. Він починається з розгляду зовнішньої події, що надходить в об'єкт, після чого простежується весь ланцюжок внутрішніх подій. При виникненні внутрішніх подій відбувається обмін повідомленнями між об'єктами-учасниками прецеденту. Використання уніфікованої мови моделювання дозволяє виконати аналіз всіх повідомлень, якими обмінюються об'єкти в процесі взаємодії.

6.4. Використання UML-FB в проектуванні та моделюванні систем керування автономними електростанціями

Існуючі технології проектування й побудови складних інформаційних систем технічного напрямку орієнтовано на відомі технології, які характеризуються значною надмірністю, високою вартістю й складністю при рішенні комплексу задач. Фактично це пояснюється складністю в узагальненому описі таких систем.

До останнього часу для рішення таких задач використовувалися автоматні методи, а також методи системного аналізу й синтезу. В останні роки для проектування, специфікації, візуалізації, конструювання й документування різних систем використовується мова UML, що представляє собою об'єднання інженерних прийомів, які раніше успішно використовувалися при моделюванні великих і складних систем. Але, як показала практика, використання інструментів UML при розробці систем реального часу, зокрема систем керування, досить складне.

З метою рішення цієї проблеми було розроблено стандарт UML-FB (IEC 61499), у якому об'єкти представлені не у вигляді класів, а у вигляді функціональних блоків (ФБ), ідеалізованих «пристроїв», що виконують ту або іншу функцію, відповідно до внутрішнього алгоритму, і, що мають вхідні й вихідні події й змінні. На основі таких ФБ можлива побудова систем реального часу будь-якої складності з контролем чисельних параметрів, а, отже, – проведення аналізу ідеалізованої системи. Однак розвиток цього стандарту для аналізу й синтезу технічних систем поки мало досліджено.

Розглянемо як приклад побудову моделі розподіленої системи керування електроенергетичної системи, що складається з 3-х дизель-генераторів з використанням програмного пакета FBDK. В якості середовища для розробки був обраний програмний продукт FBDK (Function Block Development Kit, Holobloc Inc; Version 2006-10-17).

Розглянемо побудову моделі розподіленої системи керування електроенергетичної системи, що складається з 3-х дизель-генераторів з використанням програмного пакета FBDK. В якості середовища для розробки був обраний програмний продукт FBDK (Function Block Development Kit, Holobloc Inc; Version 2006-10-17).

Першим кроком у побудові моделі є загальний аналіз системи [11]. Необхідно визначити всі об'єкти, що входять у модель, а також їх інтерфейси. Для рішення цього завдання зручно використовувати діаграму класів, на якій кожен об'єкт представлений у вигляді окремого класу зі своїми аргументами (змінними, що характеризують стан або режим роботи об'єкта керування) і методами (фактично, функціями об'єкта). Для електроенергетичної системи це такі об'єкти, як дизелі, генератори, засоби індикації стану, навантаження, блок керування. Завдання керування структурою енергетичної системи (підключення/відключення дизель-генераторів, навантаження, завдання розподілу потужності, функції захисту) виконуються блоком керування й при моделюванні реалізуються програмно.

Дизель представляє собою двигун із системою стабілізації обертів, що управляє рівнем подачі палива (залежно від навантаження на валу). Основні стани дизеля наступні:

– не запущений;

– запущений;

– навантаження на валу не перевищують максимальну потужність двигуна – номінальна частота обертання вала підтримується за рахунок збільшення подачі палива. Приводиться у відсотках від максимального значення

$$Acceleration = \frac{P_{load}}{P_{max}} \cdot 100\%;$$

– навантаження на валу більше максимальної потужності двигуна. У цьому випадку подача палива максимальна (і не зростає), частота обертання вала починає падати й визначається формулою:

$$\Omega = 0.05 \cdot (P_{max} - P_{load}) - 157,$$

де 0.05 – ваговий коефіцієнт впливу навантаження на валу на оберти двигуна, 157 – номінальна частота обертання (рад/с);

– навантаження на валу досягають критичного значення (на малих потужностях $P_{load_КРИТ} = 2P_{max}$, або частота обертання ротора падає до 100 рад/сек) – аварійна зупинка двигуна.

Таким чином, ФБ «DIESEL» має три вхідних змінних (рис. 6.8) – максимальна потужність двигуна (VAR POWER_MAX тип REAL, задається користувачем), потужність навантаження на валу (VAR POWER_LOAD тип REAL, задається валом генератора), дозвіл на запуск (VAR ENABLE_START тип BOOL, установлюється системою); і одна вхідна подія – старт (EVENT «START»). Вихідними даними будуть – поточний рівень подачі палива у відсотках від максимального значення (VAR ACCELERATION тип REAL), частота обертання валу двигуна (VAR ROT_FREQ тип REAL), сигнал про перевантаження (VAR DROPOUT тип BOOL), сигнал про аварійну зупинку (VAR EMERGENCY_STOP тип BOOL), інформація про виконання функції (EVENT «ON»).

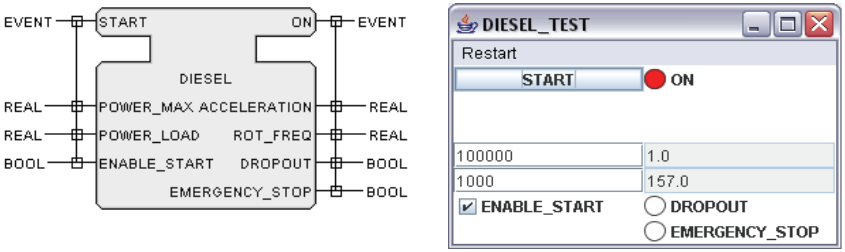


Рис. 6.8.

Генератор представляє собою сам генератор і систему стабілізації рівня вихідної напруги, за допомогою зміни струму збудження. Основні стани генератора:

– відключений (якщо вал дизеля не обертається);

– включений;

– навантаження в мережі не перевищують максимальної потужності генератора (номінальне значення рівня напруги на виході підтримується за

рахунок зміни струму порушення – значення якого приводиться у відсотках від максимального значення:

$$Stimulation = \frac{P_{load} \cdot 0.05 + 1 \cdot (157 - \Omega)}{P_{max} \cdot 0.05} \cdot 100\%,$$

де 0,05 і 1 – вагові коефіцієнти впливу зміни параметрів на рівень вихідної напруги);

– навантаження в мережі перевищують максимальну потужність генератора або/і частота обертання ротора падає до критичного значення – порушення максимальне й не зростає, рівень вихідної напруги визначається формулою:

$$U = P_{max} \cdot 0.05 - P_{load_electric} \cdot 0.05 - (157 - \Omega) \cdot 1 + 380,$$

де 157 – номінальна частота обертання (рад/с), 380 – номінальна діюча напруга на виході;

– навантаження в мережі або/і частота обертання ротора досягають критичного значення, при яких

$$\frac{P_{load} \cdot 0.05 + 1 \cdot (157 - \Omega)}{P_{max} \cdot 0.05} \cdot 100\% > 200\%,$$

або напруга на виході менше 300В – аварійне відключення від мережі.

Частота вихідної напруги буде визначатися, виходячи з кількості пар полюсів генератора, і дорівнює:

$$f = \frac{\Omega}{\pi},$$

а потужність на валу:

$$P_{load_mechanic} = \frac{P_{load_electric}}{85\%},$$

з урахуванням ККД=85%.

Таким чином, ФБ «GENERATOR» (рис. 6.9) має три вхідних змінних: максимальні потужності генератора (VAR POWER_MAX тип REAL, задається користувачем), потужність навантаження в мережі (VAR POWER_LOAD_EL тип REAL), частота обертання валу дизеля (VAR ROT_FREQ тип REAL); і одна вхідна подія – старт (EVENT «ROTATE»). Вихідними даними будуть – поточний рівень струму порушення, у відсотках від максимального значення (VAR STIMULATION тип REAL), механічне навантаження на валу генератора (VAR POWER_LOAD_MC тип REAL), інформація про вихідний ЕРС: діюче значення (VAR VOLTAGE тип REAL), частота (VAR FREQ тип REAL), сигнал про перевантаження (VAR DROPOUT тип BOOL), сигнал про аварійне відключення (VAR EMERGENCY_SHUTDWN тип BOOL), інформація про виконання функції (EVENT «ON»).

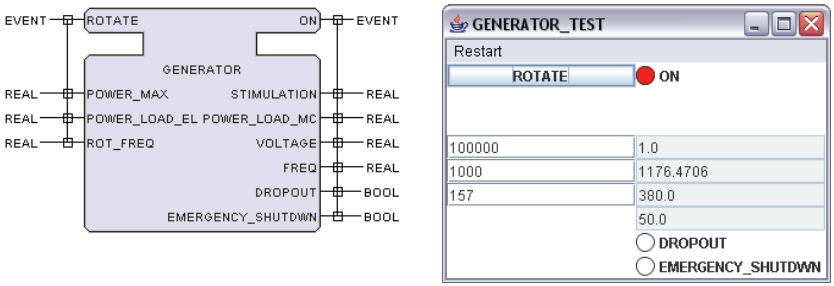


Рис. 6.9. Зовнішній вигляд ФБ «Генератор» і тестове вікно

На рис. 6.10 представлені алгоритми роботи ФБ дизеля й генератора.

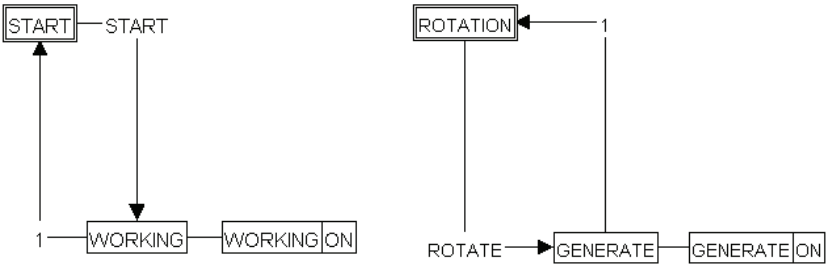


Рис. 6.10.

Після того, як створені моделі дизеля й генератора, можна створити складовий ФБ дизель-генератора. Визначивши тип блоку, і вказавши напрямку потоків даних, вийде об'єднаний ФБ, що представляє собою модель дизель-генератора. У складеному блоці відсутні алгоритми, або програми – він усього лише визначає зв'язки і їх напрямки між базовими ФБ.

Вхідними параметрами блоку є:

– максимальна потужність генератора;
– максимальна потужність дизеля;
– потужність навантаження мережі;
– дозвіл на запуск.

Вихідними параметрами блоку:

– рівень вихідної напруги;
– частота вихідної напруги;
– рівень подачі палива в дизелі;
– рівень струму збудження в генераторі;
– сигнал про аварійне відключення генератора;
– сигнал про аварійну зупинку дизеля;
– сигнал про перевантаження дизеля;
– сигнал про перевантаження генератора.

На рис. 6.11 представлено вид складеного ФБ «DIESEL_GENERATOR» і вікна моделювання.

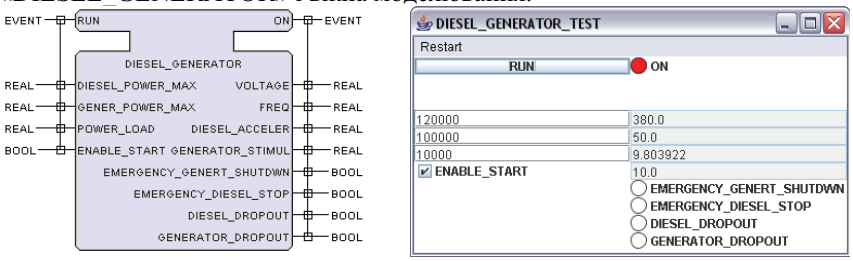


Рис. 6.11.

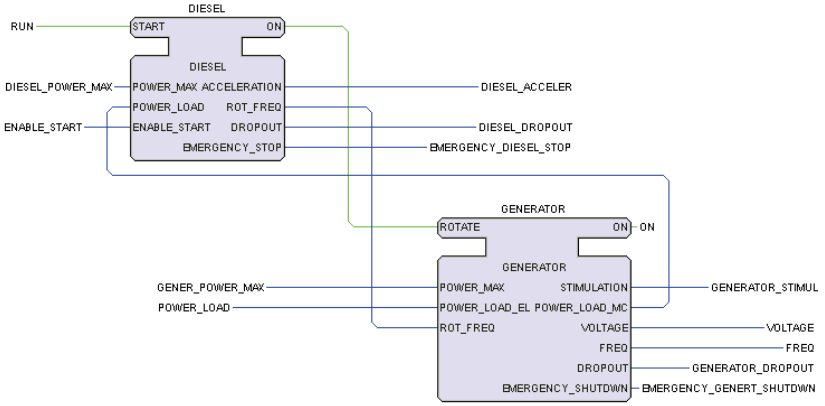


Рис. 6.12.

На рис. 6.13 представлено тестове вікно для функціонального блоку дизель-генератора.

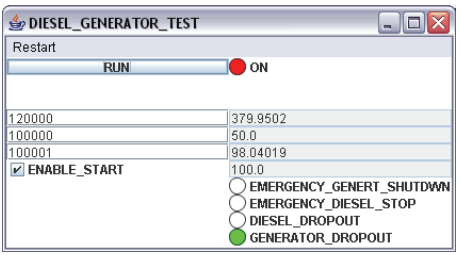


Рис. 6.13.

Вхідними параметрами моделі є потужність дизеля, потужність генератора й потужність навантаження. У наведеному прикладі потужність навантаження більше потужності генератора, тому видається сигнал про перевантаження генератора. Також відображаються значення напруги мережі, частоти, рівень подачі палива й струм збудження. Ці параметри визначаються статичними характеристиками дизеля й генератора, які описуються на етапі проектування блоку. В окремих випадках більш важливим є перевірка логіки роботи системи, чим одержання точних числових значень. Тому можна використати спрощену модель дизель-генератора, у якій частота завжди підтримується рівної 50 Гц, а напруга на шинах 380 В.

На рис. 6.14 представлений зовнішній вигляд програми FBDK і функціонального блоку дизель-генератора (спрощений).

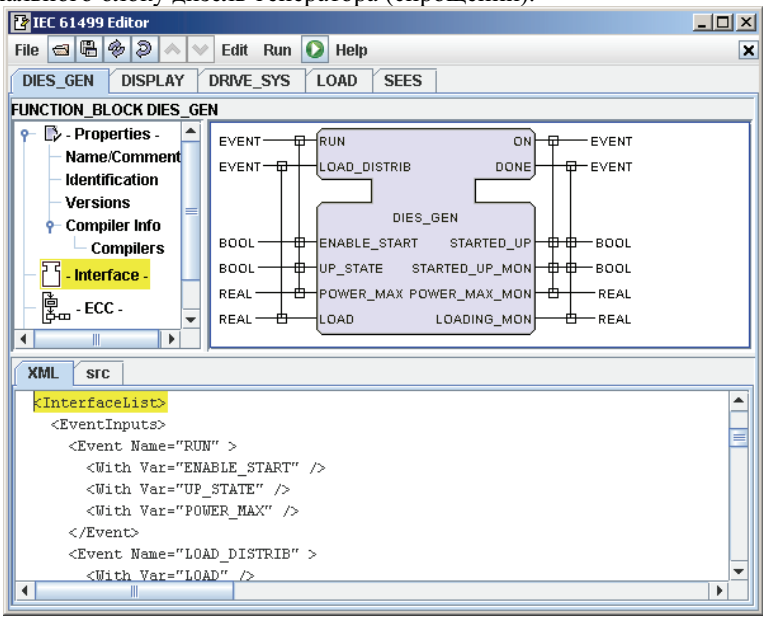


Рис. 6.14.

Вхідними сигналами блоку є сигнал запуску (RUN), розподілу потужності (LOAD_DISTRUB), вихідними – сигнал готовності (DONE). Також для блоку дизель-генератора задаються такі параметри, як потужність (POWER_MAX), навантаження (LOAD), сигнал дозволу запуску й сигнал (ENABLE_START) можливості запуску (UP_STATE). Останній використовується для моделювання відмов пуску ДГ внаслідок, наприклад, поломки. Вихідні параметри показують, запущений генератор чи ні (STARTED_UP), а також його завантаження. Ці сигнали надходять у блок індикації.

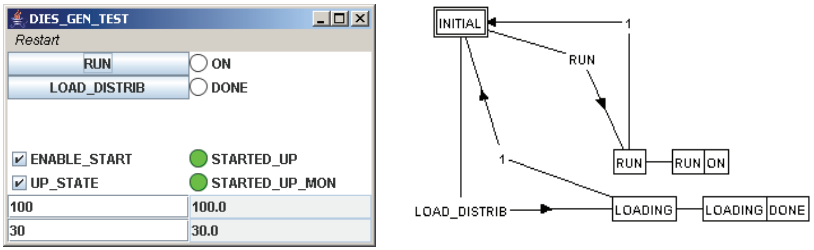


Рис. 6.15.

Опис поводження блоку в програмі FBDK виконується мовою JAVA. На рис. 6.16 представлений вигляд вікна із програмою.

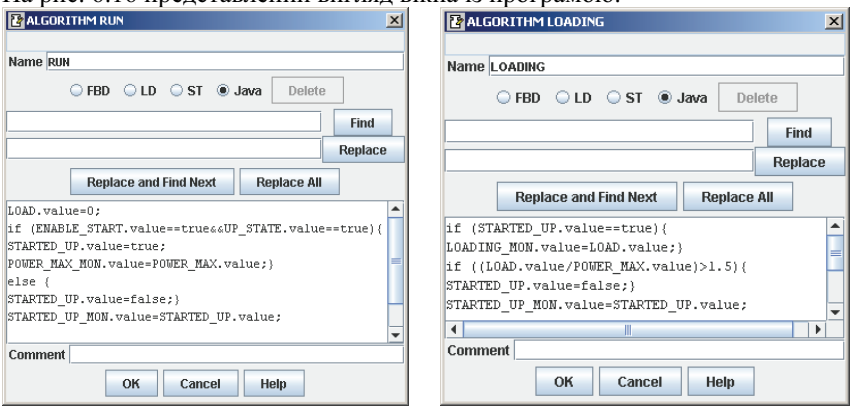


Рис. 6.16.

На рис. 6.17 представлено блок індикації стану електроенергетичної системи й алгоритм його роботи. У цей блок надходять сигнали від кожного дизель-генератора й дозволяють визначити стан агрегату, його відсоток завантаження та інші параметри.

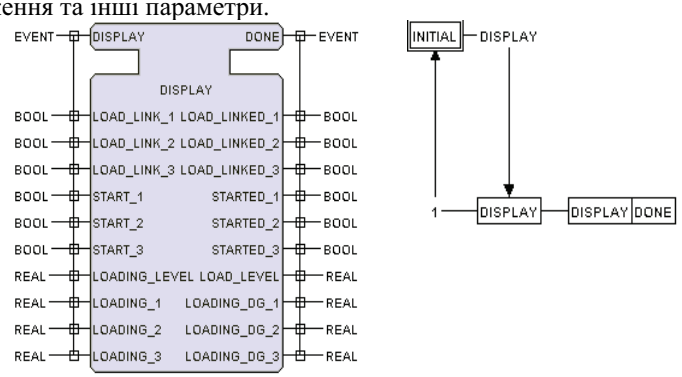


Рис. 6.17.

На рис. 6.18 представлений вид функціонально блоку системи керування. Всі завдання по керуванню структурою енергетичної системи виконує система керування.

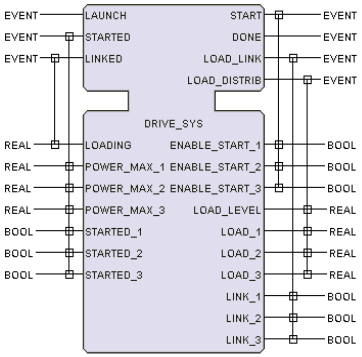


Рис. 6.18.

На рис. 6.19 представлено алгоритм роботи функціонального блоку DRIVE_SYS.

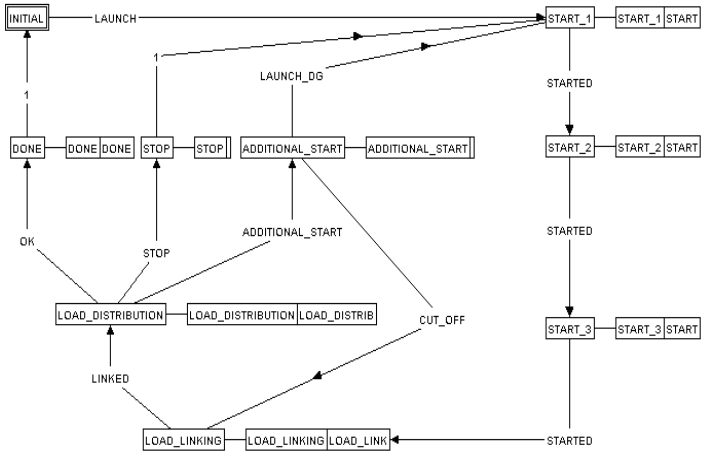


Рис. 6.20.

Тестове вікно представлено на рис. 6.21. У цьому вікні задаються сумарна потужність навантаження (у кВт), потужності кожного з дизель-генераторних агрегатів і сигнали дозволу запуску. Відображається величина завантаження кожного з дизель-генераторів. Моделювання виконується шляхом натискання на кнопки LAUNCH (пуск), STARTED (пуск ДГ), LINKED (підключення до шин). При цьому відбувається перехід з одного стану в інший відповідно до діаграми, наведеної на рис. 6.21.

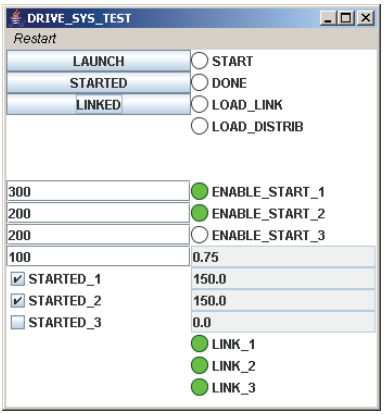


Рис. 6.21.

На рис. 6.22 представлено функціональний блок навантаження й тестове вікно.

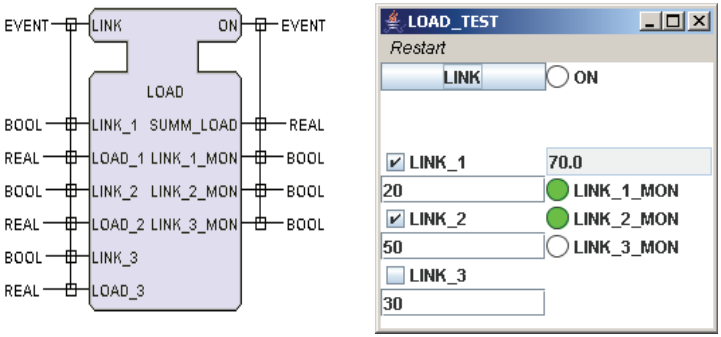


Рис. 6.22.

За допомогою цього блоку задається навантаження на дизель-генератори. У блоці можна задавати сумарну потужність 3-х груп споживачів і управляти підключенням кожної із груп.

Після того, як всі блоки описані, можна створити модель електроенергетичної системи (рис. 6.23).

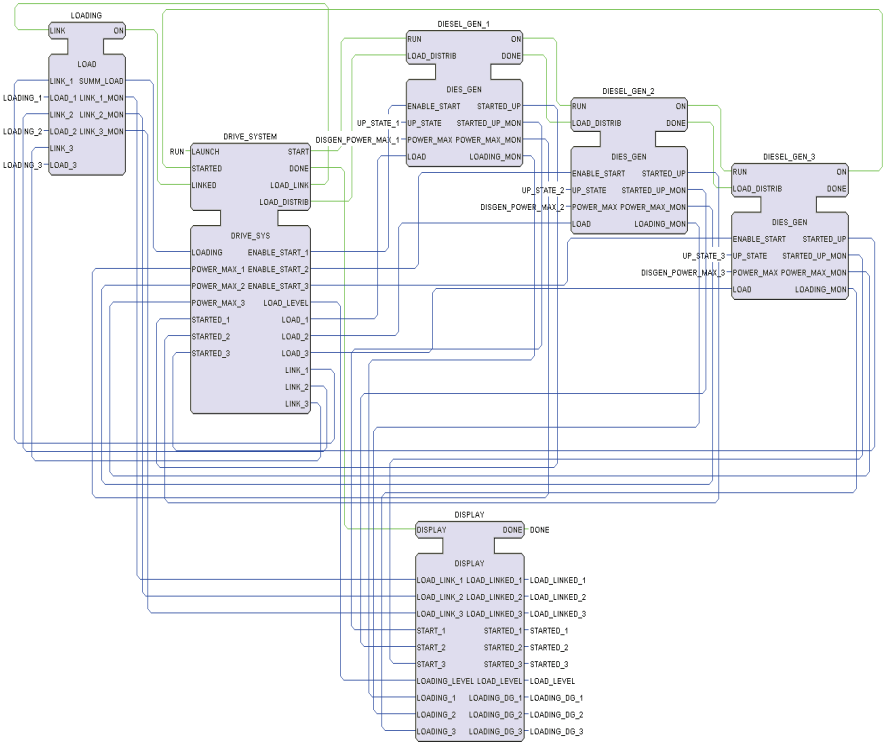


Рис. 6.23.

Існують програми, що дозволяють перетворювати створені моделі на UML-FB у моделі «класичного» UML (і навпаки). Для цього необхідно створити діаграму класів і діаграми станів для кожного класу, які описують алгоритм його роботи.

Основні переваги нового стандарту стосовно «класичного» – можливість одержання чисельних характеристик системи й наближеність ідеології до особливостей систем реального часу.

6.5. Концептуальна модель системи моніторингу за станом СЕЕС

Система моніторингу представлена у вигляді програмного комплексу, який вирішує завдання збору й обробки даних, прийняття рішень і організації баз даних.

При розробці системи моніторингу, її графічного інтерфейсу використовувалися методи концептуального моделювання інформаційних процесів і інтелектуальних систем [56].

Концептуальне моделювання інформаційного забезпечення – це модель діяльності процесів, необхідна для надання знань про структуру досліджуваної області. Дана модель враховує процеси предметної області, вхідні й вихідні інформаційні об'єкти й сукупність керуючих елементів. Подібна організація інформаційних зв'язків сприяє більше наочному переходу від неформальних знань предметної області до їхнього формального подання.

Загальною концептуальною моделлю графічного інтерфейсу можна назвати наступний ряд процесів і об'єктів:

$$S_{ГИ} = \langle P, D, G, H_p, H_d, H_g, S_p, S_d, S_g, InD, InG, OutD, OutG \rangle,$$

де $P = \{p_i\}$ – множина процесів побудови зображень і обробки інформації;

$D = \{d_i\}$ – множина інформаційних об'єктів;

$G = \{g_i\}$ – безліч графічних об'єктів;

H_p, H_d, H_g – установлюють відповідність між окремими об'єктами моделі й множиною підлеглих їм підоб'єктів;

S_p, S_d, S_g – проходження процесів, які задають частковий порядок виконання процесів;

InD – вхідні інформаційні об'єкти процесу;

InG – вхідні графічні об'єкти процесу;

$OutD$ – вихідні інформаційні об'єкти процесу;

$OutG$ – вихідні графічні об'єкти процесу.

До безлічі процесів побудови й обробки інформації $P = \{p_i\}$ можна віднести функції високого рівня, що формують крапкові зображення (растрові подання об'єктів, видимі контури об'єктів і ін.); апроксимуючі до кусково-лінійних функцій (графічні подання об'єктів, індикатори та ін.); функції обробки вхідних даних; функції формування керуючих пакетів; функції розподілу процесів на потоки, що виконуються та інші.

До множини інформаційних об'єктів $D = \{d_i\}$ можна віднести пакети даних, що надходять; змінні і їх структури, асоційовані з елементами керування; проміжні стеки даних, формовані репорти передачі даних та інші.

Множина графічних об'єктів $G = \{g_i\}$ представлено у вигляді ряду індикаторів (графічні, стрілочні, шкальні, чисельні й ін.), керуючих об'єктів (елементи введення/виводу, індикації станів, відображення строкових змінних і ін.), видимої області системи моніторингу.

Однією з головних складових графічної моделі є концепція графічних атрибутів. Набори атрибутів описують основні властивості графічних об'єктів, такі як стиль лінії, її колір, розміри контурів і колір фону. Модель графічних атрибутів може бути описана наступним рядом:

$$A_g = \langle C, F_c, N_c, T_c, n_c, t_c, InC, OutC \rangle,$$

де $C = \{c_i\}$ – множина графічних атрибутів;

F_c – графічне зображення;

N_c, n_c – множина імен графічних атрибутів;

T_c, t_c – множина імен типів графічних атрибутів;

InC – процес, що запитує значення атрибута;

$OutC$ – процес ініціалізації сукупності запитаних атрибутів графічного об'єкта. Подібним чином складається модель графічних ресурсів:

$$A_r = \langle R, F_r, N_r, T_r, n_r, t_r, InR, OutR, DelR \rangle, ,$$

де $R = \{r_i\}$ – множина графічних ресурсів;

$DelR$ – процедура відображення графічного ресурсу.

При проектуванні концептуальної моделі (рис. 6.24) використовувались визначені типи об'єктів, до яких відносяться графічні примітиви, ресурси й атрибути. У графічних бібліотеках кожному із примітивів зіставлена певна кількість атрибутів.

Для спрощення аналізу концептуальної моделі використовують шаблони предметної області. Вони представлені окремими фрагментами моделі і побудовані щодо обраних процесів, з огляду на взаємодії з іншими об'єктами.

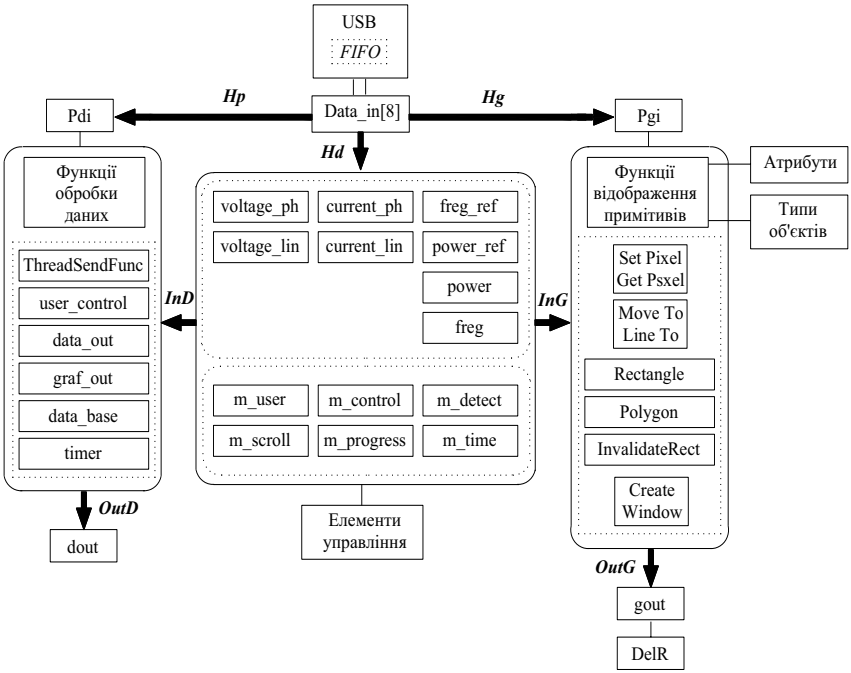


Рис. 6.24.

Аналізуючи структуру зображень щодо кожного окремо взятого процесу можна сформулювати наступні залежності вихідних графічних об'єктів:

– вихідне зображення для даного процесу є єдиним, типи ієрархій процесу й зображення збігаються;

– вихідному зображенню відповідає хоча б один процес, для якого хоча б один компонент зображення є вхідним;

– компоненти зображення, що є строго вхідним стосовно шаблону не можуть бути вихідними зображеннями підпроцесів;

– будь-який підпроцес може мати не більше одного вихідного зображення;

– вихідному даному процесу шаблона відповідає хоча б один компонент, що є вхідним для підпроцесу;

– будь-які дані, взаємодіючі з підпроцесами, є або компонентами вхідних або вихідних даних процесу, або локальними даними, використовуваними для передачі інформації між підпроцесами;

– компоненти даного, що є строго вхідним стосовно процесу не можуть виступати в якості вихідних даних підпроцесів.

– виходячи з вищевказаних залежностей концептуальна модель системи моніторингу має регулярну структуру зображень і регулярну структуру даних, тобто дана модель є регулярною.

Проходження в регулярній концептуальній моделі пов'язане з надмірністю моделі, що вносить одночасне використання відносин проходження процесів S_p і зображень S_g. Вивід зображень залежить від пріоритету проходжень, тому що ними побічно визначається порядок виконання процесів. При декомпозиції зображень відносини проходження успадковуються всіма компонентами зображень.

При виборі імен екземплярів і типів об'єктів у моделі повинні враховуватися наступні обмеження:

– два підоб'єкта повинні мати різні імена, що не збігаються з іменем головного об'єкта;

– об'єкти, що мають тотожну структуру підоб'єктів повинні мати схожі імена, що вказують їхню приналежність до загального процесу;

– щоб запобігти однойменних і однотипних екземплярів об'єктів шляхом попарного перебору роблять синтаксичний контроль моделі.

Використання концептуальної моделі дозволяє аналізувати процеси обробки й виводу зображень, зменшити кількість операцій, що виконуються шляхом виключення тотожних функцій, від початку зіставити проходження процесів і організацію неструктурованих даних, розподілити ресурси й операції на окремі потоки, що виконуються знижуючи навантаження на обчислювальні засоби. Подібна організація програмного комплексу в значній мірі збільшує швидкодію й надійність системи моніторингу.

6.6. Методика створення програмного забезпечення для систем автоматизації електроенергетичних установок

Автоматизована система керування електроенергетичною установкою має багаторівневу структуру [23]. Нижній рівень – збір інформації з датчиків і видача керуючих впливів на виконавчі механізми. Середній – рівень первинної обробки й маршрутизації пакетів. Верхній – рівень прийому, обробки й відображення даних, включає сервер збору даних й автоматизоване робоче місце оператора.

До основних функцій системи відносяться:

1. Інформаційні функції:

– вимір і контроль напруги на шинах головного розподільного щита;

– контроль гранично припустимих значень частоти напруги;

– сигналізація роботи устаткування;

– виявлення, сигналізація та реєстрація відхилень параметрів від установлених границь;

– ручне введення даних;

– формування й видача оперативних даних;

– архівування параметрів технологічного процесу вироблення електроенергії.

2. Керуючі функції:

– пряме цифрове керування контурами (параметрами) технологічного процесу;

– реалізація протиаварійних захистів;

– видача керуючих впливів (здійснюється із мнемосхем на дисплеї).

3. Допоміжні функції:

– тестування комплексу технічних засобів;

– переналаштування системи (реконфігурація програмного забезпечення).

4. Організаційно-технологічні функції:

– розрахунок споживання й вироблення електроенергії за годину, добу, з метою реалізації можливості прогнозування завантаження електростанції й рішення завдань упереджуючого керування;

– облік часу пробігу (простою) устаткування;

– оцінка діяльності технологічного персоналу;

– розрахунок техніко-економічних показників.

На етапі створення програмного забезпечення (ПЗ) необхідно визначиться з набором типових блоків, з яких буде складатися електростанція. До таких блоків відносяться різні автомати керування режимами роботи електростанції та комутуються оператором або автоматично, залежно від режиму роботи СЕЕС, генератори, дизелі, конектори, блоки відображення стану автоматів з метою діагностики, блоки контролю параметрів елементів електростанції: частоти, напруги, параметрів потужності, $\cos\varphi$.

Для кожного із блоків призначаються параметри, що характеризують його властивості:

– для генератора – потужність, напруга, оберти та ін.;

– для автомата – струм, час спрацьовування та ін.;

– для навантаження – потужність, тип навантаження та ін.

Із блоків створюється бібліотека. Оскільки програмне забезпечення виконує керування зовнішнім пристроєм, то кількість блоків повинна відповідати кількості керованих об'єктів. Зокрема, кількість автоматичних вимикачів в автономній електростанції повинне відповідати кількості блоків у бібліотеці.

Основна ідея організації програмного забезпечення вводу-виводу складається в розбивці його на кілька рівнів, причому нижні рівні забезпечують екранування особливостей апаратури від верхніх, а ті, у свою чергу, забезпечують зручний інтерфейс для користувачів.

Кожний із блоків автоматики програмно зв'язується з певним реле, що управляється автоматично. Приклад структури даних, що посилають, показаний на рис. 6.25.

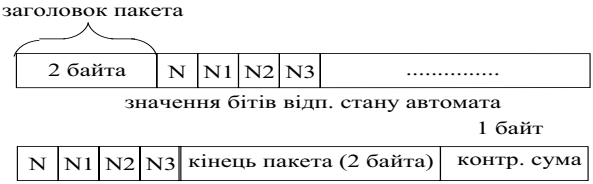


Рис. 6.25.

Оскільки не виключені одиничні помилки для підвищення надійності поле зі станом автоматів дублюється. Нехай біти позначені b_{N-1}, b_{N-2}, ….....,b_3 , b_2 , b_1 , b_0 , а в дублі C_{N-1} , C_{N-2} ,…....., C_3 , C_2 , C_1 , C_0 . Тоді перевіряється умова:

$$e = \overline{b_n \oplus C_n}.$$

Якщо $e = 1$, це означає, що при передачі даних відбулася помилка й керуючий пристрій ігнорує отриманий пакет. При цьому відправляється зворотний пакет з кодом помилки й запитом повторної передачі.

Контрольна сума *LRC* обчислюється шляхом додавання всіх байт у пакеті без обліку переповнення:

$$LRC = \left[\sum_{i=0}^{N-1} B_i \right].$$

Необхідно вирішити проблему створення топології електростанції, що у програмі створює користувач, при цьому спосіб опису топології повинен задовольняти вимогам зручності формалізації, збережена і зчитування в/із файлу, перевірки на наявність топологічних помилок (несумісні блоки не повинні бути з'єднані). Для рішення цієї проблеми кожному блоку призначається унікальний номер *Id.* Тоді вся бібліотека може бути описана набором

$$Id = \{Id_0, Id_1 .... Id_M\},$$

де M – кількість типів блоків, що використовуються в електростанції.

У табл. 6.2 представлений приклад бібліотеки блоків, що містить у собі автомати, *L*- і *T*-конектори, синхронний генератор, асинхронний двигун, активне навантаження.

Таблиця 6.2.

Блок	Id	Блок	Id
	Id_0		Id_1
	Id_2		Id_3
	Id_4		Id_5
	Id_6		Id_7

З типових блоків на робочому полі програми розроблювачем складається схема електростанції. Оскільки кожен блок має свій унікальний ідентифікатор, те вся топологія може бути описана за допомогою матриці:

$$T = \begin{bmatrix} Id_5 & Id_3 & Id_2 & \dots & \dots & Id_9 \\ Id_7 & Id_8 & \dots & \dots & \dots & \dots \\ : & \dots & \dots & \dots & \dots & \dots \\ : & \dots & \dots & \dots & \dots & \dots \\ Id_2 & \dots & \dots & \dots & \dots & Id_4 \end{bmatrix}.$$

Розмірність матриці може бути фіксованою, а може бути динамічною. На початковому етапі розробки пропонується робити матрицю фіксованої розміром 10×10.

Знаючи значення елементів матриці, на діалоговому вікні програми в заданому порядку розташовуються блоки з бібліотеки, які мають ідентифікатори такі ж, як елементи матриці. Таким чином, матриця однозначно відповідає схемі суднової електростанції. Використання матричного подання доцільно, оскільки дозволяє використовувати масиви при реалізації програ-

много коду. Крім того, математичний апарат добре відпрацьований і з використанням матриць зручно формалізувати топології, вводити додаткові перевірки на наявність помилок.

Друга проблема складається в створенні зв'язків між блоками. Рішення цієї задачі дозволить виконати перевірку на коректність схеми електростанції та попередити виникнення помилок. У цілому, можна обійтися і без цієї матриці, і виконати перевірку безпосередньо по топологічній матриці. З табл. 6.2 видно, що блоки Id_0 й Id_2 не можуть бути розташовані по вертикалі поруч, а блоки Id_1 й Id_3 не можуть бути розташовані по горизонталі поруч. На рис. 6.26 показаний приклад схеми, складеної з типових блоків.

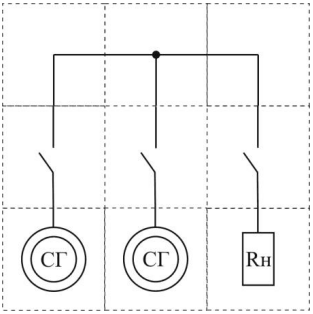


Рис. 6.26. Схема, побудована із блоків

Схема може бути описана матрицею 3×3:

$$S_1 = \begin{bmatrix} 6 & 1 & 7 \\ 2 & 2 & 2 \\ 3 & 3 & 4 \end{bmatrix}.$$

Оскільки кожен блок має свій унікальний номер, елементи матриці представлять собою ці ідентифікатори.

Інший приклад схеми представлений на рис. 6.27. У цій схемі допущені помилки при розташуванні блоків на робочому полі.

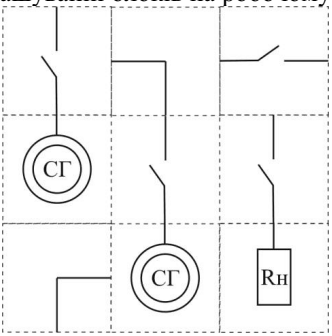


Рис. 6.27.

Матриця, що описує топологію наведеної схеми, має вигляд:

$$S_2 = \begin{bmatrix} 2 & 7 & 0 \\ 3 & 2 & 2 \\ 6 & 3 & 4 \end{bmatrix}.$$

Для першої матриці всі елементи розташовані коректно для другої матриці видно, що по вертикалі елементи 6 й 2 у першому стовпці розташовані неправильно. Можна ввести для цих блоків Id_{6nk} та Id_{2ij} умову:

$$k = j,\ n = i - 1.$$

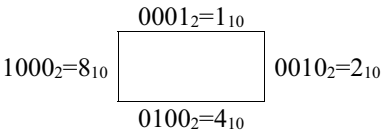
Тоді якщо $i = 2$, і блок автомат розташований в комірці, то блок *L*-конектор відносно цього блоку може бути розташований у цій же колонці, але на попередньому рядку.

Аналогічно, для елементів Id_2 та Id_4. Умова коректного розташування на робочому полі блоків Id_{2mn} і Id_{4ij} має вигляд:

$$n = j,\ i = m + 1.$$

На етапі створення схеми перевіряється умова для кожного блоку й забезпечується коректність схеми.

Оскільки кожен блок може мати вхідний і вихідний зв'язок з кожної зі сторін, для створення матриці зв'язків використовується наступний принцип: наявність або відсутність зв'язку позначається «1» або «0» відповідно. Тоді кожної зі сторін блоку призначається число, наприклад:



Для схеми на рис. 6.26 матриця зв'язків має вигляд:

$$C = \begin{bmatrix} 0110_2 & 1110_2 & 1100_2 \\ 0101_2 & 0101_2 & 0101_2 \\ 0001_2 & 0001_2 & 0001_2 \end{bmatrix}_2 = \begin{bmatrix} 7 & 14 & 12 \\ 5 & 5 & 5 \\ 1 & 1 & 1 \end{bmatrix}_{10}.$$

Для збереження у файл досить записати елементи обох матриць (топологічної матриці й матриці зв'язків), а потім у тій же послідовності зчитати. Знаючи значення елементів цих матриць, можна розташувати блоки на робочому полі та виконати їхнє з'єднання. У такий спосіб реалізується можливість збереження розробленої схеми.

Для зберігання координат блоків зручно використовувати матрицю координат. Якщо розміри блоків фіксовані, то досить зберігати тільки координати верхнього лівого кута для кожного блоку відносно робочого поля.

Якщо ж розміри блоків різні, тоді додатково вказується ширина та висота блоку. Приклад матриці координат:

$$P_{xy} = \begin{bmatrix} P_{00} & P_{01} & P_{02} \\ P_{10} & P_{11} & P_{12} \\ P_{20} & P_{21} & P_{22} \end{bmatrix},$$

де

$$P_{ij} = [x, y, Id],$$

x – координата блоку по горизонталі;
y – координата блоку по вертикалі.

Поле *Id* введене для додаткового контролю й перевірки від помилок, у загальному випадку може бути відсутнє, оскільки розміри топологічної матриці й матриці координат будуть однаковими. Якщо розміри блоків різні то елемент матриці координат представляє собою набір значень:

$$P_{ij} = [x, y, W, H, Id],$$

де W – ширина блоку,
H – висота блоку.

Використання такої структури матриці дозволяє контролювати розташування блоків на полі так, щоб вони не перетиналися. Для цього при з'єднанні елементів повинна перевірятися й виконуватися умова:

$$x_{i+1,j+1} > x_{i,j} + W,$$
$$Y_{i+1,j+1} > Y_{i,j} + H.$$

Таким чином, використання матриць дозволяє вирішувати завдання формалізації схем і виявляти помилки на етапі проектування.

Для реалізації основних вимог при побудові програмного забезпечення автоматизованих систем повинні бути дотримані принципи, які можна розділити на три групи: проектування, системотехнічні й організаційні. До принципів проектування ставляться принципи системності, комплексності, ефективності, принципи нових завдань. Системотехнічними є принципи комплексного використання обчислювальної техніки й програмних засобів, типізації рішень, єдності інформаційної бази, організації безпосереднього спілкування користувача із системою, розвитку системи. Більшість завдань варто вирішувати в діалоговому режимі.

При розробці програмного забезпечення першим етапом є збір даних про об'єкт автоматизації й здійснюваних видів діяльності, оцінка якості функціонування об'єкта, виявлення проблем, рішення яких можливо засобами автоматизації.

Завершальним етапом при розробці програмного забезпечення є аналіз функціонування системи, виявлення відхилень фактичних експлуатаційних характеристик ПЗ від проектних значень, встановлення причин цих відхилень, усунення виявлених недоліків і забезпечення стабільності експлуатаційних характеристик ПЗ, внесення необхідних змін у документацію на ПЗ.

Правильний вибір методу, засобів і технології проектування створює умови для зниження витрат і скорочення строків проектування. До числа основних методів, які доцільно використати при проектуванні організаційних завдань, варто віднести метод пошуку аналога, метод синтезу на базі модулів і типових організаційних рішень, метод моделювання.

6.7. Статична модель інтерфейсу користувача

На першому етапі створення програмного забезпечення необхідно визначиться з набором типових блоків, з яких буде складатися електростанція. До таких блоків належать різні автомати керування режимами роботи електростанції, що комутуються оператором або автоматично, залежно від режиму роботи суднової електроенергетичної системи, генератори, дизелі, коннектори, блоки відображення стану автоматів з метою діагностики, блоки контролю параметрів елементів електростанції: частоти, напруги, параметрів потужності, cosφ.

Для кожного із блоків призначаються параметри, що характеризують його властивості:

– для генератора – потужність, напруга, оберти й ін.;

– для автомата – струм, час спрацьовування й ін.;

– для навантаження – потужність, тип навантаження та ін.

Із блоків створюється бібліотека. Оскільки програмне забезпечення виконує керування зовнішнім пристроєм, то кількість блоків повинне відповідати кількості керованих об'єктів. Зокрема, кількість автоматичних вимикачів в автономній електростанції повинно відповідати кількості блоків у бібліотеці. Процес створення зв'язку показаний на рис. 6.28.

Користувач взаємодіє із системою за допомогою клавіатури та миші. Користувач ініціює наступні прецеденти (рис. 6.28):

1. *Створити Блок.*

Передумова. Даний блок є в бібліотеці.

Опис. Під створенням блоку розуміється перетаскування на робоче поле кожного з доступних у бібліотеці блоків (технологія Drag and Drop).

Альтернатива. При відсутності блоку в бібліотеці вивести повідомлення.

Постумова. Новий блок створений і виділений.

2. *Видалити Блок.*

Передумова. Блок виділений (натисканням лівої кнопки миші).

Опис. Видалити блок можна натисканням кнопки Delete на клавіатурі або викликом відповідного пункту контекстного меню блоку. Якщо блок був пов'язаний з іншими елементами, зв'язки також знищуються.

Постумова. Об'єкт зник з робочого поля, а кількість екземплярів даного елемента в бібліотеці збільшилося на 1.

3. *Перемістити Блок.*

Передумова. Блок виділений.

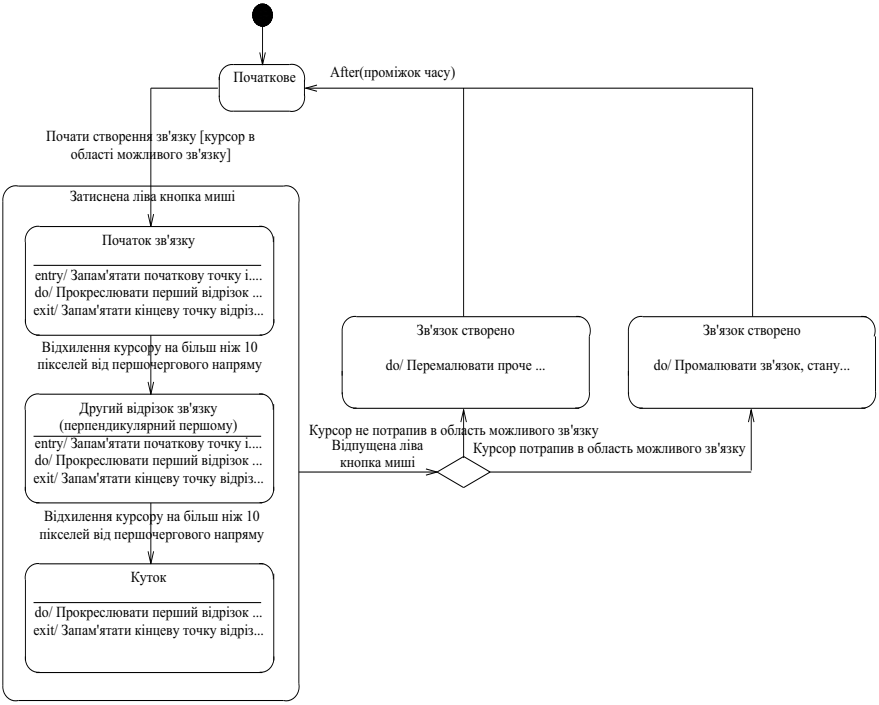


Рис. 6.28.

Опис. Переміщення блоку здійснюється слідом за курсором при затиснутій лівій кнопці миші або при натисканні кнопок «↑», «↓», «→», «←» на клавіатурі. Якщо блок має зв'язок, він повинен переміщатися разом з ним.

Альтернатива. При досягненні блоком границь робочого поля він повинен зупинятися.

Постумова. Блок змінив положення й виділений.

4. *Створити Зв'язок між Блоками*.

5. *Видалити Зв'язок між Блоками*.

Передумова. Зв'язок виділений натисканням на одному з її відрізків.

Опис. Видалення зв'язку відбувається при натисканні кнопки Delete на клавіатурі або виборі відповідного пункту контекстного меню.

Постумова. Зв'язок зник з робочого поля.

6. *Перемістити Лінію Зв'язку*.

Передумова. Лінія зв'язку виділена натисканням на одному з її відрізків.

Опис. При переміщенні курсору при затиснутій лівій кнопці миші лінія зв'язку пересувається слідом за курсором.

Альтернатива. При вилученні курсору за межі робочого поля зв'язок переміщується до краю робочого поля.

Постумова. Лінія зв'язку змінила положення й виділена.

7. *Копіювати Об'єкт або Лінію Зв'язку*.

Передумова. Блок (лінія зв'язку) виділений.

Опис

1. Користувач натискає комбінацію «Ctrl+C» або вибирає пункт контекстного меню «Копіювати».

2. Користувач натискає комбінацію «Ctrl+V» або вибирає пункт контекстного меню «Вставити».

Альтернатива. Якщо екземпляри блоку закінчилися в бібліотеці, операція не виконується й виводиться повідомлення, що повідомляє.

Постумова. Створено й виділений блок (лінія зв'язку); лінія зв'язку промальована червоними кольорами (якщо копіювалася лінія зв'язку).

8. *Створити Вузол.*

Передумова. На робочому полі є хоча б одне перетинання ліній зв'язку.

Опис. Вузол можна створити при щічці правою кнопкою миші на перетинанні ліній зв'язку й виборі відповідного пункту контекстного меню.

Альтернатива. Якщо в цьому місці вузол уже створений, операція не виконується.

Постумова. Створено вузол на перетинанні ліній зв'язку.

9. *Повернути блок.*

Передумова. Блок виділений.

Опис. Повернути блок на 90° можна при натисканні «Ctrl+R» або виборі відповідного пункту контекстного меню блоку. Якщо блок має лінії зв'язку, вони повертаються слідом за блоком.

Постумова. Блок повернутий на 90° і виділений.

Сутність структурного підходу до розробки ПС полягає в декомпозиції (розбивці) системи на функції, що автоматизуються, які у свою чергу діляться на підфункції, на завдання й так далі. Процес декомпозиції триває аж до визначення конкретних процедур. При цьому система, що автоматизується зберігає цілісне уявлення, у якому всі складові компоненти взаємозалежні.

В основі структурного методу лежать такі загальні принципи:

– розбивка системи на безліч незалежних задач, доступних для розуміння й рішення;

– ієрархічне упорядкування, тобто організація складових частин проблеми в деревоподібні структури з додаванням нових деталей на кожному рівні.

Кожен додаток, написаний для Windows, має одне головне вікно, що складається із двох компонентів – фрейму та клієнтської (робочої) області. Воно може бути як стандартним вікном Windows (наприклад, при побудові SDI- і MDI-додатків), так і звичайним блоком діалогу. У кожному разі всередині головного вікна додаток надає користувачеві всю необхідну для роботи інформацію, що представляє собою не тільки дані (блоки й зв'язки між

ними), але й елементи інтерфейсу – меню, смуги прокручування, панелі інструментів, рядок стану і т.д.

Додатки, що базуються на блоках діалогу, не мають ніякого меню, крім системного, не можуть відкривати/зберігати документи. SDI- і MDI-додатки дозволяють працювати з файлами, причому в SDI-додатку можна в кожен момент часу мати відкритим тільки один документ, а в MDI – декілька. Оскільки розроблювальне ПЗ повинне мати можливість зберігати і відновлювати топологію схеми, але при цьому в кожен момент часу може бути зібрана тільки одна схема (тому що вона повинна відповідати реальним об'єктам), то при виборі типу додатка рекомендується зупинитися на SDI-додатку.

В основі SDI-додатка лежить архітектура «документ/подання». Вона ґрунтується на трьох глобальних поняттях – фрейм, документ і подання. У самому загальному випадку, під документом розуміють ті дані, з якими працює додаток. Це може бути текст, картинка і т.п. Відображення цих даних на екрані здійснюється у фреймі документа. Розроблювачі бібліотеки MFC створили для фреймів спеціальні класи вікон – подання, які відображають дані документа і управляють взаємодією користувача з ними.

Розглянута архітектура ґрунтується на створенні відповідних фреймів документів, що містять спеціальні вікна – подання. Крім того, фрейм здійснює координацію взаємодії між поданнями різних типів, направляючи їм команди й одержуючи від них повідомлення.

Архітектура «документ/подання» розділяє між собою документи і їхні зображення. Документ представлений у вигляді об'єкта, що забезпечує простір для зберігання даних у пам'яті й відповідає за такі операції, як запис і читання документа з диска. Очевидно, що більша частина таких операцій виконується самим додатком, а не класами бібліотеки MFC. Крім того, для кожного використовуваного подання також створюється окремий об'єкт, що являє собою екранне вікно, відповідальне за взаємодію з користувачем й об'єктом документа, печатка й т.д. Спільну роботу перерахованих об'єктів координує фрейм, що у більшості випадків є головним вікном додатка.

Коли запускається додаток під Windows, користувач взаємодіє з документами за допомогою їхніх зображень у фреймах. Фрейми документів мають два основних компоненти: властиво фрейм і його вміст. Для них бібліотека MFC використає два різних класи. Клас фрейму управляє безпосередньо фреймом, а клас подання – його вмістом. При цьому вікно подання є дочірнім стосовно фрейму, тобто розміщається в його робочій області. Сам фрейм документа є вікном SDI-додатка.

Таким чином, для каркаса розроблювального додатка знадобляться чотири класи: клас фрейму (для зв'язку документа з його поданнями), клас документа (для зберігання даних, що стосуються топології схеми) і два класи подання (один – для відображення бібліотеки блоків, другий – для відображення топології схеми). Кожен блок також можна представити у вигляді окремого класу. Проаналізувавши діаграму варіантів використання

можна помітити, що існує певна сукупність методів (а виходить, і атрибутів), однакових для всіх блоків. Тому можна створити абстрактний базовий клас, спадкоємцями якого й будуть класи блоків. Тоді на початковому етапі розробки ПЗ можна створити кілька класів блоків, відробити на них основні методи, а потім уже розширювати бібліотеку.

Використовуючи поняття мови UML можна визначити, що класи документа (у розроблювальному додатку це клас CDockWndDoc), базовий клас (CBase) і його спадкоємці (CGenerator – генератор CSwitch – ключ; їх можна об'єднати в пакет Бібліотека) є сутностями; клас фрейму (CFrameWnd) – керуючим класом і класи представлень (CTreeElements й CDockWndView; їх можна об'єднати в пакет Представлення) – класами представлення. Діаграми класів, у яких показані відносини між перерахованими вище класами, представлені на рис. 6.29 й рис. 6.30.

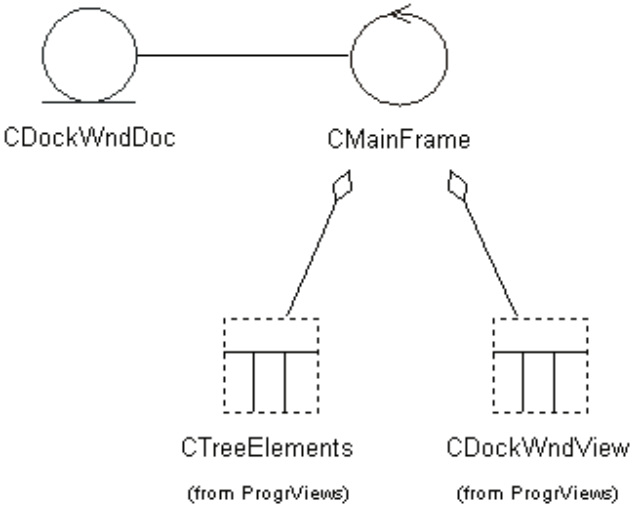


Рис. 6.29.

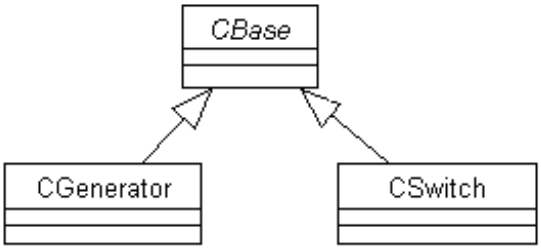


Рис. 6.30.

Оскільки кількість блоків кожного типу повинне відповідати кількості керованих об’єктів, то програма повинна давати можливість створювати строго фіксовану кількість екземплярів кожного класу бібліотеки. Для зручності роботи зі створеними блоками (а саме зчитування/запису їх із/в документ) бажано щоб екземпляри, що відносяться до одного класу, були якось зв’язані між собою. Тому пропонується для кожного типу блоків створити масив екземплярів відповідного класу, причому його розмірність повинна відповідати кількості фізичних об’єктів, що представляються цими блоками. Ці масиви (їх буде стільки, скільки типів блоків) можна зберігати в основному об’єкті представлення (CDockView). Тоді при створенні блоку буде створюватися не екземпляр класу, а тільки вікно, що відповідає йому (самі екземпляри класу будуть існувати протягом роботи додатка). (Можна, звичайно, використати динамічні масиви, і при створенні блоку просто додавати в кінець відповідного масиву новий екземпляр класу. Але тоді створення (а також видалення) блоку може зайняти багато часу, оскільки процес додавання нового елемента в масив складається з декількох етапів: резервування пам'яті під новий масив, копіювання елементів поточного масиву в новий, додавання в кінець нового масиву елемента, звільнення пам'яті.).

Для зберігання кількості блоків кожного типу будемо використати іменовані константи (наприклад, CountGen – для числа блоків «Генератор», CountSwtchs – «Ключ» і т.д.), описані в окремому файлі (total.h). Також у цьому файлі можна зберігати загальне число типів блоків (TotalCount, також іменована константа) і визначення типу (використовується для того, щоб кожному блоку поставити у відповідність не індекс, а ім'я; наприклад, ChsdElmnt {Generator = 0, Switch, Nothing} – змінні типу ChsdElmnt).

Далі варто вирішити питання створення зв’язків. У бібліотеці MFC немає класу, що інкапсулює роботу з лініями, тому створимо власний клас CLine. Він буде зберігати координати початкової й кінцевої крапок відрізка; покажчики на з’єднані з початковою й кінцевою крапками відрізку; покажчики на блоки, які з’єднує відрізок (найчастіше один з них приймає нульове значення, тому що найчастіше лінія зв’язку складається з декількох відрізків). Знаючи початкові й кінцеві крапки відрізків лінії зв’язку, можна за допомогою контексту пристрою, пов’язаного з вікном робочого поля, накреслити лінію зв’язку. Однак додаток повинен мати можливість не тільки створювати лінію зв’язку, а й працювати з нею – переміщати й видаляти. При переміщенні одного з відрізків лінії зв’язку повинні змінюватися й координати кінців пов’язаних з ним відрізків, тому виникає необхідність зберігання в об’єкті CLine покажчиків на з’єднані з його початковою і кінцевою крапками відрізки. Видалення зв’язку відбувається при виділенні одного з відрізків і наступному натисканні кнопки Delete, тому тут також потрібні покажчики на з’єднані з даним відрізком відрізки. Також при видаленні лінії зв’язку блоки, які вона з’єднувала, повинні сповіщатися про те, що вони вже не зв’язані. Тому клас CLine повинен зберігати покажчики на блоки, які даний відрізок з’єднує.

Оскільки натискання на лінії зв'язку класифікується як просто натискання на робочому полі, для розпізнання цієї події треба використати спеціальний алгоритм. Наприклад, знаючи, що кольори робочого поля й кольори лінії зв'язку не збігаються, можна одержати кольори пікселя (або сукупності пікселів, що перебувають поблизу положення курсору), на якому натиснув користувач, зрівняти його з кольорами робочого поля, і, якщо вони не збігаються, виходить, обрана лінія зв'язку. Однак при цьому неможливо ідентифікувати обрану лінію зв'язку, оскільки відомі тільки координати крапки, на якій клацнув користувач. Для рішення цієї проблеми пропонується наступний підхід. Можна створити клас бінарного дерева CTree й, оскільки всі відрізки лини зв'язку ортогональні, два його екземпляри – один для горизонтальних відрізків, другий – для вертикальних. При створенні відрізка лінії зв'язку його напрямок відомо, тому цей відрізок (точніше покажчик на нього) можна додати до відповідного бінарного дерева. Критерієм для знаходження положення при додаванні відрізка до дерева може служити його координата по осі *Oy* для горизонтальних відрізків (оскільки значення *y*-координат початкової й для кінцевої крапок відрізка збігаються) і по осі *Ox* – для вертикальних. Якщо елемент із таким значенням постійної координати в дереві вже існує, пошук місця для його додавання ведеться по другій координаті початкової крапки лінії. При цьому пріоритет віддається лініям лежачої вище (для горизонтальних) або лівіше (для вертикальних). Далі при розпізнанні події «Користувач клацнув на відрізку» можна опитуванням кольори пікселів зверху, знизу, праворуч і ліворуч з'ясувати напрямок обраного відрізка. Потім можна звернутися до відповідного цього напрямку бінарному дереву, задавши для пошуку координату крапки, що залишається постійної для даного напрямку відрізка (для горизонтального – *y*-координату, вертикального – *x*). Далі, оскільки можуть існувати кілька відрізків, що належать різним лініям зв'язку й лежать на одній прямій, необхідно провести пошук в обраній галузі дерева по другій координаті крапки, на якій клацнув користувач. І тільки тепер можна точно ідентифікувати лінію зв'язку, що виділив користувач.

Альтернативою створенню бінарних дерев є використання динамічних масивів, однак, додавання, а самий головне – пошук елементів у бінарному дереві здійснюються набагато швидше. Тому в додатку будемо використати клас CTree.

Таким чином, у програму додалися 2 класи – CLine й CTree. Відношення між ними й вищ ерозглянутими класами показано на рис. 6.31.

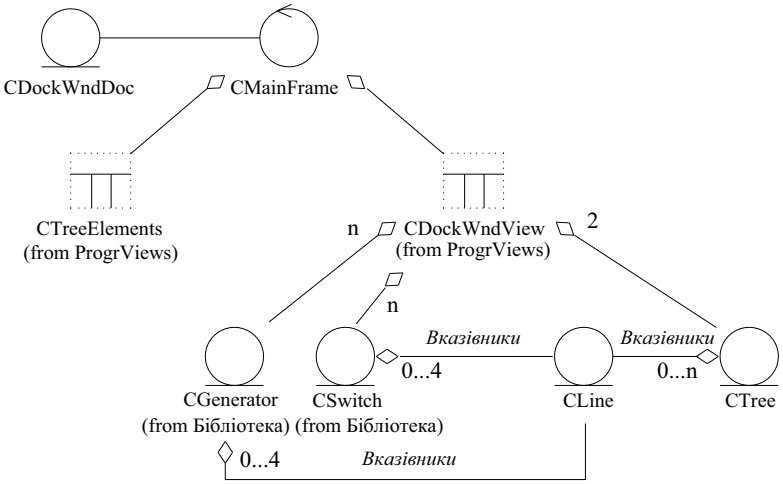


Рис. 6.31.

Клас робочого поля (CDockWnd) містить 2 дерева, елементами яких є покажчики на всі створені в ході роботи з додатком відрізки ліній зв'язку. Клас кожного із блоків містить покажчики на всі пов'язані із блоком відрізки (їх може бути від 0 до 4, оскільки кожен блок має всього 4 сторони). Крім дерев, до складу класу CDockWnd входять n_1 об'єктів класу CSwitch й n_2 – класи CGenerator.

Для класів, що представляють різні типи блоків, існує сукупність методів і атрибутів, виконуваних і використовуваних аналогічно, то був створений абстрактний клас CBase, спадкоємцями якого є класи CGenerator і CSwitch (це показано в статичній моделі додатка). Сам клас CBase є спадкоємцем класу CStatic бібліотеки MFC і містить атрибути й методи, показані на рис. 6.32. Клас CGenerator є стандартним нащадком класу CBase і містить тільки його атрибути й методи. Об'єкти класу CSwitch (який також є нащадком класу CBase) має ще 2 додаткові стани: «ключ замкнутий» і «ключ розімкнений».

На рис. 6.32 атрибути мають наступне призначення:

m_EnConnect – число, що дозволяє зв'язки (використовується для визначення чи має можливість кожна зі сторін блоку мати вхідний або вхідний зв'язок);

m_CanConnect – змінна, котра показує чи потрапив курсор в область можливого зв'язку;

m_Angle[2] – змінні, що кодують положення блоку;

m_PrevPoint – змінна для зберігання попередніх координат лівого верхнього кута блоку; використовується при обчисленні відстані, на яке перемістився блок;

m_CurPoint – змінна для зберігання поточних координат лівого верхнього кута блоку відносно робочого поля;

m_IsMoving – змінна для зберігання відомостей про переміщення об'єкта (0 – блок коштує на місці, 1 – переміщається);

m_pLines[4] – масив покажчиків на крайні відрізки зв'язку, з'єднані зі сторонами блоку.

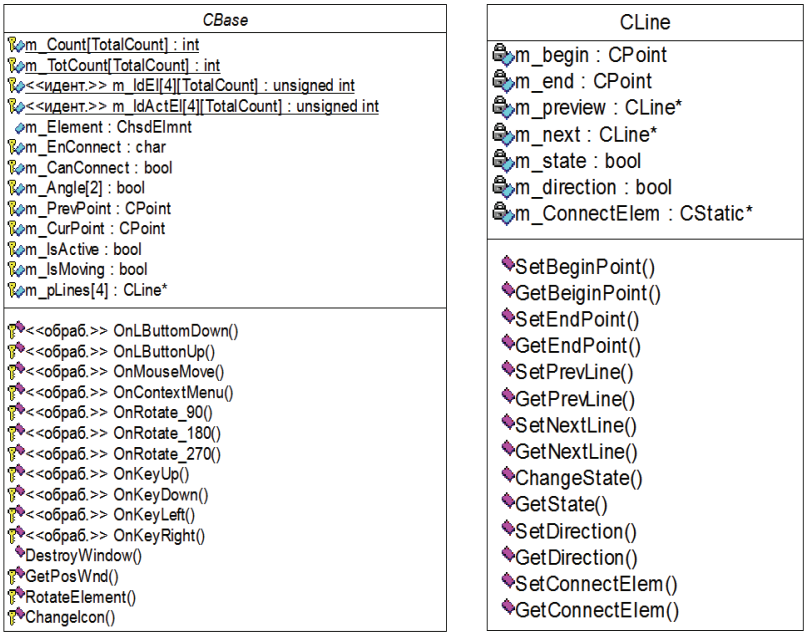


Рис. 6.32.

Інші атрибути пов'язані з тим, що для класів, що представляють різні типи блоків, немає необхідності в кожному екземплярі зберігати дані, що стосуються зображення блоків, а також кількості вже створених вікон. Ці дані повинні бути зв'язані безпосередньо із класом. Для цього можна використати змінні із кваліфікатором static. Однак такі змінні створюються тільки один раз при визначенні класу, і тому при спадкуванні класів вони не успадковуються, а є загальними для всіх класів-нащадків. Тому що з кожним класом-нащадком класу CBase повинні бути зв'язані свої іконки й своє число створених вікон, то пропонується в базовому класі створити масиви відповідних змінних (розмірність масиву буде дорівнює числу блоків у бібліотеці (іменована константа TotalCount); елементам масивів з однаковими індексами будуть відповідати дані щодо одного класу-нащадка). Всі методи, що мають стереотип <<оброб.>> є функціями-оброблювачами однойменних повідомлень, що посилають вікну блоку. Метод ChangeIcon відповідає за

зміну зображення блоку, RotateElement – за зміну змінних, що кодують положення блоку, і числа, яке дозволяє зв'язки.

Оскільки одним з базових принципів об'єктно-орієнтованого програмування є інкапсуляція, то всі атрибути даного класу є приватними змінними, а методи орієнтовані на роботу з ними об'єктами інших класів. Призначення більшості атрибутів докладно описано при описі класу CLine у статичній моделі.

6.8. Динамічна модель та аналіз повідомлень

На етапі динамічного моделювання розглядаються динамічні, або поведінкові аспекти системи. Динамічна модель є одночасно міжоб'єктною (що описує взаємодію об'єктів) і внутрішньооб'єктною (що характеризує, як залежний від стану об'єкт визначається кінцевим автоматом і зображується на діаграмі станів). Внутрішньооб'єктна сторона динамічного моделювання має відношення до кінцевих автоматів та діаграм станів.

Взаємодія між об'єктами описується за допомогою UML-діаграм кооперацій. Головна особливість діаграми кооперації полягає в можливості графічно представити не тільки послідовність взаємодії, але й всі структурні відносини між об'єктами, що беруть участь у цій взаємодії. Повідомлення на діаграмі кооперації специфікує комунікацію між двома об'єктами, один із яких передає іншому деяку інформацію. При цьому перший об'єкт очікує, що після одержання повідомлення другим об'єктом піде виконання деякої дії. Таким чином, повідомлення є причиною або стимулом для початку виконання операцій, відправлення сигналів, створення й знищення окремих об'єктів. Зв'язок забезпечує канал для спрямованої передачі повідомлень між об'єктами від об'єкта-джерела до об'єкта-одержувача.

У динамічній моделі виконується аналіз повідомлень, якими обмінюються об'єкти при виконанні кожного з варіантів використання. Для цього використовуються діаграми кооперації для кожного із прецедентів (рис. 6.33).

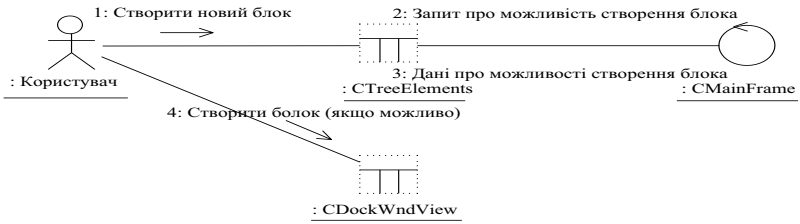


Рис. 6.33.

У процесі створення нового блоку беруть участь 3 об'єкти: при спробі створення користувач перетаскує іконку блоку з бібліотеки (об'єкт подання CTreeElements), далі об'єкт подання посилає повідомлення головному фрейму з питанням, чи є ще дані блоки в бібліотеці, і якщо головний фрейм

«дозволив» створити блок при перетаскуванні іконки на робоче поле поданню CDockView посилає повідомлення «Створити блок». У цьому повідомленні повинна втримуватися інформація щодо того, який саме блок треба створити. Зберігання цієї інформації покладає на об'єкт CMainFrame. Таким чином, одним з атрибутів цього класу буде індекс (або ідентифікатор) обраного для створення користувачем блоку.

В описі варіанта використання «Створити Блок» з'ясовано, що створення блоку відбувається в момент відпускання лівої кнопки миші на робочому полі, якщо до цього елемент був обраний у поданні бібліотеки. Тому функцію створення блоку (у програмі вона названа CreateNewElement) можна викликати у функції-оброблювачі повідомлення, що генерується при відпусканні лівої кнопки миші в робочому полі (OnLButtonUp). Аргументом функції CreateNewElement буде константа (типу ChsdElmnt), що ідентифікує обраний блок і зберігається в об'єкті CMainFrame.

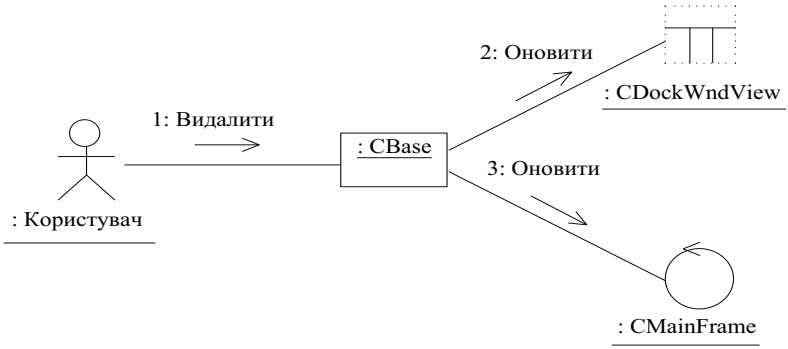


Рис. 6.34.

При видаленні блоку відправляється відповідне повідомлення блоку, після чого блок посилає повідомлення поданню й головному фрейму із проханням їхнього відновлення (для об'єкта CDockView воно полягає у видаленні ліній зв'язку, якими був зв'язаний даний блок з іншими; для CMainFrame – у відновленні інформації про кількість створених на робочому полі на даний момент блоків).

Для видалення об'єкта (рис. 6.34) можна перевизначити функцію DestroyWindow для блоку CBase таким чином, щоб при його виклику послідовно здійснювалося відновлення подання, потім даних щодо кількості блоків і безпосередньо сам виклик функції видалення вікна об'єкта (при цьому сам об'єкт не зникає).

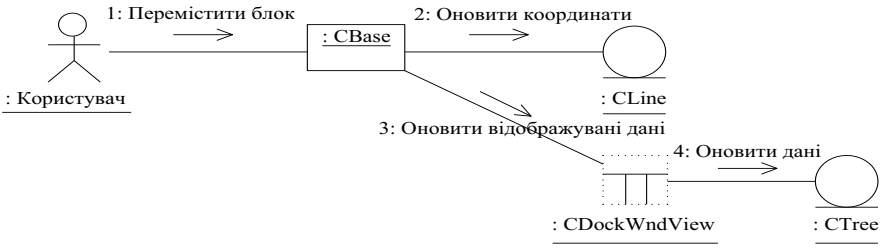


Рис. 6.35.

При переміщенні блоку (рис. 6.35) посилає відповідне повідомлення екземпляру блоку. Коли переміщення закінчене, об'єкт блоку посилає головному поданню повідомлення «Обновити» (перемалювати блок і вихідні (вхідні) з (в) його лінії зв'язку).

Розглянемо докладніше процес переміщення. Він складається з декількох етапів: спочатку користувач натискає ліву кнопку миші при розміщенні курсору над блоком, потім при затиснутій лівій кнопці миші переміщає курсор й, нарешті, відпускає ліву кнопку миші. Таким чином, для реалізації даного варіанта використання треба розглядати функції-оброблювачі трьох подій, що посилають вікну блоку: натискання лівої кнопки миші (OnLButtonDown), переміщення курсору (OnMouseMove) і відпускання лівої кнопки миші (OnLButtonUp). Процес обчислення відносного переміщення блоку має свої особливості: оскільки повідомлення, що генеруються при зміні положення курсору, посилають екземпляру блоку, координати поточного положення курсору обчислюються щодо лівого верхнього кута блоку. Тому для того щоб визначити, куди переміщується об'єкт (вправо, вліво, нагору або вниз), необхідно щораз порівнювати координати поточного положення курсору з попереднім. Наприклад, якщо в один момент координати курсору були (16, 16), а в наступний – (19, 17), те блок треба перемістити на 3 пікселя вправо та на 1 ліворуч (вісь *Ox* спрямована ліворуч праворуч, *Oy* – зверху вниз).

При перемальовуванні блоку на головному поданні треба знати координати його вершин щодо вікна подання. Оскільки розміри всіх блоків фіксовані (32×32), те досить знати координати однієї з його вершин, наприклад, лівої верхньої. Для цього можна один раз при натисканні лівої кнопки миші обчислити ці координати (використовуючи метод класу CDockView ScreenToClient; аргументом цього методу є адреса на прямокутник, що містить координати дочірнього вікна, а результатом виклику – зміна цих координат щодо зухвалого вікна), а далі при переміщенні курсору змінювати їх відповідно до вищеописаного алгоритму. При цьому поки затиснута ліва кнопка миші, курсор повинен бути захопленим вікном блоку (тобто приймати всі повідомлення від миші повинне тільки вікно блоку). Для візуального контролю переміщення можна використати нестандартний курсор, що

з’являється при початку переміщення й зникає в момент відпускання лівої кнопки миші.

Таким чином, для виконання переміщення блоку клас CBase повинен мати 3 додаткових атрибути: координати попереднього положення лівої вершини (m_PrevPoint типу CPoint), що тече положення (m_CurPoint типу CPoint) і прапор, що показує переміщається чи об’єкт ні (m_IsMoving типу bool). Сам процес переміщення можна описати наступним псевдокодом:

```
при натисканні лівої кнопки миші
виділити об’єкт
зберегти координати лівої верхньої вершини об’єкта в змінної m_PrevPoint
дозволити переміщення (установити значення m_IsMoving в «істина»)
при переміщенні миші
захопити курсор і змінити його вид
обчислити різниці між однойменними координатами крапки поточного положення курсору й крапки, що зберігає попередні координати (m_PrevPoint)
додати отримані значення до відповідних координат крапки, що зберігає координати поточного положення блоку (m_CurPoint)
зберегти поточні координати курсору в змінної m_PrevPoint
при відпусканні лівої кнопки миші
відпустити курсор і змінити його вид на стандартний
якщо в даного блоку є лінії зв’язку
змінити їхньої координати
перемалювати лінії зв’язку
перемалювати вікно відповідно до нових координат (значення координат лівого верхнього кута зберігається в змінної m_CurPoint, правого нижнього - кожна на 32 більше)
заборонити переміщення (установити значення m_IsMoving в «неправду»)
```

При аналізі цього псевдокоду може здатися, що змінна m_IsMoving не потрібна. Насправді вона використовується для відділення процесу переміщення від інших процесів (виділення об’єкта, створення зв’язку).

РОЗДІЛ 7. МОДЕЛЬНІ ДОСЛІДЖЕННЯ ЯКОСТІ РОБОТИ ПРОГРАМНО-АПАРАТНИХ ЗАСОБІВ МОНІТОРИНГУ ТА КЕРУВАННЯ СЕЕС

7.1. Формальний опис елементів автоматизованих робочих місць оператора електроенергетичної системи

Сучасні автоматизовані системи керування електроенергетичними установками − це людино-машинні системи, в яких за допомогою технічних засобів забезпечується збір, накопичення, обробка інформації, формування оптимальної стратегії управління та видача результатів людині чи групі людей, що приймають рішення з управління. З метою забезпечення можливості взаємодії людини з комп'ютером в інтерактивному режимі з'являється необхідність реалізувати в рамках АСК автоматизоване робоче місце, яке являє собою сукупність різних взаємопов'язаних підсистем. Для отримання ефективного результату розробки користувальницького інтерфейсу використовується системний підхід, при якому користувач розглядається як інтелектуальна частина системи «людина – програмний продукт». При цьому повинні бути проаналізовані питання взаємодії людини з комп'ютером з точки зору ергономіки, враховані психологічні та психофізіологічні характеристики користувача з метою підвищення ефективності інтерфейсу користувача, а екранні органи управління задовольняти ряду вимог. На ранніх етапах проектування АРМ оператора складність розробки користувальницького інтерфейсу обумовлена неможливістю підключення реального обладнання, а в разі наявності такої можливості − небезпечністю штучного створення різноманітних, у тому числі і аварійних ситуацій. Неправильне функціонування підсистеми управління або її частини може принести значний збиток. Використання моделювання та імітаційного стенда, реалізованого на декількох комп'ютерах і яке містить набір основних підсистем автоматизованої системи управління, є найбільш вдалим методом вирішення цієї проблеми. До складу імітаційного стенду включена підсистема візуалізації, і результат роботи моделі відображається на моніторі так само, як і в реальній системі в обумовленому моделями обсязі. Недоліком подібної моделі є відсутність формального опису підсистеми візуалізації. Розглянутий в роботі [57] метод синтезу математичних моделей логіко-динамічних процесів контролю та управління може бути розширений і використаний для аналітичного опису користувацьких інтерфейсів програм. Наявність аналітичної моделі інтерфейсу користувача дозволить оцінити його повноту і несуперечність, визначити відносини і типи зв'язків між підсистемами управління та візуалізації, а також описати функціональні можливості елементів окремих підсистем і алгоритми обробки даних. Перевагою розглянутого далі підходу є можливість використання різних рівнів декомпозиції для опису підсистем і елементів АСК, що дозволяє виділити найбільш важливі аспекти опису на різних стадіях проектування та аналізу системи.

Користувальницький інтерфейс будь-якої складної системи складається з комбінації типових елементів введення і виведення даних у різних формах − текстовій, графічній, звуковій. Використання спеціалізованих графічних елементів (візуальних метафор) для відображення контрольованих параметрів надає інформації принципово нову якість динамічності, тобто здатності до швидкої перебудови і безпосередньому її використанні в розв'язуваних задачах управління. У разі необхідності є можливість швидко представити будь-яку вибірку з цієї інформації, наприклад, шляхом побудови адаптивних користувальницьких інтерфейсів, використанням колірного кодування інформації та анімованих зображень. Інтерфейс автоматизованого робочого місця оператора електроенергетичної системи розглядається як множина функціональних структур, під якими розуміється закон зв'язку між функціональними елементами процесу діяльності. Під елементами розуміються локалізовані в просторі та часі активності в сенсі спрямованих дій або фаз процесу діяльності. У системному описі користувача інтерфейсу опорними точками служать: типологія елементів (введення даних, текстове або графічне відображення інформації); типи зв'язків між елементами, що характеризують порядок взаємодії елементів (вхідні зв'язку для графічних елементів відображення інформації, вихідні зв'язку для графічних елементів управління).

На рис. 7.1 представлена модель АРМ оператора, розроблена в середовищі Matlab для моделювання процесу управління і моніторингу параметрів суднового дизель-генератора, який входить до складу електроенергетичної системи. Серед графічних елементів користувальницького інтерфейсу можна виділити основні і похідні, головні і другорядні; різним рівням функціональної структури відповідають і різні групи елементів.

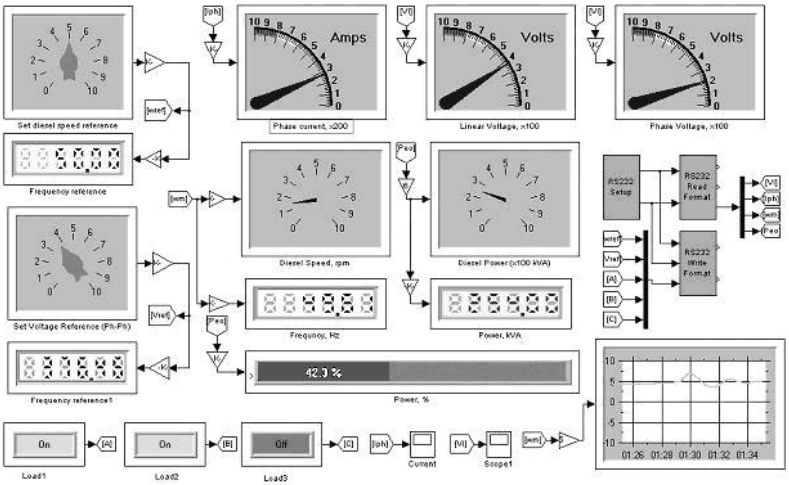


Рис. 7.1.

За допомогою екранних органів управління на моделі контролюючого обладнання (в даному випадку це два блоки «Generic Knob» і три кнопки «Square Red») задаються необхідні значення частоти напруги (обертів дизеля) і величина лінійної напруги на обмотках статора синхронного генератора. Екранні кнопки використовуються для підключення до генератора різного навантаження. Далі виконується моніторинг таких параметрів, як струм і напруга статора генератора, оберти дизеля, частота напруги мережі і потужність, споживана навантаженням (абсолютне значення потужності і відносне, у відсотках від номінальної потужності генератора) шляхом візуалізації зазначених величин за допомогою віртуальних об'єктів на екрані (семисегментний індикатор, стрілочний прилад, лінійний індикатор). Також використовується віртуальний осцилограф для відстеження характеру зміни контрольованих параметрів об'єкта.

На рис. 7.2 представлений елемент користувальницького інтерфейсу, що виконує роль екранного органу управління опорним значенням напруги генератора.

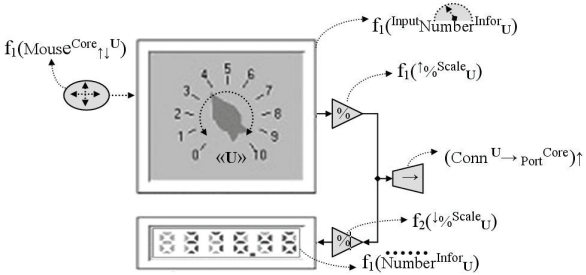


Рис. 7.2.

Оператор за допомогою маніпулятора типу «миша» має можливість змінити положення обертового регулятора. Цю дію можна описати виразом $f_1(\text{Mouse}^{\text{Core}}\uparrow\downarrow^{\text{U}})$ в якому Mouse означає тип маніпулятора «миша», Core – параметр, що задається оператором і використовується обчислювальним ядром мікроконтролера для розрахунку і моделювання керуючого впливу на об'єкт управляння (систему збудження синхронного генератора). Символи «↑↓» означають, що зміна параметрів може відбуватися, як у сторону збільшення, так і у сторону зменшення значення.Символ U несе інформацію про параметр, що задається – напругу генератора.

Встановлене за допомогою екранного органу управління значення масштабується. Ця операція описується виразом $f_1(\uparrow\%^{\text{Scale}}{}_{\text{U}})$, в якому позначення $\%^{\text{Scale}}$ описує виконання процедури масштабування, символ «↑» у верхньому індексі означає, що отримане значення збільшується. Вираз $f_2(\downarrow\%^{\text{Scale}}{}_{\text{U}})$ описує операцію масштабування, однак результуюче значення зменшується. В даному випадку це необхідно щоб підготувати дані для відображення на віртуальному індикаторі – семисегментному індикаторі, який описується виразом

$$f_1(\text{Number}^{\overset{\cdots\cdots}{\text{Infor}}}{}_{\text{U}}).$$

В цьому виразі Infor означає, що віртуальний об'єкт на екрані виконує функцію відображення інформації про значення напруги генератора (U) за допомогою 6-ти позиційного числового індикатора (Number). Це необхідно для додаткового контролю оператором процесу введення і перетворення даних, а також установки параметра із заданою точністю. Масштабування виконується для зручності передачі даних між вузлами мережі з використанням промислових інтерфейсів і протоколів.

Процес роботи оператора з екранним елементом управління (обертовим регулятором) для установки необхідного значення напруги генератора можна представити у вигляді:

$$\text{f}_1(\text{Mouse}^{\text{Core}\uparrow\downarrow\text{U}}) \to \text{f}_1(^{\text{Input}}\text{Number}^{\text{Infor}}{}_{\text{U}}) \to \text{f}_1(^{\downarrow}\%^{\text{Scale}}{}_{\text{U}}) \begin{cases} \to (\text{Conn}^{\text{U}} \to \text{Port}^{\text{Core}})\uparrow \\ \to \text{f}_2(^{\uparrow}\%^{\text{Scale}}{}_{\text{U}}) \to \text{f}_1(\text{Numb}^{\overset{\cdots\cdots}{\text{er}}\text{Infor}}{}_{\text{U}}) \end{cases}.$$

Вираз $(\text{Conn}^{\text{U}} \to \text{Port}^{\text{Core}})\uparrow$ описує вхідний зв'язок (символ «↑») між екранним елементом управління, за допомогою якого виконується керування напругою генератора (Conn^{U}), і елементом, що представляє вхідний порт мікропроцесорної системи ($\text{Port}^{\text{Core}}$), який виконує обробку введеного оператором значення і формування керуючого впливу на локальні системи управління.

На рис. 7.3, а, представлений елемент управління «лінійний індикатор», який використовується для відображення інформації про величину активної потужності, яка віддається генератором на загальне навантаження, у вигляді шкали (у відсотках від номінальної потужності генератора), а на рис. 7.3, б – елемент управління «екранна кнопка», яка представляє перемикач із двома положеннями вкл/викл. і використовується для управління навантаженням.

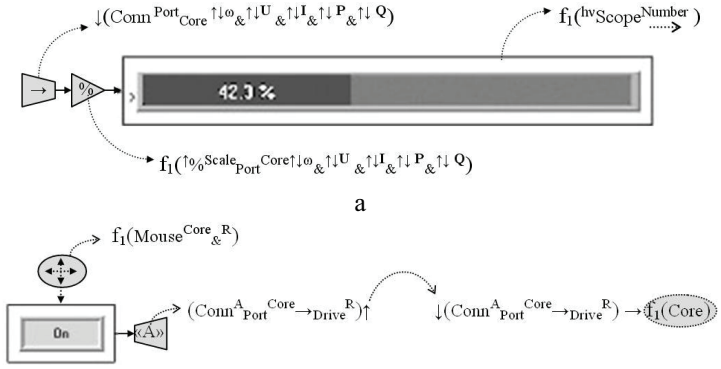


Рис. 7.3.

Екранний елемент відображення інформації «лінійний індикатор» пов'язаний (Conn) з вихідним портом мікропроцесорної системи ($Port_{Core}$). Крім інформації про активну потужність (P), вхідний зв'язок також містить інформацію і (&) про частоту (ω), напругу (U), струм (I), реактивну потужність (Q), значення яких оператор може як збільшувати, так і зменшувати («↑↓»). Символ «↓» несе інформацію про направлення даних і означає, що дані передаються в елемент управління для їх відображення. Аналітичний запис $f_1(^{hv}Scope^{Number}_{\cdots\cdots>})$ означає, що інформацію у вигляді числового значення (Number) і шкали, що дискретно змінюється ($\cdots\cdots>$), оператор отримає візуально, через зоровий аналізатор (^{hv}Scope). Збільшення значення параметра, що відображає індикатор, призводить до заповнення робочої області індикатора системним кольором в напрямку зліва направо.

Елемент управління «екранна кнопка» дозволяє оператору виконувати операції (включення, відключення і т. д.) з окремими об'єктами обладнання та групами об'єктів. Аналітичний запис дій оператора при роботі з цим елементом управління містить інформацію про тип навантаження (активне навантаження позначається символом R, індуктивне − символом L, активно-індуктивне − символами RL і т.д.), типі маніпулятора, за допомогою якого виконується управління (Mouse – «миша»), і підсистему, в яку передається інформація про стан екранної кнопки (Core – мікропроцесорна система управління).

Таким чином, запропоноване аналітичне подання екранних елементів управління і дій оператора містить опис трьох основних частин інтерфейсу користувача:

– візуальне (звукове, тактильне) оформлення, яке відповідає за подання інформації оператору про параметри, що вводяться і контролюються, в текстовому, графічному, мнемонічному та інших видах;

– функціональні можливості системи, що включають набір можливостей для виконання професійної діяльності;

– техніки взаємодії оператора з системою (миша, клавіатура, трекбол, джойстик, сенсорний екран та ін.).

При цьому функціональність системи розглядається в тісному зв'язку з призначеним для користувача інтерфейсом і елементами взаємодії користувача з системою.

На рис. 7.4 представлено опис логіко-динамічного процесу вимірювання діючого значення лінійної напруги генератора, і візуалізації виміряного значення за допомогою віртуального осцилографа в середовищі Matlab Simulink.

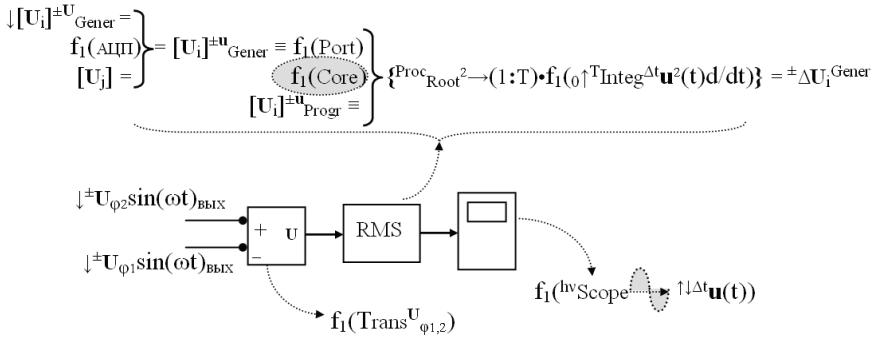


Рис. 7.4.

Аналітичний вираз $f_1(Trans^{U}{}_{\varphi 1,2})$ описує функціональну структуру вимірювального трансформатора напруги. При розгляді цього елемента системи у вигляді «чорного ящика», приховуються його деталі реалізації. Цей елемент містить також аналого-цифровий перетворювач, арифметичний пристрій, що виконує розрахунок різниці миттєвих значень фазних напруг ($U_{\varphi 1}$ и $U_{\varphi 2}$) для визначення величини лінійної напруги (u=$U_{\varphi 1,2}$). Елемент «RMS» виконує розрахунок діючого значення лінійної напруги U_{AB}.

Аналітичний запис $^{Proc}Root^2$ описує процедуру знаходження квадратного кореня. Процедура інтегрування (Integr) в границях одного періоду ($_0\uparrow^T$) напруги (**u**(t)) в обраній нотації описується виразом {(1:T)•$f_1(_0\uparrow^T Integr^{\Delta t}\mathbf{u}^2(t)d/dt)$}. Фігурні дужки вказують на програмну реалізацію алгоритму розрахунку, за аналогією з мовою Сі, в якій тіло функції записується в аналогічні дужки. Чисельне інтегрування виконується методом прямокутників з рівномірним кроком сітки Δt, який дорівнює періоду дискретизації АЦП. Таким чином, розглянутий варіант аналітичного запису дозволяє перейти від моделі «чорного ящика» до моделі «сірого ящика». Аналого-цифрове перетворення напруги виконується вбудованим в мікроконтролер периферійним пристроєм (f1(АЦП)). Отриманий в результаті масив значень $[U_i]^{\pm u}{}_{Gener}$ по внутрішній шині надходить (f_1(Port)) до арифметико-логічного пристрою, який разом з регістрами загального призначення, лічильником команд і деякими іншими елементами є ядром мікропроцесора (f_1(Core)). Далі виконується обробка масиву з даними про миттєві значеннях лінійної напруги генератора відповідно до алгоритму, який реалізовано в програмі ($[U_i]^{\pm u}{}_{Progr}$).

Розглянутий підхід дозволяє створити математичну модель АРМ оператора або його частини. Згідно з визначенням В.М. Глушкова, математична модель − це множина символічних математичних об’єктів і співвідношень між ними. У широкому сенсі до математичних моделей відносять будь-які моделі, що задовольняють визначенню В.М. Глушкова. Наприклад, вимірювання активної та реактивної потужності генератора, а також відображення

значень цих величин за допомогою графічних елементів користувальницького інтерфейсу, можна представити у вигляді комбінованої структурно-функціональної моделі (рис. 7.5).

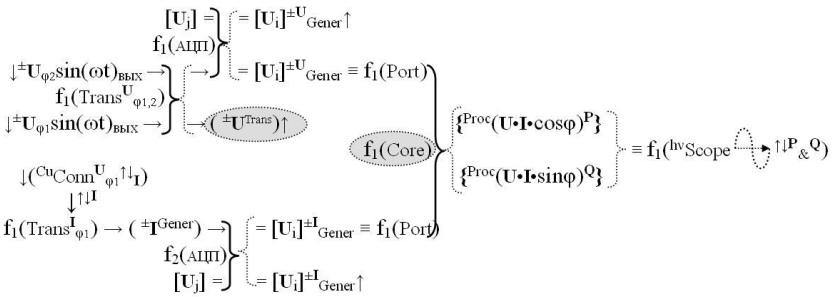


Рис. 7.5.

На рис. 7.6 представлено узагальнену модель АРМ оператора СЕЕС.

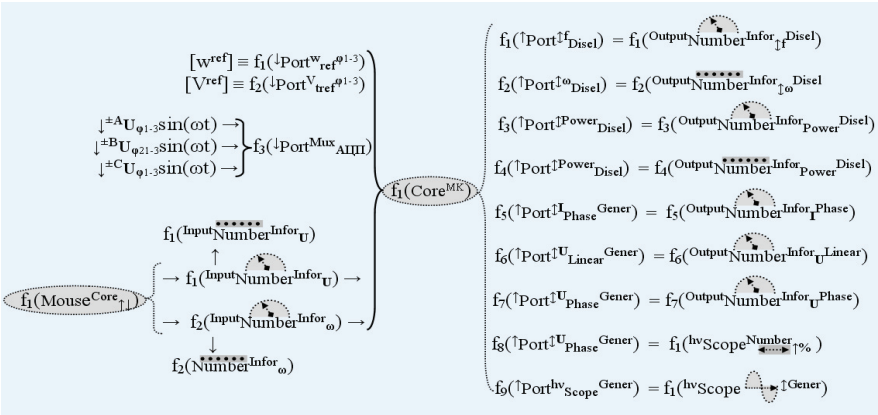


Рис. 7.6.

Математичне моделювання відіграє значну роль у перевірці адекватності моделей шляхом логічного аналізу. Опис системи у вигляді логіко-динамічного процесу перетворення інформації дозволяє виконати перевірку якісної адекватності моделі істинності гіпотез про зв'язки між параметрами і змінними моделі, внутрішню несуперечливість, коректність перетворення інформації, зокрема заміну безперервних процесів дискретними. Така перевірка є обов'язковим попереднім етапом і при встановленні адекватності кількісних моделей.

Розглянутий підхід має об'єктно-орієнтований стиль опису, містить інформацію про структуру та типи інформаційних зв'язків між елементами,

що є складовими системи. Представлені в явному вигляді позначення елементів користувальницького інтерфейсу і елементів перетворення інформації залишаються прозорими для розробника і містять інформацію про поведінку елементів, вимоги до їх взаємодії і функціональних можливостях. Використання аналітичного опису користувацького інтерфейсу дозволяє підвищити ефективність програмного продукту шляхом повного використання закладеної до програмного забезпечення функціональності. Запропонований підхід може бути використаний як для високорівневого проектування та опису взаємодії оператора з системою, так і для низькорівневого проектування та опису взаємодії елементів системи на різних рівнях декомпозиції.

7.2. Використання UML-діаграм станів для опису алгоритмів роботи компонентів програмного забезпечення

Завданням програмного забезпечення верхнього рівня є дистанційний контроль і управління електроенергетичною системою. Для реалізації всіх необхідних функцій ПЗ повинно мати засоби для створення мнемосхеми електростанції, моніторингу та управління СЕЕС в режимі реального часу, аналізу режимів роботи силових агрегатів і ПЗ. Вимоги до ПЗ для моніторингу та управління електроенергетичною системою можна розділити на три групи за діючим особам, які беруть участі у окремих етапах роботи з програмою (рис. 7.7) і умовно позначені як Оператор налаштування системи (виконує створення мнемосхеми, налаштування та коригування властивостей її компонентів), Оператор управління електростанцією (виконує моніторинг і управління енергосистемою в режимі реального часу) і Аналітик (виконує аналіз режимів роботи ПЗ і СЕЕС з метою їх оптимізації, після чого весь процес роботи з ПЗ – налаштування властивостей компонентів, робота в режимі реального часу і її аналіз – повторюється заново).

Головною дійовою особою в системі є Оператор управління електростанцією. Дії, які він має виконувати, визначають вимоги до складу бібліотеки компонентів і інтерфейсу користувача; є основою для збору даних, які використовуються для аналізу роботи системи. Далі прецеденти 1.1–1.8 розглянуто детальніше.

1. Графічне відображення процесів можна розглядати на декількох рівнях: відображення поточної структурної схеми (або мнемосхеми) електростанції; відображення електричних параметрів (потужностей, напруг, частот) і станів дискретних сигналів. Вимога відображення мнемосхеми електростанції визначає необхідність наявності візуальних метафор елементів електростанції – дизель-генератора, асинхронного двигуна, навантаження, трансформатора, шини (далі відповідні перерахованим елементам компоненти ПЗ наводяться з великої літери). Відображення електричних параметрів можна здійснити за допомогою наступних компонентів: Стрілочний і

Шкальний індикатори, Текстове поле і додаткові діалогові вікна для розширеного моніторингу параметрів дизеля і генератора; для індикації дискретних станів можна використовувати Світлодіоди.

2. Виконання пуску/зупинки дизель-генераторного агрегату пов'язано з необхідністю наявності компонента Кнопка (з фіксацією або імпульсна).

3. Захист ДГА є узагальненим поняттям і складається з захистів від зворотної потужності; втрати збудження; протікання максимального струму; зміни напруги поза допустимими межами; зміни частоти поза допустимими межами; перевантаження і недостатньої або граничної потужності, що віддається на загальне навантаження. Кожна функція повинна бути зіставлена з дискретним сигналом (або сигналами), стан якого буде змінюватися при виникненні аварійної або небажаної ситуації. Також ПЗ повинно містити засоби для установки граничних значень, при яких відбуватиметься ця зміна. Таким чином, на рівні ПЗ функції захисту ДГА можуть бути представлені одним компонентом, в якому можна задати інформаційний зв'язок з компонентом ДГА, включити або відключити окремі складові захисту і задати для кожної з них границі спрацьовування.

4. Синхронізація ДГА з шинами ГРЩ має бути представлена окремим компонентом, який містить засоби для ініціювання процесу автоматичної точної синхронізації запущеного ДГА з мережею на апаратному рівні; для відображення поточних значення напруг (діючих) і частот ДГА та мережі, фазового кута зсуву між напругами; для відображення та можливості зміни уставок синхронізації і параметрів дискретних сигналів управління системами збудження генератора та регулювання частоти обертання дизеля. Результатом закінчення процесу синхронізації є зміна стану генераторного вимикача, що на програмному рівні також має бути відображено.

5. Зміна стану навантаження полягає в підключенні або відключенні його від шин ГРЩ і може бути виконана за допомогою зміни стану компонента Автоматичний вимикач. До додаткових вимог до останнього відноситься можливість завдання часу спрацьовування.

6. До виконання розподілу навантажень (активного і реактивного) між паралельно працюючими дизель-генераторами висувається ряд вимог: розбиття ДГА на окремі незалежні секції; наявність засобів для завдання схеми розподілу і часу перехідного процесу перерозподілу навантажень для кожної з секцій; наявність засобів для установки частки навантаження, яку повинен прийняти на себе кожен ДГА, від загального навантаження секції; можливість зміни допустимої нерівномірності розподілу навантажень; наявність засобів для налаштування параметрів керуючих сигналів для зміни активної і/або реактивної потужностей, які віддаються окремими ДГА на загальне навантаження секції. Перераховані можливості можуть бути реалізовані програмно в одному компоненті.

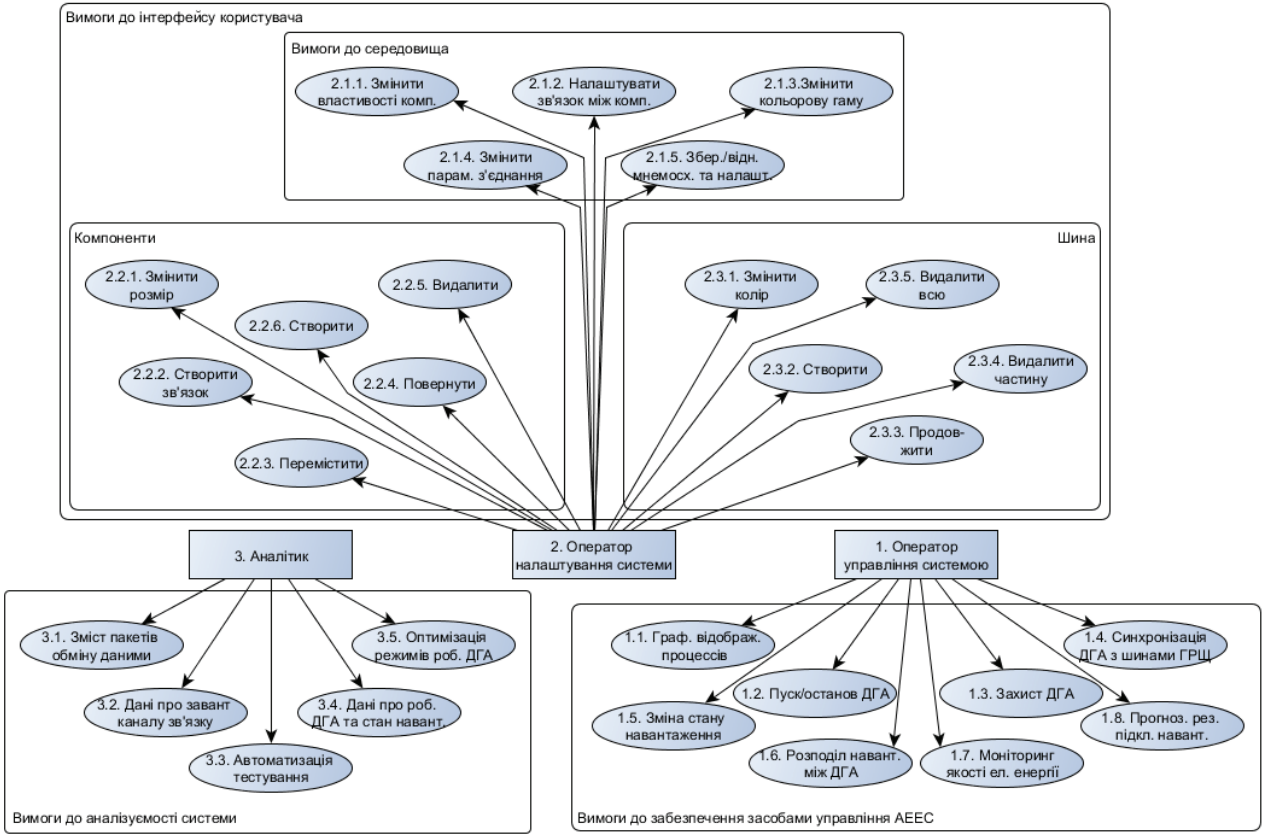

Вимоги до інтерфейсу користувача
Вимоги до середовища
2.1.1. Змінити властивості комп.
2.1.2. Налаштувати зв'язок між комп.
2.1.3.Змінити кольорову гаму
2.1.4. Змінити парам. з'єднання
2.1.5. Збер./відн. мнемосх. та налашт.
Компоненти
2.2.1. Змінити розмір
2.2.5. Видалити
2.2.6. Створити
2.2.2. Створити зв'язок
2.2.4. Повернути
2.2.3. Перемістити
Шина
2.3.1. Змінити колір
2.3.5. Видалити всю
2.3.2. Створити
2.3.4. Видалити частину
2.3.3. Продов-жити
3. Аналітик
2. Оператор налаштування системи
1. Оператор управління системою
3.1. Зміст пакетів обміну даними
3.5. Оптимізація режимів роб. ДГА
3.2. Дані про завант каналу зв'язку
3.4. Дані про роб. ДГА та стан навант.
3.3. Автоматизація тестування
Вимоги до аналізуємості системи
1.1. Граф. відображ. процессів
1.4. Синхронізація ДГА з шинами ГРЩ
1.2. Пуск/останов ДГА
1.3. Захист ДГА
1.5. Зміна стану навантаження
1.8. Прогноз. рез. підкл. навант.
1.6. Розподіл навант. між ДГА
1.7. Моніторинг якості ел. енергії
Вимоги до забезпечення засобами управління АЕЕС

Рис. 7.7.

7. Для моніторингу якості електроенергії може бути створений окремий компонент, який інформуватиме Оператора про перевищення показниками якості нормально і гранично допустимих значень, а також матиме можливість детального моніторингу стану енергосистеми в окремому діалоговому вікні.

8. Прогнозування результатів підключення навантаження відноситься до засобів інформаційної підтримки оператора і розглянуто детальніше в п. 3.4.

Щодо графічних властивостей всі компоненти можна розділити на три групи:

1) компоненти, що мають графічний зв'язок з іншими компонентами (Автоматичний вимикач, Трансформатор, Дизель, Генератор, Асинхронний двигун, Навантаження);

2) компонент для зв'язку (Шина);

3) компоненти, що мають інформаційний або логічний зв'язок з іншими компонентами (Текстове поле, Світлодіод, Шкальний індикатор, Стрілочний індикатор, Кнопка управління, Системна кнопка, Захист ДГА, Система синхронізації, Система розподілу навантажень, Система моніторингу якості електроенергії).

Підготовчим етапом перед моніторингом та управлінням електростанцією в режимі реального часу є створення мнемосхеми електростанції та налаштування властивостей її компонентів. Для здійснення перелічених дій Оператор налаштування системи повинен мати можливість взаємодії з бібліотекою компонентів для створення останніх (переміщення з бібліотеки на робоче поле за допомогою технології Drag and Drop). Кожен графічний компонент повинен підтримувати операції переміщення (за допомогою курсору миші і стрілок на клавіатурі) і видалення (натисканням на кнопку Delete або вибором відповідного пункту з контекстного меню компонента). Компонент Шина використовується для візуалізації енергетичних взаємозв'язків між елементами схеми і повинен мати можливість видалення або всієї шини, або її частин; зміни кольору для відображення незалежних секцій та продовження при додаванні нових компонентів в мнемосхему. Компоненти третьої групи (що не мають графічного зв'язку з іншими компонентами) повинні мати можливість зміни розмірів, а першої групи (що мають графічний зв'язок) − створення зв'язку з іншими компонентами за допомогою Шини. Загальними вимогами до програмного середовища на етапі підготовки системи до роботи в режимі управління електростанцією є можливість установки властивостей компонентів і каналу обміну даними для інформаційної взаємодії з апаратними засобами автоматизації, створення зв'язків між компонентами для моніторингу електроенергетичних параметрів у зручному для Оператора вигляді, зміни кольорової гами середовища і можливості збереження/відновлення створеної і налаштованої мнемосхеми.

При роботі ПЗ в режимі моніторингу та управління СЕЕС окрім виконання основних дій повинен проводитися збір інформації про роботу програми (завантаженості каналу обміну даними, змісту інформаційних пакетів) і силових агрегатів (стан і величини потужностей, що віддаються на загальне навантаження) протягом певного часу (тривалість залежить від розв'язуваних завдань). Далі збережена інформація обробляється Аналітиком з метою:

1) верифікації раніше розрахованого часу відгуку системи (п. 3.6.3) та рішення про можливість використання захистів ДГА та управління дискретними сигналами на програмному рівні; для цих цілей використовується інформація про завантаженість каналу зв'язку;

2) виявлення та усунення несправностей при обміні даними із засобами автоматизації (використовуються дані про вміст інформаційних пакетів);

3) розрахунку окремих показників якості електроенергії (середніх, мінімальних і максимальних значень і частостей провалів і сплесків потужностей в енергосистемі) на основі інформації про режими роботи ДГА;

4) виявлення такої комбінації номінальних потужностей ДГА, при якій кількість споживаного ними палива буде мінімальною, на основі аналізу режимів роботи генераторних агрегатів; ці дані можуть бути використані при реконфігурації електростанції.

Також до завдань Аналітика відноситься проведення функціонального тестування ПЗ, тому бажана наявність засобів для автоматизації цього процесу.

Оскільки використання програмного забезпечення є трьохетапним ітераційним процесом (зміна або створення мнемосхеми електростанції та налаштування властивостей її компонентів, використання програми в основному режимі, аналіз роботи системи) і перший і третій етап з точки зору користувача візуально ідентичні, процес використання ПЗ можна розділити на два режими: режим конструктора схеми і режим моніторингу та управління електростанцією.

Проектована програма є системою зі складною поведінкою, тому для її реалізації доцільно використовувати об'єктно-орієнтовану мову програмування. Далі розглядається процес проектування системи на прикладі використання уніфікованої мови моделювання UML для опису складових ПЗ. Кожна програма для Windows зазвичай містить головне вікно, що складається з двох компонентів – фрейма і клієнтської (робочої) області. Оскільки проектоване ПЗ повинно мати додаткове меню (для перемикання режимів роботи програми, налаштування параметрів з'єднання та ін.), підтримувати роботу з документами (зберігати/відновлювати мнемосхеми і налаштування) і в кожний момент часу може проводитися взаємодія лише з однією мнемосхемою (для відповідності реальним об'єктам), в якості базової була обрана SDI-програма. Робоча область проектованої програми в режимі конструктора схеми повинна мати два подання даних – для відображення

Бібліотеки компонентів (бажано у вигляді деревовидного списку) і для відображення топології схеми. Для їх поділу можна використовувати спліттер. Далі розглянуто структуру і поведінку компонентів програмного забезпечення в кожному з режимів його роботи.

Режим конструктора схеми об'єднує в собі два підрежиму роботи з програмним забезпеченням: створення (або редагування) мнемосхеми електростанції, налаштування властивостей її компонентів і програмного середовища; аналіз роботи електростанції і ПЗ. Для першого підрежиму характерними операціями, що виконуються користувачем, є створення, переміщення, видалення і зміна властивостей компонентів, створення графічних і інформаційних зв'язків між компонентами та інше. Згідно з діаграмою варіантів використання результатом процесу «Створення» має бути поява на Робочому полі нового компонента. Кожен з компонентів після створення повинен реагувати на переміщення курсору над його поверхнею, натиснення і відпускання лівої кнопки миші та інше, тобто бути вікном, батько якого – Робоче поле. Для координування взаємодії між компонентами і виконання аналітичних («надсистемних») операцій другого підрежиму зручно в об'єкті «Робоче поле» зберігати покажчики на об'єкти створених компонентів у вигляді списків. Аналіз діаграми варіантів використання і склад бібліотеки компонентів показує, що всі компоненти (крім Шини; далі під словом «компоненти» будуть матися на увазі всі компоненти, крім Шини) щодо графічного інтерфейсу в режимі конструктора схеми підтримують однаковий набір операцій – створення, переміщення, видалення – і далі поділяються на ті, які мають можливість створення зв'язку, і ті, які мають можливість масштабування. Тому при проектуванні ПЗ зручно створити два базових класи з відповідними можливостями і при реалізації компонентів використовувати механізм успадкування.

Компонент Шина в загальному вигляді являє ламану певного кольору і товщини. Програмно його можна реалізувати у вигляді структури, полями якої є регіон, список точок (початкова та кінцева точки відрізків) і колір. При виконанні деяких операцій (продовження, видалення частини) можливе виникнення труднощів, якщо об'єкт буде вікном. Тому доцільно в об'єкті Робоче поле зберігати список компонентів Шина як сукупність графічних об'єктів, а дії, пов'язані з Шинами, виконувати при виникненні відповідних повідомлень, адресованих об'єкту Робоче поле. Для виконання дій, що реалізують вимоги до середовища згідно з діаграмою варіантів використання, використовуються допоміжні діалогові вікна, які створюються при виклику відповідних пунктів контекстних меню компонентів або головного меню програми. Бібліотека компонентів повинна містити тільки назви компонентів, тому безпосереднього зв'язку з об'єктами, які їх реалізують, не має.

Для виконання дій Аналітиком необхідна наявність додаткових діалогових вікон, що містять дані про завантаженість каналу зв'язку та зміст інформаційних пакетів; що відображають збережену в базі даних інформацію про режими роботи ДГА; що управляють процесом тестування та інші.

Вище було зазначено, що Робоче поле містить покажчики на всі компоненти, тому класи, що реалізують наведені діалогові вікна, зручно зберігати як атрибути класу Робоче поле.

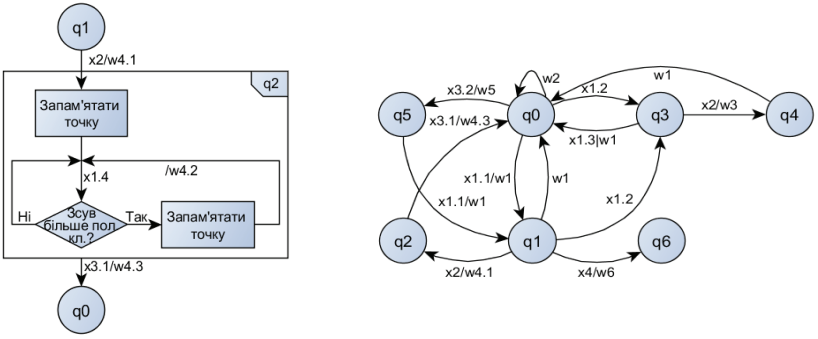
Кожен з компонентів і Робоче поле в режимі конструктора схеми є сутністю зі складною поведінкою (реакцією на вхідну дію може бути одна з кількох дій). Для розділення опису логіки їхньої поведінки від опису семантики кожної з дій при реалізації класів пропонується використовувати автоматний підхід, що в рамках об'єктно-орієнтованого програмування означає інкапсуляцію автомата в класі. Взаємодія між об'єктами в системі відбувається за допомогою повідомлень. У режимі конструктора схеми обмін повідомленнями відбувається між базовими класами (клас з можливістю зв'язку і клас з можливістю масштабування) і Робочим полем: $w1$ – зняття фокуса; $w2$ – зміна Шини; $w3$ – початок створення Шини; $w4$ – переміщення об'єкта: $w4.1$ – початок переміщення об'єкта; $w4.2$ – переміщення об'єкта на одну клітину; $w4.3$ – кінець переміщення об'єкта; $w5$ – кінець створення Шини; $w6$ – видалення об'єкта. Взаємодія компонентів і Робочого поля з користувачем здійснюється через передачу повідомлень програмному забезпеченню від операційної системи: $x1$ – переміщення курсору: $x1.1$ – над об'єктом; $x1.2$ – в області можливого зв'язку (можливого масштабування); $x1.3$ – поза областю можливого зв'язку (можливого масштабування); $x1.4$ – при затиснутій лівій кнопці миші; $x2$ – натискання лівої кнопки миші; $x3$ – відпускання лівої кнопки миші: $x3.1$ – в області можливого зв'язку (можливого масштабування); $x3.2$ – поза областю можливого зв'язку (можливого масштабування); $x4$ – натискання кнопки Delete.

Базовий клас з можливістю зв'язку

У режимі конструктора схеми вимогою до класу є інкапсуляція процесів переміщення компонента і створення з ним зв'язку (створення самого об'єкта належить до завдань Робочого поля). Логіку роботи класу в даному режимі можна представити у вигляді автомата $A_6 = \{Q, X, Y, f, g\}$, де $Q = \{q0, q1, q2, q3, q4, q5, q6\}$ – множина станів автомата: q0 – об'єкт створений (початковий стан); q1 – об'єкт у фокусі (виділений маркерами); q2 – переміщення об'єкта, стан є складним (алгоритм поведінки в стані наведено на рис. 7.8, а; під зсувом розуміється різниця між останньою збереженою точкою і координатами поточного положення курсора, половиною клітини – відстань, рівна половині кроку сітку в будь-якому з напрямів Ox, Oy або обох відразу); q3 – зв'язок з Шиною можливий; q4 – початок зв'язку з Шиною; q5 – кінець зв'язку з Шиною; q6 – об'єкт видалений (кінцевий стан); X – множина допустимих вхідних дій і Y – множина вихідних дій – розшифровані вище; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.8, б.

Поведінку наведеного класу успадковують класи компонентів Дизель, Генератор, Асинхронний двигун, Автоматичний вимикач і Трансформатор.

Дані компоненти в режимі конструктора схеми не вимагають наявності додаткових можливостей, крім зміни властивостей компонента.



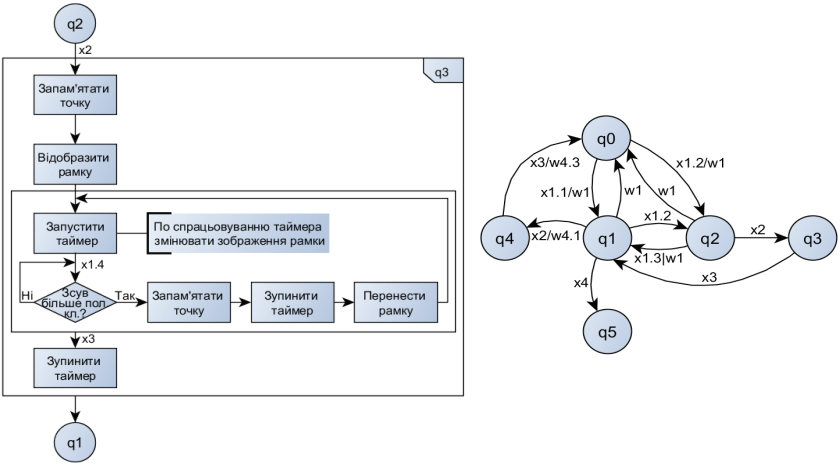
а б

Рис. 7.8.

Базовий клас з можливістю масштабування

У режимі конструктора схеми клас повинен підтримувати операції «Переміщення» і «Зміна розміру». Логіку поведінки класу можна представити у вигляді автомата $A_7 = \{Q, X, Y, f, g\}$, де $Q = \{q0, q1, q2, q3, q4, q5\}$ – множина станів автомата: q0 – об'єкт створений (початковий стан); q1 – об'єкт у фокусі (виділений маркерами); q2 – зміна розміру можлива; q3 – зміна розміру об'єкта (алгоритм поведінки об'єкта в стані наведено на рис. 7.9, а); q4 – переміщення об'єкта (алгоритм поведінки об'єкта аналогічний наведеному на рис. 7.9, а (замість стану $q2$ стан $q4$)); q5 – об'єкт видалений (кінцевий стан); X – множина допустимих вхідних дій і Y – множина вихідних дій – розшифровані вище; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.9, б.

Нащадками поведінки класу «Базовий клас з можливістю масштабування» є класи компонентів Стрілочний і Шкальний індикатори, Світлодіоди, Кнопка. Оскільки наведені компоненти в режимі конструктора схеми окрім можливостей, реалізованих у батьківському класі, вимагають лише наявності засобів для зміни їх властивостей або установки інформаційного взаємозв'язку з іншими компонентами, додаткових автоматів для опису їх поведінки не потрібно.



а б

Рис. 7.9.

Робоче поле

У режимі конструктора схеми об’єкт «Робоче поле» повинен виконувати всі операції, пов’язані з Шиною (створення, продовження, видалення повне і часткове, зміна кольору), операцію «Створення компонента» і брати участь при переміщенні і видаленні компонента. Поведінка класу є складною, тому її можна представити у вигляді автомата $A_7 = \{Q, X, Y, f, g\}$, де $Q = \{q0, q1, q2, q3, q4, q5, q6\}$ – множина станів автомата: $q0$ – об’єкт у фокусі (початковий стан); $q1$ – створення компонента: «перетягування» з Бібліотеки компонентів на Робоче поле (алгоритм поведінки наведено на рис. 7.10, а); $q2$ – об’єкт не у фокусі; $q3$ – видалення компонента; $q4$ – створення Шини (при виборі в Бібліотеці компонентів); $q5$ – створення Шини (початком є об’єкт); $q6$ – продовження Шини. Стани $q4 - q6$ пов’язані з операціями створення та продовження (з програмною точки зору – створення і приєднання до обраної) Шини й подібні між собою. При вході в кожне з них повинна виконуватися наступна послідовність дій (умовно блок entry): зміна курсору; переміщення курсору у вузол сітки; запам'ятовування положення курсора як початкової точки Шини.

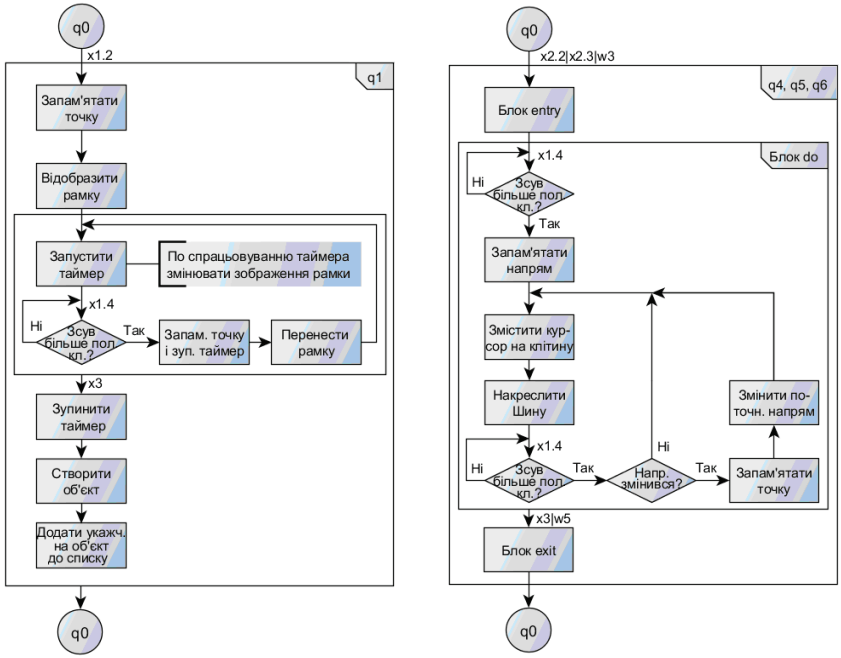


Рис. 7.10.

При вході в стан $q5$ додатково необхідно запам'ятовувати ідентифікатор об'єкта, з якого починається Шина; $q6$ – ідентифікатор обраної Шини. Поведінка класу в станах $q4-q6$ (блок do) є ідентичною і полягає в кресленні Шини відрізками з довжиною, яка дорівнює кроку сітки. Зміна напрямку креслення відбувається при зсуві курсора більш ніж на половину кроку сітки в напрямку Ox для вертикальної, або Oy – для горизонтальної поточної частини Шини. На рис. 7.10, б наведено алгоритм роботи класу в станах $q4-q6$. Вихід зі станів $q4-q6$ (блок exit) може бути ініційований відпусканням лівої кнопки миші на Робочому полі, Шині (в цьому випадку створена Шина приєднується до обраної) або на компоненті (при цьому компонент повинен надіслати Робочому полю повідомлення «Кінець створення Шини» ($w5$) і чекати у відповідь повідомлення «Зміна Шини» ($w2$) з адресою створеної Шини). Якщо автомат знаходився в стані $q6$ (продовження Шини), і Шина була закінчена на інший, відбувається об'єднання двох Шин, яке супроводжується передачею всім пов'язаним з Шиною компонентам повідомлення «Зміна Шини» ($w2$). Також при виході зі станів $q4-q6$ оновлюється зображення курсора.

– $X = \{x1.1,\ x1.2,\ x2.1,\ x2.2,\ x2.3,\ x3, w1, w3, w4.1, w4.2, w4.3, w5, w6\}$ – множина допустимих вхідних дій – і $Y = \{w1,\ w2\}$ – множина вихідних дій – розшифровані вище; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.11.

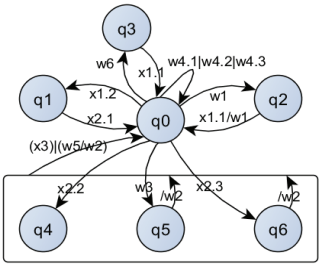


Рис. 7.11.

Режим моніторингу та управління СЕЕС

У режимі моніторингу та управління електростанцією ПЗ повинне виконувати декілька завдань (далі реалізація кожного з них розглянута окремо):

– інформаційний обмін із засобами автоматизації – передача команд оператора, зчитування, відображення і обробка виміряних апаратними засобами аналогових і дискретних значень;

– накопичення статистичних даних про завантаженість каналу зв'язку – кількостей переданих та отриманих байт за одиницю часу;

– накопичення інформації про режими роботи дизель-генераторних агрегатів – середніх значень потужностей, що віддаються ними на загальне навантаження, за певний проміжок часу.

Одиницею обміну даними є транзакція, що складається з запиту і відповіді – запису і зчитування пакетів даних з COM-порту. Для здійснення перерахованих операцій необхідно знати розміри обох пакетів і мати зарезервовану область пам'яті для зчитаного пакета. Доцільно всі дані щодо кожної окремої транзакції зберігати у вигляді структури з полями: масив даних (або покажчик на нього) для запиту; розмір масиву даних для запиту (в байтах); покажчик на область пам'яті для запису відповіді; очікуваний розмір пакета даних, одержуваних при відповіді; кількість символів (байт), отриманих при відповіді; ідентифікатор об'єкта. Для упорядкування та контролю за процесом обміну даними можна створити список покажчиків на ці структури. Оскільки Робоче поле має доступ до всіх компонентів, які беруть участь в обміні даними із засобами автоматизації, доцільно в ньому зберігати список покажчиків і проводити всі операції з COM-портом, а самі структури зберігати в компонентах. Тоді до класів останніх додаються дві вимоги − наявність структури для кожної можливої транзакції і формування вмісту пакетів даних для запиту при створенні компонета і при зміні його

мережевих властивостей (при зміні функції поля пакета запиту може змінитися розмір самого пакету, тому в структурі запропоновано зберігати покажчик на масив даних для запиту). Процес обміну даними стосовно кожного компонента в спрощеному вигляді можна представити у вигляді наступної послідовності дій: додавання покажчика на структуру в список; запис пакету даних для запиту в COM-порт (здійснює Робоче поле, використовуючи поля структури з даними та їх розміром); зчитування пакета даних з COM-порту (використовується область пам'яті, визначена в структурі, і кількість очікуваних байт; фактично зчитана кількість байт записується у відповідне поле структури); передача повідомлення «Отримано дані» компоненту (за допомогою ідентифікатора, визначеного в структурі); розшифровка компонентом отриманого повідомлення: при помилці запису або зчитування даних з COM-порту Робоче поле передає повідомлення з аргументом 0; якщо обмін даними був завершений без помилок, аргумент приймає одиничне значення, що означає істинність даних (відповіді на запит) у структурі. Якщо помилок не було, компонент здійснює розрахунок контрольної суми і, при її відповідності отриманій, розшифровку відповіді. Кількість отриманих байт використовується для виявлення пакетів з кодами помилок: якщо кількості очікуваних та отриманих байт не збігаються і контрольна сума вірна, у відповіді на запит міститься код помилки.

Компонентами, для яких можливий зв’язок із засобами автоматизації, є Дизель, Генератор, Автоматичний вимикач, Кнопка управління, Світлодіод, Захист ДГА, Система синхронізації, Система розподілу навантажень і Система моніторингу якості електроенергії. З їхньою допомогою можна здійснювати наступні операції (наведені в порядку зниження пріоритетів):

1) керування та індикація процесу синхронізації. Для своєчасного вимірювання даних і контролю стану генераторного вимикача з упередженням апаратний пристрій синхронізації має вести обмін даними з ПЗ в строго детерміновані проміжки часу. Тому на рівні програмного забезпечення при запуску Системи синхронізації і відображенні даних за допомогою деталізованого діалогового вікна інформаційний обмін з іншими засобами автоматизації тимчасово повинен блокуватися;

2) опитування одиничних аналогових значень (діючих значень напруги і струму, значень частот, коефіцієнта потужності і синуса кута між векторами струму і напруги) для подальшої обробки Захистом ДГА; зміна станів дискретних виходів при спрацьовуванні хоча б одного із захистів або зникненні аварійних або небажаних ситуацій;

3) зміна станів дискретних виходів при зміні станів Автоматичних вимикачів або Кнопок управління;

4) опитування станів дискретних входів (зворотних зв’язків від генераторних вимикачів) для визначення структури електростанції Системою розподілу навантажень (відбувається в кожному циклі обміну даними); опитування значень потужностей, що віддаються включеними на паралельну

роботу ДГА (відбувається з певним часовим інтервалом, заданим як час перехідного процесу перерозподілу навантажень, або при зміні структури електростанції), і при, необхідності, передача параметрів керуючих сигналів для їх коригування;

5) отримання станів дискретних входів (зворотних зв'язків від автоматичних вимикачів) для відображення на Світлодіодах;

6) отримання, обробка і, при необхідності, відображення значень показників несиметрії напруги Системою моніторингу якості електроенергії;

7) опитування одиничних аналогових значень (діючих значень напруг і струмів, значень частоти і коефіцієнтів потужності Генераторів);

8) опитування одиничних аналогових значень (вимірювань, пов'язаних з Дизелем; необхідні тільки при відкритому діалоговому вікні параметрів Дизеля);

9) отримання масивів аналогових значень (миттєвих значень струму і напруги Генератора; необхідні тільки при відкритому діалоговому вікні «Осцилограми»).

Таким чином, якщо для обміну даними з пристроєм синхронізації ввести окремий підрежим, список решти покажчиків на структури складатиметься з постійних елементів і змінних. Останні можна розділити на тривалі змінні (відповідні опитуванням даних при відкритих діалогових вікнах параметри Дизеля і/або Осцилограми) і короткочасні змінні (відповідні опитуванню потужностей і передачі параметрів керуючих сигналів Системою розподілу навантажень, передачі команд на зміну станів дискретних виходів Захистом ДГА, Автоматичними вимикачами та Кнопками управління).

Обмін даними із засобами автоматизації повинен відбуватися циклічно з певним проміжком часу (періодом сканування, заданим в параметрах з'єднання). Тому при вході в режим управління електростанцією в роботу включається таймер, спрацьовування якого ініціює цикл обміну даними як атомарний блок операцій. У кожному циклі відбувається послідовне виконання транзакцій для кожного елемента і передача повідомлень «Отримано дані» компонентам-батькам структур або діалоговим вікнам. Відображення даних виконується кількома способами: безпосередньо компонентами (стани дискретних входів – Світлодіодами; допустимість значень – Захистами ДГА, Системами моніторингу якості електроенергії та розподілу навантажень); за допомогою компонентів для індикації (одиничних аналогових значень – параметрів Генератора – Стрілочними і Шкальними індикаторами, Текстовим полем); за допомогою діалогових вікон (параметри Дизеля, осцилограми Генератора, несиметрія напруг). Оскільки для кожного з діалогових вікон може бути відкрито вікно лише для одного з компонентів, отримані в результаті обміну даними значення передаються цьому вікну, минаючи сам компонент. Діючі значення струму і напруги, значення частоти і коефіцієнта потужності, а також отримані на основі їх комбінацій значення потужностей можуть відображатися одночасно за допомогою декіль-

кох компонентів. Для отримання даних останніми в Генераторі пропонується зберігати список ідентифікаторів пов'язаних з ним компонентів для індикації і при отриманні коректної відповіді на запит передавати їм необхідну інформацію (повідомлення «Відобразити параметр»).

Однією з вимог до ПЗ є можливість перегляду та зберігання інформації про обмін даними – вмісту пакетів та інтенсивності інформаційного потоку. Оскільки всі дії, пов'язані з обміном даними, відбуваються в Робочому полі, пропонується всі необхідні для виконання цієї вимоги операції (підрахунок переданих та отриманих байт, відображення та зберігання даних) виконувати в ньому. Визначення інтенсивності інформаційного потоку пов'язане з накопиченням кількості байт за певний проміжок часу, тому при переході в режим управління електростанцією відбувається запуск ще одного таймера, налаштованого на переривання із заданим інтервалом.

Аналогічно вищеописаному можна здійснювати збір і зберігання інформації про режими роботи ДГА: оскільки через Робоче поле проходять запити від всіх компонентів, в ньому можна накопичувати і обчислювати середні значення потужностей, що віддаються ДГА на загальне навантаження, протягом певного інтервалу часу і далі передавати ці дані для зберігання в базі даних.

Якщо вхідні дії, пов'язані з повідомленнями від операційної системи при взаємодії оператора з програмою, роботою таймерів і діями з COM-портом, позначити як $e1$ – перехід ПЗ в режим управління електростанцією; $e2$ – перехід ПЗ в режим конструктора схеми; $e3$ – отриманий пакет даних; $e4$ – переміщення курсору в області компонента; $e5$ – зміна структури електростанції; $e6$ – спрацьовування таймера; $e7$ – завантаженість як мінімум одного ДГА перевищує допустимі границі; $e8$ – відкриття вікна синхронізації; $e9$ – отримані значення уставок; $e10$ – отримані дані про стан генераторного вимикача; $e11$ – отримані значення напруг, частот і кута зсуву між векторами напруг; $e12$ – зміна значень уставок; $e13$ – зміна параметрів керуючих сигналів; $e14$ – отриманий пакет даних (в режимі синхронізації); $e15$ – помилка при відкритті COM-порту; $e16$ – закриття вікна синхронізації; $e17$ – запуск процесу синхронізації; $e18$ – дані компонентом отримані; $e19$ – натискання лівої кнопки миші; вхідні змінні, відповідні результату виконання певних дій, як $x1$ – помилка в пакеті даних; $x2$ – помилка усунена; $x3$ – помилка невиправна без заміни апаратних засобів; $x4$ – присутній зв'язок: $x4.1$ – з компонентами індикації (як мінімум одним), $x4.2$ – з компонентом Захист ДГА; $x5$ – дані сформовані; $x6$ – черга запитів не порожня; $x7$ – дані компоненту відправлені; $x8$ – таймер працює; $x9$ – для Захисту ДГА використовуються дані, одержувані Системою розподілу навантажень; $x10$ – для Захисту ДГА використовуються дані, що вимірюються незалежно; $x11$ – пряма умова захисту виконується; $x12$ – обернена умова захисту виконується; $x13$ – ДГА працює паралельно з іншими ДГА; $x14$ – генераторний вимикач замкнутий; $x15$ – черга порожня; $x16$ – запис в базу даних дозволений; $x17$ – відображення змісту пакетів даних дозволено; $x18$ – помилка зв'язку з базою

даних; $x19$ – помилка зв'язку з базою даних усунена; $x20$ – включена система корекції частоти обертів дизеля і/або напруги збудження генератора, і вихідні дії, що відбуваються при виникненні певних комбінацій вхідних впливів і змінних, як $z1$ – додати пакети з запитами в чергу, зміст яких пов'язаний з опитуванням значень або командою: $z1.1$ – параметри Генератора (діючі значення напруги і струму, значення частоти і коефіцієнта потужності); $z1.2$ – параметри, необхідні для забезпечення функцій захистів ДГА; $z1.3$ – команда на замикання дискретного виходу; $z1.4$ – команда на розмикання дискретного виходу; $z1.5$ – уставки синхронізації і параметри керуючих сигналів; $z1.6$ – дані про стан генераторного вимикача; $z1.7$ – поточні значення частот, напруг і кута зсуву між векторами напруг шини ГРЩ і запущеного ДГА; $z1.8$ – команда на зміну уставок синхронізації; $z1.9$ – команда на зміну параметрів керуючих сигналів; $z1.10$ – дані про стани генераторних вимикачів секції; $z1.11$ – дані для розрахунку потужностей, що віддаються включеними на паралельну роботу ДГА; $z1.12$ – команда на включення коригувальних впливів для регуляторів частоти і напруги; $z1.13$ – команда на відключення коригувальних впливів для регуляторів частоти і напруги; $z1.14$ – команда на замикання автоматичного вимикача або реле; $z1.15$ − команда на розмикання автоматичного вимикача або реле; $z2$ – видалити запит з черги; $z3$ – передати дані компоненту; $z4$ – запуск таймера; $z5$ – відображення даних; $z6$ – зупинка таймера; $z7$ – створити чергу обміну даними; $z8$ – змінити колір Шини; $z9$ – передати пакет запиту апаратним засобам автоматизації; $z10$ – передати дані в стан, який ініціював обмін даними; $z11$ – видалити запит з вершини стека; $z12$ – сформувати постійну чергу запитів; $z13$ – відкрити COM-порт; $z14$ – закрити COM-порт; $z15$ – видалити запит; $z16$ – запам'ятати дані про стани навантажень і навантаження ДГА; $z17$ – відобразити вміст пакетів даних, то описана поведінка класу Робочого поля в режимі управління електростанцією може бути описана автоматом $A_8 = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q8\}$ – множина станів автомата: $q0$ – початковий стан; $q1$ – очікування початку циклу обміну даними; $q2$ – очікування відповіді на запит; $q3$ – очікування запису в базу даних; $q4$ – запис в базу даних; $q5$ – помилка зв'язку з базою даних; $q6$ – очікування синхронізації ДГА з шинами ГРЩ; $q7$ – синхронізація ДГА з шинами ГРЩ; $q8$ – очікування відключення керуючих впливів для регуляторів частоти та/або напруги; X, E, Z – множини допустимих вхідних впливів, вхідних змінних і вихідних дій, розшифровані вище; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.12.

Слід зазначити, що при необхідності переходу в режим конструктора схеми (для корекції налаштувань обміну даними, аналізу режимів роботи ПЗ і ДГА) слід спочатку передати команди на відключення сигналів корекції потужностей, що віддаються ДГА на загальне навантаження, а потім виконати перехід. У автоматі ця умова відображена введенням стану $q8$.

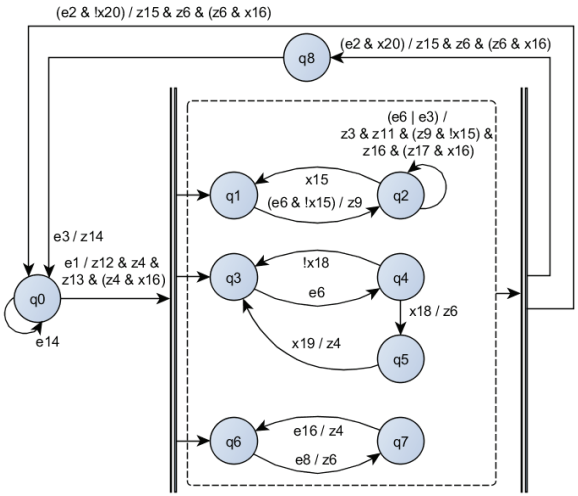


Рис. 7.12.

При отриманні компонентами повідомлення «Отримано пакет даних» від Робочого поля починається процес їх розшифрування і подальшої обробки. Для деяких компонентів (Світлодіодів, Системи моніторингу якості електроенергії) цей процес є елементарним (при отриманні повідомлення дані розшифровуються однозначно і не вимагають додаткових інформаційних обмінів), проте більшість компонетов (Генератор, Захист ДГА, Система розподілу навантажень, Автоматичний вимикач, Кнопка управління, Система синхронізації) реалізують складні схеми поведінки. Тому для опису останніх пропонується використовувати автоматний підхід. Далі розглянуто поведінку кожного з компонентів у режимі моніторингу та управління електростанцією окремо (позначення вхідних дій, вхідних змінних і вихідних дій для всіх автоматів відповідають наведеним вище).

Генератор

Поведінку компонента Генератор можна описати у вигляді автомата $A_9 = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q7\}$ – множина станів автомата: $q0$ – початковий стан; $q1$ – очікування відповіді на запит; $q2$ – перевірка пакету даних; $q3$ – відсутність зв'язку із засобами автоматизації; $q4$ – передача даних компоненту Захист ДГА; $q5$ – передача даних компонентам індикації; $q6$ – очікування відображення значень параметрів генератора; $q7$ – відображення значень параметрів генератора; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.13.

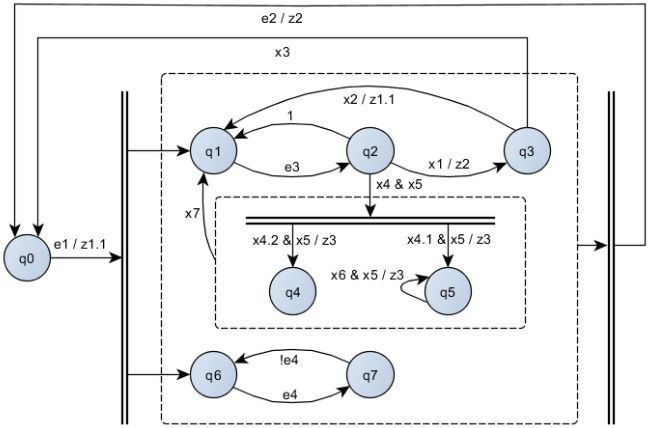


Рис. 7.13.

У початковому стані (*q*0*)* компонент очікує переходу програмного забезпечення в режим моніторингу та управління електростанцією. При виникненні цієї події починається паралельне виконання декількох процесів: обмін даними із засобами автоматизації (стани *q*1 – *q*3); одночасна передача параметрів Генератора, отриманих при успішному обміні даними, компонентам індикації (*q*4) і, якщо включений режим мінімізації інформаційних потоків, компоненту Захисту даного ДГА (*q*5); відображення параметрів Генератора у вікні підказки при переміщенні курсора над поверхнею компонента (*q*6 – *q*7). Зворотний перехід в початковий стан може відбутися в двох випадках: при переході ПЗ в режим конструктора схеми або при виникненні неполадок з обміном даними. В останньому випадку виконується індикація помилки і, при її усуненні (шляхом зміни властивостей компонента або заміни апаратного пристрою), обмін даними триває без необхідності переходу ПЗ в режим конструктора схеми.

Система розподілу навантажень

Поведінку компонента Система розподілу навантажень (на прикладі однієї секції) можна описати у вигляді автомата $A_{10} = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q11\}$ – множина станів автомата: *q*0 – початковий стан; *q*1 – очікування відповіді на запит; *q*2 – перевірка пакету даних; *q*3 – відсутність зв'язку із засобами автоматизації; *q*4 – перевірка структури секції; *q*5 – очікування перевірки завантаженостей ДГА; *q*6 – очікування прийому значень потужностей, що віддаються кожним ДГА на загальне навантаження секції; *q*7 – перевірка пакету даних; *q*8 – розрахунок відхилень реальних значень завантаженостей ДГА від ідеальних; *q*9 – очікування відповіді на команду

включення керуючих впливів; $q10$ – перевірка пакету даних; $q11$ – відсутність зв'язку із засобами автоматизації; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.14.

Перехід компонента з початкового стану ($q0$) до виконання дій з розподілу навантажень відбувається при переході програмного забезпечення з режиму конструктора схеми в режим моніторингу та управління електростанцією. Процеси, що відбуваються далі, умовно можна розділити на дві складові: опитування станів дискретних входів – зворотних зв'язків від генераторних вимикачів ($q1$ – $q4$) з метою визначення структури секції, що відбувається в кожному циклі обміну даними; опитування значень потужностей ($q5$ – $q7$) з подальшою обробкою отриманих даних ($q8$) і, при необхідності, видачею керуючих впливів на регулятори частоти та/або збудження ($q9$ – $q10$), що відбуваються або відразу ж після виявлення зміни структури секції, або після закінчення певного часового інтервалу. При виникненні несправностей при обміні даними (відсутності пакетів даних з відповідями або прийомі пакетів з кодами помилок), компонент відображає помилкову ситуацію і переходить до стану $q4$ або $q11$ залежно від того, з якою з апаратних частин виникли проблеми при обміні даними. Якщо усунення помилкової ситуації не вимагає заміни апаратних засобів, компонент очікує продовження обміну даними в наведених станах, в іншому випадку − переходить в початковий стан ($q0$). Після усунення апаратних несправностей компонент може почати обмін даними без необхідності переходу ПЗ в режим конструктора схеми.

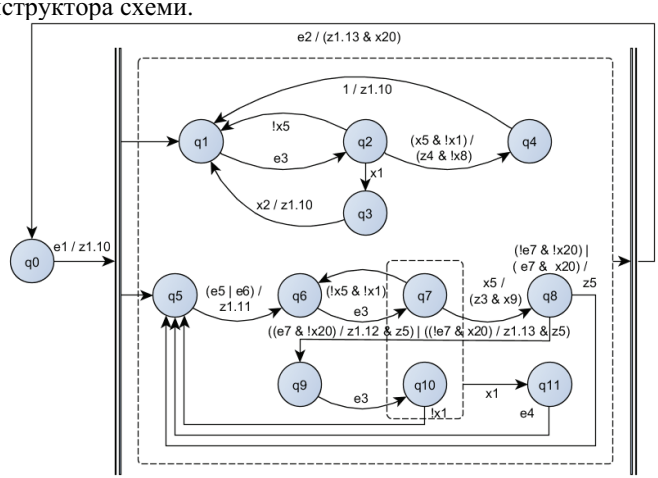
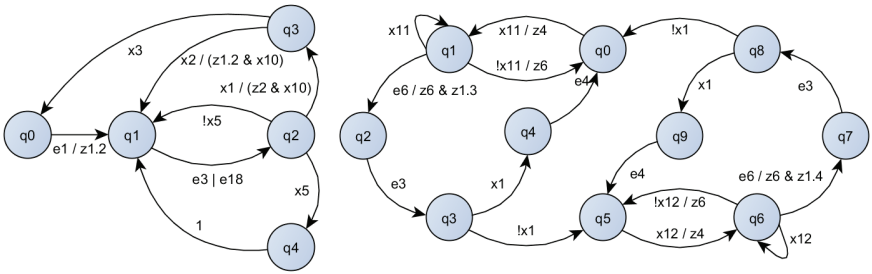


Рис. 7.14.

Перехід ПЗ в режим конструктора схеми супроводжується передачею Робочим полем Системі розподілу навантажень відповідного повідомлення. При його прийомі компонент перевіряє наявність включених керуючих впливів і, якщо вони присутні, виконує передачу команди на їх відключення.

Захист ДГА

Компонент Захист ДГА об'єднує функції шести захистів генераторного агрегату: від зворотної потужності, втрати збудження, протікання максимального струму, зміни напруги в неприпустимих межах, зміни частоти в неприпустимих межах, перевантаження по потужності. При виникненні аварійних або небажаних ситуацій компонент повинен відправити команду на зміну відповідного ситуації стану дискретного виходу. Як було наведено в розд. 2, для реалізації перерахованих захистів ДГА необхідно вимірювати однакові параметри генератора: діючі значення напруги і струму, значення частоти і кута зсуву між векторами струму і напруги. Крім того, перераховані параметри беруть участь в інформаційному обміні із засобами автоматизації для відображення параметрів Генератора і для виконання обчислень Системою розподілу навантаження. Тоді у якості джерела даних для забезпечення захистів ДГА можна використовувати один з трьох каналів: інформація, одержувана Генератором; інформація, що отримується Системою розподілу навантажень щодо даного генератора або інформація, що отримується в процесі безпосереднього обміну із засобами автоматизації. Накопичення і обчислення необхідних для реалізації функцій захистів даних можна представити у вигляді автомата $A_{11} = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q4\}$ – множина станів автомата: q0 – початковий стан; q1 – очікування відповіді на запит; q2 – перевірка пакету даних; q3 – відсутність зв'язку із засобами автоматизації; q4 – обробка даних; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.15, а.



а б

Рис. 7.15.

Після переходу програмного забезпечення в режим моніторингу та управління електростанцією компонент очікує прийому за будь-яким з трьох каналів (якщо дані мають бути отримані незалежно від інших компонентів, що вказується при налаштуванні властивостей даного компонента, відповідні запити додаються в постійну чергу Робочого поля) даних, після чого виконує їх обробку. Остання полягає в обчисленні параметрів, необхідних для подальшої обробки включеними функціями захистів.

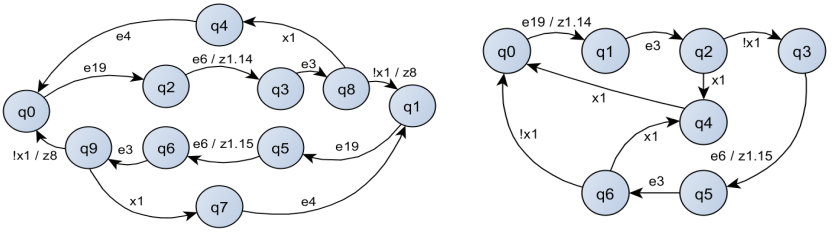
Кожна з функцій захистів, що інкапсулюються в компоненті Захист ДГА, може бути включеною або відключеною; для кожного з захистів крім інформації про відповідний дискретний вихід можна задати уставки за значенням (число, з яким буде порівнюватися параметр, що перевіряється) і за часом (час, протягом якого умова спрацьовування захисту має бути істинною для ідентифікації аварійної або небажаної ситуації). Реалізацію кожної з функцій захистів можна представити у вигляді автомата $A_{12} = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q9\}$ – множина станів автомата: $q0$ – захист відключено (дискретний вихід у початковому стані); $q1$ – очікування закінчення часу спрацьовування захисту; $q2$ – очікування відповіді на запит-команду; $q3$ – перевірка пакету даних; $q4$ – відсутність зв'язку із засобами автоматизації; $q5$ – захист включен; $q6$ – очікування закінчення часу зняття захисту; $q7$ – очікування відповіді на запит-команду; $q8$ – перевірка пакету даних; $q9$ – відсутність зв'язку із засобами автоматизації; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.15, б.

У початковому стані $q0$ автомат проводить перевірку умови спрацьовування захисту і, якщо її результат позитивний, запускає таймер для відліку часової уставки спрацьовування і переходить у стан $q1$. Під час роботи таймера триває прийом автоматом даних і перевірки умови і, якщо результат перевірки постійно позитивний, по закінченню часового інтервалу автомат передає команду для зміни стану дискретного виходу. Якщо параметр, що перевіряється, знаходиться в нормальних межах або приймає допустиме значення при запущеному таймері, автомат повертається (або залишається) в стані $q0$. Після спрацьовування захисту автомат знаходиться в стані $q5$, в якому також відбувається перевірка умови: якщо значення параметра, що перевіряється, повертається в діапазон допустимих значень, ініціюється процес зміни стану дискретного виходу в початковий стан аналогічний вищеописаному.

Алгоритм реалізації кожної з функцій захистів відповідає наведеному автомату (A_{12}). Для роботи останнього необхідно значення параметра, що перевіряється, тому в стан $q4$ автомата A_{11} пропонується вкласти автомати A_{12}, кількість яких залежить від кількості включених захистів (максимальне значення – 8, тому що функції захистів по напрузі і по частоті реалізуються за допомогою двох окремих автоматів). Тоді при переході автомата A_{11} в стан $q4$ після обробки прийнятих від засобів автоматизації даних виконуватиметься виклик всіх вкладених автоматів A_{12} з передачею їм отриманого в результаті обробки значення. Після виконання необхідних дій у вкладених автоматах автомат A_{11} перейде в стан очікування прийому наступних даних $q1$.

Автоматичний вимикач і Кнопка управління

Поведінку компонента Автоматичний вимикач можна описати у вигляді автомата $A_{13} = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q9\}$ – множина станів автомата: $q0$ – Автоматичний вимикач в початковому стані; $q1$ – Автоматичний вимикач у стані, протилежному початковому; $q2$ – очікування закінчення часу спрацьовування автоматичного вимикача; $q3$ – очікування відповіді від Робочого поля; $q4$ – очікування дій користувача щодо усунення помилки; $q5$ – очікування закінчення часу зворотного спрацьовування автоматичного вимикача; $q6$ – очікування відповіді від Робочого поля; $q7$ – очікування дій користувача щодо усунення помилки; $q8$ – перевірка пакету даних при непарній зміні стану Автоматичного вимикача; $q9$ – перевірка пакету даних при парній зміні стану Автоматичного вимикача; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.16, а.



а б

Рис. 7.16.

При переході програмного забезпечення в режим моніторингу та управління електростанцією автомат A_{13} знаходиться в початковому стані, якому відповідає зазначений у властивостях компонента стан дискретного виходу (нормально замкнутий або розімкнений). При натисканні користувачем на компоненті автомат запускає таймер на час, вказаний у властивостях компонента як час спрацьовування, по закінченні якого передається команда на зміну стану дискретного виходу. Повторне натискання на компоненті призведе до аналогічної послідовності дій, після завершення якої автомат повернеться в початковий стан.

Однією з вимог до компоненту Кнопка управління є можливість вибору між реалізацією поведінки кнопки з фіксацією або імпульсної кнопки. Алгоритм роботи кнопки з фіксацією аналогічний алгоритму роботи Автоматичного вимикача з різницею в тому, що кнопка не пов'язана з Шиною, тому логіку її поведінки можна представити у вигляді автомата A_{13}, виключивши з його вихідного алфавіту дію $z8$.

Поведінку імпульсної кнопки можна представити у вигляді автомата $A_{14} = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q9\}$ – множина станів автомата: $q0$ – Кнопка в початковому стані (відпущена); $q1$ – очікування відповіді на запит-

команду; $q2$ – перевірка пакету даних при натисканні на Кнопку; $q3$ – Кнопка натиснута, очікування закінчення часу імпульса; $q4$ – очікування дій користувача щодо усунення помилки; $q5$ – очікування відповіді на команду-запит; $q6$ – перевірка пакету даних при відтисканні Кнопки; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.16, б. При натисканні на Кнопку відбувається передача команди на зміну стану дискретного виходу і запуск таймера на час, рівний вказаному у властивостях компонента як час імпульсу. При спрацьовуванні таймера Кнопка повертається в початковий стан, передається команда на установку стану дискретного виходу в початкове і автомат переходить до стану $q0$.

Система синхронізації

Компонент Система синхронізації є кнопкою з пов'язаним з нею діалоговим вікном, яке відображається при натисканні на неї в режимі моніторингу та управління електростанцією. Як зазначено вище, у зв'язку з особливостями апаратної реалізації пристрою синхронізації при відкритті діалогового вікна інформаційний обмін з іншими засобами автоматизації повинен тимчасово блокуватися. Тоді алгоритм поведінки Системи синхронізації (діалогового вікна) можна представити у вигляді автомата $A_{15} = \{Q, X, E, Z, f, g\}$, де $Q = \{q0 - q9\}$ – множина станів автомата: $q0$ – початковий стан; $q1$ – очікування даних; $q2$ – перевірка пакету даних; $q3$ – відсутність зв'язку із засобами автоматизації; $q4$ – перевірка стану генераторного вимикача; $q5$ – синхронізація завершена; $q6$ – очікування прийому значень напруг, частот і кута зсуву; $q7$ – перевірка пакету даних; $q8$ – очікування спрацювання зворотного зв'язку від генераторного вимикача; $q9$ – відсутність зв'язку із засобами автоматизації; $q10$ – очікування зміни уставок синхронізації; $q11$ – очікування даних про результат зміни уставок; $q12$ – перевірка пакету даних з результатом виконання команди на зміну уставок синхронізації; $q13$ – очікування зміни параметрів керуючих впливів; $q14$ – очікування даних про результат зміни параметрів; $q15$ – перевірка пакету даних з результатом виконання команди на зміну параметрів керуючих впливів; $q16$ – відсутність зв'язку із засобами автоматизації; $q17$ – очікування циклу обміну даними; $q18$ – очікування відповіді на запит; f і g – функції станів і вихідних символів, представлені у вигляді графа на рис. 7.17.

При відкритті діалогового вікна автомат переходить в стани $q1$ – $q3$, відповідні опитуванню значень уставок синхронізації, параметрів керуючих впливів для корекції частоти обертання дизеля і збудження генератора. Далі в стані $q4$ проводиться опитування стану генераторного вимикача: якщо він замкнений (ДГА включений на паралельну роботу з шинами ГРЩ), Система синхронізації відображає цю ситуацію і переходить у стан $q5$, очікуючи подальших дій користувача. В іншому випадку з'являється можливість натисканням на відповідну кнопку діалогового вікна ініціювати процес

синхронізації на апаратному рівні, після чого процеси в Системі синхронізації поділяються на три складових: обмін даними з апаратними засобами ($q17$ – $q18$), зміна уставок синхронізації або параметрів керуючих впливів ($q10$ – $q16$), формування запитів і відображення поточних значень різниць частот і напруг, фазового кута між векторами напруг шини ГРЩ і генератора ($q6$ – $q8$). Перераховані підпроцеси виконуються доти, поки не відбудеться одна з подій: переривання користувачем процесу синхронізації або успішне підключення ДГА до шин ГРЩ. У першому випадку автомат повернеться в початковий стан ($q0$), у другому – перейде до стану, який індикує закінчення синхронізації ($q5$).

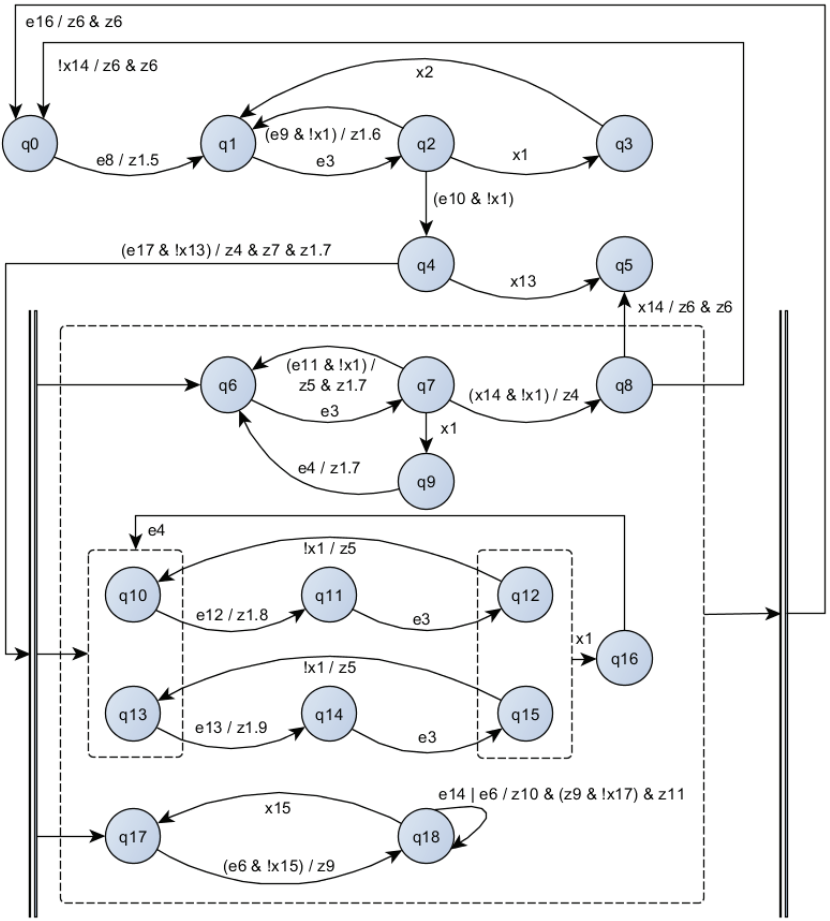


Рис. 7.17.

7.3. Моделювання процесів взаємодії компонентів програмного забезпечення за допомогою паттернових мереж

Використання персонального комп'ютера зі спеціалізованим програмним забезпеченням в якості підсистеми верхнього рівня АСК СЕЕС, а мікроконтролерів – в якості основ апаратних пристроїв автоматизації дозволяє представити структуру програмного забезпечення всіх компонентів системи управління у вигляді паттернової мережі з цілями: визначення складу і схеми взаємодії програмних об'єктів, що входять до складу системи управління; формального опису структури системи управління у вигляді комплекту утворюючих; обчислення основних показників, що характеризують інформаційний канал – часу реакції системи і завантаженості каналу зв'язку на основі значень атрибутів утворюючих і відносин зв'язку.

На рис. 7.18 наведена структура програмного забезпечення системи управління електростанцією у вигляді паттернової мережі. Умовно її можна розділити на три складових: спеціалізоване програмне забезпечення для персонального комп'ютера, що містить мнемосхему електростанції (режим моніторингу та управління електростанцією; на рис. 7.18 представлено утворюючими $g1$ – $g20$), програмне забезпечення для засобів автоматизації (утворюючі $g22$ – $g24$) та програмне забезпечення інтерфейсного пристрою (утворююча $g21$).

До складу електростанції в загальному випадку входять m секцій по n_i (i – [1, m]) блоків дизель-генераторних агрегатів, захистів ДГА, засобів відображення параметрів дизеля і генератора; n_i–1 блоків контролю і управління процесом синхронізації ДГА з шинами ГРЩ; $2n_i$ імпульсних кнопок управління станом дизеля; k_i асинхронних двигунів і навантажень; n_i+k_i автоматичних вимикачів і світлодіодів для індикації їх станів; одного пристрою розподілу навантажень між паралельно працюючими ГА.

Кожній складової електростанції відповідає певний об'єкт програмного забезпечення, реалізацію поведінки якого інкапсулює утворююча: $g1$ – Текстове поле для відображення електричних параметрів; $g2$ – Шкальний індикатор; $g3$ – Стрілочний індикатор; $g4$ – Генератор; $g5$ – Дизель (спільно з блоком Генератора утворює блок ДГА); $g6$ – діалогове вікно «Параметри дизеля»; $g7$ – діалогове вікно «Осцилограми»; $g8$ – Система синхронізації; $g9$ – Захист ГА; $g10$ – Автоматичний вимикач; $g11$ – Світлодіод; $g12$ – Навантаження; $g13$ – Асинхронний двигун; $g14$ – Кнопка управління; $g15$ – Система розподілу навантаження.

Утворюючі $g1$ – $g3$ призначені для позначення блоків індикації вимірюваних електричних параметрів; з кожним ДГА може бути пов'язано довільна, але кінцева, кількість таких блоків (позначені як l_1, l_2 і l_3 відповідно).

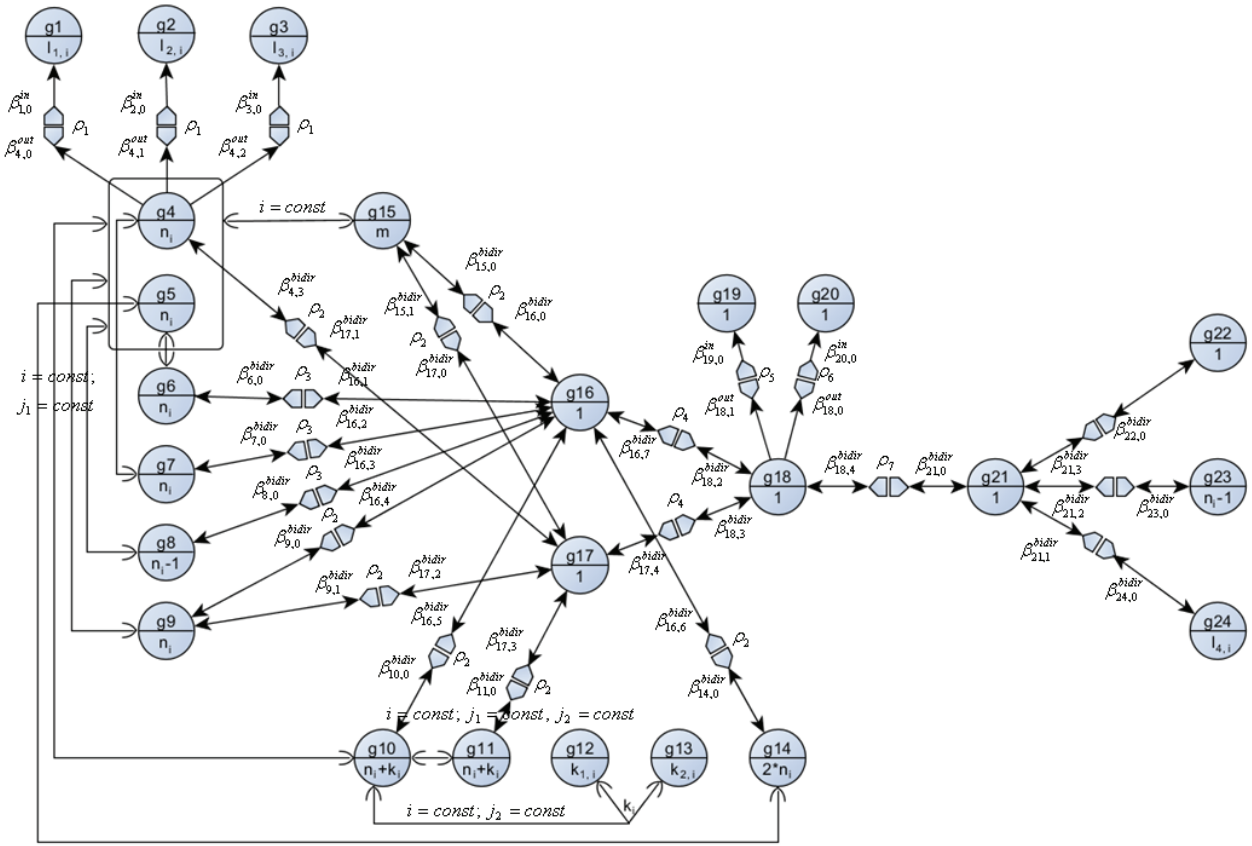

g1
$l_{1,i}$
g2
$l_{2,i}$
g3
$l_{3,i}$
g4
n_i
g5
n_i
g6
n_i
g7
n_i
g8
n_i-1
g9
n_i
g10
n_i+k_i
g11
n_i+k_i
g12
$k_{1,i}$
g13
$k_{2,i}$
g14
2*n_i
g15
m
g16
1
g17
1
g18
1
g19
1
g20
1
g21
1
g22
1
g23
n_i-1
g24
$l_{4,i}$
$i = const$
$i = const;$
$j_1 = const$
$i = const;\ j_1 = const,\ j_2 = const$
$i = const;\ j_2 = const$
k_i

Рис. 7.18.

Контроль і керування електростанцією здійснюється віддалено, шляхом відображення та обробки отриманих від апаратних засобів автоматизації даних і передачі їм управляючих впливів (сигналів на зміну станів реле захистів, автоматичних і генераторних вимикачів, частот обертання дизелів і напруг збудження генераторів). Для врахування пріоритету виконуваних операцій і контролю за процесом обміну даними необхідно створити єдині черги запитів та очікуваних відповідей.

Всю сукупність запитів можна розділити на дві множини: запити інформаційного типу (для отримання значення параметрів ДГА і станів автоматичних і генераторних вимикачів) і керуючого типу (команди на зміну структури електростанції та/або параметрів електроенергії, що віддається ДГА в мережу), тому формуванню черг запитів і відповідей відповідають три утворюючих: $g16$ – формувач черги змінної довжини; $g17$ – формувач черги постійної довжини; $g18$ – формувач кінцевої черги.

До функцій програмного забезпечення верхнього рівня відносяться також накопичення статистичних даних про інформаційні потоки (на рис. 7.18 за цю функцію відповідає утворююча $g19$) і зберігання в базі даних значень потужностей, що віддаються ДГА в мережу і споживаються окремими навантаженнями (утворююча $g20$).

Враховуючи наявність нерозривного логічного зв'язку між блоками:

1) Генераторів, Дизелів, діалогових вікон для відображення параметрів дизеля і генератора, Захистів ДГА, Систем синхронізації, Кнопок управління станом дизеля, генераторних вимикачів (представлених компонентом Автоматичний вимикач) і Світлодіодів-індикаторів станів генераторних вимикачів (на рис. 3.17 логічні зв'язки позначені закругленими стрілками; даному зв'язку відповідає i = const, j_1 = const ($j_1 \in [1, n_i]$));

2) Навантаження і Асинхронних двигунів і Автоматичних вимикачів зі Світлодіодами-індикаторами їх станів (i = const, j_2 = const ($j_2 \in [1, k_i]$));

і умовно нерозривного зв'язку між наведеними групами, структуру програмного забезпечення для комп'ютера для кожної секції (i = const) можна записати у вигляді підкомплекта:

$$G^i = G_1^i + G_2^i + G_0^i, \tag{7.1}$$

де

$$\begin{aligned} G_1^i = \Big\{ &[g1]^{l_1^i}, [g2]^{l_2^i}, [g3]^{l_3^i}, [g4]^{n_i}, [g5]^{n_i}, [g6]^{n_i}, [g7]^{n_i}, [g8]^{n_i - 1}, \\ &[g9]^{n_i}, [g10]^{n_i}, [g11]^{n_i}, [g14]^{2 \cdot n_i}, [g15]^1 \Big\}, \end{aligned} \tag{7.2}$$

$$G_2^i = \Big\{ [g12]^{k_1^i}, [g13]^{k_2^i}, [g10]^{k_i}, [g11]^{k_i} \Big\}, \tag{7.3}$$

$$G_0^i = \Big\{ [g16]^1, [g17]^1, [g18]^1, [g19]^1, [g20]^1 \Big\}, \tag{7.4}$$

і позначення $[g1]^{l_1^i}$ є скороченим записом позначення кількості екземплярів у підкомплекті ($l_1^i = \#(g1, G_1^i)$).

Тоді структура всього програмного забезпечення для управління електростанцією являє собою комплект, утворений сумою суми підкомплектов G_1^i і G_2^i і перетину подкомплектов G_0^i кожної секції:

$$G = \sum_{i=1}^{m}\left(G_1^i + G_2^i\right) + \bigcap_{i=1}^{m} G_0^i \cdot \tag{7.5}$$

Згідно дискретній теорії паттернів кожній утворюючій відповідає певна арность. Так, з j_1-тим ДГА в i-тій секції може бути зв'язано l_1^{i,j_1} Текстових полів, l_2^{i,j_1} Шкальних і l_3^{i,j_1} Стрілочних індикаторів, при цьому відношення зв'язку ρ_1 є відношенням «один до багатьох», тобто утворююча $g4$ може бути зв'язана з довільним, але кінцевим, числом утворюючих $g1$ – $g3$, а утворюючі $g1$ – $g3$ лише з однією $g4$. Звідси арності $g1$ – $g3$ рівні одиниці:

$$\omega\left(g1^{i,j_1}\right) = \omega\left(g2^{i,j_1}\right) = \omega\left(g3^{i,j_1}\right) = 1, \tag{7.6}$$

що відповідає принципу реалізму: кожен блок індикації відображає дані, що отримуються лише від одного блоку ДГА, у той час як ці дані (електричні параметри генератора) можуть бути відображені в різних комбінаціях за допомогою різних засобів індикації. Сам же блок генераторного агрегату крім зв'язків з

$$\sum_{p=1}^{3} l_p^{i,j_1}$$

блоками індикації повинен підтримувати інформаційний зв'язок із засобами автоматизації, що здійснюється за допомогою блоку формування черг і на рис. 7.18 відображено у вигляді відношення зв'язку ρ_2 множинного включення (відносини «багато до одного») – між утворюючими $g4$ і $g17$: для кожного j_1-ого ДГА в i-тій секції виконується опитування значень електричних параметрів генератора. Звідси слідує арность $g4$:

$$\omega\left(g4^{i,j_1}\right) = \sum_{p=1}^{3} l_p^{i,j_1} + 1. \tag{7.7}$$

Аналогічно арності утворюючих $g9$ – $g11$, $g14$ – $g15$:

$$\begin{gathered}\omega\left(g10^{i,j_1}\right) = \omega\left(g11^{i,j_1}\right) = \omega\left(g14^{i,j_1}\right) = 1; \\ \omega\left(g9^{i,j_1}\right) = \omega\left(g15^{i,j_1}\right) = 2.\end{gathered} \tag{7.8}$$

Відношення зв'язку ρ_3 – відношення «один з багатьох до одного» означає можливість наявності зв'язку в кожен момент часу лише одного із множини екземплярів відповідної утворюючої з іншою утворюючою. Наприклад, зв'язок $\beta_{6,0}\rho_3\beta_{16,1}$ розшифровується як з n_i-тої кількості блоків «Параметри дизеля» (утворююча $g6$) в кожен момент часу тільки один може брати участь в обміні даними із засобами автоматизації за допомогою зв'язку з блоком формування черг змінної довжини (утворююча $g16$). Таким чином, арності утворюючих $g16$ і $g17$ розраховуються як:

$$\begin{gathered}\omega\left(g16^i\right) = 4 + 4 \cdot n_i + k_i; \\ \omega\left(g17^i\right) = 1 + 3 \cdot n_i + k_i,\end{gathered} \tag{7.9}$$

тобто арность зв'язків для відношень ρ_2 дорівнює потужності відповідної утворюючої в i-тому підкомплекті, а для відношень ρ_3 – одиниці.

Утворююча $g18$ є об'єднуючою відносно утворюючих, що відображають формування черг постійної і змінної довжин ($g16$ і $g17$), і відповідає за

формування кінцевих черг запитів і відповідей для організації обміну даними по фізичному каналу зв'язку. Оскільки через неї проходить вся інформація про параметри електроенергії і стани автоматичних вимикачів, до функцій блоку, представленого утворюючою $g18$, відноситься передача статистичних даних про інформаційні потоки і параметри електроенергії у базу даних, за що відповідають блоки, представлені утворюючими $g19$ і $g20$. Таким чином, відношення зв'язку ρ_5 і ρ_6 є «чисто» інформаційними відношеннями «один до одного» без безпосереднього обміну даними із засобами автоматизації, у той час як відношення ρ_4 –відношення включення (також «один до одного») означає повне включення даних, отриманих від одних утворюючих в інші. Зв'язок програмного забезпечення для комп'ютера з пристроєм, що забезпечує маршрутизацію пакетів даних, позначений відношенням ρ_7.

Таким чином, використання апарату паттернових мереж дозволяє формально описати структуру програмного забезпечення АСК СЕЕС; подібна форма подання дає можливість:

– врахувати всі можливі схеми взаємодії програмних об'єктів при проведенні функціонального тестування системи;

– досить просто вносити зміни до складу програмного забезпечення шляхом додавання (чи видалення) утворюючих та організації їх взаємодії з існуючими утворюючими за допомогою певних відносин зв'язку;

– врахувати поведінку всіх об'єктів ПЗ, що впливають на час реакції системи на дії оператора або на зміни електромагнітних процесів в об'єкті управління і впливають на завантаженість каналу обміну даними із засобами автоматизації;

– формально виконати реорганізацію структури програмного забезпечення при необхідності зміни часу реакції системи.

7.4. Мультикомп'ютерна модель для дослідження процесів управління СЕЕС

Для всебічної перевірки алгоритмів керування електростанцією необхідна наявність декількох генераторів. Крім того, необхідною є перевірка поведінки системи в разі виходу одного з генераторів за межі допустимої області роботи. У роботі пропонується для цих цілей використовувати моделі, що імітують поведінку об'єкта автоматизації, пов'язаних з ним локальних систем управління нижнього рівня і засобів автоматизації середнього рівня як єдиного цілого. Для забезпечення максимального наближення до реальних процесів, що відбуваються в інформаційному каналі системи управління, пропонується розробити моделі, які на рівні інтерфейсу збігаються з розробленими засобами автоматизації. Тоді, якщо для моделювання роботи АСК замість кожного об'єкта автоматизації та пов'язаних з ним пристроїв використовувати окремий комп'ютер з відповідною запущеною моделлю, можна перевірити роботу підсистеми верхнього рівня АСК СЕЕС при управлінні електростанцією будь-якої топології; реакцію системи на виникнення різних

позаштатних ситуацій; заповнити базу даних відомостями для розрахунку часових характеристик системи управління.

Вимогою до всіх імітаційних моделей є наявність інтерфейсу для роботи в інформаційній мережі (підтримка протоколу Modbus RTU з можливістю зміни адреси пристрою). Інші вимоги до моделей, відповідних різним засобам автоматизації, різні і далі розглянуті окремо.

Необхідні характеристики моделі контролера автоматизації (рис. 7.19): імітація наявності дискретних входів і виходів з можливістю зміни відповідної кожному з них адреси регістру; наявність блоку установки основних параметрів силового агрегату (коефіцієнта потужності, частоти, діючих значень напруги і струму) та моделювання відповідно до встановлених значень швидкоплинних процесів для отримання осцилограм напруги та струму генератора.

Відлагодити програмне забезпечення системи синхронізації та розподілу навантажень між працюючими на загальне навантаження судновими генераторами, а також перевірити схемотехнічні блоки перетворення аналогових сигналів та індикації можна за допомогою імітаційної моделі пристрою, наведеної на рис. 7.20.

У розробленій програмі для мікроконтролера, керуючого системою моніторингу якості електроенергії, можна виділити два паралельні процеси – виконання дій, пов'язаних з визначенням коефіцієнтів несиметрії, і обмін даними з КСУ. Реалізацію першого процесу можна представити наступною послідовністю дій: вимірювання фазних напруг, вимірювання кутів зсуву між фазними напругами і розрахунок коефіцієнтів несиметрії. Для забезпечення відповідності роботи пристрою необхідним часовим характеристикам (не менше 9 спостережень за 3 сек.) ці дії повинні виконуватися циклічно з інтервалом приблизно 0,3 сек. Реалізувати наведену вимогу можна за допомогою таймера, налаштованого на переривання з необхідною періодичністю, який і буде забезпечувати циклічність виконання вимірювань та розрахунків.

Для вимірювання міжфазних кутів пропонується використання методу підрахунку імпульсів за інтервал часу. Для отримання значень кутів у системних одиницях (радіанах або градусах) наведеним способом необхідно також вимірювати і період однієї з фазних напруг. Використання переривань від нуль-детекторів фазних напруг дозволить реалізувати процеси вимірювань міжфазних кутів та періоду напруги паралельно.

Розроблені в процесі виконання проекту засоби автоматизації СЕЕС та відповідне програмне забезпечення є складовими АСК СЕЕС, яка, в свою чергу, є ієрархічною та складається з трьох вертикальних рівнів. Оскільки для зв'язку засобів автоматизації нижнього і середнього рівнів використовуються дискретні або аналогові сигнали, що допускають лише зв'язок «один до одного», а для зв'язку підсистем середнього і верхнього рівнів використовується канал, який реалізує топологію «загальна шина».

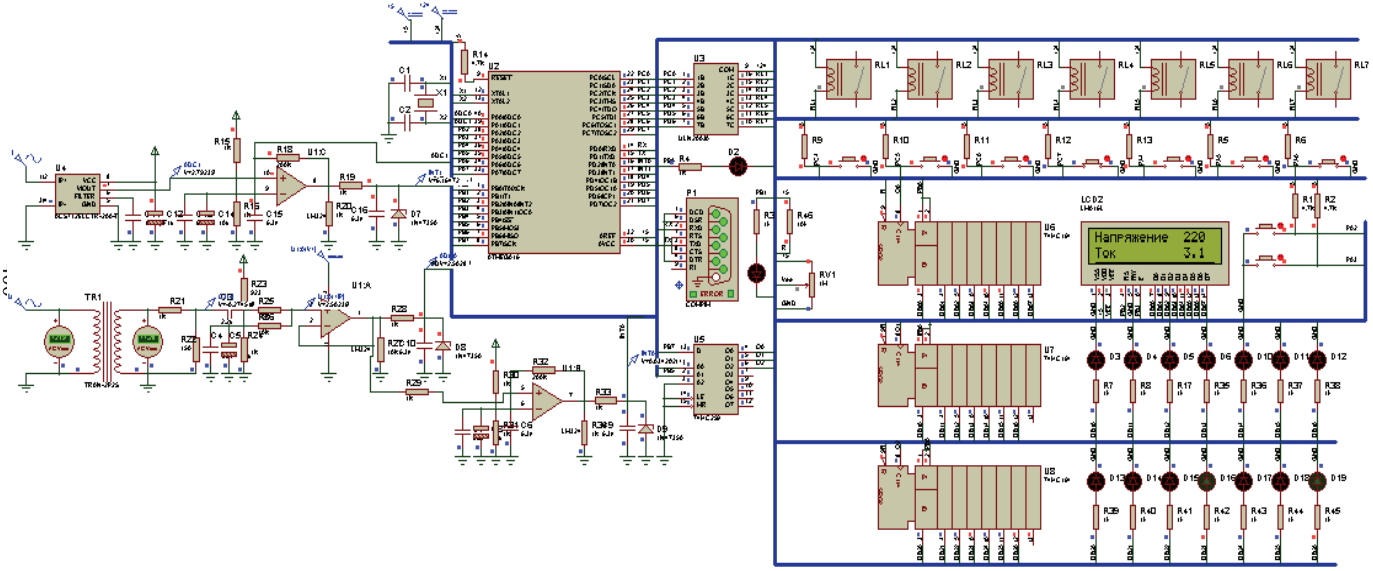


Рис. 7.19.

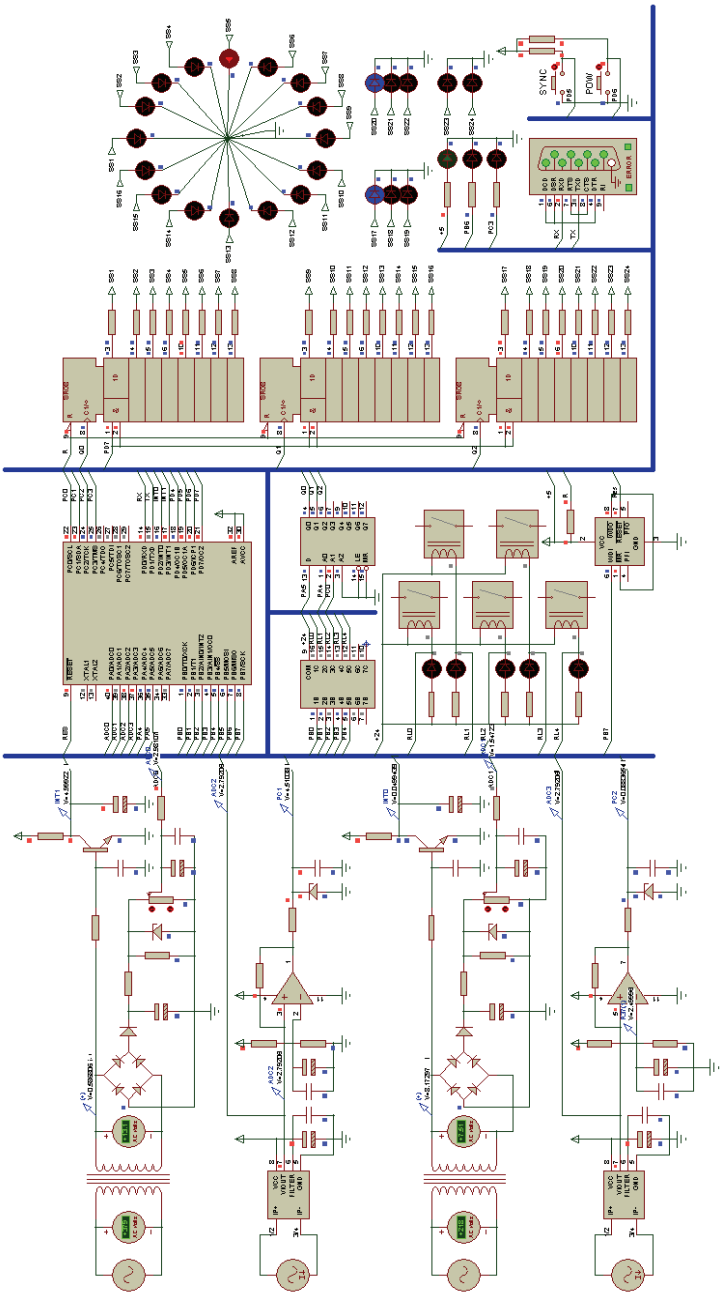


Рис. 7.20.

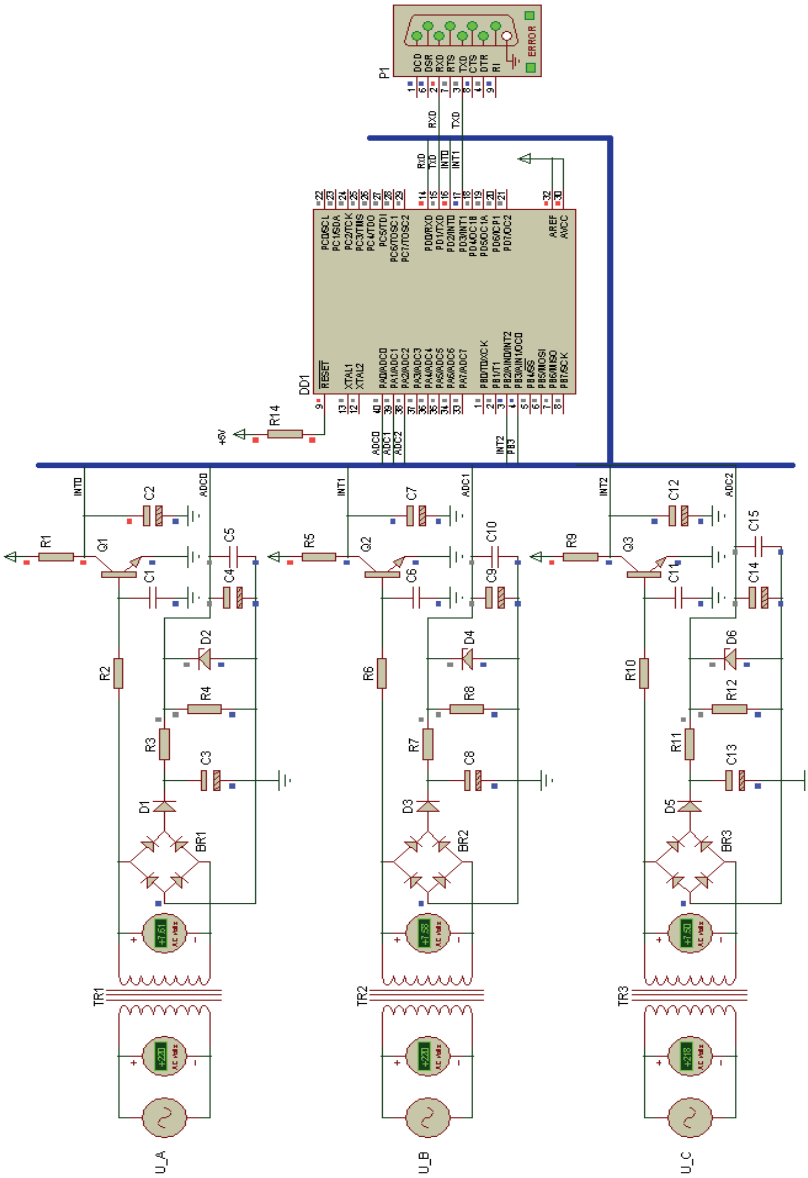

P1
DCD
DSR
RXD
RTS
CTS
DTR
RI
ERROR
RXD
TXD
DD1
RESET
XTAL1
XTAL2
+5V
R14
INT0
INT1
INT2
ADC0
ADC1
ADC2
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
Q1
Q2
Q3
D1
D2
D3
D4
D5
D6
BR1
BR2
BR3
TR1
TR2
TR3
U_A
U_B
U_C

Рис. 7.21.

Для вимірювання міжфазних кутів пропонується використання методу підрахунку імпульсів за інтервал часу. Для отримання значень кутів у системних одиницях (радіанах або градусах) наведеним способом необхідно також вимірювати і період однієї з фазних напруг. Використання переривань від нуль-детекторів фазних напруг дозволить реалізувати процеси вимірювань міжфазних кутів та періоду напруги паралельно.

Розроблені в процесі виконання проекту засоби автоматизації СЕЕС та відповідне програмне забезпечення є складовими АСК СЕЕС, яка, в свою чергу, є ієрархічною та складається з трьох вертикальних рівнів. Оскільки для зв'язку засобів автоматизації нижнього і середнього рівнів використовуються дискретні або аналогові сигнали, що допускають лише зв'язок «один до одного», а для зв'язку підсистем середнього і верхнього рівнів використовується канал, який реалізує топологію «загальна шина».

Для всебічної перевірки алгоритмів керування електростанцією необхідна наявність декількох генераторів. Крім того, необхідною є перевірка поведінки системи в разі виходу одного з генераторів за межі допустимої області роботи. У роботі пропонується для досягнення цієї мети використовувати моделі, що імітують поведінку об'єкта автоматизації, пов'язаних з ним локальних систем управління нижнього рівня і засобів автоматизації середнього рівня як єдиного цілого. Для забезпечення максимального наближення до реальних процесів, що відбуваються в системи управління, розроблено моделі, які на рівні інтерфейсу збігаються з розробленими засобами автоматизації. Тоді, якщо для моделювання роботи АСК СЕЕС замість кожного об'єкта автоматизації та пов'язаних з ним пристроїв використовувати окремий комп'ютер з відповідною запущеною моделлю, можна перевірити роботу підсистеми верхнього рівня АСК СЕЕС при управлінні електростанцією будь-якої топології; реакцію системи на виникнення різних позаштатних ситуацій; заповнити базу даних відомостями для розрахунку часових характеристик системи управління.

Вимогою до всіх імітаційних моделей є наявність інтерфейсу для роботи в інформаційній мережі (підтримка протоколу Modbus RTU з можливістю зміни адреси пристрою). Інші вимоги до моделей, відповідних різним засобам автоматизації, різні і далі розглянуті окремо. На рис. 7.22 та рис. 7.23 представлено людино-машинні інтерфейси, за допомогою яких можна керувати процесом моделювання роботи розроблених мікропроцесорних систем. На рис. 7.24 та рис. 7.25 представлені відповідні блок-схеми програм.

Необхідні характеристики моделі мікропроцесорної системи керування: імітація наявності дискретних входів і виходів з можливістю зміни відповідної кожному з них адреси регістру; наявність блоку установки основних параметрів силового агрегату (коефіцієнта потужності, частоти, діючих значень напруги і струму) та моделювання відповідно до встановлених значень швидкоплинних процесів для отримання осцилограм напруги та струму генератора.

Рис. 7.22.

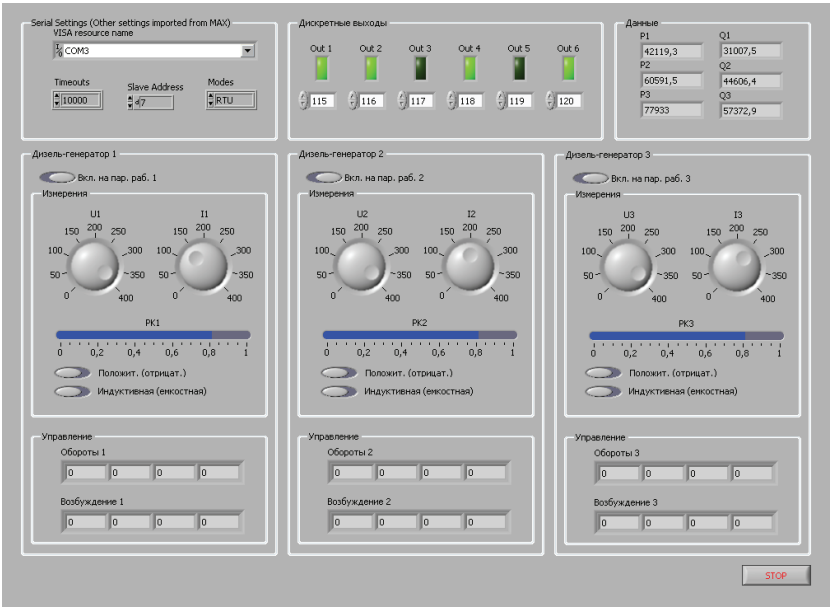


Рис. 7.23.

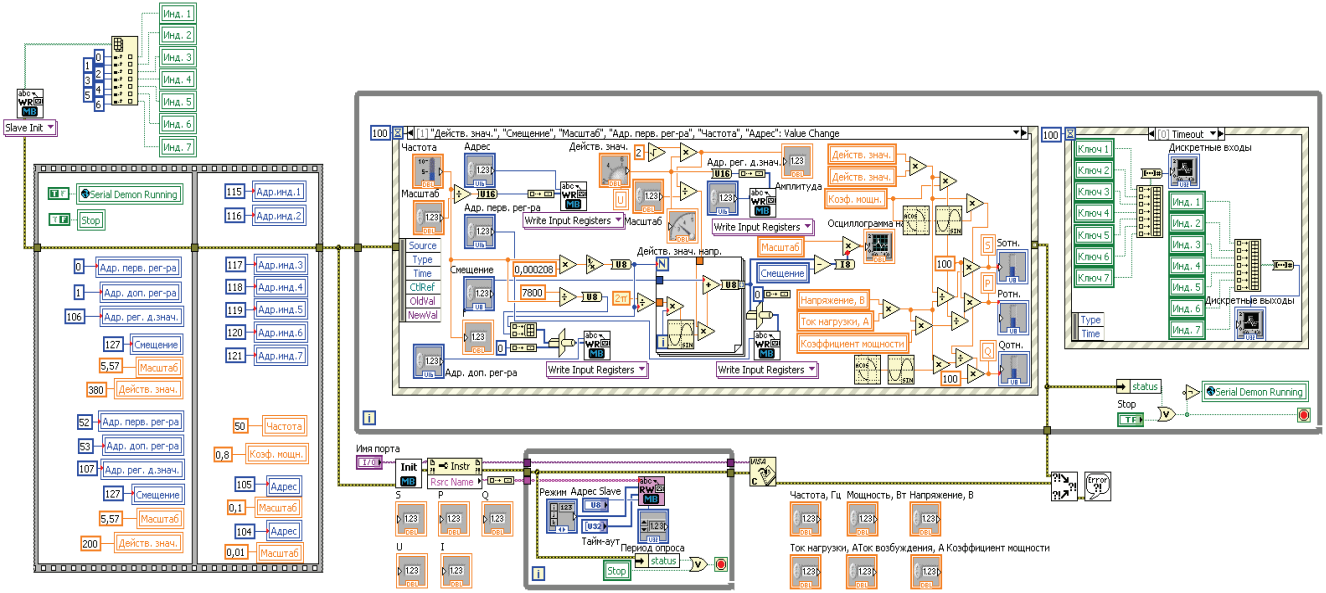

Slave Init
Инд. 1
Инд. 2
Инд. 3
Инд. 4
Инд. 5
Инд. 6
Инд. 7
Serial Demon Running
Stop
Адр. перв. рег-ра
Адр. доп. рег-ра
Адр. рег. д.знач.
Смещение
Масштаб
Действ. знач.
Адр.инд.1
Адр.инд.2
Адр.инд.3
Адр.инд.4
Адр.инд.5
Адр.инд.6
Адр.инд.7
Частота
Коэф. мощн.
Адрес
Write Input Registers
Source
Type
Time
CtlRef
OldVal
NewVal
Действ. знач. напр.
Амплитуда
Осциллограмма
Напряжение, В
Ток нагрузки, А
Коэффициент мощности
Sотн.
Pотн.
Qотн.
Ключ 1
Ключ 2
Ключ 3
Ключ 4
Ключ 5
Ключ 6
Ключ 7
Дискретные входы
Дискретные выходы
status
Имя порта
Init
Rsrc Name
Instr
Режим
Адрес Slave
Тайм-аут
Период опроса
Частота, Гц
Мощность, Вт
Напряжение, В
Ток нагрузки, А
Ток возбуждения, А
Коэффициент мощности

Рис. 7.24.

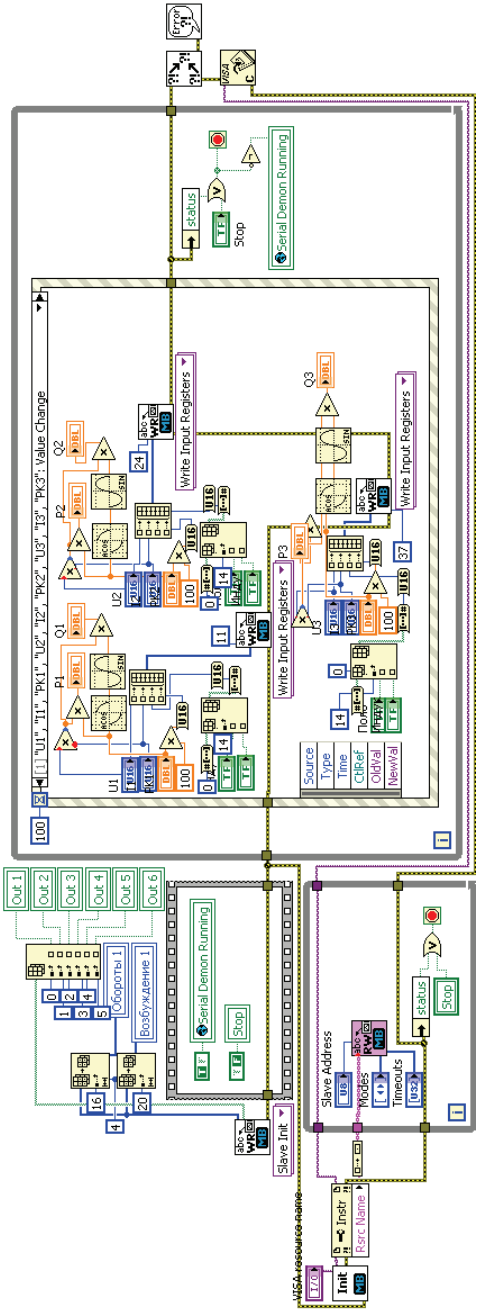

"U1", "I1", "Pk1", "U2", "I2", "Pk2", "U3", "I3", "Pk3": Value Change
Write Input Registers
Source
Type
Time
CtlRef
OldVal
NewVal
Serial Demon Running
Stop
status
Slave Address
Modes
Timeouts
Slave Init
Out 1
Out 2
Out 3
Out 4
Out 5
Out 6
Обороты 1
Возбуждение 1
VISA resource name
Instr
Rsrc Name
Init

Рис. 7.25.

Характеристики моделі системи синхронізації і розподілу навантажень: імітація включення ДГА на паралельну роботу з мережею (модель має засоби для емуляції дискретних входів або для зміни змісту регістрів, відповідних запуску або останову ДГА, ініціювання процесу його синхронізації з мережею, зміни параметрів керуючих сигналів для імітації впливів на регулятори частоти і напруги в процесі синхронізації); імітація підключення/відключення навантаження до паралельно працюючих ДГА за допомогою зміни станів дискретних виходів; імітація розподілу активного і реактивного навантаження, для управління процесом якого використовується вміст регістрів, які задають параметри сигналів керування, що надходять на регулятори частоти і напруги (імітація виражається в автоматичній зміні моделюваних діючого значення струму і значення коефіцієнта потужності); емуляція виникнення аварійних ситуацій (переходу ДГА в режим двигуна або втрата ним збудження; на рівні моделі це має бути виражене наявністю перемикачів для зміни напрямків потоків активної і реактивної потужностей, що віддаються ДГА в мережу).

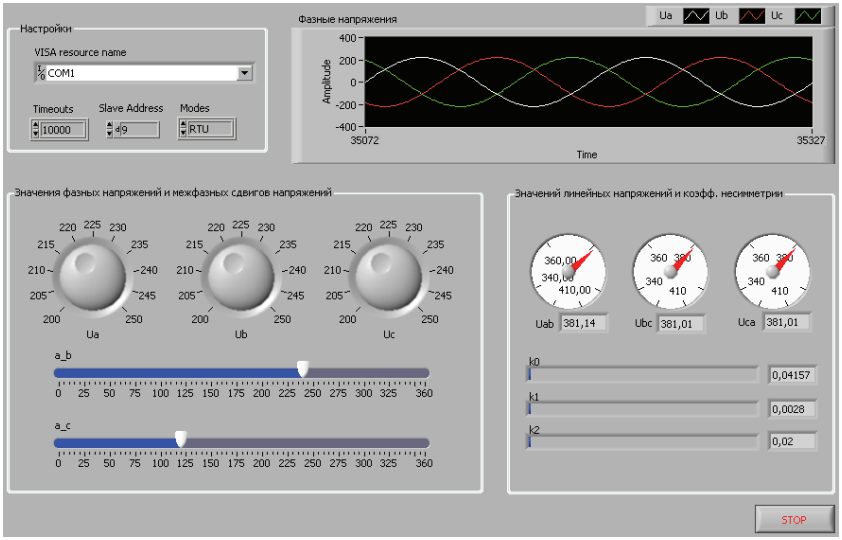


Рис. 7.26.

Характеристики моделі системи моніторингу якості електроенергії: наявність блоку установки діючих значень всіх фазних напруг мережі і значень кутів зсуву між векторами напруг; відображення осцилограм напруг трифазної мережі, виконання розрахунку коефіцієнтів несиметрії і їх відображення.

Наведені моделі можуть бути використані при проведенні функціонального тестування програмного забезпечення, а також для дослідження процесу управління судновою електроенергетичною системою.

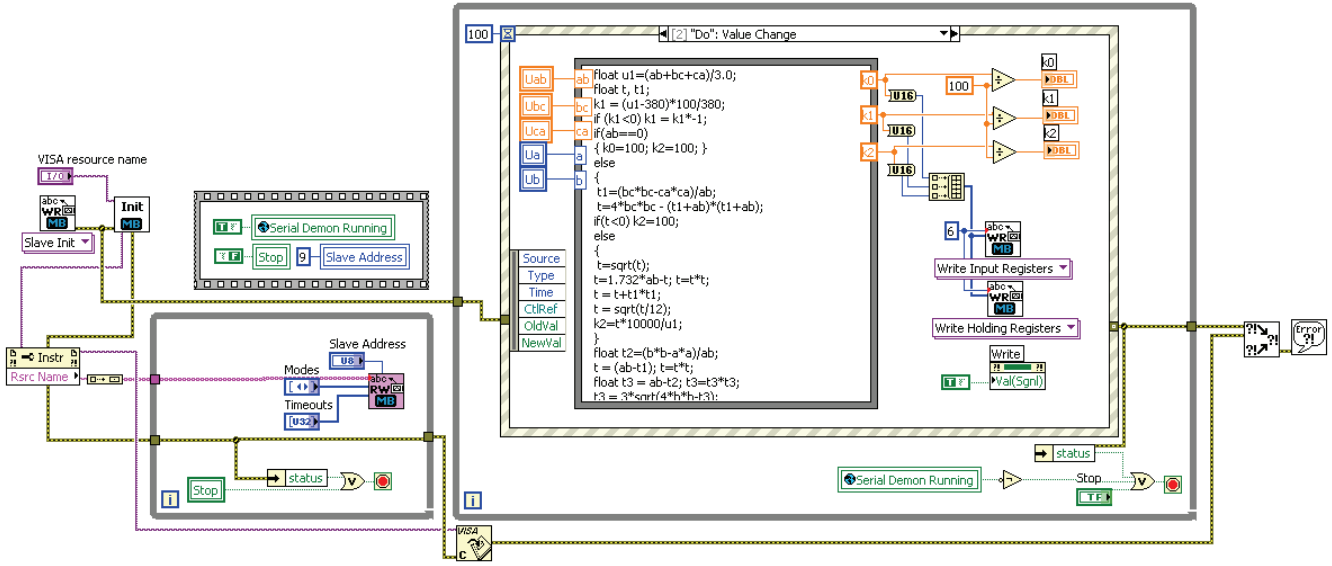

VISA resource name
Init
Slave Init
Serial Demon Running
Stop
Slave Address
Instr
Rsrc Name
Modes
Timeouts
status
Write Input Registers
Write Holding Registers
Write
Val(Sgnl)
Error
"Do": Value Change
float u1=(ab+bc+ca)/3.0;
float t, t1;
k1 = (u1-380)*100/380;
if (k1<0) k1 = k1*-1;
if(ab==0)
{ k0=100; k2=100; }
else
{
t1=(bc*bc-ca*ca)/ab;
t=4*bc*bc - (t1+ab)*(t1+ab);
if(t<0) k2=100;
else
{
t=sqrt(t);
t=1.732*ab-t; t=t*t;
t = t+t1*t1;
t = sqrt(t/12);
k2=t*10000/u1;
}
float t2=(b*b-a*a)/ab;
t = (ab-t1); t=t*t;
float t3 = ab-t2; t3=t3*t3;
t3 = 3*sqrt(4*b*b-t3);
Uab
Ubc
Uca
Ua
Ub
Source
Type
Time
CtlRef
OldVal
NewVal
Serial Demon Running
Stop
status

Рис. 7.27.

7.5. Фізична модель суднової електроенергетичної системи та її автоматизованої системи керування

Для дослідження кількісних та якісних показників розроблених систем автоматизації та методів керування було розроблено фізичну модель СЕЕС, яку розміщено в науково-дослідній лабораторії енергетичної електроніки і електромагнітної сумісності та проблем керування в електроенергетиці (рис. 7.28), що створена в Національному університеті кораблебудування ім. адм. Макарова на кафедрі програмованої електроніки, електротехніки і телекомунікацій.

Рис. 7.28.

Фізична модель СЕЕС складається з двох синхронних генераторів потужністю 2,2 кВт та приводними асинхронними двигунами, а також двигуна внутрішнього згоряння, мікропроцесорних систем керування збору, обробки та відображення інформації, що розташовано на спеціалізованих дослідницьких стендах. Стенди укомплектовані частотними електроприводами Altivar, мікропроцесорними системами синхронізації генераторів, мікропроцесорними системами збудження синхронних генераторів, цифровими приладами для контролю параметрів електроенергії. Всі засоби автоматизації об'єднані в мікропроцесорну мережу. Керування СЕЕС здійснює розроблена автоматизована система керування, яка включає в себе контролер автоматизації та мікропроцесорні системи керування, збору та обробки інформації, основними функціями яких є:

– керування режимом роботи електростанції;

– керування збудженням синхронних генераторів;

– автоматична точна синхронізація генераторів

– комутація потужних споживачів електроенергії;

– вимірювання активної та реактивної потужностей генераторів та розподіл потужностей при роботі генераторів на загальне навантаження;

– контроль параметрів силових агрегатів;

– контроль якості електроенергії в мережі;

– моніторинг частоти напруги генераторів і керування обертами приводних двигунів;

– моніторинг режиму роботи системи і аварійних ситуацій;

– статистична обробка показників якості електроенергії.

На основі структурних схем і алгоритмів програм засобів автоматизації, були розроблені експериментальні зразки контролера автоматизації та системи синхронізації і розподілу навантажень.

Розроблені засоби автоматизації використовуються в складі щита керування генераторним агрегатом (рис. 7.29, а), який входить до складу АСК фізичною моделлю суднової електроенергетичної системи (рис. 7.29, б).

а

б

Рис. 7.29.

До складу моделі CEEC окрім щита управління входять приводний асинхронний двигун з частотним електроприводом Altivar 38, синхронний генератор потужністю 2,2 кВт із системою збудження і асинхронний двигун потужністю 2,2 кВт, який використовується в якості навантаження контроль параметрів та керування роботою приводних двигунів, синхронних генераторів і двигунів може здійснюватися як місцево, за допомогою пристроїв, розташованих на стенді, так і дистанційно з пульта керування (рис. 7.30).

Рис. 7.30.

Пульт керування СЕЕС складається з трьох секцій:

– секція моніторингу параметрів синхронних генераторів та контролю якості електроенергії;

– секція керування режимами роботи електроенергетичної установки, керування засобами автоматизації та силовим обладнанням. забезпечує керування пуском/зупинкою двигунів, керування збудженням синхронних генераторів та процесом їх синхронізації з мережею, виконує функції розподілу потужності між паралельно працюючими генераторами;

– секція керування та контролю параметрів приводних двигунів генераторів та керування навантаженням. Забезпечує контроль параметрів двигунів, плавний пуск та зупинку асинхронних двигунів, виконує відображення аварійних сигналів, дозволяє програмувати режими роботи приводних двигунів.

Програмні засоби моніторингу забезпечують відображення осцилограм напруги та струму кожного з генераторів, виконують спектральний аналіз напруги та струму. На основі цих даних розраховується коефіцієнт гармонік, який є показником якості електроенергії. Виконується розрахунок значень активної, реактивної й повної потужності генератора, а також cosφ. Використання стилізованих індикаторів дозволяє відобразити параметри у зручній для оператора формі.

На рис. 7.31 наведений алгоритм проведення фізичного моделювання, метою якого була перевірка розроблених апаратно-програмних засобів для моніторингу та управління СЕЕС.

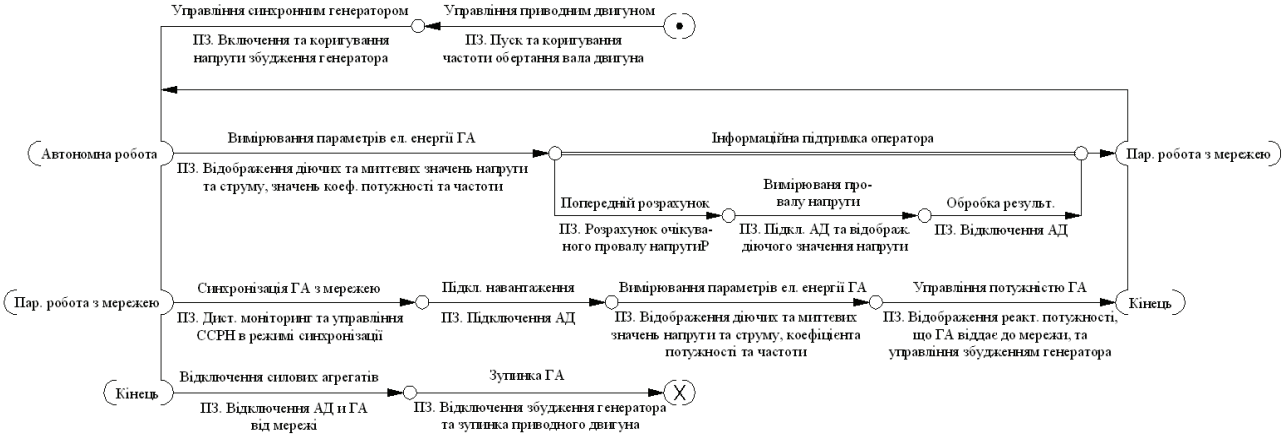

Управління синхронним генератором
Управління приводним двигуном
ПЗ. Включення та коригування напрути збудження генератора
ПЗ. Пуск та коригування частоти обертання вала двигуна
Автономна робота
Вимірювання параметрів ел. енергії ГА
Інформаційна підтримка оператора
Пар. робота з мережею
ПЗ. Відображення діючих та миттєвих значень напруги та струму, значень коеф. потужності та частоти
Попередній розрахунок
Вимірювання провалу напруги
Обробка результ.
ПЗ. Розрахунок очікуваного провалу напругиР
ПЗ. Підкл. АД та відображ. діючого значення напруги
ПЗ. Відключення АД
Пар. робота з мережею
Синхронізація ГА з мережею
Підкл. навантаження
Вимірювання параметрів ел. енергії ГА
Управління потужністю ГА
Кінець
ПЗ. Дист. моніторинг та управління ССРН в режимі синхронізації
ПЗ. Підключення АД
ПЗ. Відображення діючих та миттєвих значень напруги та струму, коефіцієнта потужності та частоти
ПЗ. Відображення реакт. потужності, що ГА віддає до мережи, та управління збудженням генератора
Кінець
Відключення силових агрегатів
Зупинка ГА
ПЗ. Відключення АД и ГА від мережі
ПЗ. Відключення збудження генератора та зупинка приводного двигуна

Рис. 7.31.

7.6. Використання фізичної моделі CEEC для підготовки фахівців з експлуатації суднового електрообладнання

Для ефективного управління електроенергетичною системою програмні засоби, які для цього використовуються, повинні відповідати нормам якості програмного забезпечення, визначених у стандартах серії ISO / IEC (25000-25099); задовольняти вимогам до людино-машинних інтерфейсів, визначених у стандартах серії ISO / IEC 9241 та мати засоби для дистанційного контролю та управління електроенергетичними процесами, регламентовані Правилами класифікації та побудови морських суден. Однак для безпечного використання сучасних АСК суднові механіки та електромеханіки також повинні мати відповідні професійні знання та навички, забезпечуючи як вахтенне, так і безвахтенне обслуговування суднового обладнання і засобів автоматики. Вимоги до оператора управління судновою електроенергетичною установкою визначені в Міжнародній конвенції про підготовку і дипломування моряків та несення вахти. Згідно з цими ж вимогами судновий механік повинен пройти курс практичної підготовки тривалістю не менше 12 місяців (6 з яких – на судні), причому навчання має проходити в спеціально обладнаних лабораторних і тренажерних комплексах морських навчальних закладів.

В роботі [58] наведено опис тренажера суднової автоматизованої електроенергетичної системи, реалізованого на базі SCADA-програми M-Vision, головного розподільного щита із засобами автоматизації фірми Selco, трьох основних і одного аварійного генераторних агрегатів і суднових споживачів електроенергії. Описаний тренажер забезпечує адекватне відтворення експлуатаційних ситуацій з технічного обслуговування реального суднового устаткування і призначений для спеціальної підготовки суднових електромеханіків. У дослідженні пропонується використання двоступеневого навчання студентів морських навчальних закладів: первинного з використанням реального програмного забезпечення верхнього рівня системи управління та віртуальних моделей засобів і об'єктів автоматизації всіх інших рівнів; і вторинного − на основі реально функціонуючого і максимально наближеного до сучасної конфігурації суднового устаткування.

Цільовими функціями людино-машинної системи при навчанні оператора за допомогою апаратно-програмної моделі автоматизованої системи управління судновою електроенергетичною установкою є: забезпечення оператора адекватною інформацією про режими роботи електрообладнання; формування навичок і закріплення знань у оператора, пов'язаних з його діями як в штатному режимі роботи електроенергетичної системи, так і при виникненні аварійних ситуацій. Оскільки управління електроенергетичною системою пов'язане з використанням спеціалізованого програмного забезпечення верхнього рівня, пропонується його використання на обох етапах навчання.

Завданням первинного навчання студентів морських навчальних закладів є вивчення алгоритмів пуску і зупинки дизель-генераторних агрегатів, синхронізації ДГА з мережею, розподілу активного і реактивного навантаження між паралельно працюючими генераторними агрегатами; алгоритмів перевірки і регулювання систем захистів дизель-генератора від перевантаження по потужності, втрати збудження, зворотної потужності та ін. Після вивчення теоретичного матеріалу (в першу чергу відповідних нормативних документів) необхідна його перевірка і закріплення. З цими цілями пропонується формування класів нееквівалентних станів енергосистеми (введення в дію генераторного агрегату для автономної роботи, ручне і автоматизоване управління паралельною роботою дизель-генераторів, виникнення умов спрацювання систем захистів дизель-генераторів та ін.) і далі створення декількох наборів пар «мнемосхема електроенергетичної системи для програмного забезпечення з усіма необхідними налаштуваннями – імітаційна модель електроенергетичної системи», що містять по одній ситуації з кожного класу для першого набору, по дві – для другого і т.д. Для забезпечення ефективного навчання імітаційна модель електроенергетичної системи повинна відтворювати зовнішні та внутрішні зв'язки, відповідні вихідному об'єкту з максимальною точністю. АСК, що розглядається, є багаторівневою і просторово розподіленою. Тому необхідно створити імітаційні моделі об'єктів управління спільно з їх системами управління нижнього і середнього рівнів як єдиного цілого, які на рівні інтерфейсу зв'язку з програмним забезпеченням автоматизованого робочого місця оператора будуть повністю відповідати апаратним засобам автоматизації середнього рівня. Відтворення внутрішніх зв'язків електроенергетичної системи моделлю має бути виражене у вигляді адекватного математичного моделювання електроенергетичних процесів у системі, що виникають при зміні її структури: стрибків струму при недотриманні умов підключень генераторного агрегату до суднової мережі, провалів напруги і частоти при підключенні потужних споживачів електроенергії, неузгодженості у розподілі навантаження між паралельно працюючими генераторними агрегатами при зміні навантаження енергосистеми та ін. Оскільки математичне моделювання електроенергетичних процесів досить ресурсоємний процес, а імітаційні моделі повинні здійснювати інформаційний обмін з програмним забезпеченням верхнього рівня з адекватними, відповідними реальним засобам, часовими затримками, в роботі пропонується створення мультикомп2ютерної моделі електроенергетичної системи, керування роботою якої буде виконуватися від інструкторської станції. Тоді мультикомп2ютерна модель електроенергетичної системи буде складатися з інформаційних моделей, відповідних контролеру автоматизації, системі синхронізації і розподілу навантажень, системі моніторингу якості електроенергії та пов'язаних з ними об'єктів автоматизації з автоматичними регуляторами частоти й напруги.

Завданнями інструкторської станції є моделювання зміни навантаження енергосистеми і максимально точне відтворення параметрів відповідних електроенергетичних процесів. Для вирішення останнього завдання необхідна наявність математичної моделі і засобів для розрахунку системи диференціальних рівнянь, які описують поточну конфігурацію суднової електроенергетичної системи.

Оскільки виконання розрахунків передбачає використання чисельних методів, які вже реалізовані в програмних комплексах Matlab Simulink і Scilab Scicos, в роботі пропонується до кожної пари «мнемосхема електроенергетичної системи – набір імітаційних моделей об'єктів і апаратних засобів автоматизації» додати імітаційну модель всієї електроенергетичної системи у вигляді графічної блок-діаграмми, створеної в одному з перерахованих програмних засобів. До складу цієї моделі повинні увійти блоки силових агрегатів, засобів автоматизації (регуляторів частоти і напруги, пристроїв синхронізації і розподілу навантажень), еквівалентних активно-індуктивного поточного і що підключається навантажень електростанції. Для можливості управління конфігурацією моделі також необхідний відповідний блок, який керуватиме засобами автоматизації і станами автоматичних вимикачів.

Для автоматизації процесу навчання пропонується для кожної зв'язки «мнемосхема – набір інформаційних моделей засобів і об'єктів автоматизації – модель електроенергетичної системи» скласти множину станів системи (автономна робота дизель-генератора, необхідність підключення додаткового дизель-генератора, перехід генераторного агрегату в руховий режим та ін.) і створити файл, кожен рядок якого буде містити відповідні певному стану енергосистеми дані про її конфігурацію: стани автоматичних вимикачів і потужності підключених до суднової мережі споживачів електроенергії. Тоді, використовуючи принцип особливих моментів в якості схеми моделювання, можна в певні моменти часу генерувати номер стану енергосистеми. Потім, використовуючи інформацію про відповідну цьому стану конфігурацію енергосистеми, можна виконувати запуск імітаційної моделі електроенергетичної системи і дані, які повертаються в результаті моделювання (діючі значення напруг і струмів, величини активних і реактивних потужностей, частоти струмів генераторних агрегатів) передавати далі імітаційним моделям об'єктів і засобів автоматизації. Таким чином, вивчення і закріплення основних алгоритмів керування судновою електроенергетичною установкою на первинному етапі підготовки оператора буде проходити відповідно до схеми, наведеної на рис. 7.32.

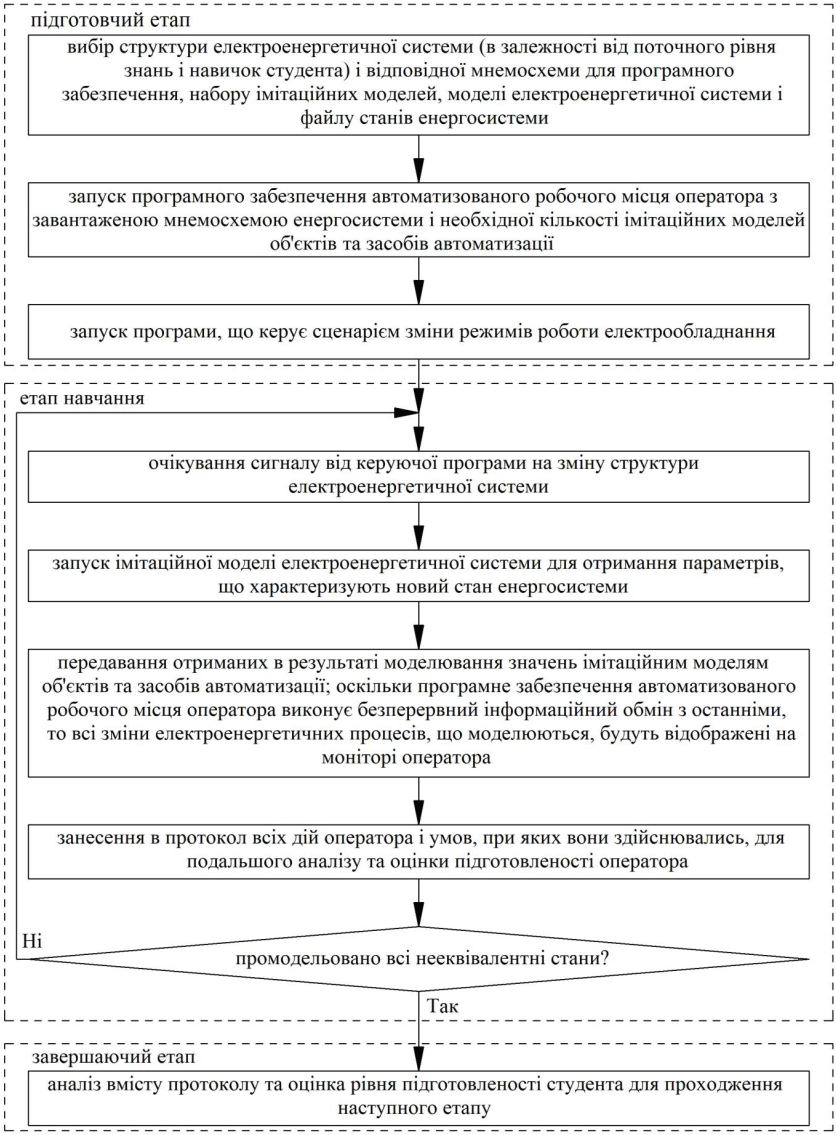


Рис. 7.32.

Завданнями вторинного етапу підготовки фахівців з експлуатації суднового електрообладнання є:

– вивчення схем підключення та алгоритмів налаштування автоматичних регуляторів частоти дизельних двигунів і напруги синхронних генераторів; систем синхронізації дизель-генераторів з мережею і розподілу навантажень між паралельно працюючими генераторними агрегатами;

– дослідження систем плавного пуску і гальмування, частотно-регульованих електроприводів асинхронних двигунів: вивчення структури та принципу роботи, схем підключення двигунів і керуючих сигналів, особливостей управління системами за допомогою апаратних засобів і аналогових або дискретних сигналів;

– дослідження контролерів автоматизації і систем моніторингу параметрів дизельних двигунів: вивчення структурних схем і принципів роботи, схемотехніки пристроїв і типових схем підключення, особливостей налаштування пристроїв для роботи в єдиній інформаційно-керуючій мережі розподіленої системи керування; вивчення типів і інтерфейсів підключення промислових датчиків температури, рівня та ін.;

– ознайомлення зі схемами підключення і принципами роботи програмованих логічних контролерів (ПЛК) та модулів розширення; вивчення мов програмування ПЛК та принципів об'єднання наведених пристроїв в підсистеми для реалізації складних локальних алгоритмів управління обладнанням;

– використання знань і навичок, отриманих в результаті вивчення основ роботи вищенаведених систем автоматизації, а також спеціалізованого програмного забезпечення для дослідження розподіленої системи керування фізичною моделлю суднової електроенергетичної системи.

Для вирішення поставлених завдань при навчанні студентів у роботі пропонується використання відповідним чином укомплектованих стендів. Для дослідження системи збудження синхронного генератора і синхронізації генераторного агрегату з мережею, а також принципів роботи з частотно-регульованими електроприводами, можливе використання стенду, що містить синхронний генератор і приводний асинхронний двигун потужністю 2 кВт, частотний електропривод Altivar 38, перераховані системи з виведеними контрольними точками для можливості регулювання їх налаштувань, універсальним вимірювальним приладом Lovato DMK32 і комп'ютера зі спеціалізованим програмним забезпеченням для зв'язку з мікропроцесорними системами. Вивчення систем плавного пуску і гальмування асинхронних двигунів можна проводити на основі стендів, укомплектованих двигунами, пристроями Altistart 01 і Altistart 48, тумблерами для зміни режимів роботи цих пристроїв (включення/відключення функції BOOST, аварійна зупинка двигуна), кнопками зміни і індикаторами відображення поточних станів систем, вимірювальними приладами для аналізу впливу зміни налаштувань систем плавного пуску на величини пускових струмів і сплесків потужності, персональними комп'ютерами з необхідними програмними засобами (програмою PowerSuite, засобами для зв'язку з осцилографами). Для

дослідження інтерфейсів і схем підключення промислових датчиків в лабораторії використані термоперетворювачі ТСМ фірми Овен і, оскільки інтерфейси всіх промислових датчиків стандартизовані, для вивчення принципів реалізації складних алгоритмів керування об'єктами автоматизації в залежності від зовнішніх умов реалізована імітація датчиків різних фізичних величин за допомогою змінних опорів. Для імітації датчиків з виходом типу «сухий контакт» використані тумблери і імпульсні кнопки; відображення станів дискретних виходів пристроїв реалізовано за допомогою світлодіодних індикаторів. Вивчення мов реалізації алгоритмів для ПЛК та принципів побудови підсистем, в яких ПЛК є головним, може відбуватися за допомогою стендів, укомплектованих засобами імітації датчиків і пристроями фірм Shneider Electric (TWDLMDA40DTK, TM2DRA8RT, TM2AMI8HT, TWBNOZ485D), Zelio Logic (SR2B201BD) і Овен (ПЛК100, МВА8, МДВВ, СП270, ЕКОН131). Після дослідження систем автоматизації на окремих стендах навчання переходить в завершальну стадію − відпрацювання навичок створення, налаштування і використання розподіленої системи управління, що складається з пульта управління з автоматизованим робочим місцем оператора (комп'ютерами зі спеціалізованим програмним забезпеченням, сенсорними панелями та аварійними вимикачами) на верхньому рівні; ПЛК, систем моніторингу параметрів мережі і синхронізації на середньому рівні; систем збудження генераторів та перетворювачів частоти на нижньому рівні. Всі наведені стенди реалізовані в лабораторії кафедри теоретичної електротехніки та електронних систем Національного університету кораблебудування ім. адм. Макарова та ознайомитися з їх комплектацією більш детально можна на сайті кафедри.

Поділ процесу навчання фахівців з експлуатації суднового електроустаткування на два послідовних етапи дозволить підвищити його ефективність шляхом інкрементального повторення алгоритмів керування судновою електроенергетичною системою. Для запобігання виникнення аварійних ситуацій запропоновано використання на першому етапі навчання інформаційних моделей апаратних засобів і об'єктів автоматизації спільно із засобами автоматизованого робочого місця оператора. Другий етап навчання передбачає покрокове вивчення апаратних засобів автоматизації на локальних стендах і далі – у складі розподіленої системи управління фізичною моделлю суднової електроенергетичної системи.

СПИСОК ЛІТЕРАТУРИ

1. Рябенький В. М., Драган С. В., Солобуто Л. В. Основи моделювання систем і процесів в електротехніці. (Використання пакета прикладних програм MATLAB/Simulink) : навч. посібник. Львів : Новий Світ-2000, 2008. 385 с.

2. Буч Р., Якобсон А., Рамбо Дж. UML. Классика CS. 2-е изд. СПб.: Питер, 2006. 736 с.

3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования, паттерны проектирования. СПб.: Питер, 2001. 368 с.

4. Гнеденко Б. В., Коваленко И. Н. Введение в теорию массового обслуживания. 3-и изд., испр. и доп. М.: Эдиториал УРСС, 2005. 400 с.

5. Горелик Г. Б. Автоматизированные системы управления судовыми энергетическими установками. Хабаровск: Изд-во ХГТУ, 2001. 21 с.

6. Дубинин В. Н., Вяткин В. В. Проектирование распределенных систем управленния промышленными процессами с использованием UML-FB // Известия высших учебных заведений. Поволжский регион. Технические науки. 2004. № 2. С. 136–146.

7. Дьяконов В., Круглов В. Математические пакеты расширения Matlab: специальный справочник. СПб.: Пітер, 2001. 480 с.

8. Зайцев Д. А., Шинкарчук Т. Н. Моделирование телекоммуникационных сетей в системе NS // Наукові праці ОНАЗ ім. О. С. Попова. 2006. № 2. С. 35–43.

9. Коваленко І. І., Рябенькій В. М., Кудін О. А. Паттернові мережі та деякі приклади їх застосування. *Проблеми інформаційних технологій*. № 2(4). Херсон: Вид-во ХНТУ, 2008. С. 170–175.

10. Коваленко І. І., Кудін О. О. Подання структур обчислювальних систем патерновими мережами. *Збірник наукових праць НУК*. Миколаїв: НУК, 2009. № 6(429). С. 164–172.

11. Коваленко І. І., Бідюк П. І., Гожий О. П. Вступ до системного аналізу. Миколаїв: Видавництво МДГУ ім. Петра Могили, 2004. 146 с.

12. Олзоева С. И. Распределенное моделирование в задачах разработки АСУ. Улан-Уде: изд-во ВСГТУ, 2005. 219 с.

13. Рябенький В. М., Ушкаренко О. О. Verilog. Практика проектування цифрових пристроїв на ПЛІС. Миколаїв: Іліон, 2007. 324 с.

14. Рябенький В. М., Ходаков В. Е. Ушкаренко А. О. Комп'ютерне керування зовнішніми пристроями через стандартні інтерфейси. Херсон: Олді-плюс, 2008. 380 с.

15. Рябенький В. М., Юрченко О. М., Ушкаренко О. О., Сопель М. Ф. Інформаційне забезпечення розподілених систем керування автономними електроенергетичними установками: монографія. Київ: Інститут електродинаміки, 2012. 208 с.

16. Рябенький В. М., Юрченко О. М., Ушкаренко О. О., Сопель М. Ф. Оптимізація систем керування автономними електроенергетичними установками з газодизель-генераторними агрегатами: монографія. Київ: Інститут електродинаміки, 2013. 226 с.

17. Схемотехніка електронних пристроїв та систем: у 6 т. Том 2: Прикладна теорія цифрових автоматів / В. М. Рябенький, М. Т. Фісун, О. О. Ушкаренко. Миколаїв: Вид-во ЧДУ ім. Петра Могили, 2010. 380 с.

18. Схемотехніка електронних пристроїв та систем: у 6 т. Том 3: Мікропроцесорна техніка / В. М. Рябенький, О. О. Ушкаренко, В. С. Буряк. Миколаїв: Іліон, 2012. 445 с.

19. Схемотехніка електронних пристроїв та систем: у 6 т. Том 6: Апаратно-програмні засоби відображення інформації / В. М. Рябенький, О. О. Ушкаренко. Миколаїв: Іліон, 2013. 464 с.

20. Рябенький В. М., Жуйков В. Я., Ямненко Ю. С., Борисов О. В. Схемотехніка: пристрої цифрової електроніки: підручник: у 2 т. Т. 1. Київ: НТУУ «КПІ», 2015. 399 с.

21. Рябенький В. М., Жуйков В. Я., Ямненко Ю. С., Борисов О. В. Схемотехніка: пристрої цифрової електроніки: підручник: у 2 т. Т. 2. Київ: НТУУ «КПІ», 2015. 360 с.

22. Рябенький В. М., Ушкаренко О. О. Програмна реалізація алгоритмів цифрової обробки сигналів: монографія. Миколаїв: НУК, 2016. 244 с.

23. Рябенький В. М., Дорогань О. І., Ушкаренко О. О. Автоматизація процесів моніторингу параметрів та керування автономними електроенергетичними системами з дизель-генераторними агрегатами: монографія. Миколаїв: НУК, 2017. 240 с.

24. Рябенький В. М., Ушкаренко О. О., Островерхов М. Я., Махмуд М. С. Аль-Суод. Програмна реалізація методів розрахунку та аналізу електричних кіл: навч. посіб. Київ: Видавництво Ліра-К, 2021. 462 с.

25. Голишев Л. К. Системний підхід до аналізу та проектування складних систем. Системний проект: монографія. К.: Інформ.-аналіт. агентство, 2011. 554 с.

26. Стеценко І. В. Моделювання систем: навч. посіб. Черкаси: ЧДТУ, 2010. 399 с.

27. Черных И. В. Simulink: среда создания инженерных приложений. М.: Диалог-МИФИ, 2003. 496 с.

28. Щерба А. А., Рябенький В. М., Кучеренко М. Є., Побєдаш К. К., Чибеліс В. І., Кінаш А. Т., Солобуто Л. В. Електротехніка та електроніка. Київ: Корнійчук, 2007. 490 с.

29. Campbell St. L., Chancelier J.-P., Nikoukhah R. Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4. New York: Springer, 2006. 330 p.

30. Gomma H. Designing Concurrent, Distributed, and Real-Time Applications with UML. New York: Addison-Wesley, 2013. 1060 p.

31. Gardner C., Johnson D., Provine J. Networked Intelligent Instrumentation & Control for Switchboards. *Electric Ship Technologies Symposium.* USA, 2007. P. 510–518.

32. Математичні моделі та методи аналізу електронних кіл: навч. посібник / Ю. Я. Бобало та ін.; за ред. д-ра техн. наук, проф. Ю. Я. Бобала та д-ра техн. наук, проф. Б. А. Мандзія. Львів: Видавництво Львівської політехніки, 2013. 320 с.

33. Учебное пособие по решению задач по теоретической электротехнике. Часть II / под общей ред. доц. А. В. Корощенко. Донецк: ДонНТУ, 2008. 237 с.

34. Рибалко М. П., Есауленко В. О., Костенко В. І. Теоретичні основи електротехніки: лінійні електричні кола: підручник. Донецьк: Новий світ, 2003. 513 с.

35. Pierre Brémaud. Markov chains., New York: Springer-Verlag, 1999. 455 p.

36. Søren Asmussen. Applied Probability and Queues. Science & Business Media, 2003. 438 p.

37. Son, Seok-Geum. (2016). A Study on Improving High-Power Induction Motor Starting. *The Journal of Korea Institute of Information, Electronics, and Communication Technology*. 9. 178-184. 10.17661/jkiiect.2016.9.2.178.

38. Kim, Jong-Gyeum. (2020). A Study on the Variation of the Power Factor according to the Multiples of the 3-phase Induction Motor Starting Current. *The transactions of The Korean Institute of Electrical Engineers*. 69. 1773–1777. 10.5370/KIEE.2020.69.11.1773.

39. Wu, Di & Wu, Hongbin & Dongt, Hao. (2018). Influence of Induction Motor Starting on Microgrid. 376-381. 10.1109/APPEEC.2018.8566305.

40. Hernández-Guzmán, Victor & Silva-Ortigoza, Ramón & Orrante-Sakanassi, Jorge. (2021). Energy-Based Control of Electromechanical Systems: A Novel Passivity-Based Approach. 205–276. 10.1007/978-3-030-58786-4.

41. Palka, Ryszard & Woronowicz, Konrad. (2021). Linear Induction Motors in Transportation Systems. *Energies*. 14. 2549. 10.3390/en14092549.

42. Mirošević, Marija & Sumina, Damir & Bulić, Neven. (2011). Impact of Induction Motor Starting on Ship Power Network. *International Review of Electrical Engineering*. 6. 186–197.

43. Son, Seok-Geum. (2017). Induction Motor Starting Characterization with Power Factor Correction Capacitors. *The Journal of Korea Institute of Information, Electronics, and Communication Technology*. 10. 206–212. 10.17661/jkiiect.2017. 10.3.206.

44. Liang, Xiaodong & Ilochonwu, Obinna. (2011). Induction Motor Starting in Practical Industrial Applications. *IEEE Transactions on Industry Applications*. 47. 271–280. 10.1109/TIA.2010.2090848.

45. Council of European Energy Regulators. (2012). Guidelines of Good Practice on the Implementation and Use of Voltage Quality Monitoring Systems for Regulatory Purposed, Ref: C12-EQS-51-03.

46. ISO 50001. (2011). Energy Management Systems – Requirements with guidance for use.

47. Brémaud, P. (2020) Generalities on Random Processes. *Probability Theory and Stochastic Processes*. Universitext. Springer, Cham. 199–220. 10.1007/978-3-030-40183-2_5.

48. Korobko, Grigory & Korobko, Ivan & Shilov, Maksim. (2018). Application in autonomous power plants of a diesel generator set with a variable frequency of rotation. *Vestnik of Astrakhan State Technical University*. Series: Marine engineering and Technologies. 42–48. 10.24143/2073-1574-2018-4-42-48.

49. Zabihian, Farshid. (2021). Power Plant Engineering. *CRC Press*. 1250. 10.1201/9780429069451.

50. Gomez, J. & Reineri, Claudio & Campetelli, G. & Morcos, M. M. (2004). A Study of Voltage Sags Generated by Induction Motor Starting. *Electric Power Components and Systems*. 32. 645–653. 10.1080/15325000490228423.

51. Mahmoud M. S., Al-Suod & Ushkarenko, O. O. (2016). Analytical Representation of Control Processes of Induction Motor and Synchronous Generator in Power Plants. *Jordan Journal of Electrical Engineering*. 2. 4. 278–288.

52. Mahmoud M. S., Al-Suod & Ushkarenko, A. (2017). Optimization of model structure of induction motor control system. *WSEAS Transactions on Power Systems*. 12. 316–323.

53. Mehdi R. Zargham. Computer Architecture: single and parallel systems. N.J.: Prentice-Hall International, Inc. 1996. 472 p.

54. Гома Хасан. UML. Проектирование систем реального часу, распределенных и параллельных приложений / пер. с англ. М.: ДМК Пресс, 2002. 704 с.

55. Шаповалова Е. А. Моделирование с использованием языка UML. Пакет Rational Rose [Электронный ресурс]. URL: https://masters.donntu.ru /2005/kita/shapovalova/ind/index.htm?ysclid=lr9g300de9796804865.

56. Ушкаренко О. О. Концептуальна модель системи моніторингу за станом автономної електроенергетичної установки. *Інновації в суднобудуванні та океанотехніці*: матеріали ІІ міжнародної науково-технічної конференції. Миколаїв: НУК, 2011. С. 541–544.

57. Рябенький В. М., Ушкаренко О. О. Метод структурного синтезу та аналізу моделей пристроїв автономної електроенергетики. *Технічна електродинаміка*. Тем. випуск. 2012. Ч. 4. С. 130–133.

58. Муха Н. І., Дранкова А. О., Волошин В. М., Миська А. Р., Дудко С. А. Тренажер суднової автоматизованої електроенергетичної системи. *Авіаційно-космічна техніка та технологія*. Харків, 2011. № 9(86). С. 207–210.

59. Рябенький В. М., Ушкаренко О. О., Воскобоєнко В. І. Моделювання мікропроцесорних систем управління газодизель-генераторними установками. *Вісник Національного університету «Львівська політехніка»*. 2009. № 637. С. 78–82.

60. Рябенький В. М., Ушкаренко О. О., Сулейман А. Знаходження оптимальної стратегії доступу до каналу зв'язку в бездротових мережах. *Проблеми інформаційних технологій*. 2008. № 4. С. 176–180.

61. Рябенький В. М., Ушкаренко О. О., Нгуєн Ван Тхань. Дослідження провалів і сплесків напруги в автономних електроенергетичних системах. *Технічна електродинаміка*. Тем. випуск. 2010. Ч. 1. С. 182–186.

62. Рябенький В. М., Ушкаренко О. О., Сулейман А. Дослідження аналітичної моделі мережі при зміні параметрів інформаційних потоків. *Вістник ХНТУ*. 2010. № 2(38). С. 250–253.

63. Рябенький В. М., Ушкаренко О. О., Аль-Суод Махмуд Мохаммад, Халед Омар Ганнам. Імітаційне моделювання процесів комутації споживачів в автономних електроенергетичних установках. *Збірник наукових праць НУК*. 2011. № 4. С. 123–129.

64. Рябенький В. М., Ушкаренко О. О., Халед Омар Ганнам Моделювання гібридних компонентів системи управління автономною електроенергетичною установкою. *Технічна електродинаміка.* Тем. випуск. 2012. Ч. 2. С. 127–130.

65. Рябенький В. М., Ушкаренко О. О., Дорогань О. І. Моделювання режимів робіт суднових споживачів електроенергії з використанням діаграм станів. *Енергозбереження. Енергетика. Енергоаудит.* Спец. випуск. 2013. Т. 2. № 8(114). С. 84–89.

66. Рябенький В. М., Ушкаренко О. О., Язід Джамал Ісмаїл Альшайх. Комп'ютерна модель автономної електроенергетичної установки для дослідження процесу синхронізації генераторів. *Проблеми інформаційних технологій*. 2014. № 2(016). С. 62–69.

67. Рябенький В. М., Ушкаренко О. О., Воскобоєнко В. І., Нгуєн Ван Тхань. Оптимізація параметрів регулятора системи збудження при різкозмінних навантаженнях. *Інновації в суднобудуванні та океанотехніці*: матеріали II міжнародної науково-технічної конференції. Миколаїв: НУК, 2011. С. 533–537.

Навчальне видання

Р Я Б Е Н Ь К И Й
Володимир Михайлович
У Ш К А Р Е Н К О
Олександр Олегович
А Л Ь - С У О Д
Махмуд Мохаммад Салем

МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ, ЕЛЕКТРОТЕХНІЧНИХ ТА ЕЛЕКТРОЕНЕРГЕТИЧНИХ СИСТЕМ

Навчальний посібник

Рисунки – О. Ушкаренко

Формат 60×84$^{1}/_{16}$. *Ум. друк. арк.* 24,6. *Тираж* 100 пр. *Зам.* № 534-717.

В И Д А В Е Ц Ь І В И Г О Т О В Л Ю В А Ч
Товариство з обмеженою відповідальністю фірма «Іліон».
54038, м. Миколаїв, вул. Бузника, 5/1.
Свідоцтво суб'єкта видавничої справи ДК № 1506 від 25.09.2003 р.